Interpolation and Interpolation-Hash Searching

Gaston H. Gonnet

Department of Computer Science
University of Waterloo

Research Report CS-77-02

January 1977.

Work supported in part by Canadian
National Research Council grant A8984.

Interpolation and Interpolation-Hash Searching

Gaston H. Gonnet

Department of Computer Science
University of Waterloo

Research Report CS-77-02

January 1977.

Work supported in part by Canadian
National Research Council grant A8984.

Interpolation and Interpolation Hash Searching

Gaston H. Gonnet

Abstract

Interpolation search is an algorithm to search a table containing numerically ordered keys using the value of the key and assumptions on the distribution of keys in the table to select each probe position.

First we describe the model of a table we will use, consisting of random uniform ordered real variables, together with some immediate generalizations to arbitrary continuous distributions as well as to discrete distributions.

The general interpolation search scheme is presented with the corresponding algorithm. Some restrictions of the interpolation formula lead to the definition of coherent algorithms.

The main emphasis is given to the simplest of the linear interpolation algorithms, for which we derive: average asymptotic behaviour $O(\log_2(\log_2(n)))$ (done by a conventional approach and also by information theory); the exact average behaviour and the complexity of its computation; an asymptotic probability density function of the number of probes to find an element, from which we derive that its variance is $O(1)$, later corrected to obtain better approximations; numerical approximations to the exact average number of probes and simulations that confirm the theoretical results. Several results concerning asymptotic expansions of probability distributions and entropies are derived as a by product.

Some variations of the algorithm are studied in detail: binary interpolation search, the optimal search algorithm, the unsuccessful search case which is also $O(\log_2(\log_2(n)))$, the optimal known successful search, binary interpolation search, and the one-interpolation then-sequential algorithm, the latter being $O(n^{1/2})$.

The second main section merges the interpolation search algorithm idea with hash searching techniques to produce an interpolation-hash searching algorithm. The search algorithm is presented as well as an approximate construction algorithm. The optimal configuration, addition and deletion of elements, and the complexity of the algorithm are discussed. The method has the advantages of being $O(1)$ for any fixed occupancy rate (as are hash coding techniques) and $O(\log_2(\log_2(n)))$ for a full table, keeping the table in order. Simulation results are presented to corroborate the theoretical ones.

Table of Contents

1. - Introduction.

The main purpose of this thesis is a detailed study of the interpolation search techniques. This involves the description of the basic algorithm, the complexity analysis of it and some related algorithms, as well as some extensions that lead to the definition of a special kind of hash searching.

The interpolation search algorithm can be used to search ordered tables. The search procedure involves, at each step, the selection of a probe position based on the value of the searched key and the values of the keys that delimit the subfile. This selection is usually accomplished by linear interpolation. After the selected location is inspected, we either have found the element or we repeat the search in the corresponding subfile.

The search is so natural when dealing with numerical keys, that it makes little sense trying to attribute it to somebody. W.W. Peterson [1957] was the first one to describe the search algorithm in the published literature and he attempted to compute its running time by computing the uncertainty of a file under consideration. Unfortunately there is no detailed description or code for the algorithm, and this approximation to the entropy contains an error. In a survey of table lookup techniques, Price [1971] describes a variation of the interpolation search algorithm that we describe as one-interpolation-then-sequential. There is no detailed description or code of the algorithm. Price also considers the case when the distribution of the key values is not uniform, and suggests possible remedies by approximation with polynomials. In chapter III of D.E. Knuth's book [1973], we find only a brief verbal description of the algorithm without much enthusiasm. H.S.M. Kruijer [1974] presents a paper on Interpolation Search in which he describes the algorithm in words and with some detail. He claims that "in certain cases [the algorithm] behaves better than binary search". He also points out the relation between the search problem and the zerofinding problem. R.P. Brent [1973] also realizes this latter relation in his book and defines a function for each searched key, whose zero is its index. The author wrote a term paper on search techniques [Gonnet 72], that remains unpublished, in which the $\log_2(\log_2(n))$ average behaviour of interpolation search was proven with the aid of information theory. Whitt and Sullenberger [1975] discuss an application of interpolation for external search. Unfortunately there is no detailed description of the algorithm or analysis of its running time. Recently A.C. Yao and F.F. Yao presented a paper [1976] that shows an upper and a lower bound of $\log_2(\log_2(n))+O(1)$ on the average number of accesses.

It is important to note that there is no complete and correct description of the algorithm in the literature. The algorithm, however, is not trivial, and difficult to code correctly without some insight into the problem. It is a particularly loop-prone algorithm. This fact together with the requirement of multiplication and division are probably the causes of the lack of interest in the algorithm.

In section 2 we define the basic tools needed to study the algorithm. Section 3 contains a study of the interpolation search algorithm for full tables. We derive correct versions of the algorithm, exact running times, asymptotic behaviour, empirical formulas and simulation results. Some variations of the basic algorithm are also studied. Section 4 is concerned with the use of the interpolation search algorithm for partially filled tables. This algorithm may be visualized also as a particular hash coding technique. The algorithms for construction, searching, addition and deletion are studied as well.

Finally appendix I contains a collection of asymptotic expansions used in connection with the analysis in section 3.

It is customary in a Ph.D. thesis to state explicitly which results are original and presented here for the first time. All the results for which there is no explicit reference given in the text are, to the best of my knowledge, new. Clearly not all of them are of the same importance. The most significant new results are in chapters three and four, that except for the concurrent work of A.C. Yao and F.F. Yao [1976] contain original results. Some results in chapter two, notably the lower bounds on hash searching as well as other formulae plus most of the non trivial asymptotic expansions in Appendix I are also new.

2. - Basic Concepts.

In this chapter we will define the basic assumptions, definitions and common tools we will use later. First we define the model of the search procedure we will analyze. Secondly we will review some theoretical lower bounds on the number of accesses required by different algorithms. This will give us the idea of which algorithms cannot be better than interpolation search. In the third part we discuss the general problem of real distributions and how to translate them to the uniform case. Section 4 contains some basic as well as specific probability tools to deal with ordered random variables, and finally in Section 5 we consider the discrete model of keys.

2.1 - Definitions.

Notation.

The notation is consistent throughout the chapters. For most of the well known constants and functions we adopt the notation used by Abramowitz and Stegun [1964], with the exception of $lg(x)$ that will denote the base 2 logarithm of x.

The notation for probability functions is fairly standard; in all cases it is the same as Johnson and Kotz [1969].

Capital letters X and Y, possibly subscripted, denote an instance of a random variable. Their most common appearance is as keys of a file or table.

Consistently we use α as the name of the key for which we are searching. Any probability that is a function of α should be interpreted as: probability of ... given that we are searching for α.

Particular functions defined inside chapters use upper case letters e.g. A, B, C, or lower case greek, e.g. $\phi(...)$.

Inside programs we will use bold face type to denote all keywords of the language.

Formal Definitions.

We will define a *file* or *table* as a finite homogeneous collection of records or elements. A *record* or *element* is a nonempty set of not necessarily homogeneous variables. There exists at least one variable in a record called the *key* that serves to identify the record.

A record or element has an *index*, which is its ordinal number in the file. The index will range from 1 to n.

The record with index i will be denoted by R_i. The key value of R_i will be denoted K_i.

The *search problem* consists of: given a variable α, either find at least one i such that $K_i=\alpha$, or prove that there is no i such that $K_i=\alpha$. The first outcome will be called the *successful* search, the latter the *unsuccessful* search.

The *known successful* search problem is: find i such that $K_i=\alpha$ knowing that there exists at least one i such that $K_i=\alpha$.

The *known unsuccessful* search is: prove that there is no i such that $K_i=\alpha$, given that α is not one of the keys. The above definitions sound trivial in the mathematical sense. However we should understand "find" or "prove" as algorithmic actions rather than theorems.

We define a file to be *ordered* when there exists an order relation R such that $K_i \, R \, K_{i+1}$ for all i such that $1 \leqslant i < n$.

We define an *access* to an element or record in the file as any use of any component of the record by the algorithm. Our measure of *complexity* of a search algorithm is defined by the number of accesses required to answer a search problem.

Other measures of complexity may be relevant or interesting. They are, however, either closely related to the implementation or easily derived from the one analyzed here. E.g. number of multiplications and divisions; index computations etc. Consequently we will restrict our study to the number of accesses.

We will not base our conclusions or study on real files, but on models of files. Moreover, the search procedures are concerned only with the key portion of the record, so hereafter we will ignore non-key parts of the record.

We will *model* our files with a set of n independent random variables from the same probability distribution. If the files under consideration are ordered, we relabel or sort the keys according to the order relation. There are some other ways to model files (or ordered files) e.g. $K_i = K_{i-1}$ + {independent random variable} etc., but we feel that the above modelling is the one that reflects more closely real files, besides being almost a standard for complexity analyses.

We will refer to the number of accesses needed to solve a search problem for a given key α and a given file as the *number of accesses*. We can define several possible *average number of accesses*. We will restrict ourselves to the *average number of accesses* **given** α for any instance of our model, and the average number of accesses **for any** α and any instance of our model.

To find the latter we will need to know or assume a particular distribution of successful and unsuccessful searches. Since algorithms may differ for these situations, and so their analyses, both averages will be independently computed for the successful and unsuccessful case. This provides more information than assuming a distribution and giving a single answer.

The *uncertainty* of the position in a successful search is the information-theoretic measure defined by the entropy of the index i given α. Formally the uncertainty will be defined by the entropy of a random source [Ash 64] that produces i (index of α) for any random file of size n ($1 \leqslant i \leqslant n$) that contains α. Uncertainties will be normally measured in bits.

2.2 - Theoretical Lower Bounds.

In this subsection we will analyze average lower bounds on the number of accesses required for some search algorithms that are comparable to interpolation search. Later we can compare these lower bounds with the actual behaviour of interpolation search and decide which algorithms will never be better.

In all the following models we will use the three way comparison of keys, i.e. we decide on $X>\alpha$, $X=\alpha$ or $X<\alpha$. We will further assume that each comparison requires one access to the record.

Lower Bound on Binary Search.

To find the lower bound on the average number of comparisons we construct, for any algorithm, the corresponding decision tree. This tree will have 3 branches at each node, one of which (the equal outcome) cannot be spanned further.

First we will consider the successful search case. The tree must have at least n leaves. It is easy to show [Knuth 73] that the shortest tree will contain non-central leaves only at two levels. Moreover if

(2.2.1) $\quad h = \lfloor lg(n) \rfloor,$

where h is the height of the tree, there are $2^{h-1}-1$ interior nodes (branches on =),

(2.2.2) $\quad 2^{h-1}-\lceil (n-2^h+1)/2 \rceil$ leaves at level h-1, and

(2.2.3) $\quad \lceil 3(n-2^h+1)/2 \rceil$ leaves at level h.

Finally we compute the average number of accesses to be

(2.2.4) $\quad E[\text{accesses}] = (h+1+nh-2^{h+1}+\lceil (n+1)/2 \rceil)/n \sim h-\frac{1}{2},$ or

(2.2.5) $\quad h-\frac{1}{2} \leqslant E[\text{accesses}] < h.$

This result differs from the one in Knuth [1973] because we are considering a lower bound and do not require some redundant comparisons.

The unsuccessful case is analyzed in the same way, except that now each node of the tree has only two possible branches and the complete tree has to have $n+1$ leaves. The height of the optimal tree is $h+1$.

There are $2^{h+1}-n-1$ leaves at level h and $2n+2-2^{h+1}$ leaves at level $h+1$ giving an average number of accesses of

(2.2.6) $E[\text{accesses}] = h+2-2^{h+1}/(n+1)$, or

(2.2.7) $h+1 \leqslant E[\text{accesses}] < h+3/2.$

Lower Bound for Hash Coding with Separate Overflow Chaining.

This algorithm has several implementations [Knuth 73]. In all cases the collisions are stored in a linked list. This implies that after the first probe we search sequentially through all the records that "hashed" to the same place. The average behaviour or lower bounds for this algorithm are not affected by the order in which the table is created.

Let β denote the *occupation factor* i.e. {number of records} / {size of table}. Assuming that the hash function is uniform enough so that the number of elements that hash to a given position are Poisson distributed we derive

(2.2.8) $$E[\text{accesses}] = \tfrac{1}{2}\sum_{i=1}^{\infty}\beta^i e^{-\beta} i(i+1)/i! \;/\; \sum_{i=1}^{\infty}\beta^i e^{-\beta} i/i!$$

$$= 1 + \beta/2,$$

for the successful search and

(2.2.9) $$E[\text{accesses}] = e^{-\beta} + \sum_{i=1}^{\infty}\beta^i e^{-\beta} i/i!$$

$$= \beta + e^{-\beta},$$

for the unsuccessful case.

For the unsuccessful case we are assuming that we can detect whether an element is the head or the tail of its corresponding list with only one access.

Lower Bounds for Hash Coding.

We will consider here the hash coding search without any clustering [Knuth 73] of n records placed in a table of m entries. The occupation factor is defined as $\beta = n/m \leqslant 1$.

In this case the order in which the table is created alters the properties of the average and maximum number of accesses for the successful case. Consequently we need to define and analyze two of the possible optimums of this hashing technique and then conclude its bounds.

The *minimax* hash coding problem is defined as follows: given a file and a hashing function, find the order in which elements have to be inserted so that the maximum number of accesses to locate any single element is minimized. This minimum maximum number of accesses will be called the *minimax* value.

We will first consider the case of a full table, i.e. $\beta=1$. We will require that the hashing function produce for any random key, an independent sequence of probes distributed discrete rectangular in (1,m). This is slightly different from the usual hashing functions since we allow probe positions to be repeated.

A necessary, but not sufficient condition, to solve the minimax problem with k accesses is that the k×n probe positions "occupy" all possible values from 1 to m. Since this is a necessary condition, minimax$\geqslant$k.

Given k, the probability of all table positions (1,m) appearing in between k×n probe positions, is an occupancy distribution, also known as Arfwedson's distribution [Arfwedson 51, Stevens 37]

$$(2.2.10)\quad \Pr\{x\leqslant k\} = \sum_{i=0}^{n}(-1)^i\binom{n}{i}(1-i/n)^{kn}$$

$$\sim (1-e^{-k})^n.$$

The expected value of x for the above distribution is

$$(2.2.11)\quad E[x] = \sum_{k=0}^{\infty} k[\Pr\{x\leqslant k\}-\Pr\{x\leqslant k-1\}]$$

$$= -\sum_{k=0}^{\infty}\sum_{i=1}^{n}(-1)^i\binom{n}{i}(1-i/n)^{kn}$$

$$= -\sum_{i=1}^{n}(-1)^i\binom{n}{i}[1-(1-i/n)^n]^{-1}$$

$$= ln(n) + 1.07... + o(1).$$

When we do not have a full table, using the expected value of the occupancy distribution

(2.2.12) $E[x] = m[1-(1-1/m)^{nk}]$

we can approximate the expected value of the lower bound on the minimax from

(2.2.13) $n = m[1-(1-1/m)^{nk}] \Longrightarrow$

$$k \sim -\beta^{-1}ln(1-\beta).$$

The above approximation is valid when $m\rightarrow\infty$ and $0<\beta<1-\epsilon$.

The second possible optimum is the average number of accesses. The problem is then to find an order in which elements have to be inserted so that the average number of accesses is minimized. We will find a lower bound on the number of accesses for the case of a full table. In this case our model of the hashing function is one that produces, for each key, a random permutation of the locations 1 through m.

Since the table is full, n=m, each location must contain a key. Consequently, whatever the order of insertion is, the location i will contain a key that needed a number of accesses greater or equal to the smallest order of appearance of i in any hash probe sequence, i.e. in any of the n permutations of 1 through m. The probability that location i does not appear in position 1,2,..,x of any random permutation is

(2.2.14) Pr{first appearance of i is after x probes} = $[(m-x)/m]^m$.

The expected value of the "first appearances" is a lower bound of the expected value of the number of accesses and is equal to

(2.2.15) $E[\text{first appearance}] = \sum_{k=0}^{m-1}[(m-k)/m]^m = e/(e-1) + O(m^{-1}) \sim 1.5819...$

This lower bound can be further improved if we consider the collision of unique "first appearances". More precisely, we define a "first appearance" of position i in x probes as

critical when only one key probes to i in x probes. For each collision (two critical) we will need at least on more access to fill both critical table positions. Counting collisions we obtain

(2.2.16) $E[\text{collisions}] = \Pr\{2 \text{ critical for each key}\} + 2\times\Pr\{3 \text{ critical}\} + 3\times\ldots$

$= E[\text{critical}] - 1 + \Pr\{\text{no critical}\}$

$= (e-1)^{-1} - 1 + \prod_{i=1}^{\infty}(1-e^{-i}) + O(m^{-1}).$

The lower bound is now

(2.2.17) $2/(e-1) + \prod_{i=1}^{\infty}(1-e^{-i}) \sim 1.66838\ldots$

The unsuccessful search case depends only on the load factor of the table. Under the natural assumption that probe positions do not repeat, the average number of accesses needed to complete the unsuccessful search is equal to the number of access needed to find an empty position, i.e. [Knuth 73]

(2.2.18) $E[\text{accesses}] = \sum_{k=1}^{n} k(m-n)n!(m-k)!/[(n-k+1)!m!]$

$= (m+1)/(m-n+1) \sim (1-\beta)^{-1}$ if $m>n$,

$= n$ if $m=n$.

2.3 - Distribution of Keys.

In the following sections we will assume that the keys of the record are independently generated from a U(0,1) distribution.

In this section we will analyze how real keys relate to the probability distributions and how we can relate these distributions to the U(0,1).

In actual applications, keys are far from being random variables; they are assigned or derived from records and play an important role in the identification of the record. We will now formalize some concepts about keys and their relation to probability distributions. We will call the set of all possible key values the *domain of the keys*. The keys of a certain file are a subset (almost always a proper subset) of the domain. The *key probability distribution* specifies, for each key in the key domain, the probability that a randomly selected record has that key value. The file contains records selected independently of the distribution of the keys.

To clarify these definitions, let us follow a complete example. Let the record be a collection of information that partially describes a living human being. Our key will be its birthdate (for example taken in Julian form). The *domain* of the key is the set of all integers less than or equal to the current date. The *probability distribution of the keys*, for this example, is a composition of a birth rate and a mortality distribution. A file is a set of such records. This distribution applies to a file as long as we do not choose the records based in any way on people ages; i.e. the subset is independently chosen from the distribution of the keys.

We find some problems with our model. Real numbers do not exist in the practical world, only approximations of them. Secondly the model presented fails to describe situations in which keys are interdependent; e.g. assignment of *different* keys to each member of the population (sampling without replacement). Finally, although distributions may exist, they may be constantly changing, as in the example above, and/or may be very difficult to obtain.

The second step to convert the key probability distribution to a U(0,1), is almost trivial. We will assume that the key cumulative probability distribution is continous and has an inverse. Let Y be a key, and let

$$F(a) = \Pr\{Y \leqslant a\}. \tag{2.3.1}$$

If we define the key transformation

$$X = F(Y), \tag{2.3.2}$$

then

(2.3.3) $\Pr\{X \leqslant b\} = \Pr\{F(Y) \leqslant b\} = \Pr\{Y \leqslant F^{-1}(b)\} = F(F^{-1}(b)) = b,$

and we verify that X is distributed U(0,1). More detailed analysis would show that the only requirement to perform the transformation is that F(x) should be continous.

As said before, we will never work with real numbers but only with approximations to them (e.g. floating point numbers). Consequently it will be equally approximate to "continuize" discrete distributions. In section 2.5 we will take a closer look at one type of discrete distribution.

When the distribution of the keys is unknown we may approximate it based on a sample of points or on other type of approximation. Isaac and Singleton [1956], Kronmal and Tarter [1965 & 1968], Price [1971], Simon and Guiho [1972] discuss several types of approximations to cumulative distribution functions. Knott [1975] surveys some of these approximations in connection with hashing schemes.

Finally let us consider another example of a real distribution. We will call it a *decayed sequential* distribution, and results from the following model. Keys are integers assigned sequentially to records at a constant birth rate b. Records "die" at a constant probability rate d, i.e.

(2.3.4) $\Pr\{\text{a given record dies in } (t,t+\Delta t)\}/\Delta t = d; \qquad \Delta t \rightarrow 0.$

When the system is stabilized we find

(2.3.5) $\Pr\{\text{record i still exists}\} = e^{-d(K-i)/b},$

(2.3.6) $E[\text{number of records}] = [1-e^{-d/b}]^{-1}$, and

(2.3.7) $\Pr\{x \leqslant i\} = e^{-d(K-i)/b},$

where K is the largest of the keys, the one just assigned, and x is the key value of a randomly chosen alive record. The records are then geometrically distributed, and if b/d is rather large, we can well approximate the above with an exponential distribution.

2.4 - Ordered Uniform Real Random Variables.

In the following chapters we will use properties of a set of ordered U(0,1) random variables. Therefore we will describe here some of the formulas involved that will ease later computations. We will study the properties of a set of n random independent variables $Y_1, Y_2, ..., Y_n$ from a U(0,1) distribution. We order the random variables in increasing order, and relabel them so that: $0 \leqslant X_1 \leqslant X_2 \leqslant ... \leqslant X_n \leqslant 1$.

The probability density function (p.d.f.) of X_j is then

$$(2.4.1) \qquad \text{pdf}(X_j) = x^{j-1}(1-x)^{n-j}/B(j,n-j+1) = I_x'(j,n-j+1),$$

where $B(a,b)$ is the Beta function, $I_x(a,b)$ is the Incomplete Beta function and $I_x'(a,b)$ is the derivative of the Incomplete Beta function respect to x.

The cumulative distribution function (c.d.f.) of X_j is

$$(2.4.2) \qquad \Pr\{X_j \leqslant x\} = I_x(j,n-j+1).$$

We find also

$$(2.4.3) \qquad E[X_j] = x_0 = j/(n+1),$$

$$(2.4.4) \qquad \text{Var}(X_j) = j(n-j+1)/((n+1)^2(n+2)),$$

and in general the s^{th} moment of X_j is

$$(2.4.5) \qquad E[X_j^s] = \mu'_s = \Gamma(j+s)\Gamma(n-j+1)/[\Gamma(j)\Gamma(n-j+s+1)].$$

The truncated moments are given by

$$(2.4.6) \qquad \int_0^x x^s I_x'(j,n-j+1)dx = \mu'_s I_x(j+s,n-j+1).$$

From 2.4.5, expanding the binomial, we find that the central moments are

$$(2.4.7) \qquad E[(X_j - x_0)^s] = \mu_s,$$

(2.4.8) $\mu_0 = 1,$

(2.4.9) $\mu_1 = 0,$

(2.4.10) $\mu_2 = \mathrm{Var}(X_j) = j(n-j+1)/((n+1)^2(n+2)),$

(2.4.11) $\mu_3 = 2j(n-j+1)(2j-n-1) \,/\, ((n+1)^3(n+2)(n+3)),$ etc.

From a general asymptotic expansion for $I_x(a,b)$ given by Temme [1975], assuming that $\epsilon<x_0<1-\epsilon$ when $n\to\infty$, we derive

(2.4.12) $I_{x_0}(j,n-j+1) = \frac{1}{2} + (n+1-2j)[2\pi j(n-j+1)(n+1)]^{-\frac{1}{2}}/3 + O(n^{-3/2}),$

(2.4.13) $I_{x_0}'(j,n-j+1) = [(n+1)^3/(2\pi j(n-j+1)]^{\frac{1}{2}} \times [1 + O(n^{-1})].$

Let $I_\rho(j,n-j+1) = \frac{1}{2}$; then

(2.4.14) $\rho = x_0 + (2j-n-1)/(3(n+1)^2) + O(n^{-2}).$

It is also of interest to know the central signed moments. Using 2.4.6, 2.4.7 and 2.4.12 we find

(2.4.15) $E[\mathrm{sgn}(X_j-x_0)(X_j-x_0)^s] = \nu_s,$

and

(2.4.16) $\nu_0 = 1-2I_{x_0}(j,n-j+1) \sim 2(2j-n-1)[2\pi j(n-j+1)(n+1)]^{-\frac{1}{2}}/3 + O(n^{-3/2}).$

The mean deviation is

(2.4.17) $\nu_1 = 2j^j(n-j+1)^{n-j+1}\Gamma(n+1) \,/\, [\Gamma(j)\Gamma(n-j+1)(n+1)^{n+2}]$

$\sim [2j(n-j+1)/(\pi(n+1)^3)]^{\frac{1}{2}} + O(n^{-3/2})$

$\sim O(n^{-\frac{1}{2}}),$

$$(2.4.18)\qquad \nu_2 = \mu_2\nu_0 - \nu_1(2j-n-1)/[(n+1)(n+2)]$$

$$\sim 4(n+1-2j)[j(n-j+1)/(2\pi(n+1)^5)]^{1/2} / (3(n+2)) + O(n^{-5/2})$$

$$\sim O(n^{-3/2}),$$

and

$$(2.4.19)\qquad \nu_3 = \mu_3\nu_0 + 2\nu_1[jn(n-j+1)+j^2+(n-j+1)^2]/[(n+1)^2(n+2)(n+3)]$$

$$\sim 4[-3j^2n+2j^2+3jn^2+jn-2j+2n^2+4n+2] \times [j(n-j+1)/(2\pi(n+1)^7)]^{1/2} / (3(n+2)(n+3)) + O(n^{-5/2})$$

$$\sim O(n^{-3/2}),$$

where $\sim$ denotes asymptotic expansion of the expression.

Some nontrivial sums involving the Incomplete Beta function are

$$(2.4.20)\qquad \sum_{j=k}^{n} I_x(j,n-j+1) = nxI_x(k-1,n-k+1) - (k-1)I_x(k,n-k+1),$$

$$(2.4.21)\qquad \sum_{j=1}^{n} I_x(j,n-j+1) = nxI_x(0,n) = nx,$$

and

$$(2.4.22)\qquad \sum_{j=1}^{k} I_x(j,n-j+1) = nx[1-I_x(k,n-k)] + kI_x(k+1,n-k).$$

Let $F_n(x_1,x_2,\ldots,x_n) = \Pr\{X_1 \leqslant x_1; X_2 \leqslant x_2; \ldots; X_n \leqslant x_n\}$ be the joint c.d.f. of the n U(0,1) independent ordered random variables. Let us assume also that $0 \leqslant x_1 \leqslant x_2 \leqslant \ldots \leqslant x_n \leqslant 1$ so we can simplify the notation. Then we have

$$(2.4.23)\qquad \begin{aligned} F_n(x_1,x_2,\ldots,x_n) = {} & x_1F_{n-1}(x_2,x_3,\ldots,x_n) + \\ & (x_2-x_1)F_{n-1}(x_1,x_3,\ldots,x_n) + \\ & \cdots\cdots\cdots + \\ & (x_n-x_{n-1})F_{n-1}(x_1,x_2,\ldots,x_{n-1}). \end{aligned}$$

This recursion formula comes from studying the insertion of the n^{th} element between the n-1 already ordered. There is no known compact form to write F_n explicitly for the general case.

An associated problem is to find the probability that the element α is in position j. That is, given a randomly generated file, find the distribution of the location of α, that is one of the X's each being equally probable 'a priori', conditional to the value α. Let $S = \{X_1, X_2, \ldots, X_n\}$ be the file we search, we find that

$$(2.4.24) \quad \Pr\{X_j = \alpha \mid \alpha \in S\} = \binom{n-1}{j-1} \alpha^{j-1} (1-\alpha)^{n-j} = I_\alpha'(j, n-j+1)/n$$

is a binomial distribution, whence

$$(2.4.25) \quad E[j] = (n-1)\alpha + 1,$$

$$(2.4.26) \quad \text{Mode}\{j\} = \lceil n\alpha \rceil \text{ or } \lfloor n\alpha + 1 \rfloor,$$

$$(2.4.27) \quad \text{Var}(j) = (n-1)\alpha(1-\alpha),$$

and

$$(2.4.28) \quad \mu_3\{j\} = (n-1)\alpha(1-\alpha)(1-2\alpha).$$

A closely related problem is to find the probability of locating a new element α (different from all X's) in between X_{j-1} and X_j. That is, given a random file, find the distribution of the location of α that is not one of the X's, conditional to the value α. We find

$$(2.4.29) \quad \Pr\{X_{j-1} < \alpha < X_j \mid \alpha \notin S\} = \binom{n}{j-1} \alpha^{j-1} (1-\alpha)^{n-j+1},$$

which is also a binomial distribution. This implies

$$(2.4.30) \quad E[j] = n\alpha + 1,$$

$$(2.4.31) \quad \text{var}\{j\} = n\alpha(1-\alpha),$$

and

$$(2.4.32) \quad \mu_3\{j\} = n\alpha(1-\alpha)(1-2\alpha).$$

Further information about the Incomplete Beta function can be found in: Abramowitz and Stegun [1964]; Johnson and Kotz [1969]; Temme [1975]; Dingle [1973]; Luke [1969].

2.5 - Discrete Distributions of Keys.

When the keys come from a discrete distribution, we do not usually have a transformation like 2.3.1 to change it to a canonical form. Consequently the study of any particular discrete distribution will not throw much light on other discrete distributions, in contrast with the continuous case.

As a good example, and as one of the most common discrete distributions, we will analyze the discrete rectangular distribution. Moreover we will specify a rather natural assumption for real files; that is, that the keys are not repeated; i.e. they are sampled without replacement. The file is sorted so that $1 \leqslant X_1 < X_2 < \ldots X_n \leqslant m$. For the successful search, the distribution of the location of α is

$$(2.5.1) \quad \Pr\{X_j = \alpha\} = \binom{\alpha-1}{j-1}\binom{m-\alpha}{n-j} / \binom{m-1}{n-1},$$

a hypergeometric distribution with parameters $m-1$, $n-1$ and α (α is now an integer, $1 \leqslant \alpha \leqslant m$). The expected value is

$$(2.5.2) \quad E[j] = (n-1)(\alpha-1)/(m-1) + 1,$$

and the mode of the distribution of j is

$$(2.5.3) \quad \text{mode}\{j\} = \lceil n\alpha/(m+1) \rceil, \quad \text{or}$$

$$= \lfloor n\alpha/(m+1) \rfloor + 1.$$

The probabilities of $X_j > \alpha$ and $X_j < \alpha$ are hypergeometric cumulative distributions. These and their approximations are well described in Johnson and Kotz [1969, chap. 6].

The distribution of the j^{th} ordered variable is the same as 2.5.1, except that now we consider α to be the variable.

The distributions related to the unsuccessful case, when α is not one of the keys, are

$$(2.5.4) \quad \Pr\{X_{j-1} < \alpha < X_j\} = \binom{\alpha-1}{j-1}\binom{m-\alpha}{n-j+1} / \binom{m-1}{n},$$

also a hypergeometric distribution except now with parameters $m-1$, n and α (α is an integer, $1 \leqslant \alpha \leqslant m$), and $\Pr\{X_j < \alpha\}$, a cumulative hypergeometric distribution.

These distributions are well approximated by corresponding Incomplete Beta distributions, as in 2.4, such that the first two moments coincide [Johnson 69]. Consequently, based on a numerical approximation argument, we can translate the problem to a similar continuous one.

3. - Pure Interpolation Search.

3.1 - Motivation.

In this chapter we consider the search of an ordered table or file whose keys are scalars, using this fact to improve the average number of accesses. As a model of the file we will consider a set of n random independent variables, $Y_1, Y_2, \ldots Y_n$, drawn from the same distribution. The only restriction we will impose on our model is that the source cumulative distribution function, $F(y) = Pr\{Y \leqslant y\}$, should be continuous.

As we saw in 2.3, without loss of generality, using the transformation

(3.1.1) $\quad X = F(Y),$

we need only to consider keys drawn from a U(0,1) uniform distribution.
It is immediate that the variable X is distributed U(0,1) and furthermore if $Y_1 \leqslant Y_2$ then $X_1 = F(Y_1) \leqslant X_2 = F(Y_2)$, so the transformation preserves order. Later we will only use an interval of the range, e.g. [a,b]; in this case the transformation of the variables in the interval will be

(3.1.2) $\quad X = (F(Y)-F(a)) \;/\; (F(b)-F(a)).$

Actually we do not need to convert the keys, but only the functions applied over them to achieve this transformation.

3.2 - Definition of the Search Algorithm.

Let $0=X_0 \leqslant X_1 \leqslant X_2 \leqslant \ldots \leqslant X_n \leqslant X_{n+1}=1$ be our file obtained from n ordered random variables. The interpolation search algorithm will be defined by the function

(3.2.1) $\phi(\alpha,n) = j,$

where α is the search key, n is the size of the file and $1 \leqslant j \leqslant n$ is the probe position to inspect. We will assume that the keys X are contained in a vector k, that k(0) and k(n+1) contain the limit values of the keys, and $k(0)<\alpha<k(n+1)$. The algorithm, for U(0,1) random variables, is defined in a pseudo-language as follows:

```
select (α)
        case <k(0) or >k(n+1): return(FAIL);
        case k(0): return(0);
        case k(n+1): return(n+1)
        end select;
low := 0;  high := n+1;
while (high-low>1) do
        j := φ( (α-k(low)) / (k(high)-k(low)), high-low-1) + low;
        select (k(j))
                case α: return(j);
                case <α: low := j;
                case >α: high := j
                end select
        end while
return(FAIL);
```

If the variables are not U(0,1), then we use the transformation [3.1.2] for the computation of ϕ, i.e.

```
j := φ( (F(α)-F(k(low))) / (F(k(high))-F(k(low))), high-low-1) + low;
```

Note that from the point of view of our definition of number of accesses, the select-case statement allows us to use only one access per location inspected. In some low level languages, or in languages that allow three way branches, this will also mean only one comparison. The first select-case statement may be completely avoided if we know, without need of verification, that $k(0)<\alpha<k(n+1)$.

To prove this algorithm correct we first note that a successful exit only occurs when $k(j)=\alpha$. Furthermore, if the file is ordered, the condition $\{k(low)<\alpha<k(high)$ **and** $high-low\leqslant 1\}$ implies failure. We find that $\{k(low)<\alpha<k(high)\}$ is an invariant relation, if we do not return, in the body of the **while** statement. Also inside the loop, if $1\leqslant\phi(\alpha,n)\leqslant n$, then high–low is a strictly decreasing magnitude, hence the **while** will terminate. To complete the proof of correctness, upon exit from the loop, given the initial conditions, we verify the failure condition. Note that the condition $1\leqslant\phi(\alpha,n)\leqslant n$ for all α and n, is crucial to guarantee convergence of the algorithm.

We will define a *coherent algorithm* as one for which ϕ satisfies:

(3.2.2) a) $\phi(1-\alpha,n) = n+1-\phi(\alpha,n)$ (symmetry),

(3.2.3) b) $\phi(0^+,n) = 1$ (boundary).

A general linear interpolation is defined by

(3.2.4) $\phi(\alpha,n) = \lfloor a_0+a_1\alpha+a_2n+a_3\alpha n\rfloor$.

By coherency we have at least the conditions:

(3.2.5) $a_2 = 0,\quad a_3 = 1,\quad 1\leqslant a_0<2,\quad -a_0<a_1\leqslant 1-a_0$.

The symmetry condition will not be satisfied if we use floor functions for all the range of α. There are two solutions to this problem. Use $\phi(\alpha,n)$ for $0\leqslant\alpha\leqslant ½$ and $n+1-\phi(1-\alpha,n)$ for $½<\alpha\leqslant 1$, or ignore the lack of symmetry, since it occurs over a finite set of points with total probability equal to zero.

The interpolation formula

(3.2.6) $\phi(\alpha,n) = \lfloor n\alpha\rfloor+1$,

or the symmetric

(3.2.7) $\phi(\alpha,n) = \lceil n\alpha\rceil$,

are the simplest coherent linear interpolation formulas. They have also the interesting property that the position searched is the mode of the distribution of the location of α [2.4.26], i.e. $\Pr\{X_j=\alpha\}$ is maximum.

The formula 3.2.6 has a slight advantage over 3.2.7 since the floor function is almost always available, for positive arguments, in a high level language using real-integer conversion. We will now describe a FORTRAN interpolation search algorithm for a table of U(0,1) random keys using the interpolation formula 3.2.6.

```
      FUNCTION ISEAR(K,LBOUND,HBOUND,ALPHA)
      REAL K(1), ALPHA
      INTEGER LBOUND, HBOUND, LOW, HIGH, J, ISEAR
      LOW = LBOUND
      HIGH = HBOUND
      IF( ALPHA - K(LBOUND) )80, 60, 10
   10 IF( K(HBOUND) - ALPHA )80, 70, 20
   20 IF( HIGH-LOW .LE. 1 )GO TO 80
      J = (ALPHA-K(LOW)) / (K(HIGH)-K(LOW)) * FLOAT(HIGH-LOW-1)
      J = J+LOW+1
      IF( K(J) - ALPHA )30, 50, 40
   30 LOW = J
      GO TO 20
   40 HIGH = J
      GO TO 20
   50 ISEAR = J
      RETURN
   60 ISEAR = LBOUND
      RETURN
   70 ISEAR = HBOUND
      RETURN
C     SEARCH FAILURE, ALPHA NOT IN K.
   80 ISEAR = -1
      RETURN
      END
```

Note that the computation of ϕ should be done by itself. The addition of LOW+1 in the same statement does not guarantee, because of rounding, the correct evaluation of the floor function. The above code makes use of the unpopular arithmetic three way if statement. This construct usually produces a shorter and more efficient code.

The *rational* interpolation search algorithm, is the algorithm to search when we have a set of keys from a discrete rectangular distribution R(1,m) drawn without replacement. This distribution is described in 2.5. The corresponding FORTRAN algorithm to 2.5.3 is the same

as the above except for the declarations that become

```
INTEGER K(1), ALPHA, LBOUND, HBOUND, LOW, HIGH, J, ISEAR
```

and the interpolation function that becomes

```
J = (ALPHA-K(LOW))*(HIGH-LOW-1)/(K(HIGH)-K(LOW)) + LOW + 1
```

When the keys are R(1,m) but drawn **with** replacement, the equivalent interpolation function is given by

$$j = \lceil n \times ln((m-\alpha)/(m-\alpha+1))/[ln((\alpha-1)/\alpha)+ln((m-\alpha)/(m-\alpha+1))]\rceil .$$

This formula is clearly impractical and expensive to compute.

It should be noted that despite its appearance this algorithm is not trivial. In our opinion, the lack of a correct published algorithm, the difficulties involving transformation of non-uniform keys and not knowing, until recently, its asymptotic behaviour are among the major reasons for the lack of popularity of interpolation search.

It is practical, for some special interpolation functions, to define $\phi(\alpha,n)$ by its *break* points.
Iff $\lambda_{i,n}$ is a break point of the function $\phi(\alpha,n)$, then

$$(3.2.8) \quad \lambda_{i-1,n} < \alpha < \lambda_{i,n} \Rightarrow \phi(\alpha,n) = i.$$

For the interpolation formulas 3.2.6 and 3.2.7, $\lambda_{i,n} = i/n$.

3.3 - Information-Theoretic View.

One way to compute the expected number of accesses to find a given key in an ordered table is with the aid of information theory. The general scheme we will follow is: we compute the uncertainty of the search at a given step as a function of the key value, size of the table etc.; secondly we compute the information gain in any step of the algorithm; if we are successful in expressing this gain in terms of the same parameters as the uncertainty, we then can compute the number of steps needed to reduce the uncertainty to zero, i.e. to find the element in the table.

The uncertainty of a search is expressed in simple terms by observing that the probability of the key α (known to be in the file) of being in position j is given by 2.4.24. The uncertainty, which formally is the entropy of the random variable "location of α" is expressed [Ash 64] by

$$(3.3.1)\quad H(\alpha,n) = -\sum_{j=1}^{n} \Pr\{X_j=\alpha\} lg(\Pr\{X_j=\alpha\})$$

$$= H(1-\alpha,n).$$

Although exact, this formula is of little help, so we will use an asymptotic expansion in n of $H(\alpha,n)$. The first term of this expansion was obtained by Peterson [1957], but he had a mistake in the $O(1)$ term that deteriorated greatly the, otherwise good, numerical approximation. His approach was based on approximations of the binomial distribution [Feller 57; Johnson 69]. To obtain further terms in the expansion we need to use a more powerful method. The best suited seems to be a parallel of the ones described by Dingle [1973].

Since $\Pr\{X_j=\alpha\}$ is a binomial distribution, $H(\alpha,n)$ is the expected value of $-lg(\Pr\{X_j=\alpha\})$ in that distribution. For the binomial distribution the main contribution to $H(\alpha,n)$ is given by the terms near its expected value, in our case $(n-1)\alpha+1 = \mu'_1$. Then we expand the slowly varying factor of the summand in a Taylor series at μ'_1

$$(3.3.2)\quad -lg(\Pr\{X_j=\alpha\}) = \sum_{i=0}^{\infty} d_i (j-\mu'_1)^i.$$

If higher derivatives of the slowly varying factor imply lower order in the asymptotic variable, by Watson's lemma we invert the order of integration and obtain the formula

$$(3.3.3)\quad H(\alpha,n) = \sum_{i=0}^{\infty} \mu_i d_i.$$

The central moments of a binomial distribution are given in 2.4.7; higher order moments can be obtained from the recurrence formula [Romanovsky 23]:

$$(3.3.4) \qquad \mu_{i+1} = \alpha(1-\alpha)[n\times i\mu_{i-1} + d\mu_i/d\alpha],$$

which implies that $\mu_i = o(n^{\lfloor i/2 \rfloor + 1})$. For d_i we write

$$(3.3.5) \qquad -ln(2)lg(Pr\{X_j=\alpha\}) = ln(\Gamma(n))-ln(\Gamma(j))-ln(\Gamma(n-j+1))+(j-1)ln(\alpha)+$$
$$(n-j)ln(1-\alpha),$$

and then we compute

$$(3.3.6) \qquad d_1 = -\psi(j)+\psi(n-j+1)+ln(\alpha/(1-\alpha))\,|_{j=\mu'_1} = O(ln(n)),$$

$$(3.3.7) \qquad d_2 = -\psi'(j)-\psi'(n-j+1)\,|_{j=\mu'_1} = O(n^{-1}),$$

and

$$(3.3.8) \qquad d_3 = -\psi''(j)+\psi''(n-j+1)\,|_{j=\mu'_1} = O(n^{-2}).$$

We conclude then that $d_i = O(n^{1-i})$. Each term of the infinite sum 3.3.2 is $o(n^{\lfloor (4-i)/2 \rfloor})$ and has an asymptotic characteristic. The computation of terms is tedious and error prone; in this case the ALTRAN algebraic manipulator [Brown 71] was used to obtain

$$(3.3.9) \qquad H(\alpha,n) = \tfrac{1}{2}lg(2\pi e(n-1)\alpha(1-\alpha)) - (1-2\alpha)^2/(12ln(2)(n-1)\alpha(1-\alpha)) -$$
$$(\alpha^4+(1-\alpha)^4)/(24ln(2)[(n-1)\alpha(1-\alpha)]^2) + O(n^{-3}).$$

Historical note.

One of the first derivations of the expected number of accesses using information theory was done by Gonnet [1972], and although it has some weaknesses it is interesting to follow the approach. From the information-theoretic point of view, we gain information about the location of the variable in the table in two steps: when we know the value of the variable with respect to the bounds of the file, and secondly when we examine the selected location and discard part of the table.

The first gain is given by the difference between the entropy of the search when we do not know the value of the key, namely:

$$(3.3.10) \quad -\sum_{j=1}^{n}(1/n)lg(1/n) = lg(n),$$

and the entropy when we do know the key value, i.e. 3.3.1. This difference is

$$(3.3.11) \quad g_1 = \tfrac{1}{2}lg(n/[2\pi e\alpha(1-\alpha)]) + O(n^{-1}).$$

The second gain is given by the information gained in a three way decision, namely $X_j=\alpha$, $X_j<\alpha$ or $X_j>\alpha$. The first event occurs with probability $[2\pi n\alpha(1-\alpha)]^{-1/2} + O(n^{-1})$, and the last two with probability $\frac{1}{2} + O(n^{-1/2})$. Then the information gained with the actual comparison with the variable in the table is

$$(3.3.12) \quad g_2 = 1 + \tfrac{1}{2}[2\pi n\alpha(1-\alpha)]^{-1/2}lg(2\pi n\alpha(1-\alpha)) + O(n^{-1}).$$

More detailed analysis shows that if $n\times\min(\alpha,1-\alpha)>1$ then $g_2 \geqslant 1$. The total information gain is

$$(3.3.13) \quad g_t = \tfrac{1}{2}lg(2n/[\pi e\alpha(1-\alpha)]) + O(n^{-1/2}) \geqslant \tfrac{1}{2}lg(n) - 0.09419... + O(n^{-1}).$$

Thus we can say, with good approximation, that the uncertainty is halved on the first step of the search. There was an arithmetic error in the original work that made a stronger inequality, and consequently a slightly more optimistic result. If we make the assumption that a search with uncertainty h bits is equivalent to a search between 2^h equally probable elements (that has uncertainty h), we can repeatedly use the reduction formula for the entropy.

The above formula breaks down when $n\times\min(\alpha,1-\alpha)\leqslant 1$ but as we shall see in section 3.4, when this happens, the average number of accesses is less than $1+\min(\alpha,1-\alpha)$. For the asymptotic analysis let us assume that after a fixed threshold h_0 is reached we do bisections to find the element. The total average number of accesses performed is

$$(3.3.14) \quad \lceil lg(lg(n)/h_0)\rceil + h_0 + o(1).$$

Information gain in a general step.

The uncertainty after one access is the expected value of the uncertainties of the three possible outcomes:

1) $\alpha = X_j \implies$ uncertainty 0,

2) $\alpha < X_j \implies \int_{\alpha}^{1} I_w'(j-1,n-j+1)\, H(\alpha/w,j-1)\, dw \,/\, [1-I_\alpha(j-1,n-j+1)]$,

3) $\alpha > X_j \implies \int_{0}^{\alpha} I_w'(j,n-j)\, H((1-\alpha)/(1-w),n-j)\, dw \,/\, I_\alpha(j,n-j)$

(according to 2.4.1, 2.4.2 and 3.3.1).

The expected uncertainty after one comparison is then

$$(3.3.15)\quad h^* = \int_{\alpha}^{1} I_w'(j-1,n-j+1) H(\alpha/w,j-1)\, dw + \int_{0}^{\alpha} I_w'(j,n-j) H((1-\alpha)/(1-w),n-j)\, dw$$
$$= (s_1+s_2)/ln(2).$$

(Note that the second summand is the symmetric analog of the first one under the transformation $(\alpha,j) \implies (1-\alpha,n-j+1)$). This formula is inconvenient even to compute numerical values of h^*, so we will deduce a simpler formula.

Let p_k denote $\Pr\{X_k=\alpha/w\}$ $(1 \leqslant k \leqslant j-1)$. We have

$$(3.3.16)\quad p_k = w^{2-j}\binom{j-2}{k-1}\alpha^{k-1}(w-\alpha)^{j-1-k},$$

and

$$(3.3.17)\quad ln(p_k) = ln\left(\binom{j-2}{k-1}\right) - (j-2)ln(w) + (k-1)ln(\alpha) + (j-1-k)ln(w-\alpha),$$

whence we can write

(3.3.18)
$$s_1 = \int_{\alpha}^{1} -\sum_{k=1}^{j-1}\binom{j-2}{k-1}\alpha^{k-1}(w-\alpha)^{j-1-k} ln(p_k)(1-w)^{n-j}/B(j-1,n-j+1)\, dw.$$

Decomposing $ln(p_k)$ according to functions of w we find that the summand corresponding to $(j-1-k)ln(w-\alpha)$ is

$$= \int_{\alpha}^{1} -\sum_{k=1}^{j-2}\binom{j-2}{k-1}\alpha^{k-1}(w-\alpha)^{j-1-k}(j-1-k)ln(w-\alpha)(1-w)^{n-j}/B(j-1,n-j+1)\,dw$$

$$= -\sum_{k=1}^{j-1}\binom{j-2}{k-1}\alpha^{k-1}\int_{\alpha}^{1}(w-\alpha)^{j-1-k}(1-w)^{n-j}ln(w-\alpha)/B(j-1,n-j+1)\,dw.$$

Using the definite integral

$$(3.3.19)\quad \int_{a}^{b} ln(x-a)(x-a)^{m}(b-x)^{n}\,dx =$$

$$(b-a)^{m+n+1}B(m+1,n+1)[ln(b-a)+\psi(m+1)-\psi(m+n+2)]$$

and rearranging the factorials we get

$$(3.3.20)\quad -\sum_{k=1}^{j-1}(j-1-k)\binom{n-1}{k-1}\alpha^{k-1}(1-\alpha)^{n-k}[ln(1-\alpha)+\psi(j-k)-\psi(n-k+1)].$$

For the summand $(j-2)ln(w)$ we obtain

$$(3.3.21)\quad \int_{\alpha}^{1}\sum_{k=1}^{j-1}\binom{j-2}{k-1}\alpha^{k-1}(w-\alpha)^{j-1-k}(j-2)ln(w)(1-w)^{n-j}/B(j-1,n-j+1)\,dw.$$

Summing the inner binomial this simplifies to

$$(3.3.22)\quad (j-2)\int_{\alpha}^{1} I_{w}'(j-1,n-j+1)ln(w)\,dw.$$

Finally for the summands independent of w, using the definite integral

$$(3.3.23)\quad \int_{a}^{b}(x-a)^{m}(b-x)^{n}\,dx = (b-a)^{m+n+1}B(m+1,n+1),$$

and rearranging factorials we obtain

$$(3.3.24)\quad -\sum_{k=1}^{j-1}\binom{n-1}{k-1}\alpha^{k-1}(1-\alpha)^{n-k}[ln(\binom{j-2}{k-1}) + (k-1)ln(\alpha)].$$

Collecting 3.3.20 and 3.3.24 we finally obtain

$$(3.3.25)\quad s_1 = (j-2)\int_{\alpha}^{1} I_{w}'(j-1,n-j+1)ln(w)\,dw +$$

$$\sum_{k=1}^{j-1}\binom{n-1}{k-1}\alpha^{k-1}(1-\alpha)^{n-k}[(j-1-k)(\psi(n-k+1)-\psi(j-k))-ln(p_{k}(w=1))].$$

Using the symmetry of the search the expression for s_2 is

$$(3.3.26)\quad s_2 = (n-j-1)\int_0^{\alpha} I_w'(j,n-j)\ln(1-w)\,dw + \sum_{k=j+1}^{n}\binom{n-1}{k-1}\alpha^{k-1}(1-\alpha)^{n-k}[(k-j-1)(\psi(k)-\psi(k-j))-\ln(p_k)],$$

where now p_k stands for $\binom{n-j-1}{k-j-1}\alpha^{k-j-1}(1-\alpha)^{n-k}$.

The above formulas allow the numerical computation of the information gain equation. The following table shows some selected file sizes and key values with the corresponding information measures.

n	α	initial uncertainty	$s_1/\ln(2)$	$s_2/\ln(2)$	information gain
5	0.5	2.031	0.204	0.204	1.624
15	0.5	2.950	0.599	0.599	1.752
51	0.5	3.869	0.978	0.978	1.913
101	0.5	4.369	1.166	1.166	2.037
201	0.5	4.869	1.341	1.341	2.187
501	0.5	5.530	1.556	1.556	2.418
991	0.5	6.023	1.706	1.706	2.610
51	0.1	3.162	0.528	0.861	1.773
51	0.2	3.557	0.752	0.960	1.846
51	0.3	3.747	0.868	0.994	1.885
51	0.4	3.840	0.937	0.997	1.906

These results show that the information gain in one step of the search, which is always larger than binary search, is a significant fraction of the initial uncertainty.

However, these formulas are not appropriate for obtaining asymptotic expansions of the information gain. The reason lies in the difficulty of summing the terms containing the $\psi(j-k)$ function. The main contribution for such sums is made where its argument is small, hence preventing one from obtaining higher asymptotic orders. The problem may be solved by a method similar to the one described earlier in the chapter with the aid of the definite integrals A1.4.1 through A1.4.7 and our "Euler-Riemann" [A1.5.2] summation formula. We will use in this case the integral expression for the information gain and a proper approximation of $H(\alpha,n)$.

The main problem with the asymptotic expansion for $H((w-\alpha)/w,j-1)$ is that it has a pole at $w=\alpha$. This causes an unbounded error term for s_1 since $I_w'(j-1,n-j+1)$ is $O(n^{1/2})$ near $w=\alpha$. Consequently we need better asymptotic approximations of $H(x,n)$ when $x\to 0$. For $x\to 0$ we find that

(3.3.27) $\quad H(x,n) = n\times x(1-ln(nx)) + O(n^2x^2).$

Hence we conclude that there is no asymptotic power expansion for H(x,n) in terms of x. Even subtracting the $ln(nx)$ term, the series integrated term by term with $I_w'(j-1,n-j+1) = O(e^{-a^2x^2})$ does not show the desired asymptotic behaviour

(3.3.28) $\quad \int_0^\infty e^{-a^2x^2} n^i x^i \, dx = O(n^{i/2}); \qquad a^2 = O(n).$

The problem involved with the pole is solved by changing the series 3.3.9 to equivalent ones but with $lg(1+2\pi...)$ instead of $lg(2\pi e...)$ and powers of $n\alpha(1-\alpha)+1$ instead of $n\alpha(1-\alpha)$. The resulting asymptotic series is

(3.3.29)

$$H(\alpha,n) = lg(1+2\pi e(n-1)\alpha(1-\alpha)) - [1/(4\pi e)+(1-2\alpha)^2/12]/(ln(2)[n\alpha(1-\alpha)+1])+$$

$$O([n\alpha(1-\alpha)+1]^{-2}).$$

Let $A_k(\alpha,n)$ be the asymptotic expansion 3.3.29 up to order k; i.e. $H(\alpha,n)-A_k(\alpha,n) = O([n\alpha(1-\alpha)+1]^{-k-1})$. For any fixed z we find that

(3.3.30) $\quad H(z/n,n) = A_0(z/n,n)+O(1) = O(1) = z(1-ln(z)) + e^{-z} \sum_{i=1}^{\infty} z^i ln(i!)/i! \, .$

Before going further we will show the general approach to find asymptotic expansions of integrals of the type

(3.3.31) $\quad I(f) = \int_\alpha^1 I_w(j-1,n-j+1) f(w) \, dw.$

The method is carefully described in Dingle [1973]. We first rewrite the integral as

(3.3.32) $\quad I(f) = I_\alpha'(j-1,n-j+1) \int_0^\infty e^{-a^2x^2} G(x) \, f(x) \, dx,$

with

(3.3.33) $\quad G(x) = (\alpha/w)^\beta f(w)/f(x) dw/dx \qquad$ and

(3.3.34) $\quad -a^2x^2 = (j-2-\beta)ln(w/\alpha)+(n-j)ln((1-w)/(1-\alpha)).$

We choose $\beta=(1+\Delta-2\alpha)/(1-\alpha)$ to avoid a discontinuity of dw/dx at x=0, $\Delta = nx-\lfloor nx \rfloor$, $j=\lfloor nx \rfloor+1$, and $a^2=(n-j)/[2\alpha(1-\alpha)^2]$ so that dx/dw=1.

The idea is to group slowly varying terms in G(x), then expand

$$G(x) = \sum_{i=0}^{\infty} g_i x^i, \text{ and finally we obtain}$$

$$(3.3.35) \quad I(f) = I_\alpha'(j-1,n-j+1)\sum_{i=0}^{\infty} g_i \int_0^{\infty} e^{-a^2x^2} x^i f(x)\, dx.$$

Back to our problem, we conclude that

$$(3.3.36) \quad I(H) = I(A_k) + R(\alpha,n,k),$$

where

$$(3.3.37) \quad R(\alpha,n,k) = O(I_\alpha'(j-1,n-j+1)) \times \int_0^{\infty} e^{-a^2x^2}[1+nx]^{-k-1}\, dx)$$

$$= O(n^{-1/2} ln(n)) \quad \text{for } k=0 \quad (A1.4.7), \text{ and}$$

$$= O(n^{-1/2}) \quad \text{for } k \geq 1 \quad (A1.4.8\text{-}A1.4.9).$$

Hence, with this scheme, we cannot do much better with $A_k(\alpha,n)$ than with $A_0(\alpha,n)$. Finally to compute s_1 we define

$$(3.3.38) \quad G(x) = (\alpha/w)^{\beta} ln(1+\theta(w-\alpha)/w^2)/ln(1+bx)dw/dx$$

with $\theta=2\pi e(j-2)\alpha$, and $b=\theta/\alpha^2$.

From 3.3.34 we expand x in powers of $w-\alpha$. G(x) can be easily expressed in terms of powers of x and $w-\alpha$ and with the aid of an algebraic manipulator we find

$$(3.3.39) \quad g_0 = 1; \quad g_1 = (2\alpha-\Delta-2)/[\alpha(1-\alpha)], \quad \text{etc.}$$

Using A1.4.4 we obtain after substituting u = bx, the term corresponding to g_0 is

$$(3.3.40) \quad b^{-1}\int_0^{\infty} e^{-(a^2/b^2)u^2} ln(1+u)\, du =$$

$$-\pi^{1/2}/(4a)[\gamma+2ln(2a/b)] + O(ln(n)/n),$$

where γ is Euler's constant and the term corresponding to g_1 using A1.4.5 is

$$(3.3.41)\quad I_\alpha'(j-1,n-j+1)\int_0^\infty e^{-(a^2/b^2)u^2}\, ln(1+u)\, u\, du = O(n^{-1/2}ln(n)).$$

Using Stirling's approximation we obtain

$$(3.3.42)\quad I_\alpha'(j-1,n-j+1) = [n/(2\pi\alpha(1-\alpha))]^{1/2}(1+O(1/n)),$$

and regrouping the above terms we have

$$(3.3.43)\quad s_1 = [ln(2\pi^2e^2(j-2)\alpha(1-\alpha)]-\gamma)]/8 + O(n^{-1/2}ln(n)),$$

or without changing the order:

$$(3.3.44)\quad s_1 = [ln(2\pi^2e^2(n-1)\alpha(1-\alpha))-\gamma]/8 + O(n^{-1/2}ln(n)).$$

By the symmetry in the search we find

$$(3.3.45)\quad s_1+s_2 = [ln(2\pi^2e^2(n-1)\alpha(1-\alpha))-\gamma]/4 + O(n^{-1/2}ln(n)),$$

or the total information gain in one step

$$(3.3.46)\quad h^* = H(\alpha,n)/2 + [ln(\pi e)-\gamma]/(4ln(2)) + O(n^{-1/2}ln(n))$$

$$\sim H(\alpha,n)/2 + 0.56536... = H(\alpha,n)/2 + \epsilon.$$

The analyzed case is a general step of the search algorithm, so if we define, as in 3.3.14 a threshold h_0 after which we bisect, the expected number of probes required by the algorithm will be:

$$(3.3.47)\quad \sim lg(\lceil[\tfrac{1}{2}lg(2\pi e(n-1)\alpha(1-\alpha))-2\epsilon]/(h_0-2\epsilon)\rceil) + h_0$$

$$\sim lg(\lceil[\tfrac{1}{2}lg(n\alpha(1-\alpha))+0.9163..]/(h_0-2\epsilon)\rceil) + h_0.$$

3.4 - Average Number of Accesses for a Successful Search.

In the following section we will assume that we are searching for the key α, known to be in the table $0 \leqslant X_1 \leqslant \ldots \leqslant X_n \leqslant 1$.

Exact Average Number of Accesses.

Let $B(\alpha,n)$ be the average number of accesses needed to locate the position of α in the file of n random variables. Let A denote the number of accesses required to solve a search problem in the file $S=\{X_1,\ldots,X_n\}$. Then by analyzing one step of the algorithm we find the recursive formula

$$(3.4.1) \quad B(\alpha,n) = E[A|\alpha\in S]$$

$$= E[A|X_j<\alpha,\alpha\in S]\times Pr\{X_j<\alpha|\alpha\in S\} +$$

$$Pr\{X_j=\alpha\} +$$

$$E[A|X_j>\alpha|\alpha\in S]\times Pr\{X_j>\alpha|\alpha\in S\},$$

where $j = \phi(\alpha,n)$ is the interpolation probe position. Using the fact that X_j is distributed like the $(j-1)^{th}$ random ordered variable of a set of $n-1$ (we know α is one of the variables) for the first case and as the j^{th} of $n-1$ in the second, and simplyfing the conditional distributions, we derive

$$(3.4.2) \quad B(\alpha,n) = 1+\int_{\alpha}^{1} I_w'(j-1,n-j+1)B(\alpha/w,j-1)\,dw +$$

$$\int_{0}^{\alpha} I_w'(j,n-j)B((\alpha-w)/(1-w),n-j)\,dw$$

with the boundary conditions

$$(3.4.3) \quad B(\alpha,1) = 1,$$

$$(3.4.4) \quad B(0,n) = 1,$$

and

(3.4.5) $B(\alpha,0) = 0.$

The last one is used to have a uniform notation even when j is 1 or n and one of the integrals disappears. For coherent algorithms we find as immediate results that

(3.4.6) $B(\alpha,n) = B(1-\alpha,n),$

and

(3.4.7) $B(\alpha,2) = 1 + \min(\alpha,1-\alpha).$

Theorem:

$B(\alpha,n)$ is sectionally defined by a polynomial in α of degree at most n–1.

This is proved by induction on n. It is true for $n=1$ since $B(\alpha,1)=1$, is a polynomial of degree 0. Using the recurrence formula for $B(\alpha,n)$, for the first integral, the first factor can be written as

$$I_w'(j-1,n-j+1) = A\times w^{j-2}(1-w)^{n-j}.$$

By the induction hypothesis, the function $B(\alpha/w,j-1)$ is a polynomial in α/w of degree at most j–2. The product of both, after cancelling all negative powers of w in $B(\alpha/w,n)$ gives a polynomial of degree n–2 in w. Also for any term of the form $\alpha^i w^j$ we have $i+j\leqslant n-2$. After integration and evaluation at α and 1 the degree of the result is at most n–1. The same arguments apply for the second integral with 1–w in the role of w.

Consequently there is nothing peculiar or contrived about the function $B(\alpha,n)$; it is just a polynomial. Its only complexity lies on the fact that it is defined sectionally by different polynomials.

The first $B(\alpha,n)$ for the simplest linear interpolation algorithm are given to illustrate their simplicity, and at the same time their complexity.

(3.4.8) $B(\alpha,3) = 1+2\alpha, \quad 0\leqslant\alpha<1/3,$

$= 2(\alpha^2-\alpha+1), \quad 1/3\leqslant\alpha<2/3,$

$= B(1-\alpha,3), \quad 2/3\leqslant\alpha<1.$

$$\int_0^1 B(\alpha,3)\,d\alpha = 1.39506...$$

(3.4.9) $B(\alpha,4) = 1+3\alpha, \quad 0\leqslant\alpha<1/4,$

$= -2\alpha^3+6\alpha^2-3\alpha+2, \quad 1/4\leqslant\alpha<½,$

and it is symmetric with respect to $\alpha=½$.

$$\int_0^1 B(\alpha,4)\,d\alpha = 1.50390...$$

(3.4.10) $B(\alpha,5) = 1+4\alpha, \quad 0\leqslant\alpha<1/5,$

$= 2\alpha^4-8\alpha^3+12\alpha^2-4\alpha+2, \quad 1/5\leqslant\alpha<1/3,$

$= 221\alpha^4/16-131\alpha^3/4+219\alpha^2/8-31\alpha/4+37/16, \quad 1/3\leqslant\alpha<2/5,$

$= -20\alpha^4+40\alpha^3-24\alpha^2+4\alpha+2, \quad 2/5\leqslant\alpha<½.$

$$\int_0^1 B(\alpha,5)\,d\alpha = 1.59347...$$

(3.4.11) $B(\alpha,6) = 1+5\alpha, \quad 0\leqslant 1/6,$

$= -2\alpha^5+10\alpha^4-20\alpha^3+20\alpha^2-5\alpha+2, \quad 1/6\leqslant\alpha\leqslant 1/4,$

$= (-5030\alpha^5+16030\alpha^4-18680\alpha^3+10160\alpha^2-2035\alpha+527)/243,$

$1/4\leqslant\alpha\leqslant 1/3,$

$= (959\alpha^5-2410\alpha^4+2230\alpha^3-860\alpha^2+115\alpha+30)/16, \quad 1/3\leqslant\alpha\leqslant ½.$

$$\int_0^1 B(\alpha,6)\,d\alpha = 1.66859...$$

We see that for this algorithm, and likely for any reasonable algorithm, that we have discontinuities at $\alpha=i/n$ (of $B(\alpha,n)$ or its derivatives). Moreover if $(j-1)/n<\alpha<j/n$, $B(\alpha,n)$ reflects also the discontinuities of $B(\alpha,j-1)$ and $B(\alpha,n-j)$. For example, $B(\alpha,20)$ has at least 60 discontinuities (is defined by at least 60 different polynomials).

The following graph plots $B(\alpha,n)$ for $0\leqslant\alpha\leqslant ½$ and $2\leqslant n\leqslant 7$.

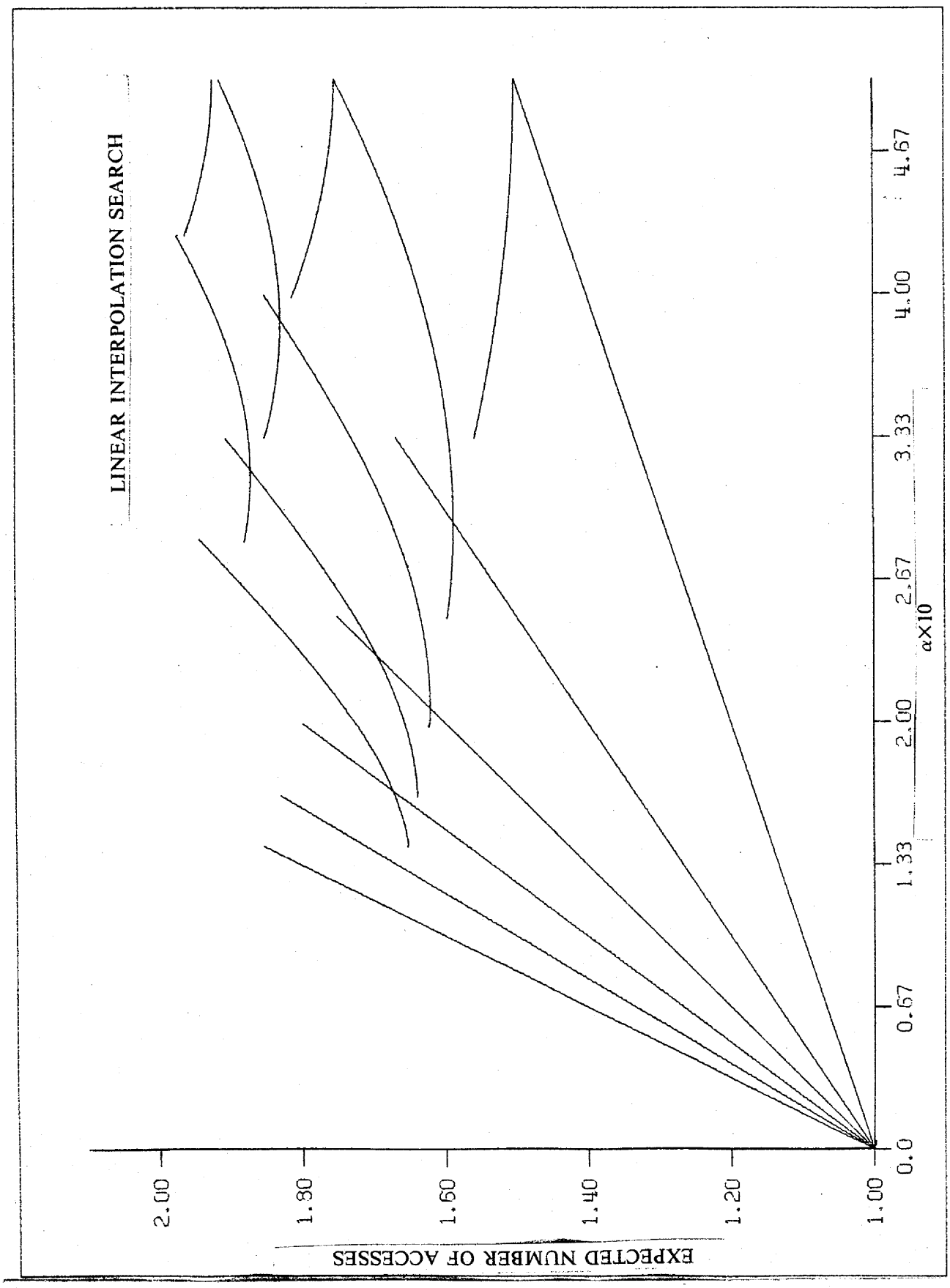
LINEAR INTERPOLATION SEARCH
EXPECTED NUMBER OF ACCESSES
2.00
1.80
1.60
1.40
1.20
1.00
0.0
0.67
1.33
2.00
2.67
3.33
4.00
4.67
αX10

Figure 3.4.I

We prove by induction on n that

$$(3.4.12)\qquad B(\alpha,n) = 1 + (n-1)\alpha, \quad \text{iff} \quad n\alpha<1$$

$$B(\alpha,n) = 1 + (n-1)\int_0^{\alpha} (1-w)^{n-2}B((\alpha-w)/(1-w),n-1)\,dw$$

$$= 1 + (n-1)\int_0^{\alpha} (1-w)^{n-2}[1+(n-2)(\alpha-w)/(1-w)]\,dw$$

$$= 1 + (n-1)\alpha.$$

Applying the recursion formula one step and using the previous result we find that if $1/n\leqslant\alpha<1/(n-2)$ then

$$(3.4.13)\qquad B(\alpha,n) = 1 + \int_{\alpha}^{1} I_w'(1,n-1)\,dw + \int_0^{\alpha} I_w'(2,n-2)[1+(n-3)(\alpha-w)/(1-w)]\,dw$$

$$= 2(1-\alpha)^{n-1} + (n-1)\alpha.$$

Optimal Interpolation Search.

The discontinuity of $B(\alpha,n)$ at $\alpha=1/n$ has a jump of

$$(3.4.14)\qquad B((1/n)^{-},n)-B((1/n)^{+},n) = 1-2e^{-1}(1+1/(2n)) + O(n^{-2})$$

As we can see in figure 3.4.I, these discontinuities also happen for other break points for small n. This means that the break point to decide to search in position 1 or 2 is not optimal. If we break in the point λ, such that $B(\lambda,n)$ has no jump, we reduce the expected number of accesses. The break point λ satisfies the equation

$$(3.4.15)\qquad 1+(n-1)\lambda = 2(1-\lambda)^{n-1}+(n-1)\lambda,$$

$$\lambda = 1 - 2^{-[1/(n-1)]} = \ln(2)/n + O(n^{-2}).$$

The average number of accesses gained by using λ instead of $1/n$ as a break point is

$$(3.4.16)\qquad \int_{\lambda}^{1/n} [1-2(1-x)^{n-1}]\,dx = (2e^{-1}-\ln(2))/n + O(n^{-2}),$$

$$\sim 0.0426/n.$$

After noticing this peculiarity we are lead to the definition of the optimal algorithm. We want to determine the optimal $\phi(\alpha,n)$ that will minimize the corresponding $B(\alpha,n)$ for any α or n. Let $B^*(\alpha,n)$ be the optimal average number of accesses needed to locate an entry in the table and $\phi^*(\alpha,n)$ the interpolation formula that defines the optimal algorithm. We have the boundary conditions $B^*(\alpha,1) = 1$ and $B^*(\alpha,2) = 1+\min(\alpha,1-\alpha)$. Note that for n=2 there is no jump in $B(\frac{1}{2},2)$.

In the recursion formula 3.4.2, $I_w'(...)$ and $B(\alpha/w,...)$ are always nonnegative. Hence the optimal $B^*(\alpha,n)$ will imply the use of optimal $\phi^*(...,j-1)$ and $\phi^*(...,n-j)$. Let $B^*_j(\alpha,n)$ be the result of searching position j looking for α between n elements using ϕ^* after the first probe. Let $\lambda^*_{i,n}$ be the break point of the optimal interpolation formula as defined in section 3.2, then

(3.4.17) $$B^*_j(\lambda^*_{j,n},n) = B^*_{j+1}(\lambda^*_{j,n},n).$$

The first few B^* and their corresponding $\lambda^*_{i,n}$ are

(3.4.18) $$B^*(\alpha,3) = 1+2\alpha, \quad 0\leqslant\alpha<1-2^{-\frac{1}{2}} = \lambda^*_{1,3} = 0.292893...$$

$$= 2(1-\alpha+\alpha^2), \quad \lambda^*_{1,3}\leqslant\alpha<1-\lambda^*_{1,3}.$$

$$\int_0^1 B^*(\alpha,3)\,d\alpha = 1.39052...$$

(3.4.19) $$B^*(\alpha,4) = 1+3\alpha, \quad 0\leqslant\alpha<\lambda^*_{1,4} = 1-2^{-1/3} = 0.206299...$$

$$= -2\alpha^3+6\alpha^2-3\alpha+2, \quad \lambda^*_{1,4}\leqslant\alpha<\tfrac{1}{2}.$$

(3.4.20)

$$\int_0^1 B^*(\alpha,4)\,d\alpha = 1.49695... \quad B(\alpha,5) = 1+4\alpha, \quad 0\leqslant\alpha<\lambda^*_{1,5} = 0.159103...$$

$$= 2\alpha^4-8\alpha^3+12\alpha^2-4\alpha+2, \quad \lambda^*_{1,5}\leqslant\alpha<0.292893...$$

$$= 14\alpha^4-(32\lambda^*_{1,3}+24)\alpha^3+96\lambda^*_{1,3}\alpha^2-(96\lambda^*_{1,3}+20)\alpha+32\lambda^*_{1,3}-7,$$

$$0.292893\leqslant\alpha<\lambda^*_{2,5} = 0.387457...$$

$$= -20\alpha^4+40\alpha^3-24\alpha^2+4\alpha+2, \quad \lambda^*_{2,5}\leqslant\alpha<\tfrac{1}{2}.$$

$$\int_0^1 B^*(\alpha,5)\,d\alpha = 1.58511...$$

As a corollary from 3.4.20 we see that the optimal interpolation search algorithm is defined by an interpolation formula which is not linear for $n=5$.

The following graph plots $B^*(\alpha,n)$ for $0 \leqslant \alpha \leqslant ½$ and $2 \leqslant n \leqslant 7$.

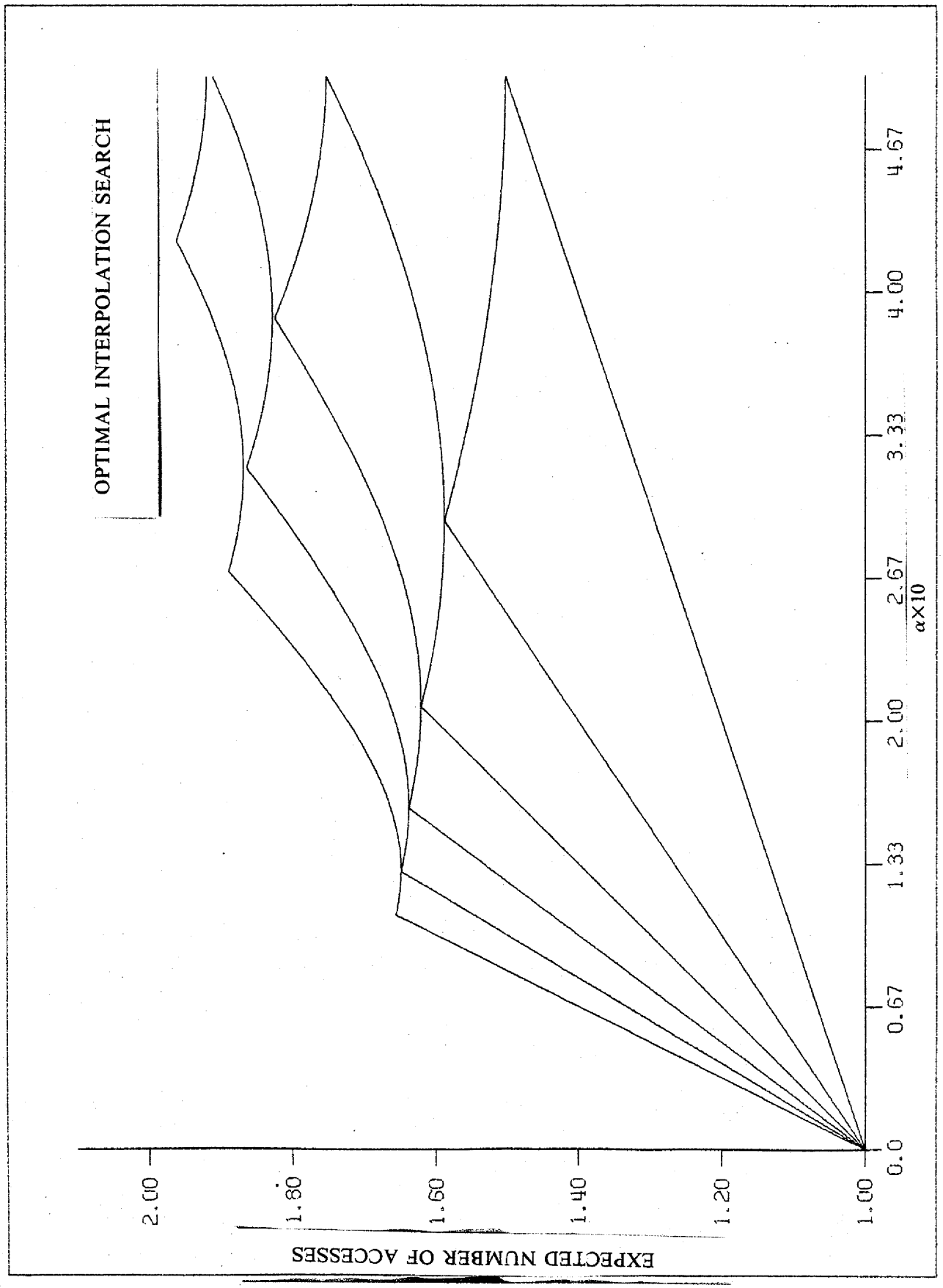
OPTIMAL INTERPOLATION SEARCH
EXPECTED NUMBER OF ACCESSES
2.00
1.80
1.60
1.40
1.20
1.00
0.0
0.67
1.33
2.00
2.67
3.33
4.00
4.67
α×10

Figure 3.4.II

It should be noted that the differences between the optimal and the simple linear algorithms are practically negligible. The following table compares the average for any α of both methods and the percentage difference with respect to the optimal up to n=7.

n	$\int_0^1 B(x,n)dx$	$\int_0^1 B^*(x,n)dx$	% difference
2	1.25	1.25	0.0
3	1.39506	1.39052	0.326
4	1.50390	1.49695	0.465
5	1.59348	1.58511	0.528
6	1.66859	1.65903	0.576
7	1.73192	1.72121	0.622

Numerical Approximations.

The computation of the exact $B(\alpha,n)$ for even moderate n seems a hopeless task. Before studying its asymptotic behaviour we may try to find numerical approximations of $B(\alpha,n)$. We will consider in the following analysis the linear interpolation algorithm 3.2.6. In this case the function $B(\alpha,n)$ has a continuous first derivative in each $(i-1)/n \leqslant \alpha < i/n$ interval. For large n, intuitively, the function in these intervals is rather "smooth".

We should not attempt to use a high order numerical method of integration that relies on high derivatives, since we may have discontinuities in the derivatives of $B(\alpha,n)$ that will cause singularities in higher derivatives. We will choose here a 3-point sectional definition of $B(\alpha,n)$, and a Simpson type quadrature formula. The approximation can be done with any other numerical method of any number of points [Davis 67]. Let the j^{th} continuous section of $B(\alpha,n)$ $[(j-1)/n \leqslant \alpha < j/n]$ be described by the set of points

$$(3.4.21) \qquad B_j(k,n) \sim B(\alpha_k,n), \qquad (1 \leqslant k \leqslant 3),$$

where $\alpha_k=(2j+k-3)/(2n)$.

To approximate the integrals that define $B(\alpha,n)$ we will integrate separately in each interval where $B(\alpha,j-1)$ or $B(\alpha,n-j)$ are continuous. Moreover since $I_w'(j-1,n-j+1)$ may introduce substantial errors because of its large derivatives, we will approximate $B(\alpha,...)$ with a parabola (midpoint and both extremes, as in Simpson's rule) and integrate the resulting rational polynomial with an approximate or exact method. For the first integral we obtain

(3.4.22)

$$\int_{\alpha}^{1} I_w'(j-1,n-j+1)B(\alpha/w,j-1)\,dw = \sum_{i=m}^{j-1}\int_{a(i)}^{b(i)} I_w'(j-1,n-j+1)B(\alpha/w,j-1)\,dw.$$

According to the discontinuities of B(x,j-1), we conclude that

(3.4.23) $\quad m = \lfloor (j-1)\alpha \rfloor + 1,$

and

(3.4.24) $\quad (i-1)/(j-1) \leqslant \alpha/w \leqslant i/(j-1),$

whence

(3.4.25) $\quad a(i) = \alpha(j-1)/i \quad \text{and} \quad b(i) = \min[\alpha(j-1)/(i-1),1].$

Let $B_i(x,j-1)$ denote the i^{th} interval polynomial approximation of $B(\alpha,j-1)$.

Similarly for the second integral we obtain

(3.4.26) $$\int_{0}^{\alpha} I_w'(j,n-j)B((\alpha-w)/(1-w),n-j)\,dw = \sum_{i=1}^{m}\int_{a(i)}^{b(i)} I_w'(j,n-j)B_i((\alpha-w)/(1-w),n-j)\,dw,$$

where

(3.4.27) $\quad m = \lceil (n-j)\alpha \rceil,$

(3.4.28) $\quad (i-1)/(n-j) \leqslant (\alpha-w)/(1-w) \leqslant i/(n-j),$

and

(3.4.29) $\quad a(i) = \max[0,(\alpha(n-j)-i)/(n-j-i)], \quad b(i) = [\alpha(n-j)-i+1]/(n-j-i+1).$

The computation of $B(\alpha,n)$ requires the computation of all $B(\alpha,j)$ for $j<n$, hence requiring $1.5(n+1)n$ values to be stored and $O(n^3)$ time.

The approximations may be enhanced by incorporating the exact results obtained in 3.4.8–3.4.13.

Formally there is no way of guaranteeing convergence of these approximations. However we can compute exact and approximate values of $B(\alpha,n)$ for small n and observe the actual errors. These errors happen to be encouragingly small as shown in the following table. The table also shows the remarkably good empirical approximation $lg(lg(n+3))$.

n	$\int_0^1 B(\alpha,n)\, d\alpha$	approximation	$lg(lg(n+3))$
1	1.0	1.0	1.00000000
2	1.25	1.25	1.21532329
3	1.39506173	1.39506173	1.37014335
4	1.50390625	1.50390625	1.48921147
5	1.59347704	1.59320666	1.58496250
6	1.66858659	1.66848482	1.66444871
7	1.73191780	1.73203984	1.73202085
8		1.7864062	1.7905350
9		1.8339951	1.8419580
10		1.8764515	1.8876967
11		1.9148106	1.9287891
12		1.9497450	1.9660209
13		1.9817598	2.0000000
14		2.0112674	2.0312056
15		2.0386100	2.0600214
16		2.0640689	2.0867591
17		2.0878730	2.1116751
18		2.1102112	2.1349823
19		2.1312421	2.1568598
20		2.1511005	2.1774592
25		2.2363592	2.2652433
30		2.3045439	2.3346810
35		2.3610800	2.3917478
40		2.4091944	2.4399594
45		2.4509556	2.4815476
50		2.4877650	2.5180115
55		2.5206141	2.5504035
60		2.5502289	2.5794891

Figure 3.4.III

Numerical Bound on the Average.

For a fixed first probe position, the function $B(\alpha,n)$ has a minimum (the optimal key value to search in that location). Let us make the natural assumption, without demonstration, that in the interval $(i-1)/n \leqslant \alpha \leqslant i/n$ this minimum is unique and $B''(\alpha,n)>0$. Consequently, as we can also verify for small n, the maximum of $B(\alpha,n)$ in this interval occurs either at $\alpha=(i-1)/n$ or at $\alpha=i/n$.

Using the recursion formula 3.4.2 we can numerically bound the values of $B(\alpha,n)$ for each interval and also bound the average $B(\alpha,n)$ for any α. Unfortunately the bound is only numerical and not tight enough to deserve more attention.

The following table compares exact values of the average number of accesses and the numerical bound.

n	exact value	numerical bound
1	1.0	1.0
2	1.25	1.5
3	1.3950	1.6296
4	1.5039	1.7822
5	1.5935	1.8969
6	1.6686	1.9775
7	1.7319	2.0409
8	1.7864	2.0892
9	1.8340	2.1366
10	1.8765	2.1765
11	1.9148	2.2131
12	1.9497	2.2472
13	1.9818	2.2795
14	2.0113	2.3085
15	2.0386	2.3353
16	2.0640	2.3599
17	2.0879	2.3829
18	2.1102	2.4042
19	2.1312	2.4242
20	2.1511	2.4429

Figure 3.4.IV

Asymptotic Behaviour of the Average Number of Accesses.

Several measures on the algorithm indicated that the quantity $n\alpha(1-\alpha)$ was of key importance. Intuitively we find that after the first step of the search, the key tends to be located at the extremes of the interval, rather than near the middle. This fact together with the symmetry of the search are well reflected by $n\alpha(1-\alpha)$.

Consequently, let us regard the quantity $\theta = n\alpha(1-\alpha)$ as a basic parameter in the search and relate it to any step in the algorithm. The expected value of $\theta_1 = n_1\alpha_1(1-\alpha_1)$ (where n_i and α_i will indicate the size and relative key value of the file after the i^{th} iteration) is given by

$$(3.4.30)\quad E[\theta_1] = 0\times Pr\{X_j=\alpha\} + \int_{\alpha}^{1} I_w'(j-1,n-j+1)(j-1)\alpha(w-\alpha)dw/w^2 + \int_{0}^{\alpha} I_w'(j,n-j)(n-j)(\alpha-w)(1-\alpha)dw/(1-w)^2.$$

As in section 3.3, we will separate the rapidly varying and slowly varying parts of the integral. We note that the expansion of the slowly varying factor contains terms of the type $|w-\alpha|$, so we will slightly modify the Incomplete Beta function so that α coincides with its average, and then apply the central signed moments described in equation 2.4.15. We find

$$(3.4.31)\quad E[\theta_1] = \int_{0}^{1} I_w'(j-1+\beta,n-j)G(w)\,dw,$$

where

$$(j-1+\beta)/(n-1+\beta) = \alpha \implies \beta = (\Delta-\alpha)/(1-\alpha),$$

$$\Delta = n\alpha - \lfloor n\alpha \rfloor,$$

and

$$G(w) = d_0 + d_1(w-\alpha) + e_1|w-\alpha| + d_2(w-\alpha)^2 + e_2|w-\alpha|(w-\alpha) + \ldots .$$

The coefficients d_i and e_i are obtained by equating the corresponding Taylor expansions at $w=\alpha$ for $w>\alpha$ and $w<\alpha$. We note that at most $d_i = O(n)$ and $e_i = O(n)$. Consequently if we want an asymptotic expansion up to $o(1)$, we need d_0, d_1, d_2 and e_1. Hence using the notation of 2.4 we obtain

$$(3.4.32)\quad d_0 = 0,$$

(3.4.33) $d_1 = (\Delta-\alpha)/[\alpha(\alpha-1)] + O(\theta^{-1}),$

(3.4.34) $d_2 = -3n/[2\alpha(1-\alpha)] + O(1),$

(3.4.35) $e_1 = n + O(1),$

and

(3.4.36) $E[\theta_1] = d_0 + e_1\nu_1 + d_2\mu_2 + O(\theta^{-1/2})$

$= [2\theta/\pi]^{1/2} - 3/2 + O(\theta^{-1/2}).$

Let h_0 be a threshold such that for $\theta>h_0$ we have

(3.4.37) $E[\theta_1] \leqslant [2\theta/\pi]^{1/2}.$

At this point we should be careful to notice that θ_1 is a random variable in contraposition to θ, which is a function of α and n. To recursively use the above inequality, we observe that $E[\theta_1]$ is a convex function of θ and we can apply Jensen's inequality to obtain

(3.4.38) $E[\pi\theta_i/2] \leqslant [\pi\theta/2]^{2^{-i}}.$

When $\theta_i \leqslant h_0$ we change the algorithm, and search sequentially from the left if $\alpha \leqslant 1/2$ or from the right otherwise. Under this condition, applying 2.4.24, the remaining average number of accesses is

(3.4.39) $E[\text{accesses}] = n\times\min(\alpha,1-\alpha)+1 \leqslant 2h_0+1.$

Finally the total number of accesses is

(3.4.40) $E[\text{accesses}] \leqslant 2h_0+1+lg(lg(\pi\theta/2)/lg(\pi h_0/2))$

$\leqslant lg(lg(n\alpha(1-\alpha))) + O(1).$

More precisely, if we construct

(3.4.41) $G^*(w) = (2/\pi)^{1/2}|w-\alpha|/\nu_1^2 + d_1(w-\alpha),$

then

$$(3.4.42)\qquad \int_0^1 I_w'(j-1+\beta,n-j)G^*(w)\,dw = [2\theta/\pi]^{1/2}.$$

We find, rather laboriously, that

$$(3.4.43)\qquad G(\alpha) = G^*(\alpha),$$

$$(3.4.44)\qquad [G(w)-G^*(w)]_w' \geqslant 0, \quad \text{for } w<\alpha,$$

and

$$(3.4.45)\qquad [G(w)-G^*(w)]_w' \leqslant 0, \quad \text{for } w>\alpha.$$

Thus $G(w)\leqslant G^*(w)$ and finally 3.4.37 is true for any θ. Consequently we can set $h_0=1$ and we find

$$(3.4.46)\qquad E[\text{accesses}] \leqslant lg(lg(\pi\theta/2)) + 3.6181\ldots .$$

We find a slightly different approach in Yao and Yao's work [1976]. First, their interpolation formula is defined by $\lfloor\alpha(n-1)+1\rfloor$ for $\alpha\leqslant\frac{1}{2}$ and $\lceil\alpha(n-1)+1\rceil$ for $\alpha>\frac{1}{2}$. The main proof is for an upper and lower bound on the average number of accesses in the unsuccessful search only. Also of interest is the claim that small perturbations of the probe position do not affect the order of its running time. This result follows from our analysis too, since as long as $\Delta=o(n^{1/2})$, we have the same asymptotic results [eqs. 3.4.30–3.4.36].

3.5 - Distribution of the Number of Accesses.

The main motif of this section is to find the probability function of the number of accesses needed to locate an element known to be in the table. We already know that the average of this distribution is $lg(lg(n))+O(1)$ so our interest will be mainly to prove asymptotic properties of higher moments and/or to obtain more accurate approximations of the average.

We will start by analyzing the probability of success in the first trial for the linear interpolation algorithm. That is, given the key α, and

$$(3.5.1) \qquad \phi(\alpha,n) = \lfloor n\alpha \rfloor + 1 = j,$$

the probability of success in the first trial [2.4.24] is

$$(3.5.2) \qquad \Pr\{X_j=\alpha\} = I_\alpha'(j,n-j+1)/n.$$

Computing the average value for any α for any random file we obtain

$$(3.5.3) \quad \Pr\{\text{success in first trial}\} = \int_0^1 \Pr\{X_j=\alpha\}\, d\alpha$$

$$= n^{-1}\sum_{j=1}^{n}[I_{j/n}(j,n-j+1)-I_{(j-1)/n}(j,n-j+1)]$$

$$= n^{-1}[1 + \sum_{j=1}^{n-1} I_{j/n}(j,n-j+1)-I_{j/n}(j+1,n-j)]$$

$$= n^{-1}[1 + \sum_{j=1}^{n-1}\binom{n}{j} j^j(n-j)^{n-j}n^{-n}]$$

$$= [\pi/(2n)]^{1/2}[1+1/(12n)] - 1/(3n) + O(n^{-2}),$$

which is a very satisfactory asymptotic expansion.

However when we compute the probability of exactly 2 trials we find

$$(3.5.4) \quad \Pr\{2 \text{ trials}\} = 2/n\sum_{j=1}^{n}\sum_{k=j+1}^{p}\int_{(k-1)/n}^{k/n} I_w'(j,n-j+1)$$

$$[I_a(k-j,n-k+1)-I_b(k-j,n-k+1)]\, dw,$$

where

(3.5.5) $a = \min(w(k-j)/[(j-1)(1-w)],1),$

$b = \max(w(k-j-1)/[j(1-w)],0),$

$p = \lceil [nj]^{1/2} \rceil .$

This formula is very difficult to handle, and for further trials it seems even more complicated. Consequently we resort to a different approach. Let $\theta_i = n_i\alpha_i(1-\alpha_i)$ be a basic parameter of the file we are searching, as we already used in 3.4. We know a recursion formula that relates θ_{i+1} to θ_i. This recursion formula is independent of the outcome of the search. We now give a recursion formula, conditioned to the failure of the trial, i.e.

(3.5.6) $\theta_{i+1} = \{[2\theta_i/\pi]^{1/2} - 3/2 + O(\theta^{-1/2})\} / (1-\Pr\{X_j=\alpha\}).$

We will first express the probability of finding the element in the i^{th} trial in terms, if possible, of θ_i. Using Stirling's expansion for $ln(\Gamma(n))$ we find that

(3.5.7) $ln(\Pr\{X_j=\alpha\}) = -\frac{1}{2}ln(2\pi\theta) + 1/(12n) + [6\Delta(1-\Delta)-1]/(12\theta) +$

$\Delta(2\Delta-1)(\Delta-1)(2\alpha-1)/(12\theta^2) + O(\theta^{-3}),$

where $\Delta = n\alpha - \lfloor n\alpha \rfloor$. This is an asymptotic expansion in terms of θ, but unfortunately, also a function of n and α (not only θ). For practical purposes, or to compute average situations, we assume that Δ is a random variable U(0,1). This is true for $\theta \to \infty$ and fixed α. We may then compute the average value, with respect to Δ, of each term of the expansion, yielding

(3.5.8) $\Pr\{X_j=\alpha\} = [2\pi\theta]^{-1/2}[1 + 1/(12n) + 1/(288n) + 1/(1440\theta^2) + O(\theta^{-3})].$

This expansion suggests that the approximation

(3.5.9) $\Pr\{X_j=\alpha\} = [2\pi\theta]^{-1/2}$

should be very good on the average.

The following table shows some values of the exact probability and its approximation for $0.5 \leqslant \theta \leqslant 2.5$, which can be considered an extreme situation.

θ	n=5	n=10	n=∞	approximation
0.5	0.6198	0.6138	0.6065	0.5642
0.75	0.4439	0.4645	0.4724	0.4607
1.	0.4189	0.3897	0.3679	0.3989
1.5		0.3258	0.3347	0.3257
2.		0.2857	0.2707	0.2821
2.5		0.2461	0.2565	0.2523

Table 3.5.I

Let $\xi_i = 2/(\pi\theta_i)$, and $\xi = 2/[\pi n\alpha(1-\alpha)]$. The probability of success in the first trial is approximated by

(3.5.10) $\Pr\{X_j=\alpha\} = \frac{1}{2}\xi^{1/2}$.

In case the probe is unsuccessful we compute the conditional expected value of the resulting θ,

(3.5.11) $\theta_{i+1} = [2\theta_i/\pi]^{1/2} + \pi^{-1} - 3/2 + O(\theta_i^{-1/2})$.

Using 3.4.38 we find

(3.5.12) $\xi_i \leqslant \xi^{2^{-i}}$.

The probability of success in the second trial, conditioned to an unsuccessful first trial and using equation 3.5.9 is

(3.5.13) $\Pr\{\text{success in second trial}\} = [2\theta_1/\pi]^{1/2} \geqslant \frac{1}{2}\xi^{1/4}$.

Similarly,

(3.5.14) $\Pr\{\text{success in third trial}\} \geqslant \frac{1}{2}\xi^{1/8}$,

etc.

The probability of no success in k trials is

$$(3.5.15)\quad \Pr\{\text{no success in k trials}\} \leqslant \prod_{i=1}^{k}(1-\tfrac{1}{2}\xi^{2^{-i}}).$$

Consequently we can bound all positive moments of the distribution of the number of accesses. We can check the probability of finding an element in the first probe with the exact result. In this case

$$(3.5.16)\quad \Pr\{X_j=\alpha\} \sim [2\pi n\alpha(1-\alpha)]^{-\frac{1}{2}},$$

and for any α we have

$$(3.5.17)\quad \Pr\{\text{success in 1 probe}\} \sim [2\pi n]^{-\frac{1}{2}}\int_0^1 [\alpha(1-\alpha)]^{-\frac{1}{2}}\,d\alpha' = [\pi/(2n)]^{\frac{1}{2}},$$

which coincides with the first asymptotic term of 3.5.3.

Doubly Exponential Distribution.

We will study in some detail the characteristics of the probability distribution defined before, namely

$$(3.5.18)\quad \Pr\{x>k\} = \prod_{i=1}^{k}(1-\tfrac{1}{2}\xi^{2^{-i}}),\ 0<\xi\leqslant 1.$$

The expected value

$$(3.5.19)\quad E[x] = T(\xi) = \sum_{i=0}^{\infty}\prod_{j=1}^{i}(1-\tfrac{1}{2}\xi^{2^{-j}})$$

satisfies the functional equation

$$(3.5.20)\quad T(\xi^2) = T(\xi)(1-\xi/2) + 1,$$

with

$$(3.5.21)\quad T(0) = \infty,$$

and

(3.5.22) $T(1) = 2.$

Let $H(\xi)$ be the solution of the homogeneous part of the functional equation for $T(\xi)$. We have

(3.5.23) $H(\xi)(1-\xi/2) = H(\xi^2) \Longrightarrow$

$$H(\xi) = [1 + \xi/2 + 2\xi^2/4 + 3\xi^3/8 + 15\xi^4/16 + ..]\times P(lg(-lg(\xi))),$$

where $P(x)$ is any periodic function with period 1. Fortunately we find that for this distribution $|P(x)-C| \sim 10^{-6}$ where C is a constant and we can safely ignore this periodic factor for practical purposes. Notice that

(3.5.24) $T(\alpha) = T(1-\epsilon) = 2 - 2\epsilon/5 + O(\epsilon^2)$

is a continuous function when $\epsilon \to 0$. This supports our contention that we may safely ignore the periodicity of $P(lg(-lg(\xi)))$.

We compute then

(3.5.25) $T(\xi) = [C + lg(-lg(\xi))]H(\xi) + R(\xi),$

where

(3.5.26) $R(\xi)(1-\xi/2) + 1 - H(\xi^2) = R(\xi^2) \Longrightarrow$

$$R(\xi) = \xi^2/2 + \xi^3/4 + 11\xi^4/8 + ...,$$

and

(3.5.27) $C = 1.3580..$.

Clearly for $\xi \to 0$ we have

(3.5.28) $E[x] = T(\xi) = lg(-lg(\xi)) + 1.3580.. + O(\xi).$

For the variance of x, we first compute the second moment,

(3.5.29) $E[x^2] = V(\xi) = \sum_{i=0}^{\infty}(2i+1)\prod_{j=1}^{i}(1-\frac{1}{2}\xi^{2^{-j}})$.

The second moment satisfies the functional equation

(3.5.30) $[V(\xi) + 2T(\xi)](1-\xi/2) + 1 = V(\xi^2)$,

where

(3.5.31) $V(0) = \infty$,

(3.5.32) $V(1) = 6$,

and if

(3.5.33) $V(\xi) = G_2(\xi)[lg(-lg(\xi))]^2 + G_1(\xi)lg(-lg(\xi)) + G_0(\xi)$,

we find that $G_2(\xi)$ and $G_1(\xi)$ have the same general solution as $H(\xi)$.

Computing the general homogeneous solution to 3.5.30 and evaluating the constants involved we can compute the variance:

(3.5.34) $var\{x\} = 3.876..P(lg(-lg(\xi))) + o(1)$.

The following table shows some exact results of the mean, variance, skewness and excess of the distribution for various ξ.

ξ	E[x]	$E[x^2]$	var{x}	skewness	excess
0.5000	2.43848	8.6006	2.6543	1.6300	-0.8778
0.1000	3.27465	14.1975	3.4741	1.4399	-1.4736
0.50E-01	3.56448	16.3424	3.6369	1.3910	-1.6074
0.10E-01	4.11092	20.7095	3.8098	1.3155	-1.8020
0.10E-02	4.67741	25.7451	3.8669	1.2554	-1.9494
0.10E-03	5.09037	29.7866	3.8748	1.2207	-2.0330
0.10E-04	5.41206	33.1662	3.8758	1.1980	-2.0876
0.10E-05	5.67507	36.0823	3.8759	1.1817	-2.1268
0.10E-06	5.89746	38.6560	3.8759	1.1694	-2.1566

0.10E-07	6.09011	40.9654	3.8759	1.1597	-2.1803
0.10E-08	6.26003	43.0639	3.8759	1.1517	-2.1997
0.10E-09	6.41203	44.9901	3.8759	1.1451	-2.2161
0.10E-10	6.54953	46.7723	3.8760	1.1394	-2.2301
0.10E-11	6.67506	48.4325	3.8760	1.1345	-2.2422
0.10E-12	6.79055	49.9875	3.8760	1.1302	-2.2529
0.10E-13	6.89746	51.4509	3.8759	1.1264	-2.2624
0.10E-14	6.99700	52.8340	3.8759	1.1230	-2.2710
0.10E-15	7.09011	54.1456	3.8759	1.1200	-2.2787
0.10E-16	7.17757	55.3935	3.8759	1.1172	-2.2857
0.10E-17	7.26003	56.5840	3.8759	1.1147	-2.2922
0.10E-18	7.33803	57.7227	3.8759	1.1123	-2.2981
0.10E-19	7.41203	58.8142	3.8759	1.1102	-2.3036

Table 3.5.II

The main conclusion of the above section is that the variance of the number of accesses is $O(1)$.

Approximate Distribution of the Number of Accesses.

So far we avoided the problem of retaining some degree of accuracy when θ is small. To accomplish this, in a numerical way, we will study the distribution of the number of accesses when $n\alpha \leqslant 1$. From 2.4.24 this is

$$(3.5.35) \quad \Pr\{\text{k accesses}\} = \binom{n-1}{k-1}\alpha^{k-1}(1-\alpha)^{n-k}.$$

The algorithm to compute a more approximate distribution is to compute the probability of exactly k accesses using equation 3.5.9 and the recursion formula 3.5.12 and then, when $n\alpha \leqslant 1$, use 3.5.35.

The following table was obtained by this procedure.

θ	E[x]	$E[x^2]$	var{x}	skewness	excess
10.00	2.39788	6.7405	0.9906	1.2701	-1.1056
100.0	3.24137	11.6635	1.1570	1.1639	-1.4971
1000.	3.35907	12.1719	0.8886	1.1293	-1.5961
1.00E04	4.18445	18.7022	1.1926	1.1026	-1.6981
1.00E05	4.34638	20.0031	1.1121	1.0921	-1.7274
1.00E07	4.98395	25.9686	1.1288	1.0682	-1.8048
1.00E09	5.26824	28.9285	1.1742	1.0655	-1.8111
1.00E10	5.33687	29.6065	1.1243	1.0620	-1.8205
1.00E14	5.96079	36.6432	1.1122	1.0474	-1.8665
1.00E15	6.06544	37.9617	1.1722	1.0487	-1.8625
1.00E19	6.30007	40.8478	1.1568	1.0456	-1.8704
1.00E20	6.33166	41.2201	1.1301	1.0443	-1.8737

Table 3.5.III

Note that the adjustment of the tail makes an important reduction in the variance that now coincides nicely with the simulation results in 3.9.

These results are for a specific θ, i.e. a combination of size and key value. However due to the flatness of $lg(lg(\theta))$, for large θ, these results are very close to the average for the corresponding size.

3.6 - Average Number of Accesses for an Unsuccessful Search.

In this section we will study the search for the key α, distributed U(0,1), that is known *not* to be in the table $0 \leqslant X_1 \leqslant \ldots \leqslant X_n \leqslant 1$. We will use the algorithm described in 3.2 expecting a FAIL outcome.

Exact Average Number of Accesses.

Let $C(\alpha,n)$ be the average number of accesses needed by the interpolation search algorithm to complete the unsuccessful search.

Analyzing one step of the algorithm we derive a recursion formula similar to 3.4.2, namely

$$(3.6.1)\quad C(\alpha,n) = 1 + \int_{\alpha}^{1} I_w'(j,n-j+1)C(\alpha/w,j-1)\,dw + \int_{0}^{\alpha} I_w'(j,n-j+1)C((\alpha-w)/(1-w),n-j)\,dw,$$

where $j = \phi(\alpha,n) = \lfloor n\alpha \rfloor + 1$, is the search position.

The boundary conditions are

$$(3.6.2)\quad C(\alpha,1) = 1,$$

and

$$(3.6.3)\quad C(\alpha,0) = 0.$$

By an analysis similar to the one done in section 3.4 we conclude that $C(\alpha,n)$ is sectionally defined by a polynomial of degree at most n.

The discontinuities of $C(\alpha,n)$ appear at the same α's as in the $B(\alpha,n)$ since the recursion formulas differ only in their continuous terms.

The first $C(\alpha,n)$ for the linear interpolation algorithm [3.2.6] are

$$(3.6.4)\quad C(\alpha,2) = -\alpha^2+2\alpha+1, \quad 0 \leqslant \alpha \leqslant \tfrac{1}{2}.$$

(3.6.5) $C(\alpha,3) = -\alpha^3+3\alpha+1, \quad 0\leqslant\alpha<1/3,$

$= 2, \qquad 1/3\leqslant\alpha\leqslant ½.$

(3.6.6) $C(\alpha,4) = -\alpha^4+4\alpha+1, \quad 0\leqslant\alpha<1/4,$

$= -3\alpha^4+4\alpha^3+2, \quad 1/4\leqslant\alpha\leqslant ½.$

(3.6.7) $C(\alpha,5) = -\alpha^5+5\alpha+1, \quad 0\leqslant\alpha<1/5,$

$= 2\alpha^5-10\alpha^4+10\alpha^3+2, \quad 1/5\leqslant\alpha<1/3,$

$= (-202\alpha^5+815\alpha^4-1240\alpha^3+850\alpha^2-230\alpha+55)/16, \quad 1/3\leqslant\alpha<2/5,$

$= -48\alpha^5+70\alpha^4-20\alpha^3-10\alpha^2+5\alpha+2, \quad 2/5\leqslant\alpha\leqslant ½.$

The following graph plots $C(\alpha,n)$ for $0\leqslant\alpha\leqslant ½$ and $2\leqslant n\leqslant 7$.

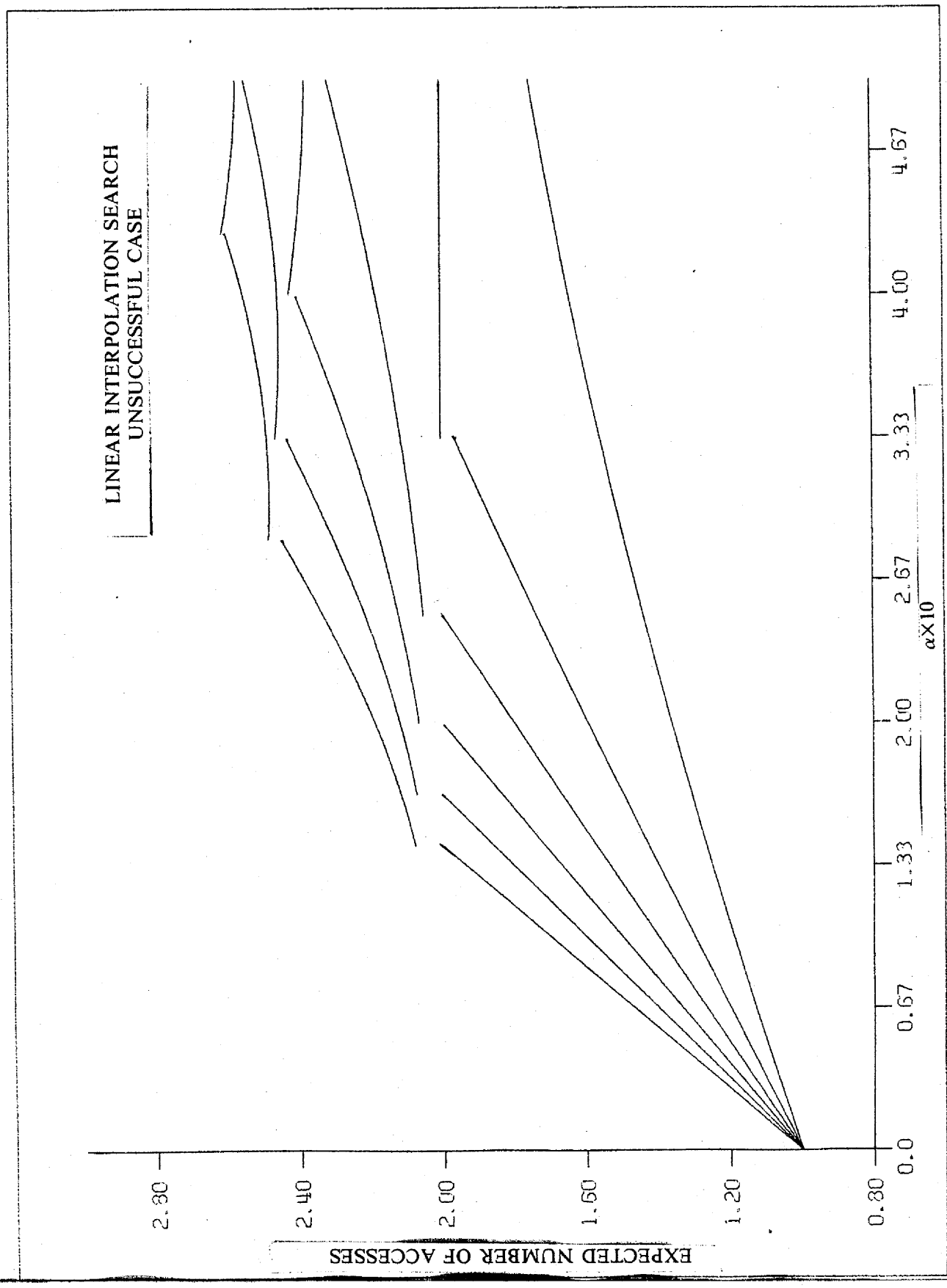
LINEAR INTERPOLATION SEARCH
UNSUCCESSFUL CASE
EXPECTED NUMBER OF ACCESSES
2.30
2.40
2.00
1.60
1.20
0.80
0.0
0.67
1.33
2.00
2.67
3.33
4.00
4.67
αX10

Figure 3.6.I

We prove by induction on n that

$$(3.6.8)\quad C(\alpha,n) = 1 + n\alpha - \alpha^n \quad \text{iff} \quad n\alpha \leqslant 1$$

$$= 1 + \int_0^{\alpha} I_w'(1,n)C((\alpha-w)/(1-w),n-1)\,dw$$

$$= 1 + n\int_0^{\alpha} (1-w)^{n-1}[1+(n-1)(\alpha-w)/(1-w) - [(\alpha-w)/(1-w)]^{n-1}]\,dw$$

$$= 1 + n\alpha - \alpha^n.$$

Applying the recursion formula one step and the previous result we find that for $1/n \leqslant \alpha \leqslant 1/(n-2)$ we have

$$(3.6.9)\quad C(\alpha,n) = 1 + \int_{\alpha}^{1} I_w'(2,n-1)\,dw + \int_0^{\alpha} I_w'(2,n-1)C((\alpha-w)/(1-w),n-2)\,dw$$

$$= [2+(n-2)\alpha](1-\alpha)^{n-1} + n\alpha - \alpha^n.$$

The discontinuity of $C(\alpha,n)$ at $\alpha=1/n$ has a jump of

$$(3.6.10)\quad C((1/n)^+,n) - C((1/n)^-,n) = (3-2/n)(1-1/n)^{n-1} - 1$$

$$= 3/e - 1 + O(1/n) \sim 0.1036 - 0.184/n.$$

The optimal break point, $\lambda_{1,n}$, is given by the equation

$$(3.6.11)\quad [2+(n-2)\lambda_{1,n}](1-\lambda_{1,n})^{n-1} = 1$$

$$\Rightarrow \quad \lambda_{1,n} = \beta/n + O(n^{-2}) \sim 1.1462/n$$

where $2+\beta=e^{\beta}$. We notice now that $\lambda_{1,n} > 1/n$, contrary to the successful case. This is a rather natural conclusion of the search.

Numerical Approximation.

Following the numerical method described in equations 3.4.22–29, changing only the arguments in the Incomplete Beta functions, we derive a method for computing approximate values of $C(\alpha,n)$ and its average. The following table shows a comparison between the exact values, the numerical approximation and the empirical formula $lg(lg(n))+0.58$.

n	$\int_0^1 C(\alpha,n)\, d\alpha$	numerical approximation	$lg(lg(n))+0.58$
1	1	1	
2	1.4166667	1.4166667	0.58
3	1.6604938	1.6604938	1.2444
4	1.8304688	1.8304036	1.5800
5	1.9610972	1.9608513	1.7953
6	2.0665118	2.0661508	1.9501
7	2.1537838	2.1535058	2.0692
10		2.3469752	2.3120
15		2.5539017	2.5460
20		2.6922305	2.6917
25		2.7945151	2.7953
30		2.8748405	2.8748
35		2.9405102	2.9388
40		2.9957660	2.9919
45		3.0432753	3.0373
50		3.0848178	3.0767
55		3.1216359	3.1114
60		3.1546291	3.1424

Figure 3.6.II

Asymptotic Behaviour of the Average Number of Accesses.

Following the same development as in 3.4 we define

(3.6.12) $\theta_i = n_i\alpha_i(1-\alpha_i)$

to be considered a basic parameter of the search, where the subscript i indicates the respective value after step i of the search algorithm. We have

$$(3.6.13)\quad E[\theta_1] = \int_{\alpha}^{1} I_w'(j,n-j+1)(j-1)\alpha(\alpha-w)\,dw/w^2 + \int_{0}^{\alpha} I_w'(j,n-j+1)(n-j)(\alpha-w)(1-\alpha)\,dw/(1-w)^2.$$

We separate the rapidly varying factor of the Incomplete Beta function and modify it so that α coincides with the average. We expand the remaining factor in a power series in $(w-\alpha)$ and $|w-\alpha|$ obtaining

$$(3.6.14)\quad E[\theta_1] = \int_{0}^{1} I_w'(j+\beta,n-j+1)G(w)\,dw,$$

where $(j+\beta)/(n+1+\beta) = \alpha \implies \beta = (\Delta-1+\alpha)/(1-\alpha)$,

$$\Delta = n\alpha-\lfloor n\alpha\rfloor,$$

$$j = \phi(\alpha,n) = \lfloor n\alpha\rfloor+1,$$

and

$$G(w) = d_0 + d_1(w-\alpha) + e_1|w-\alpha| + d_2(w-\alpha)^2 + e_2|w-\alpha|(w-\alpha) + \ldots .$$

The coefficients d_i and e_i are obtained by equating the corresponding Taylor expansions at $w=\alpha$ for $w>\alpha$ and $w<\alpha$. We note that at most $d_i = O(n)$ and $e_i = O(n)$. Consequently if we want an asymptotic expansion up to $o(1)$, we need d_0, d_1, d_2 and e_1. Hence using the notation of section 2.4 we obtain

$$(3.6.15)\quad d_0 = 0,$$

$$(3.6.16)\quad d_1 = -(\Delta-\alpha)/[2\alpha(1-\alpha)] + O(\theta^{-1}),$$

$$(3.6.17)\quad d_2 = -n/[\alpha(1-\alpha)] + O(1),$$

$$(3.6.18)\quad e_1 = n + O(1),$$

and

(3.6.19) $E[\theta_1] = d_0 + e_1\nu_1 + d_2\mu_2 + O(\theta^{-1/2})$

$= [2\theta/\pi]^{1/2} - 1 + O(\theta^{-1/2})$.

Since we are also interested in the case of small θ we want a formula of the type

(3.6.20) $E[\theta_1] \leqslant [2\theta/\pi]^{1/2}$.

This inequality is valid for all $\theta>0$ and is demonstrated, rather tediously as in 3.4.41–45. The remaining analysis is exactly the same as in 3.4 since the reduction formula 3.6.20 coincides with 3.4.37. The expected number of accesses for the unsuccessful search is bounded by

(3.6.21) E[accesses] $\leqslant lg(lg(\pi\theta/2)) + 3.6181\ldots$.

This asymptotic bound confirms the validity of the empirical approximation $lg(lg(n))+0.58$.

Simulation results corresponding to this section appear in section 3.9.

3.7 - Average Number of Accesses for Known-Successful Search.

Normal search algorithms do not rely on the fact that it may be known that the element is in the table. Surprisingly, the optimum algorithm varies greatly if we consider this assumption.

This assumption will normally be considered "too dangerous", since we want to construct fail-proof algorithms. Nevertheless it is theoretically interesting because of the low average number of comparisons required, and because it is the optimal search algorithm under these conditions (ordered random table and element known to be in it).

The algorithm, coded in a pseudo language, is the same as the one described in 3.2 except for the **while** statement that is changed to

while (high-low>2) **do**

and the final **return** statement that becomes

return(low+1);

Let $D(\alpha,n)$ be the average number of accesses needed to find α among n ordered elements using the optimal algorithm. Note that the normal linear interpolation algorithm is not interesting for this type of search, since it is almost the same as the case described in 3.4.

The function $D(\alpha,n)$ follows the same recurrence formula as $B(\alpha,n)$ (3.4.2)

$$(3.7.1)\quad D(\alpha,n) = 1+\int_{\alpha}^{1} I_w'(j-1,n-j+1)D(\alpha/w,j-1)\,dw + \int_{0}^{\alpha} I_w'(j,n-j)D((\alpha-w)/(1-w),n-j)\,dw,$$

but with the boundary conditions

$$(3.7.2)\quad D(\alpha,1) = 0,$$

$$(3.7.3)\quad D(\alpha,2) = 1,$$

and

$$(3.7.4)\quad D(\alpha,3) = 1.$$

Let $\lambda_{i,n}$ be the break points, as in 3.2.8, that define this optimal algorithm. It follows immediately that

(3.7.5) $\lambda_{1,2} = ?$ (any value is optimal),

(3.7.6) $\lambda_{1,n} = 0, \quad n>2,$

and

(3.7.7) $\lambda_{j,n} = 1-\lambda_{n-j+1,n}, \quad n>2.$

Using 3.7.1 we find that

(3.7.8) $D(\alpha,4) = -2\alpha^3+3\alpha^2+1, \quad 0\leqslant\alpha\leqslant\frac{1}{2},$

(3.7.9) $D(\alpha,5) = 3\alpha^4-8\alpha^3+6\alpha^2+1, \quad 0\leqslant\alpha<\lambda_{2,5}=0.462475,$

$= -6\alpha^4+12\alpha^3-6\alpha^2+2, \quad \lambda_{2,5}\leqslant\alpha<\frac{1}{2},$

and

(3.7.10) $D(\alpha,6) = -8\alpha^5+20\alpha^4-20\alpha^3+10\alpha^2+1, \quad 0\leqslant\alpha<\lambda_{2,6}=0.363553..,$

$= 10\alpha^5-30\alpha^4+30\alpha^3-10\alpha^2+2, \quad \lambda_{2,6}\leqslant\alpha\leqslant\frac{1}{2}.$

The following graph shows $D(\alpha,n)$ for $4\leqslant n\leqslant 7$ and $0\leqslant\alpha\leqslant\frac{1}{2}$.

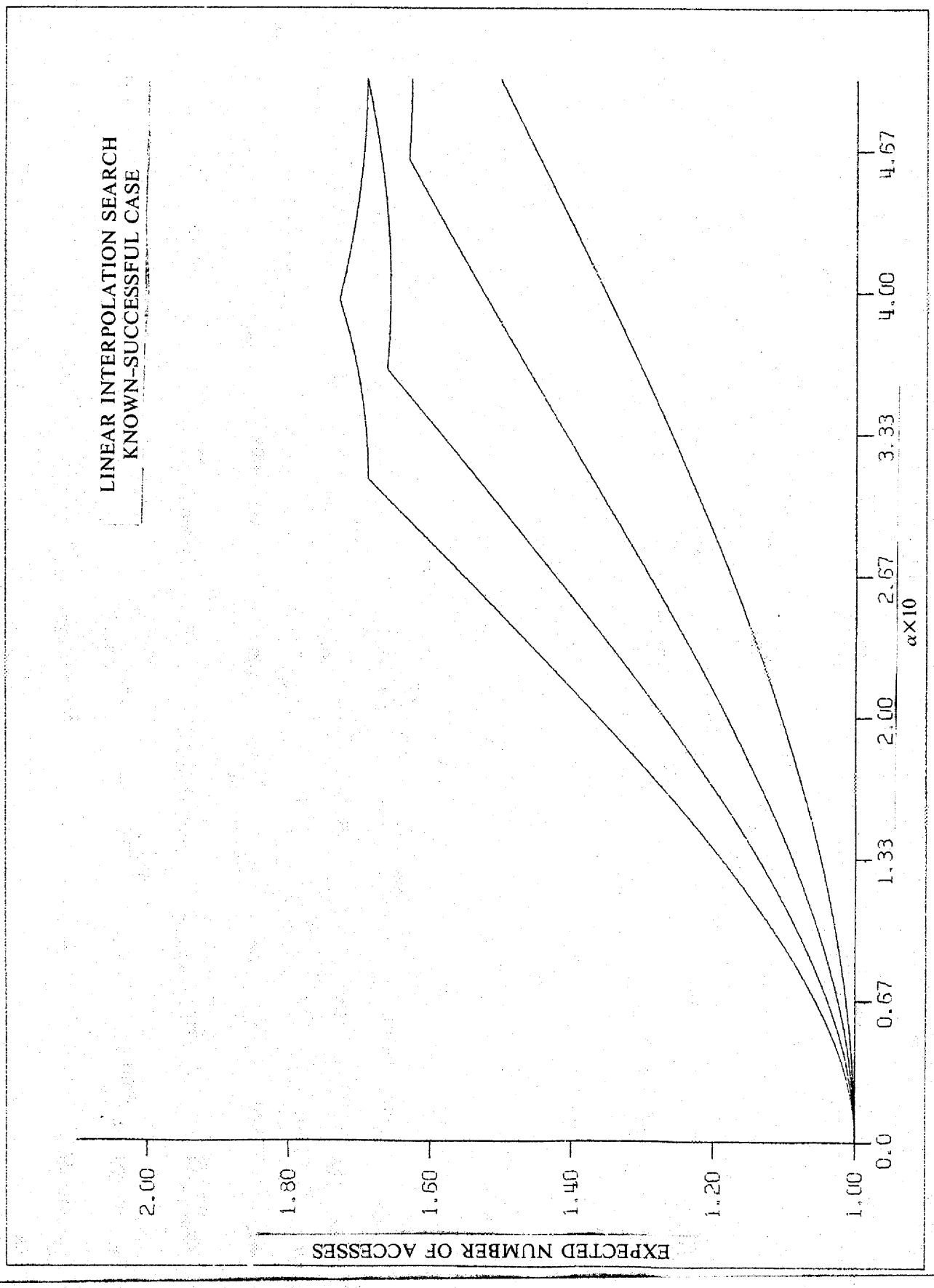
LINEAR INTERPOLATION SEARCH
KNOWN-SUCCESSFUL CASE
EXPECTED NUMBER OF ACCESSES
2.00
1.80
1.60
1.40
1.20
1.00
0.0
0.67
1.33
2.00
2.67
3.33
4.00
4.67
α×10

Figure 3.7.I

It is surprising to find that we need, for example, only 1.43 access on the average to locate one element in a table of 7 entries!

The following table compares the pure interpolation with this optimal algorithm and with the average uncertainty of the search (3.3.1).

n	$\int_0^1 B(\alpha,n)\,d\alpha$	$\int_0^1 D(\alpha,n)\,d\alpha$	$\int_0^1 H(\alpha,n)\,d\alpha$
2	1.25	1.	0.7213475
3	1.3950617	1.	1.1093617
4	1.5039062	1.1875	1.3715613
5	1.5934770	1.285078	1.5683976
6	1.6685866	1.371263	1.7254522
7	1.7319178	1.434874	1.8558517

3.8 - Variations of the Algorithm.

Binary Interpolation Search.

The idea behind this algorithm is to bisect the file into equally probable remainders as nearly as possible. That is, to arrange that

(3.8.1) $\Pr\{X_j<\alpha\} \sim \Pr\{X_j>\alpha\}$.

This goal is achieved if both probabilities are less than or equal to ½. This sole condition defines the function $\phi(\alpha,n)$. The first few break points $\lambda_{j,n}$, as defined by 3.2.8, are given by

(3.8.2) $I_{\lambda_{j,n}}(j,n-j) = ½$,

(3.8.3) $\lambda_{j,n} = \lambda_{n-j+1,n}$,

(3.8.4) $\lambda_{1,n} = 1-2^{-[1/(n-1)]}$,

(3.8.5) $\lambda_{n,2n} = ½$,

(3.8.6) $½ = [1+(n-1)\lambda_{2,n}](1-\lambda_{2,n})^{n-1}$,

$\lambda_{2,5} = 0.38572..$,

$\lambda_{2,6} = 0.31381..$,

$\lambda_{3,7} = 0.421402..$,

etc.

(3.8.7) $\lambda_{j,n} = j/n + (2j-n)/(3n^2) + O(n^{-2})$.

The latter result is derived from 2.4.14.

Notice that $\lambda_{I,n}$ coincides with the optimal algorithm, and the other values are much closer to the optimal than the linear interpolation. However, as mentioned before, these differences are only of theoretical interest, and in this case the computation of the exact $\lambda_{j,n}$ represents a very expensive task compared with the whole search process!

Interval Reducing Method.

The rationale behind this algorithm is to minimize the resulting interval; i.e., to bracket as soon as possible the searched value. The interpolation formula, $\phi(\alpha,n) = j$ is defined by the j that minimizes the expected resulting length.

$$(3.8.8)\quad E[\text{resulting length}] = (j-1)\Pr\{X_j>\alpha\} + 0\times\Pr\{X_j=\alpha\} + (n-j)\Pr\{X_j<\alpha\}$$

$$= (j-1)I_\alpha(j-1,n-j+1) + (n-j)[1-I_\alpha(j,n-j)].$$

There does not seem to exist a simple expression for $\phi(\alpha,n)$ in this case, except for the implicit minimum function.

Simulation results in section 3.9 show that this algorithm is inferior to pure interpolation.

There is an intuitive argument to explain this behaviour. When we search for elements near either end of the table, this algorithm probes in positions closer to the centre than the normal algorithm. This is caused by the imbalance of the possible resulting lengths, i.e., the algorithm really tries to minimize the interval. This usually prevents this algorithm from being successful in the first steps. The argument is that although the interval may be big, the probability of finding the element may still be significant.

One Interpolation then Sequential.

The interpolation algorithm described by Price [1971] performs only a first interpolation, and then, in case the element is not in the position searched, searches sequentially towards the appropriate end of the table. This variation of the algorithm does not follow the general algorithm given in 3.2. Consequently we will give the corresponding algorithm in pseudo-language.

```
j := φ(α,n);
select (k(j))
        case α: return(j);
        case <α: for j:=j+1 until n do if k(j)≥α then break end for;
        case >α: for j:=j-1 downto 1 do if k(j)≤α then break end for;
        end select;
if k(j)=α then return(j) else return(FAIL);
```

In the analysis of this algorithm we will first consider the successful search. Consider the search of the j^{th} key, $X_j=\alpha$. The expected number of accesses is given by

$$(3.8.9)\quad E[\text{accesses to find } X_j] = \int_0^1 [1+|\phi(\alpha,n)-j|]I_\alpha'(j,n-j+1)\,d\alpha$$
$$= 1+\int_0^1 |\phi(\alpha,n)-j|\,I_\alpha'(j,n-j+1)\,d\alpha.$$

The average number of accesses to find any X_j is

$$(3.8.10)\quad E[\text{accesses}] = 1 + 1/n\sum_{j=1}^{n}\int_0^1 |\phi(\alpha,n)-j|\,I_\alpha'(j,n-j+1)\,d\alpha.$$

To continue the analysis we need to define $\phi(\alpha,n)$ for this search. In this case we will use the linear interpolation algorithm defined in 3.2.7 $\phi(\alpha,n) = \lceil n\alpha \rceil$, to obtain

$$(3.8.11)\quad E[\text{accesses}] = 1 + 1/n\sum_{j=1}^{n}[I_{1/n}(j,n-j+1)+I_{2/n}(j,n-j+1)+\ldots$$
$$+I_{(j-1)/n}(j,n-j+1)+1-I_{j/n}(j,n-j+1)+1-\ldots+1-I_{(n-1)/n}(j,n-j+1)].$$

Using the symmetry relation $I_x(a,b) = 1-I_{1-x}(b,a)$ we transform 3.8.11 into

$$(3.8.12)\quad E[\text{accesses}] = 1 + 2/n\sum_{j=1}^{n}\sum_{k=1}^{j-1} I_{k/n}(j,n-j+1).$$

Reversing the order of summation and using the formula 2.4.20 we derive

$$(3.8.13)\quad E[\text{accesses}] = 1 + 2/n\sum_{k=1}^{n-1}[kI_{k/n}(k,n-k)-kI_{k/n}(k+1,n-k)]$$
$$= 1 + 2/n\sum_{k=1}^{n-1}\Gamma(n)/[\Gamma(k)\Gamma(n-k)][k/n]^k[(n-k)/n]^{n-k}.$$

So far the result is exact, but not very useful, so we will look for an asymptotic expansion in terms of n.

Using Stirling's approximation we decompose each summand into

$$(3.8.14)\quad [2/(n^3\pi)]^{1/2}\times[1+1/12n+1/(288n^2)+O(n^{-3})]\times$$

$$[1-n/[12k(n-k)]+n^2/(288[k(n-k)]^2)+O(n/[k(n-k)]^2)]\times[k(n-k)]^{1/2}.$$

(Note that with this method we would not be able to obtain an order higher than $n^{-1/2}$.) Using the summation formulas A1.2.1 through A1.2.3 we finally obtain

$$(3.8.15)\quad E[\text{accesses}] = 1 + [n\pi/32]^{1/2}[1-7/12n] + O(n^{-1})$$

$$\sim 1 + [(n-1)\pi/32]^{1/2} + O(n^{-1}),$$

the latter being an equivalent asymptotic expansion but with better numerical approximation to the exact result.

The method described above does not apply to the unsuccessful case. The approach we will take is slightly different and is as follows:

$$(3.8.16)\quad E[\text{accesses for }\alpha] = 2\times Pr\{X_{j-1}<\alpha<X_j\} + 3\times Pr\{X_{j-2}<\alpha<X_{j-1}\} + \ldots +$$

$$2\times Pr\{X_j<\alpha<X_{j-1}\} + 3\times Pr\{X_{j+1}<\alpha<X_{j+2}\} + \ldots$$

$$- Pr\{-\infty<\alpha<X_1\} - Pr\{X_n<\alpha<+\infty\}.$$

$$= \sum_{k=1}^{n+1}[3/2+|j-k+1/2|]Pr\{X_{k-1}<\alpha<X_k\} - \alpha^n - (1-\alpha)^n.$$

Averaging for any α we obtain

$$(3.8.17)\quad E[\text{accesses}] = \int_0^1 (\sum_{k=1}^{n+1}[3/2+|j-k+1/2|]Pr\{X_{k-1}<\alpha<X_k\} - \alpha^n - (1-\alpha)^n)\,d\alpha.$$

After transformations similar to those used in the successful case we obtain

$$(3.8.18)\quad E[\text{accesses}] = 2/(n+1)[1+\sum_{k=1}^{n-1}\{(n+1)k/n I_{k/n}(k+1,n-k) -$$

$$(k+1)I_{k/n}(k+2,n-k)\}]$$

$$= [n\pi/32]^{1/2} + O(1)$$

$$\sim 1.5 + [n\pi/32]^{1/2}.$$

This search, as expected, behaves as $O(n^{1/2})$.

In both the successful and unsuccessful cases, the constant that multiplies the $n^{1/2}$ term is rather small; consequently, this search behaves better than binary search for n up to about 1000.

Unless we are searching in a blocked file in secondary storage or other special situation, this algorithm seems less efficient than linear interpolation.

3.9 - Simulation Results.

This section contains the results of several simulations of the different interpolation search algorithms and different source distributions. In all cases the simulation pattern is roughly the same. A file of the stated size, n, is generated with pseudo-random numbers, and then the file is sorted into ascending order. Each element of the file is then searched with the corresponding interpolation algorithm, keeping a histogram of the number of searches needed to locate the element.

For each file we record the average number of accesses, its variance, and the maximum number of accesses needed to locate any single element. For each size we repeat this experiment several times, recording statistics of the average number of accesses, variance of the number of accesses and maximum. A typical output of this phase is shown below:

```
SIZE OF FILE=  17=  15+2
STATISTICS OF AVERAGE NUMBER OF ACCESSES
NUMBER OF SAMPLE POINTS=  1200  (different files)
MEAN=  2.045833          STANDARD DEVIATION=  0.358526
SKEWNESS=  0.391013          EXCESS=  0.217053
MINIMUM=  1.066667           MAXIMUM=  3.333333
```

Uniformly we will display the results of a simulation run in a single line, always in the same order. These values displayed are

- size of the file being searched,
- the best known theoretical result, for n up to 7 it is the exact value, for $8 \leqslant n \leqslant 60$ is a numerical approximation, and for $n > 60$ an empirical formula;
- the mean average number of accesses found by simulation with a 95% central confidence interval
- the sample size, i.e. number of different files created and analyzed;
- the maximum average number of accesses for any of the files;
- the maximum number of accesses needed to locate any single element in any of the files, and
- an estimate of the variance of the number of accesses needed to locate any element.

The calculation of most of these measures is straightforward. The confidence limits are set as

$$(3.9.1) \quad \pm \text{var}\{\text{average number of accesses}\}/[\text{samplesize}-1]^{1/2} \times 1.96,$$

and the variance of the number of accesses is [Kalbfleisch 75]

(3.9.2) = var{average number of accesses} + E[variance of number accesses].

Pure Interpolation Search Simulation (successful case).

The following table contains the results for various file sizes (n) of the interpolation search algorithm described in section 3.2. The interpolation formula is $\phi(\alpha,n) = \lfloor n\alpha \rfloor + 1$.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
7	1.731918	1.7406±0.0203	1200	3.000	5	0.5615
10	1.876452	1.8831±0.0208	1200	3.900	6	0.6800
15	2.038610	2.0458±0.0203	1200	3.333	7	0.7790
20	2.151100	2.1534±0.0204	1200	3.600	6	0.8525
25	2.236359	2.2301±0.0200	1200	3.560	8	0.9171
30	2.304544	2.3088±0.0193	1200	3.567	8	0.9420
35	2.361080	2.3530±0.0201	1200	3.800	8	0.9855
40	2.409194	2.4171±0.0194	1200	3.700	8	1.0081
45	2.450956	2.4476±0.0194	1200	3.711	8	1.0278
50	2.487765	2.4863±0.0197	1200	3.920	10	1.0564
55	2.520614	2.5249±0.0190	1200	3.982	9	1.0747
60	2.550229	2.5524±0.0180	1200	3.533	9	1.0694
70	2.629898	2.6099±0.0319	400	3.943	8	1.1256
80	2.672434	2.6376±0.0304	400	3.713	9	1.1189
90	2.709105	2.6802±0.0314	400	3.600	9	1.1610
100	2.741251	2.6877±0.0205	800	3.690	9	1.1268
120	2.795458	2.7640±0.0411	200	3.675	9	1.1559
140	2.839934	2.8056±0.0382	200	3.629	8	1.1926
160	2.877495	2.8476±0.0367	200	3.863	8	1.1662
180	2.909907	2.9017±0.0392	200	3.861	9	1.2115
200	2.938349	2.9289±0.0404	200	3.700	9	1.2049
250	2.996930	2.9744±0.0353	200	3.788	10	1.2342
300	3.043200	3.0275±0.0340	200	3.677	9	1.2244
350	3.081259	3.0672±0.0327	200	3.774	9	1.2111
400	3.113473	3.1211±0.0350	200	3.727	10	1.2639
450	3.141329	3.1312±0.0323	200	3.751	9	1.2298
500	3.165818	3.1613±0.0312	250	4.022	10	1.2368

600	3.207270	3.2332±0.0452	100	3.793	9	1.2634
700	3.241441	3.2603±0.0457	100	4.187	10	1.2908
800	3.270418	3.2929±0.0419	100	3.759	10	1.2696
900	3.295515	3.3271±0.0437	100	3.900	10	1.2842
1000	3.317609	3.3270±0.0297	200	3.906	10	1.2824
2000	3.455222	3.5184±0.0575	50	3.952	10	1.3162
3000	3.530094	3.5940±0.0488	50	3.951	11	1.3340
4000	3.580973	3.5956±0.0448	50	3.944	10	1.3018
5000	3.619246	3.6262±0.0356	100	4.005	11	1.2945
6000	3.649784	3.6644±0.0516	20	3.857	10	1.3031
7000	3.675111	3.7021±0.0727	20	4.026	10	1.3163
8000	3.696698	3.7342±0.0788	20	3.989	10	1.3236
9000	3.715475	3.8374±0.0847	20	4.179	13	1.3598
10000	3.732068	3.7688±0.0431	50	4.214	11	1.3272

Table 3.9.I

Several conclusions can be drawn from these results. The most significant are:

– For all exact or numerically approximated values, the mean average number of accesses falls within the confidence limits of the predicted theoretical values. For files of size greater than 60 we use the empirical formula. Comparing its value at 60 with the numerical approximation, we may conclude that for $n>30$ $lg(lg(n))$ may be a better approximation. This explains the fact that the simulation results, for n in the range 70–200, are below the empirical values.

– The maximum average number of accesses follows its prediction closely based on a normal distribution of the mean and the size of the sample (number of files simulated).

– The estimate of the variance of the average number of accesses is rather small. Consequently the maximum average number of accesses for a reasonable size sample is, in practice, close to the mean average number of accesses.

– The standard deviation of the average number of accesses noticeably decreases, which is natural, with an increment in the size of the file. Therefore we conclude that the maximum average number of accesses for a finite sample is $lg(lg(n))+O(1)$ (e.g. for 99% of the files of size 10000 the average number of accesses is less than 4.075 and for 99.9% of the files the average number of accesses is less than 4.181; the average is 3.732). This is a very valuable practical consideration.

– The maximum average number of accesses needed to find any single element remains reasonably small, even for the large samples. Surprisingly, for files larger than 1000, this maximum is lower than the one required by binary search. This is also a useful practical consideration.

Pure Interpolation Search Simulation (unsuccessful case).

The following table contains the simulation results for the unsuccessful search case. The simulation was done by searching, for each file of size n, n new random variables different from those in the file.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
7	2.153784	2.14787±0.0193	1200	3.556	6	1.0094
10	2.346975	2.35917±0.0310	500	3.917	6	1.0581
15	2.553902	2.54588±0.0315	500	3.824	6	1.1150
20	2.692230	2.69591±0.0288	500	3.909	7	1.1150
25	2.794515	2.79763±0.0297	500	3.926	7	1.1338
30	2.874840	2.85850±0.0271	500	3.969	7	1.1186
35	2.940510	2.94897±0.0275	500	3.973	8	1.1314
40	2.995766	2.97771±0.0271	500	4.024	7	1.1415
45	3.043275	3.03745±0.0275	500	3.936	8	1.1557
50	3.084818	3.07450±0.0261	500	4.019	8	1.1349
55	3.112164	3.13207±0.0261	500	4.246	9	1.1484
60	3.154629	3.15116±0.0264	500	4.581	8	1.1573
100	3.312021	3.33098±0.0514	100	4.010	9	1.1516
200	3.514301	3.56376±0.0775	50	4.129	9	1.1934
300	3.620685	3.73185±0.0589	50	4.460	9	1.1645
400	3.691675	3.76940±0.0681	50	4.328	10	1.2034
500	3.744430	3.79964±0.0633	50	4.376	9	1.1890
1000	3.896983	4.03663±0.0581	50	4.559	9	1.1564
2000	4.034937	4.05415±0.0576	10	4.230	9	1.0416
3000	4.109914	4.19773±0.1152	10	4.541	10	1.1380
4000	4.160843	4.25390±0.1006	10	4.565	10	1.1368
5000	4.199144	4.33465±0.1246	10	4.699	10	1.1687
10000	4.312021	4.48869±0.0603	20	4.757	10	1.1096

Table 3.9.II

Reducing Algorithm.

This algorithm is described in detail in section 3.8. The computation of the exact interpolation formula requires a lot more work than in the previous cases. This is why the sample sizes for this algorithm are considerably smaller.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	2.0210±0.0496	100	2.600	4	0.5672
20	2.177459	2.4275±0.0436	100	2.950	5	0.7642
30	2.334681	2.7043±0.0384	100	3.067	5	0.8366
40	2.439959	2.8828±0.0350	100	3.250	5	0.9028
50	2.518011	3.0230±0.0361	100	3.460	5	0.9236
60	2.579489	3.0794±0.0543	30	3.367	5	0.9817
70	2.629898	3.2014±0.0479	30	3.457	5	0.9615
80	2.672434	3.3121±0.0725	30	3.675	6	1.0386
90	2.709105	3.3559±0.0560	30	3.644	6	1.0286
100	2.741251	3.4197±0.0547	30	3.730	6	1.0930
200	2.938349	3.8198±0.0516	30	4.090	7	1.0981

Table 3.9.III

These results show, beyond any doubt, that the reducing algorithm behaves worse than pure interpolation.

Special Distributions.

In the following examples we will test the interpolation search algorithm when we search a file that was not originated by a uniform distribution. Even though we know how to adjust the algorithm to the proper distribution [2.3.2 and 3.1.2], these simulations provide some insight into the behaviour of the algorithm in these non typical conditions. We will find that for some distributions we have a significant departure from the $lg(lg(n))$ behaviour.

Discrete Rectangular Distribution.

This is a discrete distribution of keys, i.e. $0 \leqslant X_1 \leqslant \ldots \leqslant X_n \leqslant m-1$, where X_i are integers and m is usually bigger than n.

The simulation was done setting $m=3n$ and allowing duplicate keys to occur. For large n, assuming a Poisson distribution of the number of keys with a given value, the number of duplicates is

(3.9.3) $e^{-\beta}+\beta-1 \sim 0.0498..$ where $\beta = n/m = 1/3$.

The discretization of keys should improve the number of accesses; however it is uncertain how the repeated keys may affect the algorithm, since when we search for a duplicated key we will stop as soon as we find any, but on the other hand, while searching, probing in repeated keys provides no information to the search.

The following table shows the results for different file sizes of the various runs.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	1.78144±0.0206	1200	4.333	6	1.0385
100	2.741251	2.58589±0.0292	400	3.430	9	1.3074
500	3.165818	3.02289±0.0428	100	3.704	10	1.3133
1000	3.317609	3.18181±0.0582	50	3.630	8	1.3065
10000	3.732068	3.58564±0.0666	20	3.815	11	1.2871

Table 3.9.IV

Compound Rectangular Distribution.

In this case the table was obtained from random variables distributed as

(3.9.4) $f(x) = 1$, iff $0 \leqslant x \leqslant ½$,

$f(x) = 1/3$, iff $½ < x \leqslant 2$,

f(x) = 0, otherwise.

This is an example where the algorithm behaves poorly. Notice that roughly 50% of the keys fall in the first 25% of the interval e.g. a key of value 0.5 will likely be located at the middle of the table, although its search starts at 1/4 of the file.

The following table shows the simulation results for this distribution.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	2.28000±0.0318	1200	4.900	7	1.4629
100	2.741251	4.49040±0.0602	400	6.230	16	2.5416
500	3.165818	6.24966±0.0851	100	7.786	21	3.3790
1000	3.317609	6.83294±0.0829	50	7.548	20	3.5521
10000	3.732068	9.31388±0.0714	20	9.783	27	5.1731

Table 3.9.V

Rectangular Bimodal Distribution.

For this simulation the table is constructed with random variables distributed as

(3.9.5) f(x) = 1, iff $0 \leqslant x \leqslant 1/4$,

f(x) = 3, iff $1/2 \leqslant x \leqslant 3/4$,

f(x) = 0, otherwise.

This case is probably the most difficult for the algorithm, since in addition to the problem of different densities we have a jump in the possible values of the keys. The results follow.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	2.65833±0.0354	1200	4.600	8	1.8488
100	2.741251	6.42387±0.0779	400	8.460	20	4.0328

500	3.165818	9.90544±0.1258	100	12.042	29	5.6307
1000	3.317609	11.44168±0.1412	50	12.528	33	6.2336
10000	3.732068	16.78742±0.1214	20	17.350	47	8.7970

Table 3.9.VI

Truncated Exponential Distribution.

In this case the table is generated from variables distributed

(3.9.6) $f(x) = e^{-x}$, iff $0 \leqslant x \leqslant T$,

$f(x) = 0$, otherwise.

The bound T is selected only for practical reasons, and is chosen to be

(3.9.7) $T = -ln(ln(2)/n)/2$

which is half the value of the median of the maximum of n such random variables. Only $[ln(2)n]^{1/2}$ variables, on the average, are greater than T, while creating a file of size n.

The following table displays the results for this type of file.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	2.10650±0.0278	1200	4.200	7	1.2835
100	2.741251	5.41450±0.0616	400	7.530	16	2.4662
500	3.165818	9.91264±0.1153	100	12.856	23	3.4499
1000	3.317609	12.47884±0.1345	50	13.770	25	3.8643

Table 3.9.VII

Triangular Distribution.

For this simulation the table is constructed with random variables distributed as

(3.9.8) $f(x) = x$, iff $0 \leqslant x \leqslant 1$,

$f(x) = 2-x$, iff $1 \leqslant x \leqslant 2$,

$f(x) = 0$, otherwise.

This variable is obtained as $Y = X+Z$, where X and Z are random variables distributed U(0,1). Simulation results follow.

table size	pure interpolation	average number of accesses	sample size	maxim. average	maxim. probes	variance
10	1.887697	2.12708±0.0220	1200	3.700	6	1.1917
100	2.741251	4.04153±0.0379	400	5.370	11	1.8712
500	3.165818	6.05710±0.0612	100	6.764	15	2.1453
1000	3.317609	7.09846±0.0681	50	7.947	16	2.2105
10000	3.732068	10.37196±0.0683	20	10.756	20	2.2127

Table 3.9.VIII

In all the above simulations a good random number generator was used, based on a congruence method with modulus 2^{35} and multiplicative constant 5^{15} [Coveyou 67]. It is not known how "dependent" these pseudo-random variables are, when generated by the thousands and sorted. More precise details of the values analyzed by simulation (i.e. increasing the sample size) may be doomed with deviations due to the dependencies of these pseudo-random variables.

3.10 - Conclusions.

In this chapter we analyzed in detail the interpolation search algorithm and its variations to search in a full ordered table.

The main conclusions of the algorithm are related to its asymptotic behaviour. The expected number of accesses needed to locate an element in the table is $lg(lg(n))+O(1)$, or the good empirical approximation $lg(lg(n+3))$. The variance of the number of accesses is $O(1)$. Numerical study of the distribution of number of accesses and simulation strongly suggest that its value is close to 1.10. From this we conclude, for example, that 99.9% of the accesses to elements in a table require no more than the average plus 3 accesses.

The function $lg(lg(n))$ is an extremely flat function. For all applications in the near future, the average can be considered bounded by 5 (direct accessible files of $2^{32} = 4.3\times10^9$ elements are still too large for the actual technology). For all practical applications of tables in core, the average may be considered bounded by 4 (for tables of up to 65536 entries).

The unsuccessful search has almost exactly the same properties as the successful case. The average number of accesses is empirically approximated by $lg(lg(n))+0.58$ and the variance is near 1.10. The search time is not greatly affected by the presence or absence of the element in the table in contrast, to some methods, such as hashing, where the presence or absence of the element in the table may change its order.

Comparing the algorithm with hash searching we have the advantages of a full table (that takes only $O(n\times ln(n))$ to construct), an ordered table, a smaller variance on the number of accesses against a slightly higher number of accesses. However for certain types of real keys, it may be very difficult or impossible to find its distribution; in this case the interpolation search algorithm may not be a viable alternative.

It should be noted that the algorithm is rather sensitive to an incorrectly predicted distribution. Simulation results involving the linear algorithm and random keys from non uniform distributions, suggest that the search may even lose the $lg(lg(n))$ behaviour.

On the other hand the discretization of uniformly distributed keys slightly reduces the expected average number of accesses.

4. - Interpolation–Hash Searching.

4.1 - Definition and Motivation.

The idea of interpolation hash search can be visualized in two different ways. From the point of view of the interpolation search, we extend the set of keys by adding empty elements in order to improve the average number of accesses. From the point of view of hash code searching techniques it is a special definition of the hashing and secondary probe functions and also a particular creation algorithm that produces an ordered table.

This technique will allow us to gain the speed of hashing while preserving an order relation in the file or table. This becomes quite important in non–scientific computing, where most of the data is stored and processed in an ordered sequential way.

A similar algorithm, viewed from the point of hash code searching, is discussed by Amble and Knuth [1973]. Their first algorithm is mainly intended to reduce the number of accesses in the unsuccessful case. Their bidirectional algorithm has a closer resemblance to interpolation search, and except for the use of a key mapping function, instead of the transformations suggested in section 2.3, their method is an exact extension of the one–interpolation–then–sequential search to non full tables.

4.2 - Description of the Search Algorithm.

Before going into the detailed description of the algorithm we will analyze its differences with respect to pure interpolation search and hash coding.

Comparing it to interpolation search, the only difference lies in the treatment of an empty position. For any general configuration, we cannot guarantee that all generated probes are to fall in occupied positions in the file. If we probe an empty position, this access does not contribute much information, and we have to select a new probe position with an ad hoc function.

Comparing it to hash code searching, the main differences are the one mentioned above, and the fact that at each step the increment is a new function of the key and the limits. That is, with the key alone, we cannot predict more than one probe position.

We will define the *property H* of an interpolation hash table as the condition that the sequence of probe positions for any key belonging to the file does not go through an empty element.

It follows immediately that the algorithm to search a table with property H is the same as the one described in 3.2 with the addition of a failure return in the case that a probe location is empty.

However we cannot guarantee that property H may be obtained for all files. We will demonstrate, with a counter example, that there are files for which property H will never hold. This is not a trivial example, and only can be found for files 9 or larger in size.

Let 0.35, 0.53, 0.56, 0.95, 0.96, 0.97, and 0.98 be the keys of our file to be placed in a table of size 9, k(1),...,k(9), with the limits of the table defined as k(0)=0 and k(10)=1.

By calculating the first probe positions of each key, we find that positions 4,5,6 and 9 should not be empty. Since k(9) is not empty, the only choice is k(9)=0.98. Now the second probe of key 0.97 is k(8), consequently k(8) should not be empty and the only choice is k(8)=0.97. Similarly k(7)=0.96, k(6)=0.95, k(5)=0.56 and k(4)=0.53. The key 0.35 will probe position 2 in its second attempt, consequently k(2) = 0.35. All the previous allocations were forced, so there is no possible variation. We verify that 0.56 will probe position 6, and then position 3, which is empty.

k(0)	k(1)	k(2)	k(3)	k(4)	k(5)	k(6)	k(7)	k(8)	k(9)	k(10)
0		0.35		0.53	0.56	0.95	0.96	0.97	0.98	1

It should be noticed that the configuration is rather contrived. However this result prevents us from looking for algorithms that will create a file with property H. Instead we will look for good heuristic algorithms and remedies to this problem.

The general search algorithm should contemplate the situation of probing an empty position. For this we need to select a non empty position, if any, to continue the search. For efficiency reasons we want to select this position as closely as possible to the original interpolation probe position. This is achieved by the following portion of code:

```
jtemp := j;
for i := 1 until high-low do
        jtemp := i-jtemp;
        jabs := abs(jtemp);
        if low<jabs and high>jabs then
                if k(jabs)≠null then return(jabs) end if;
                end if;
        end for;
return(FAIL);
```

Certainly we do not want this kind of algorithm, since besides its length and complexity, it may be very inefficient.

From now on, we will study only *corrected files* that satisfy property H. A *corrected file* is an instance of an interpolation hash file that has extra keys, specially marked, added so that property H holds. These extra keys will be added only when needed, and the only requirement is that they preserve the order relation between keys. Our previous example will be corrected with any key whose value is in the range (0.35, 0.53); for example:

k(0)	k(1)	k(2)	k(3)	k(4)	k(5)	k(6)	k(7)	k(8)	k(9)	k(10)
0		0.35	0.40*	0.53	0.56	0.95	0.96	0.97	0.98	1

Note that these corrective keys do not form part of the file, so it is irrelevant whether their search probe sequence goes through an empty record. For corrected interpolation hash files we can use the normal search algorithm described in 3.2 with the addition of

```
case null: return(FAIL);
```

in the second select statement.

A simple and efficient way of assigning values to the corrective keys is to compute the interpolated value between the two nearest non null records. In our previous example this will yield k(3)=0.44, or in general if we want to compute k(j) where k(a) and k(b) (a<j<b) are the nearest non empty elements, then

(4.2.1) $k(j) = k(a) + (j-a)[k(b)-k(a)]/(b-a).$

4.3 - Optimal Interpolation Hash Configuration.

We will consider optimality based on the average number of accesses needed to locate any element of the file. The optimal configuration will be an assignment of the n records to the m locations such that the average number of accesses is minimal.

We shall restrict ourselves to study the optimality of configurations that have the property H. This may appear to be a significant restriction but the assignment of corrective keys increases considerably the "degrees of freedom" to search for the optimum.

There are very few generalities we can say about optimal configurations.

Lemma I

The optimal configuration may not be unique, e.g. let the file 0.49, 0.51 be assigned to k(1), k(2), k(3)

k(0)	k(1)	k(2)	k(3)	k(4)
0		0.49	0.51	1

or

0	0.49	0.51		1,

are both optimal, requiring an optimal average number of accesses of 1.5.

Lemma II

For the same file allocated in tables of different sizes, we cannot conclude that if we increase the size of the table, the average number of accesses will decrease or remain equal. As a counter example, taking the same file as before, allocating it in a two element table we have an average of 1, i.e.

k(0)	k(1)	k(2)	k(3)
0	0.49	0.51	1,

instead of 1.5 as for a size 3.

Lemma III, Contiguity.

Let $K_1<K_2$ be two adjacent keys (there is no key, K, in the file such that $K_1<K<K_2$) such that both probe initially to the same location. Then any allocation that preserves property H,

including the optimal, will place K_1 and K_2 in contiguous locations in the table, if they are allocated in their first or second probe locations.

To prove this lemma we observe that if K_1 and K_2 are adjacent keys, the only possible alternative for them not to be contiguous in the table is to be separated by an empty location. This may not happen since

(4.3.1) $\lceil K_1 m \rceil = a; \quad \lceil K_2 m \rceil = a;$

and therefore k(a) is not empty. Assume that $K_2 \leqslant k(a)$, (note that the only possible remaining configuration is the symmetric $K_1 \geqslant k(a)$), then

(4.3.2) $0 < K_1 < K_2 \leqslant k(a),$

(4.3.3) $a-1 < K_1 m \leqslant a,$ and $a-1 < K_2 m \leqslant a.$

The next probe position will be

(4.3.4) $\lceil K_1(a-1)/k(a) \rceil$ for K_1 and

(4.3.5) $\lceil K_2(a-1)/k(a) \rceil$ for K_2; now

(4.3.6) $(K_2-K_1)(a-1)/k(a) < (K_2-K_1)m < 1,$

and the new probe positions will be equal or contiguous.

If there are more than two steps involved we cannot guarantee this contiguity property. The following, a rather contrived counterexample, shows how the keys 0.626 and 0.687, that interpolate to the same initial position, end up placed in non contiguous locations for the optimal configuration.

k(0)	k(1)	k(2)	k(3)	k(4)	k(5)	k(6)	k(7)	k(8)	k(9)	k(10)
0				0.626		0.687	0.95	0.96	0.961	0.962

k(11)	k(12)	k(13)	k(14)	k(15)	k(16)	k(17)
0.97	0.971	0.972	0.973	0.974	0.975	1

(4.3.7) $m = 16; \quad \lceil 0.626 \times m \rceil = 11; \quad \lceil 0.687 \times m \rceil = 11;$

(4.3.8) $\lceil 0.626/k(11)\times 10\rceil = 7; \quad \lceil 0.687/k(11)\times 10\rceil = 8;$

(4.3.9) $\lceil 0.626/k(7)\times 6\rceil = 4,$ and $\lceil 0.687/k(8)\times 7\rceil = 6.$

This type of configuration is extremely unusual, actually only files of size 16 or larger may show this behaviour.

Lemma IV, Partitioning.

The intention of this lemma is to provide the tools to subdivide the optimal configuration problem. Let k(1), k(2), ..., k(m) denote the entries of the table. We will define two vectors $k_n(i)$ and $k^x(i)$ through the following steps

A. Initially $k_n(i)$ contains the minimum value of all the keys that interpolate in their first probe to location i. If no key interpolates to i then $k_n(i)$ is not defined. The vector $k^x(i)$ is defined in the same way, except that it contains the maximum.

B. For each key in the file we follow two possible paths using the interpolation search algorithm with the vectors, taken as files, k_n and k^x. For each probe position we adjust k^x and k_n, if necessary with the value of the key being searched.

C. Finally we repeat step B until no more changes occur in k_n or k^x.

To better illustrate this algorithm we will follow a complete example. Let k(1), k(2), ..., k(10) be a table which we want to fill with the key values 0.44, 0.49, 0.60, 0.63, 0.65, 0.71, 0.75 and 0.99. The first step defines the vectors to be

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
k^x					0.49		0.65	0.75		0.99
k_n					0.44		0.60	0.71		0.99

The blank position indicate undefined values. After one B step we obtain

k^x	0.44	0.49	0.63	0.71	0.75	0.75	0.99	
k_n	0.44	0.44	0.49	0.60	0.65	0.75	0.99	

After the second step we obtain

k^x	0.44	0.60	0.65	0.71	0.75	0.75	0.99
k_n	0.44	0.44	0.49	0.60	0.63	0.65	0.71

And finally

k^x	0.44	0.49	0.60	0.63	0.65	0.71	0.75	0.75	0.99
k_n	0.44	0.44	0.44	0.44	0.49	0.60	0.63	0.65	0.71

Note that different order of keys in step B may change the number of iterations. From the construction of k^x and k_n it is concluded that $k^x(i)$ and $k_n(i)$ hold the limit values of any key that may be located in position i. Furthermore if $k^x(i)$ is null, and consequently $k_n(i)$, k(i) will be an empty entry in the table, and the problem can be subdivided into two smaller ones. If $k^x(i)=k_n(i)$ the position i will contain $k^x(i)$ or be empty. If $k^x(i)=k_n(i)<k^x(i+1)=k_n(i+1)$ the file can also be subdivided in two, namely 1,2,...,i and i+1,...,m.

There is no guarantee that we can partition a given file. Unfortunately simulation shows that only sparse files can be successfully, and usefully, partitioned.

Lemma V, Border Conditions.

If a given key probes initially to position 1 (or m) then this location should not be empty and the only choice is to locate the smallest (largest) key in it. The allocation of k(1) (or k(m)) is solved. This procedure may be repeatedly applied (as can be verified in the counterexamples for Lemma 3).

Finally the optimality problem, in view of the above lemmas, can only be reduced to compute the average number of accesses over all possible configurations. If after all possible reductions we have n elements to locate in m entries and the n keys hit h different locations in their first probe ($n \geqslant h$), we must analyze

$$(4.3.10) \quad C(m,n,h) = \binom{m-h}{m-n}$$

possible configurations. This behaviour is exponential for a fixed occupation factor, and consequently unacceptable for practical algorithms.

4.4 - Construction Algorithm.

In this section we will analyze the problem of constructing an interpolation hash table starting with all the elements available. The construction will not be optimal, but we will use good heuristics. The central idea in the construction will be to minimize the average distance for all elements, from their initial interpolation location to their final position. This does not guarantee by itself optimality, but preserves the contiguity property, allocates correctly all sparse situations and seems intuitively good.

The allocation problem arises when we have, for a given interval in the table, more keys that interpolate initially to those positions than entries, e.g.

...	k(i)	k(i+1)	k(i+2)	k(i+3)	...
	3 keys	4 keys		8 keys	

In this example it becomes apparent that the first three keys will be "pushed" to the left because of the "pressure" of the others. We will also expect the 8 keys to be spread more to the right of k(i+3) than to the left for the same reason.

To solve this problem we will borrow the concept of torque from physics and apply it to keys and their initial interpolation position. To obtain equilibrium we want to have a null torque. More precisely we define h(i) to be the number of keys that interpolate initially to k(i). We will place these h(i) keys contiguous, and their torque contribution will be

(4.4.1) $h(i) \times a$

where a is the distance between the centre of "gravity" of the h(i) keys and the centre of the location i. For any sequence of keys that will be placed contiguously in the table, the total torque is

(4.4.2) $[a + \frac{1}{2}h(j)]\sum h(i) + \sum(i-1)h(i) - \frac{1}{2}[\sum h(i)]^2,$

where now a represents the leftwards displacement of the centre of the h(j) keys with respect to the centre of position j, and all sums are for i=j,j+1,....,k. Since we want a total null torque, we derive

(4.4.3) $a = \frac{1}{2}\sum h(i) - \frac{1}{2}h(j) - \sum(i-1)h(i) / \sum h(i).$

If b is the left edge of the first of the h(j) keys, then

(4.4.4) $b = a + h(j)/2 - ½, \quad$ for $h(j)>0$,

and

(4.4.5) $b = ½\sum h(i) - ½ - \sum(i-1)h(i) / \sum h(i)$.

The value b is normally a rational, we must interpret it as follows: if $q+½>b>q-½$ then the equilibrium (or the nearest) position is obtained by shifting q positions; if $q+½ = b$ then $q+1$ and q are both equilibrium displacements.

In the above example if we consider k(i) and k(i+1) only, we find

(4.4.6) $b = 7/2 - ½ - 4/7 = 17/7$,

and the first of the keys that interpolate to k(i) will be displaced to k(i–2). If we consider all the keys in the example

(4.4.7) $b = 15/2 - ½ - 28/15 = 77/15 \sim 5$,

and the first of the keys will be shifted now to k(i–5). It can be easily verified, assuming enough empty positions at the right and the left, that these are optimal placements.

From these considerations we derive the following creation algorithm

```
set j=0,  nextpos=1, and nextkey=1.
        compute h(i) for all i.
        clear out.

while (remaining keys < remaining locations and j < m) do
        sequentially search for next h(j)≠0.
        if ( h(nextpos) > 0  or  nextpos ⩾ j)
                assign h(j) keys from in(nextkey) to out(nextpos)
        else
                compute maximum b for j,j+1,...,m
                compute starting position = min ( max ( j–b, nextpos), m–remaining keys)
                assign h(j) keys from in(nextkey) to out(starting position).
        end while
assign remaining keys, if any, sequentially.
```

The whole computation is $O(m^2)$ due to the computation of all the b's. It is intuitive that b attains its maximum value in the first steps. A rather safe procedure is to compute b_j, b_{j+1} until $b^*_{j+i} < \max(b_j, b_{j+1}, \ldots, b_{j+i})$, where b^* is the corresponding b that assumes that h(j+i) is "large" (e.g. 10). This means that we take the decision of not shifting to the left any more when we analyze up to certain position and even if we have 10 hits in the next single location this will not cause further shifting of the current element. This modification makes the algorithm work in time $O(m)$ on the average, for $\beta = n/m < 1-\epsilon$. Using this short cut, the algorithm in pseudo-language is as follows

```
/* generates a good interpolation hash configuration from the
        in(1:n) sorted keys, placing them in out(1:m) */
for i := 1 until m do h(i) := 0; out(i) := null end for;
for i := 1 until n do j := φ(in(i),m); h(j) := h(j) + 1 end for;
nextkey := 1;  nextpos := 1;  j := 0;
while ( 0 < n-nextkey < m-nextpos ) do
        repeat j := j + 1 until h(j) > 1;
        if ( h(nextpos) = 0 and nextpos < j-(h(j)-1)/2 ) then
                bmax := 0;  i := 1;  sumh := h(j);  sumih := 0;
                while ( i+j≤m and bmax-j>nextpos and
                                (sumh+10)/2-(sumih+10*i)/(sumh+10) > bmax) do
                        sumh := sumh + h(j+i);
                        sumih := sumih + h(j+i)*i
                        bmax := max( bmax, int(sumh/2-sumih/sumh) );
                        i := i + 1
                        end while;
                nextpos := min( max( j-bmax, nextpos), m-n+nextkey )
                end if;
        for i := 1 until h(j) do
                out(nextpos) := in(nextkey);
                nextpos := nextpos + 1;  nextkey := nextkey + 1
                end for;
        end while;
while ( nextkey < n ) do
        out(nextpos) := in(nextkey);
        nextpos := nextpos + 1;  nextkey := nextkey + 1
        end while;
```

4.5 - Average Number of Accesses for Interpolation Hash.

In this section we will evaluate the average number of accesses needed to find an element in an interpolation hash table. Notice that the problem, although similar, cannot be attacked with the same tools as those used to analyze hash searching.

To find a lower bound on the number of accesses, we will study the average number of accesses in the ideal situation that no elements are "pushed" out of their initial positions. We know that this is going to be a rough estimate when the table is full, but a good one when the occupation factor is low.

If we succeed in minimizing the total distance of all keys to their initial probing position we have an asymptotic average number of accesses

$$(4.5.1)\quad E[\text{accesses}] = [0\times e^{-\beta} + 1\times\beta e^{-\beta} + \ \ldots\ + [\lceil i/2\rceil \times \lfloor i/2\rfloor + i]\beta^i e^{-\beta}/i! + \ \ldots$$

$$= (2\beta^2+10\beta-1+e^{-2\beta})/(8\beta) \sim 1 + \beta/2 - \beta^2/6 + \beta^3/12 \ \ldots,$$

assuming a Poisson distribution of the number of keys that probe initially to a given location.

The same idea can be extended to consider two, three, etc. adjacent locations at a time, improving the analysis with respect to the interdependence of adjacent keys. However we have no guarantee now that the search, after the first probe, will be in sequential positions, and the result is no longer a lower bound.

The average number of accesses considering two adjacent locations is given by

$$(4.5.2)\quad E[\text{accesses}] = \sum[i+j + \lceil (i+j)/2\rceil \times \lfloor (i+j)/2\rfloor - \min(i,j)]e^{-2\beta}\beta^{i+j}/(i!j!)$$

The above results are not satisfactory for practical purposes, so we will look for a more accurate answer. Observe that if we have a contiguous sequence

$$k(i-1)=\text{null},\quad k(i)\neq\text{null},\quad k(i+1)\neq\text{null},\ldots,k(j)\neq\text{null},\quad k(j+1)=\text{null},$$

where we want to count the average number of accesses to finding those keys, the problem is equivalent to find those elements, with interpolation search, in a full table of size $i-j+1$. The average number of accesses is then

$$(4.5.3)\quad E[\text{accesses}] = \sum_{k=1}^{\infty} kP(k)B(k)$$

where P(k) is the probability of finding a contiguous sequence of length k and B(k) is the average number of accesses to search any key in a full table of size k [3.4]. We consider all keys equally likely of being searched.

Since B(k) is convex, using Jensen's inequality we derive

(4.5.4) $\quad E[\text{accesses}] \leqslant B(E[k])$.

Since B(k) is a very "flat" function this bound is also a good approximation. The expected length of a contiguous sequence is rather difficult to compute. If we are interested in the asymptotic analysis, and ignore the particular cases of sequences starting in the ends of the table, we can use 6.4.41 [Knuth 73] and derive

(4.5.5) $\quad E[k] = Q_1(m,n) \sim (1-\beta)^{-2}$.

Using the empirical formula $B(k) = lg(lg(k+3))$ we derive

(4.5.6) $\quad E[\text{accesses}] = lg(lg(3+(1-\beta)^{-2}))$.

The following table summarizes all the values discussed in this chapter for some selected occupation factors and compares them with the linear probing and double hashing techniques.

	E[accesses]				
occupation	interpolation hash			linear	double
factor	[4.5.1]	[4.5.2]	[4.5.6]	probing	hashing
100%	$lg(lg(n))$	$lg(lg(n))$	$lg(lg(n))$	$[m\pi/8]^{1/2}$	$ln(n)+\gamma-1$
99%	1.39	2.86	3.73	50.5	4.65
95%	1.38	2.71	3.11	10.50	3.15
90%	1.36	2.53	2.74	5.50	2.56
85%	1.34	2.35	2.45	3.83	2.23
80%	1.33	2.18	2.23	3.0	2.01
70%	1.29	1.84	1.92	2.17	1.72
50%	1.22	1.23	1.50	1.50	1.39

From the results obtained in 3.5 about the variance of the number of accesses, we conclude that for this algorithm the variance is also $O(1)$.

The above table shows that if we are only interested in the average case, the interpolation hash scheme is only worthwhile to use when we reach very high levels of occupancy. On the other hand if we are concerned with the variance and its consequences on the probable worst case, interpolation hash is a very good choice.

4.6 - Simulation Results.

This section summarizes the simulation results for for the interpolation hash algorithm described in 4.5. For all the cases studied we computed the average number of accesses needed to find any element and its 95% confidence limits, the sample size, i.e. number of random files generated, the maximum average number of accesses for any file in the sample, the maximum number of accesses needed to locate any single element in any of the samples and an estimate of the variance of the number of accesses.

The simulation was done for occupation factors 99%, 95%, 90%, 85%, 80%, 70% and 50% combined with table sizes 10, 20, 40, 100, 200, 500 and 1000.

number keys	table size	average accesses	sample size	maxim average	maxim access	variance
Occupation factor β=99%						
99	100	2.6240±0.0343	300	3.5152	8	1.142
198	200	2.8782±0.0595	100	3.8081	9	1.275
495	500	3.0478±0.0555	50	3.5152	8	1.310
990	1000	3.1980±0.0523	50	3.6283	9	1.350
Occupation factor β=95%						
19	20	1.9823±0.0220	1000	3.6316	7	0.775
38	40	2.1757±0.0290	500	3.1842	7	0.940
95	100	2.3648±0.0337	300	3.6105	9	1.103
190	200	2.4400±0.0558	100	3.3211	9	1.190
475	500	2.6088±0.0681	50	3.3326	8	1.282
950	1000	2.6734±0.0520	50	3.2242	9	1.376

Occupation factor β=90%

9	10	1.6588±0.0213	1000	2.8889	5	0.545
18	20	1.8517±0.0206	1000	3.0000	6	0.703
36	40	1.9949±0.0285	500	3.1389	7	0.862
90	100	2.0950±0.0285	300	2.9778	8	0.958
180	200	2.1878±0.0483	100	2.8611	8	1.052
450	500	2.2690±0.0480	50	2.6422	8	1.149
900	1000	2.2738±0.0434	50	2.5900	7	1.178

Occupation factor β=85%

9	10	1.6731±0.0206	1000	3.0000	5	0.540
17	20	1.7281±0.0195	1000	3.2353	6	0.628
34	40	1.8214±0.0251	500	2.8529	6	0.731
85	100	1.9119±0.0260	300	2.6824	6	0.821
170	200	1.9771±0.0396	100	2.8235	7	0.906
425	500	1.9797±0.0403	50	2.3200	8	0.927
850	1000	2.0050±0.0264	50	2.2671	8	0.958

Occupation factor β=80%

8	10	1.5185±0.0187	1000	2.6250	5	0.439
16	20	1.6413±0.0179	1000	3.0000	5	0.560
32	40	1.7031±0.0218	500	2.7188	7	0.638
80	100	1.7794±0.0233	300	2.6625	7	0.727
160	200	1.7919±0.0292	100	2.2875	6	0.742
400	500	1.8594±0.0328	50	2.1650	6	0.842
800	1000	1.8316±0.0233	50	2.0513	9	0.809

Occupation factor β=70%

7	10	1.4016±0.0166	1000	2.4286	4	0.339
14	20	1.4695±0.0159	1000	2.9286	5	0.423
28	40	1.5235±0.0191	500	2.5000	5	0.488
70	100	1.5560±0.0164	300	2.2143	6	0.516
140	200	1.5909±0.0221	100	1.9500	5	0.560
350	500	1.5891±0.0196	50	1.7571	5	0.556
700	1000	1.5948±0.0145	50	1.7229	5	0.570

Occupation factor $\beta=50\%$

5	10	1.2194±0.0134	1000	2.6000	4	0.197
10	20	1.2590±0.0112	1000	2.3000	5	0.236
20	40	1.2765±0.0115	500	1.8000	4	0.250
50	100	1.3128±0.0113	300	1.6400	5	0.294
100	200	1.3050±0.0129	100	1.5000	4	0.282
250	500	1.3138±0.0138	50	1.4400	4	0.297
500	1000	1.3096±0.0088	50	1.3980	4	0.293

In all cases, the simulation shows a growth in the average number of accesses with the size. This growth can be attributed to the edge effects, that are ignored in the analysis, and become significant for small sizes, or for large occupation factors.

As a conclusion we may say that in general 4.5.2 provides a more accurate estimate of the average, while 4.5.6 is too pessimistic.

4.7 - Addition and Deletion of Records.

We will analyze in this section the addition of a new record to an interpolation hash table created with the algorithm described in section 4.4.

Following a search sequence we may find that the new record probes into an empty position or we find two contiguous keys that bracket the new key value. In the first case the insertion is trivial and we need only to make sure that the record is in the right order. In the second case we follow a procedure similar to the one described for the creation of the table. Since our main objective is to keep the elements as close as possible to their initial probe positions, we want to move records in the direction (which is possible) and minimizes the total "torque" of the keys. Consequently we find the two empty positions that delimit the contiguous sequence of keys where we want to insert the new key, and compute the displacement of any of the edges of this sequence.

For example suppose that the following is part of a file of size 10

...	k(3)	k(4)	k(5)	k(6)	k(7)	k(8)	...
		0.43	0.44	0.45	0.56		
	h(3)	h(4)	h(5)	h(6)	h(7)		
	0	0	3	1	0		

and we want to insert the new key 0.52. Then $k(6)<0.52<k(7)$.

The contiguous sequence is delimited by k(3) and k(8). Computing the displacement at k(4) we find

(4.7.1) $h(6) = 2$, (with the new key),

(4.7.2) $b = 5/2 - \frac{1}{2}(3+4)/5 \sim 1$

So the optimal allocation for this example, as can be verified, is obtained by shifting k(4) one position to the left yielding

k(3)	k(4)	k(5)	k(6)	k(7)	k(8).
0.43	0.44	0.45	0.52	0.56	

If no empty position is found towards one end, we are forced to move in the opposite direction. If we find no empty positions towards any end, the table is full and the addition, obviously, is not possible.

The deletion of an element is trivial; however, if we want to preserve the minimal "torque" property we may need some further reorganization.

Let k(j) be the table entry to be deleted. If k(j–1) or k(j+1) is not empty, we compute the rightmost displacement of k(j–1) and the leftmost displacement of k(j+1). If any of the displacements round to 1 or greater, we move the keys whose displacement was largest to the vacated position. The shifting of records continues by computing the displacement for each key, until it is null.

To illustrate the procedure with an example, we will use the same file as before and delete the key 0.52.

After deletion, the file looks like:

k(3)	k(4)	k(5)	k(6)	k(7)
0.43	0.44	0.45		0.56.

For k(7) the computed displacement is

(4.7.3) $b_7 = ½ - ½ - (-1 \times 1)/1 = 1.$

Note that we have to include the key 0.56 that interpolates initially to k(6). The rightmost displacement of k(5) is

(4.7.4) $b_7 = 3/2 - ½ - 0/1 = 1.$

In this case both displacements are equal, and we can easily verify that any of the shifts minimizes the total distance from the keys to their initial interpolation position.

Both algorithms bear a close resemblance to the corresponding algorithms described in [Amble 73] with the change in the hash function, for an interpolation function, and a different criteria instead of the "torque" concept.

4.8 - Application to Sorting.

From the previous analysis we find enough arguments to look for a sorting method. We will derive two methods, the first arises from the construction algorithm defined in 4.4 and the second one from the repetitive addition of elements into an initially empty, interpolation hash table. These methods are not totally original, since we have in the literature descriptions of several algorithms making use of "address computation". Close references to our work are I. Flores [1960], Isaac and Singleton [1956] and Kronmal and Tarter [1965].

The first method, for the case of a full table, is greatly simplified since all positions will be occupied. With this in mind, the sorting algorithm for U(0,1) keys in a pseudo-language is as follows.

```
/*  sorts in(1:n) in ascending order into out(1:n)
        using the auxiliary integer array iwk(1:n) */
for i := 1 until n do iwk(i) := 0  end for;
for i := 1 until n do  j := φ(in(i),n);
                                   iwk(j) := iwk(j) + 1;
                                   end for;
for i := 2 until n do iwk(i) := iwk(i-1) + iwk(i) end for;
for i := 2 until n do j := φ(in(i),n);
                                   out(iwk(j)) := in(i);
                                   iwk(j) := iwk(j) - 1;
                                   end for;
for i := 2 until n do
        if out(i-1) > out(i) then k := i;
                repeat out(k-1) :=: out(k); k := k - 1
                        until k = 1 or out(k-1) ⩽ out(k);
                end if;
        end for;
return
```

The operation :=: means interchange. The assignment j := φ(in(i),n) will be normally implemented as

(4.8.1) j := in(i)*n + 1;.

To evaluate the algorithm we first decide which are the critical quantities to evaluate. These will be the number of function evaluations, the number of comparisons and the number of interchanges.

The total number of interpolation function evaluations is constant and equal to 2m. The total number of key comparisons is m−1+{number of interchanges}. Interchanges can only occur between elements that interpolated initially to the same position. If we have h(i) elements probing in position i we have then

(4.8.2) E[interchanges for elements probing in i] = h(i)[h(i)−1]/4.

The random variable h(i) has a binomial distribution, hence the expected number of interchanges for the elements falling into any position is

$$(4.8.3)\quad E[\text{interchanges}] = 1/4\sum_{k=0}^{m} k(k-1)\binom{m}{n} m^{-k}[(m-1)/m]^{n-k}$$

$$= (m-1)/(4m).$$

Consequently the total average number of interchanges is

(4.8.4) E[interchanges] = (m−1)/4,

and the total number of comparisons is

(4.8.5) E[total comparisons] = 5(m−1)/4.

To compute the variance of the number of interchanges we find first that

(4.8.6) var{interchanges for elements probing in i} = h(i)[2h(i)^2+3h(i)−5]/72,

similar to the bubble sort algorithm applied to each "bucket" [Knuth 73].

To compute the total variance we have to compute the average [4.8.6] plus the variance of the mean. In this case we should notice the dependence between all variables, and compute

(4.8.7) $E[\sum h(i)^2] = 2m-1,$

(4.8.8) $E[\sum h(i)^3] = [5m^2-6m+2]/m,$

(4.8.9) $E[(\sum h(i)^2)^2] = [4m^3-2m^2-3m+2]/m,$

and finally

(4.8.10) $\text{var}\{\text{interchanges}\} = [20m^2-33m+13]/(72m)$.

The standard deviation is approximately $0.53m^{1/2}$ and the coefficient of variation is $O(m^{-1/2})$ which proves that the algorithm is very stable in its running time.

The following code is a FORTRAN implementation of this routine for U(0,1) variables.

```
      SUBROUTINE SORT(AIN,AOUT,N,IWK)
      DIMENSION AIN(N), AOUT(N), IWK(N)
      DO 10 I=1,N
         IWK(I) = 0
   10 CONTINUE
      DO 20 I=1,N
         J = AIN(I)*N+1
         IWK(J) = IWK(J)+1
   20 CONTINUE
      DO 30 I=2,N
         IWK(I) = IWK(I)+IWK(I-1)
   30 CONTINUE
      DO 40 I=1,N
         J = AIN(I)*N+1
         AOUT(IWK(J)) = AIN(I)
         IWK(J) = IWK(J) - 1
   40 CONTINUE
      DO 60 I=2,N
         IF(AOUT(I-1) .LE. AOUT(I))GO TO 60
         K = I
   50       AUX = AOUT(K-1)
            AOUT(K-1) = AOUT(K)
            AOUT(K) = AUX
            IF(K .LE. 2)GO TO 60
            K = K-1
         IF(AOUT(K-1) .GT. AOUT(K))GO TO 50
   60 CONTINUE
      RETURN
      END
```

Running time comparisons of this algorithm with Quicksort, implemented as suggested in Knuth [1974, 8A], indicate that both run at the same speed for files of size approximately 30. Given the different orders, and also found by simulation, the relation between the average running time is approximately 2:1 against Quicksort for files of size 1000. The comparisons were done sorting pseudo-random numbers distributed U(0,1).

The second sorting algorithm is derived from the repetitive addition of keys into an empty table. This algorithm has the advantage that does not need to read the keys twice but as a tradeoff, we will need some extra storage to guarantee some efficiency. Still, this sorting algorithm cannot be considered an "in place sort" unless we obtain the keys from an external source, since the table needs to be empty at the beginning of algorithm.

We will depart somewhat from the exact replica of the addition of elements to an empty table in two aspects. First, we will insert elements in the first empty location we find with interpolation. This does not guarantee complete ordering, but the very few resulting inversions, will be corrected when we contract the table to a sorted list. Furthermore, to avoid the special cases caused by the ends of the table, we will leave some positions to overflow records at both ends.

With these considerations, the algorithm in pseudo-language is as follows.

```
for i := 1 until n do
        low := 5; lowkey := 0;
        high := n/0.8+6; highkey := 1;
        key := newkey(....);
        while high-low > 1 do
                j := φ((key-lowkey)/(highkey-lowkey),high-low-1);
                if out(j) = null then go to allocate end if;
                if out(j) < key then do low := j; lowkey := out(j);
                        else do high := j; highkey := out(j) end if;
                end while;
        j := low;
        for k from 2 while out(abs(j)) ≠ null do j := -(j+i); end for;
        j := abs(j); incr := sign(1,low-j);
        repeat out(j) := out(j+incr); j := j+incr until j = low or j = high;
allocate:
        out(j) := key;
        end for;
```

```
/* final compression and possible inversions */
j := 0;
for i := 1 until n do
        repeat j := j+1 until out(j) ≠ null;
        out(i) := out(j);
        for k := i-1 downto 1 do
                if out(k) > out(k+1) then out(k) :=: out(k+1)
                        else break; end if;
                end for;
        end for;
return
```

For the above algorithm the maximum level of occupation is 80%, but this can be changed to any desired value. The following table shows simulation results from several runs of the number of $\phi(..)$ function evaluations, the number of moves needed to make space for the new element, and the number of inversions required in the final compression.

file size	sample size	moves	function evaluations	inversions
10	1000	2.541±0.107	13.698±0.110	0.0810±0.0197
50	1000	24.809±0.825	73.292±0.318	0.5040±0.0541
100	500	57.23±2.20	148.062±0.674	1.126±0.119
500	100	361.8±20.5	753.65±3.26	6.010±0.768
1000	100	730.8±26.2	1503.35±4.59	13.020±0.971

5. - Conclusions.

In this thesis we have exhaustively analyzed a family of search algorithms characterized by the use of an interpolation function to decide new positions to probe. These algorithms are known by the name of *interpolation search*, or *estimated entry search*.

In the second chapter, we formally introduced the model of a search algorithm, a file, and what we expect to measure from the algorithm. Particular attention was given to some general lower bounds of other search algorithms. A simple transformation over the keys of a file allowed us to consider a wide variety of possible distributions of keys. Finally we derived several formulas related to the distribution of ordered random uniform variables and ordered discrete rectangular variables.

The third chapter is related to the interpolation search algorithms used to search a full table. The basic interpolation search algorithm for a full table runs in $lg(lg(n))+O(1)$ time, in both the successful and unsuccessful case. A very good empirical formula was found, that has the correct asymptotic behaviour and gives also very good approximation for small values, for the expected number of accesses in a successful search, $lg(lg(n+3))$. The asymptotic distribution of the number of accesses was found, from which we derived the variance of the number of accesses that is $O(1)$. Numerical study of the distribution of number of accesses and simulation strongly suggested that its value is close to 1.10. From this we concluded, for example, that 99.9% of the accesses to elements in a table require no more than the average plus 3 accesses. This shows the stability of the algorithm which is better than that of some hashing schemes. The function $lg(lg(n))$ is an extremely flat function. For all practical applications of tables in core, the average may be considered bounded by 4. The unsuccessful search has almost exactly the same properties as the successful case. The average number of accesses is $lg(lg(n))+0.58$ and the variance near 1.10. The search time is not greatly affected by the presence or absence of the element in the table in contrast, for example, with hashing that may change its order. These asymptotic results were obtained by an information theoretic argument as well as a conventional approach. We also derived the exact average which becomes exponentially complicated to compute, and a numerical approximation of the exact average that showed remarkably good agreement with the simulation results. The unsuccessful search case and the known successful search case were studied separately. Some variations of the main algorithm were analyzed: the optimal interpolation search, the one-interpolation then sequential algorithm, the reducing algorithm which selects the probe position to minimize the expected resulting length of the interval in which the key is located, and the binary interpolation search.

The fourth chapter analyzes the interpolation search algorithm when it is used to search a non-full table. This algorithm may be viewed as a special kind of hash searching. The optimal configuration was analyzed and it was found that even simple properties, like probing

always to a non empty location when searching an element located in the table, may not be guaranteed by any construction algorithm. The construction, addition and deletion of elements was described based on heuristics that produce good configurations. These heuristics are based on the principle of "torque" applied to keys. The average running time of the search algorithm was approximated with several formulas, and later they were verified by simulation runs. From this algorithm we derived two new sorting schemes. The first resembles bucket sort, without lists, and is derived from the creation algorithm. The second sorting algorithm is derived from the successive addition of elements into an empty table. Both sorting algorithms run in time $O(n)$.

6. - References.

[Abramowitz 64] Abramowitz, M., and Stegun, I.A. *Handbook of Mathematical Functions.* Dover Publications, New York. 1964.

[Amble 73] Amble, O. and Knuth, D.E. Ordered Hash Tables. The Computer Journal. 17–2, (May 1974), 135–142.

[Arfwedson 51] Arfwedson, G. A Probability Distribution Connected with Stirling's Second Class Numbers. Skandinavisk Aktuarietidskrift. 34–3 (1951), 121–132.

[Ash 64] Ash, R.B. *Information Theory.* Interscience, New York, 1964.

[Brent 73] Brent, R.P. *Algorithms for Minimization Without Derivatives.* Prentice Hall, Englewood Cliffs, 1973.

[Brown 71] Brown, W.S. ALTRAN User's Manual. Bell Laboratories, New Jersey, 1971.

[Coveyou 67] Coveyou, R.R. and MacPherson, R.D. Fourier Analysis of Uniform Random Number Generators. JACM, 14–1 (Jan. 1967), 100–119.

[Davis 67] Davis, P.J. and Rabinowitz, P. *Numerical Integration.* Blaisdell, Waltham, 1967.

[Dingle 73] Dingle, R.B. *Asymptotic Expansions: Their Derivation and Interpretation.* Academic Press, London, 1973.

[Feller 57] Feller, W. *An Introduction to Probability Theory and its Applications.* John Wiley, New York, 1957, Vol.I.

[Flores 60] Flores, I. Computer Time for Address Calculation Sorting. JACM, 7–3, (Oct. 1960) 389–409.

[Gonnet 72] Gonnet, G.H. Unpublished notes on Average Number of Accesses to Files. (in Spanish),(Aug. 1972).

[Isaac 56] Isaac, E.J. and Singleton, R.C. Sorting by Address Calculation. JACM, 3–2, (July 1956), 169–174.

[Johnson 69] Johnson, N.L., and Kotz, S. *Distributions in Statistics.* Houghton Mifflin, Boston, 1969, Vol. I, (Discrete Distributions).

[Johnson 70] Johnson, N.L. and Kotz, S. *Distributions in Statistics.* Houghton Mifflin, Boston, 1970, Vol. III, (Continuous Distributions–2).

[Kalbfleisch 75] Kalbfleisch, J.G. *Probability and Statistical Inference.* University of Waterloo, Waterloo, 1975.

[Knott 75] Knott, G.D. Hashing Functions. The Computer Journal, 18–3, (August 1975), 265–278.

[Knuth 73] Knuth, D.E. *The Art of Computer Programming.* Addison–Wesley, Reading, 1973.

[Knuth 74] Knuth, D.E. Structured Programming with Go To Statements. Computing Surveys, 6–4, (Dec. 1974), 261,302.

[Kronmal 65] Kronmal, R.A. and Tarter, M.E. Cumulative Polygon Address Calculation Sorting. Proceedings of the 20th National ACM Conference, 1965, 376–384.

[Kruijer 74] Kruijer, H.S.M. The Interpolated File Search Method. Informatie, 16–11 (Nov. 1974), 612–615.

[Luke 69] Luke, Y.L. *The Special Functions and their Approximation.* Academic Press, New York, 1969.

[Peterson 57] Peterson, W.W. Addressing for Random–Access Storage. IBM Journal of Research and Development, 1–4 (April 1957), 130–146.

[Price 71] Price, C.E. Table Lookup Techniques. Computing Surveys, 3–2 (June 1971), 56–58.

[Romanovsky 23] Romanovsky, V. Note on the Moments of the Binomial about its Mean. Biometrika, 15, 410–412.

[Simon 72] Simon, J.C. and Guiho, G. On Algorithms Preserving Neighborhood, to File and Retrieve Information in a Memory. Int. Journal of Computer and Information Science, 1-1, (1972), 3-15.

[Stevens 37] Stevens, W.L. Significance of Grouping. Annals of Eugenics, 8 (1937), 57-69.

[Tarter 68] Tarter, M.E. and Kronmal, R.A. Estimation of the Cumulative by Fourier Series Methods and Application to the Insertion Problem. Proceedings of the 23rd National ACM Conference, (1968), 491-497.

[Temme 75] Temme, N.M. Uniform Asymptotic Expansions of the Incomplete Gamma Functions and the Incomplete Beta Function. Mathematics of Computation, 29-132 (Oct. 1975), 1109-1114.

[Whitt 75] Whitt, J.D., and Sullenberger, A.G. The Algorithm Sequential Access Method: An Alternative to Index Sequential. Communications ACM, 18-3 (March 1975), 174-176.

[Yao 76] Yao, A.C. and Yao, F.F. The Complexity of Searching an Ordered Random Table. Proceedings of the Symposium on Foundations of Computer Science. Houston, (Oct. 1976), 173-176.

7. - Appendix I: Asymptotic Expansions.

This appendix contains a fairly loose collection of asymptotic expansions of common functions or expressions. The criteria used for the length of the expansion i.e. order, is rather artificial and depends upon computability, number of terms in the numerator and is at most 7.

It is assumed that the expansions are for $n\to\infty$ unless otherwise specified. It is also assumed that a,b,c and z are all $O(1)$ when $n\to\infty$.

A1.1 - Exponential type expansions.

(A1.1.1)

$$(1 + z/n)^n = e^z \left[1 - z^2/2n + z^3(3z+8)/24n^2 - z^4(z^2+8z+12)/48n^3 + \right.$$

$$z^5(15z^3+240z^2+1040z+1152)/5760n^4 - z^6(3z^4+80z^3+680z^2+2112z+1920)/11520n^5 +$$

$$z^7(63z^5+2520z^4+35280z^3+211456z^2+526176z+414720)/2903040n^6 -$$

$$\left. z^8(9z^6+504z^5+10500z^4+102592z^3+485856z^2+1027584z+725760)/5806080n^7 \ \ldots \right]$$

(A1.1.2)

$$(1 + 1/n)^n = e \left[1 - 1/(2n) + 11/24n^2 - 7/16n^3 + 2447/5760n^4 - 959/2304n^5 + 238043/580608n^6 - \right.$$

$$\left. 67223/165888n^7 \ \ldots \right]$$

$$= e \left[1 - 1/[2(n+1)] - 1/24(n+1)^2 - 1/48(n+1)^3 - 73/5760(n+1)^4 - 11/1280(n+1)^5 - \right.$$

$$\left. 3625/580608(n+1)^6 - 5525/1161216(n+1)^7 \ \ldots \right]$$

(A1.1.3)

$$(1 - 1/n)^n = e^{-1} \left[1 - 1/(2n) - 5/24n^2 - 5/48n^3 - 337/5760n^4 - 137/3840n^5 - 67177/2903040n^6 - \right.$$

$$\left. 18289/1161216n^7 \ \ldots \right]$$

(A1.1.4)

$$(1 + a/n + b/n^2 + c/n^3)^n = e^a \left[1 - (a^2-2b)/2n + (3a^4+8a^3-12a^2b-24ab+12b^2+24c)/24n^2 - \right.$$

$$\left. (a^6+8a^5-6a^4b+12a^4-40a^3b+12a^2b^2- 48a^2b+24a^2c+48ab^2+48ac-8b^3+24b^2-48bc)/48n^3 \right.$$

$$\left. \ldots \right]$$

(A1.1.5)

$$(1 + b/n^2 + c/n^3)^n = \left[1 + b/n + (b^2+2c)/2n^2 + b(b^2-3b+6c)/6n^3 + \right.$$

$$(b^4-12b^3+12b^2c-24bc+12c^2)/24n^4 +$$

$$(b^5-30b^4+20b^3c+40b^3-180b^2c+60bc^2-60c^2)/120n^5 +$$

$$\left. (b^6-60b^5+30b^4c+330b^4-720b^3c+180b^2c^2+720b^2c-1080bc^2+120c^3)/720n^6 \ldots \right]$$

(A1.1.6)

$$(1 + a/n + b/n^2)^n = e^a \left[1 - (a^2-2b)/2n + (3a^4+8a^3-12a^2b-24ab+12b^2)/24n^2 - \right.$$

$$\left. (a^6+8a^5-6a^4b+12a^4-40a^3b+12a^2b^2-48a^2b+48ab^2-8b^3+24b^2)/48n^3 \ldots \right]$$

(A1.1.7)

$$(1 + c/n^3)^n = \left[1 + c/n^2 + c^2/2n^4 - c^2/2n^5 + c^3/6n^6 - c^3/2n^7 \ldots \right]$$

(A1.1.8)

$$(1 + b/n^2)^n = \left[1 + b/n + b^2/2n^2 + b^2(b-3)/6n^3 + b^3(b-12)/24n^4 + b^3(b^2-30b+40)/120n^5 + \right.$$

$$\left. b^4(b^2-60b+330)/720n^6 + b^4(b^3-105b^2+1470b-1260)/5040n^7 \ldots \right]$$

A1.2 - Asymptotic Expansions of Sums.

In the following $\zeta(z)$ is the classical Riemman Zeta function, defined by

$$\zeta(z) = \sum_{n=1}^{\infty} n^{-z}.$$

(A1.2.1)

$$\sum_{k=1}^{n-1}[k(n-k)]^{-\frac{1}{2}} = \pi + n^{-\frac{1}{2}}[\,2\zeta(\tfrac{1}{2}) + \zeta(-\tfrac{1}{2})/n + 3\zeta(-3/2)/4n^2 + 5\zeta(-5/2)/8n^3 +$$

$$35\zeta(-7/2)/64n^4 + 63\zeta(-9/2)/128n^5 + 231\zeta(-11/2)/512n^6 + \ldots\,]$$

(A1.2.2)

$$\sum_{k=1}^{n-1}[k(n-k)]^{\frac{1}{2}} = n^2\pi/8 + n^{\frac{1}{2}}[\,2\zeta(-\tfrac{1}{2}) - \zeta(-3/2)/n - \zeta(-5/2)/4n^2 - \zeta(-7/2)/8n^3 -$$

$$5\zeta(-9/2)/64n^4 - 7\zeta(-11/2)/128n^5 - 21\zeta(-13/2)/512n^6 - \ldots\,]$$

(A1.2.3)

$$\sum_{k=1}^{n-1}[k(n-k)]^{-s} = (n/2)^{1-2s}\pi^{\frac{1}{2}}\Gamma(1-s)/\Gamma(3/2-s) + 2n^{-s}[\zeta(s) + s\zeta(s-1)/n + s(s+1)\zeta(s-2)/2n^2$$

$$+ \ldots + \Gamma(s+i)\zeta(s-i)/\Gamma(s)i!n^i + \ldots\,] \quad [s\neq 2,3,4,\ldots]$$

(A1.2.4)

$$\sum_{k=1}^{n}k^k = n^n\,[\,1 + 1/en + (e+2)/2e^2n^2 + (7e^2+48e+24)/24e^3n^3 +$$

$$(9e^3+160e^2+216e+48)/48e^4n^4 + (743e^4+30720e^3+84240e^2+46080e+5760)/5760e^5n^5 +$$

$$(1075e^5+97792e^4+486000e^3+491520e^2+144000e+11520)/11520e^6n^6 + \ldots\,]$$

(A1.2.5)

$$\sum_{k=1}^{n}k^{-s} = \zeta(s) + n^{-s}\,[\,n/(1-s) + \tfrac{1}{2} - s/12n + \Gamma(s+3)/720\Gamma(s)n^3 - \Gamma(s+5)/30240\Gamma(s)n^5 +$$

$$\Gamma(s+7)/1209600\Gamma(s)n^7 \ldots\,] \qquad [s\neq 1]$$

(A1.2.6)

$$\sum_{k=1}^{n}z^k/k = -ln(1-z) + (z-1)^{-1}z^{n+1}/(n+1) + (z-1)^{-2}z^{n+2}/(n+1)(n+2) + \ldots +$$

$$(z-1)^{-i}z^{n+i}B(i,n+1) + \ldots$$

$$= -ln(1-z) + z^{n+1}[\,1/(z-1)n + 1/(z-1)^2n^2 + (z+1)/(z-1)^3n^3 + (z^2+4z+1)/(z-1)^4n^4$$

$$+ (z^3+11z^2+11z+1)/(z-1)^5n^5 + (z^4+26z^3+66z^2+26z+1)/(z-1)^6n^6 +$$

$$(z^5+57z^4+302z^3+302z^2+57z+1)/(z-1)^7n^7 + \ldots\,] \quad [0\leqslant z<1]$$

(A1.2.7)

$$\sum_{k=1}^{n}(1-z/n)^k/k = ln(n/z) - E_1(z) + e^{-z}[(z+1)/2n - (3z^3+z^2+2z+2)/24n^2 + z^4(z-3)/48n^3 - (15z^7-135z^6+230z^5-2z^4-8z^3-24z^2-48z-48)/5760n^4 + \ldots] \quad [z>0]$$

A1.3 - Gamma Type Expansions.

We will denote Euler's constant, 0.57721... by γ.

(A1.3.1)

$$\sum_{k=1}^{n}1/k = \psi(n+1) + \gamma = \gamma + ln(n) + 1/(2n) - 1/12n^2 + 1/120n^4 - 1/252n^6 + 1/240n^8 \ldots$$

(A1.3.2)

$$\sum_{k=1}^{n}ln(k) = ln(\Gamma(n+1)) = (n+\tfrac{1}{2})ln(n) - n + ln(2\pi)/2 + 1/12n - 1/360n^3 + 1/1260n^5 - 1/1680n^7 \ldots$$

(A1.3.3)

$$n! = \Gamma(n+1) = n^n(2\pi n)^{1/2}e^{-n}[1 + 1/12n + 1/288n^2 - 139/51840n^3 - 571/2488320n^4 + 163879/209018880n^5 + \ldots]$$

$$= n^n[2\pi(n+1/6)]^{1/2}e^{-n}[1 + 1/144n^2 - 23/6480n^3 + 5/41472n^4 + 4939/6531840n^5 + 11839/1343692800n^6 - 1110829/1881169920n^7 + \ldots]$$

A1.4 - Asymptotic Expansions of Sums and Definite Integrals Containing e^{-x^2}

(A1.4.1)

$$\int_1^{\infty} e^{-x^2/n}\,dx = [n\pi]^{1/2}/2 - 1 + 1/3n - 1/10n^2 + 1/42n^3 - \ldots.$$

(A1.4.2)

$$\int_1^\infty e^{-x^2/n}\,dx/x = -\gamma/2 + \tfrac{1}{2}ln(n) + 1/(2n) - 1/(8n^2) + 1/(36n^3) - \ldots$$

(A1.4.3)

$$\int_1^\infty e^{-x^2/n}\,dx/x^s = e^{-1/n}/(s-1) - 2/[(s-1)n]\int_1^\infty e^{-x^2/n}\,dx/x^{s-2} \qquad (s>1)$$

(A1.4.4)

$$\int_0^\infty e^{-x^2/n}ln(1+x)\,dx = (n\pi)^{1/2}[ln(n/4)-\gamma]/4 - \gamma/2 + 1 + \tfrac{1}{2}ln(n) + (\pi/n)^{1/2}/2 - [ln(n)+5/3-\gamma]/(6n)$$
$$- (\pi/n^3)^{1/2}/6 + \ldots$$

(A1.4.5)

$$\int_0^\infty e^{-x^2/n}ln(1+x)x\,dx = \tfrac{1}{2}n\int_0^\infty e^{-x^2/n}[1+x]^{-1}\,dx \quad [A1.4.7]$$

(A1.4.6)

$$\int_0^\infty e^{-x^2/n}\,dx = (\pi n)^{1/2}/2$$

(A1.4.7)

$$\int_0^\infty e^{-x^2/n}\,[x+1]^{-1}\,dx = \tfrac{1}{2}ln(n) - \gamma/2 + (\pi/n)^{1/2} - [1+ln(n)-\gamma]/(2n) - 2(\pi/n^3)^{1/2}/3 +$$
$$[ln(n)+3/2-\gamma]/(4n^2) + 4(\pi/n^5)^{1/2}/15 - \ldots$$

(A1.4.8)

$$\int_0^\infty e^{-x^2/n}\,[x+1]^{-2}\,dx = 1 - (\pi/n)^{1/2} + [ln(n)-\gamma]/n + 2(\pi/n^3)^{1/2} - [ln(n)+1-\gamma]/n^2 -$$
$$4(\pi/n^5)^{1/2}/3 + [ln(n)+3/2-\gamma]/n^3 + \ldots$$

(A1.4.9)

$$\int_0^\infty e^{-x^2/n}\,[x+1]^{-s}\,dx = T(s) = 1/(s-1) + 2[T(s-2)-T(s-1)]/(n(1-s)) \quad [s>1]$$

(A1.4.10)

$$\sum_{k=1}^\infty e^{-k^2/n}ln(k+1) = (\pi n)^{1/2}[ln(n/4)-\gamma]/4 - \gamma/2 + \tfrac{1}{2}ln(2\pi n) + (\pi/n)^{1/2}/2 + [G-ln(n)/6]/n - (\pi/n^3)^{1/2}/6 +$$
$$\ldots$$

where $G = 1/3-\frac{1}{2}ln(2\pi)-2\zeta'(-1)+\zeta'(-2)+\gamma/6 = -0.1890087587119...$

A1.5 - Summation Formulas.

Euler-MacLaurin

(A1.5.1)

$$\sum_{k=1}^{n-1} f(k) = \int_1^n f(x)\,dx + \sum_{i=1}^{\infty} B_i/i!\, f^{(i-1)}(x)\Big|_{x=1}^{x=n}$$

Euler-Riemman

(A1.5.2)

$$\sum_{k=1}^{\infty} f(k) = \int_0^{\infty} f(x)\,dx + \sum_{i=-\infty}^{\infty} a_i \zeta(-i)$$

where the coefficients a_i are all finite and

$$f(x) = \sum_{i=-\infty}^{\infty} a_i x^i.$$