Good problems

[DPV] 2.4. Suppose you are choosing between the following three algorithms:

- Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
- Algorithm B solves problems of size n by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
- Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in big-O notation), and which would you choose?

[DPV] 2.17. Given a sorted array of distinct integers $A[1,\ldots,n]$, you want to find out whether there is an index i for which $A[i]=i$. Give a divide-and-conquer algorithm that runs in time $O(\log n)$.

[DPV] 2.28. The *Hadamard matrices* $H_0, H_1, H_2, \ldots$ are defined as follows:

- H_0 is the 1×1 matrix $\left[1\right]$
- For $k>0$, H_k is the $2^k \times 2^k$ matrix

$$H_k = \left[\begin{array}{c|c} H_{k-1} & H_{k-1} \\ \hline H_{k-1} & -H_{k-1} \end{array}\right]$$

Show that if v is a column vector of length $n=2^k$, then the matrix-vector product $H_k v$ can be calculated using $O(n\log n)$ operations. Assume that all the numbers involved are small enough that basic arithmetic operations like addition and multiplication take unit time.

[DPV] 2.29. Suppose we want to evaluate the polynomial $p(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n$ at point x.

(a) Show that the following simple routine, known as *Horner's rule*, does the job and leaves the answer in z.

$z = a_n$
for $i=n-1$ downto 0:
  $z = zx + a_i$

(b) How many additions and multiplications does this routine use, as a function of n? Can you find a polynomial for which an alternative method is substantially better?

[KT] **2.** Recall the problem of finding the number of inversions. As in the text, we are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions between two orderings.

Interesting problems

[DPV] 2.22. You are given two sorted lists of size m and n. Give an $O(\log m + \log n)$ time algorithm for computing the kth smallest element in the union of the two lists.

[DPV] 2.23. An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you *can* answer questions of the form: "is $A[i] = A[j]$?" in constant time.

(a) Show how to solve this problem in $O(n \log n)$ time. (*Hint:* Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

(b) Can you give a linear-time algorithm? (*Hint:* Here's another divide-and-conquer approach:

- Pair up the elements of A arbitrarily, to get $n/2$ pairs
- Look at each pair: if the two elements are different, discard both of them; if they are the same, keep just one of them

Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if and only if A does.)

[CLRS] ***9-2 Weighted median***

For n distinct elements $x_1, x_2, \ldots, x_n$ with positive weights $w_1, w_2, \ldots, w_n$ such that $\sum_{i=1}^{n} w_i = 1$, the ***weighted (lower) median*** is the element x_k satisfying

$$\sum_{x_i < x_k} w_i < \frac{1}{2}$$

and

$$\sum_{x_i > x_k} w_i \le \frac{1}{2} .$$

For example, if the elements are $0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2$ and each element equals its weight (that is, $w_i = x_i$ for $i = 1, 2, \ldots, 7$), then the median is 0.1, but the weighted median is 0.2.

a. Argue that the median of $x_1, x_2, \ldots, x_n$ is the weighted median of the x_i with weights $w_i = 1/n$ for $i = 1, 2, \ldots, n$.

b. Show how to compute the weighted median of n elements in $O(n \lg n)$ worst-case time using sorting.

c. Show how to compute the weighted median in $\Theta(n)$ worst-case time using a linear-time median algorithm such as SELECT from Section 9.3.

The ***post-office location problem*** is defined as follows. We are given n points $p_1, p_2, \ldots, p_n$ with associated weights $w_1, w_2, \ldots, w_n$. We wish to find a point p (not necessarily one of the input points) that minimizes the sum $\sum_{i=1}^{n} w_i \, d(p, p_i)$, where $d(a, b)$ is the distance between points a and b.

d. Argue that the weighted median is a best solution for the 1-dimensional post-office location problem, in which points are simply real numbers and the distance between points a and b is $d(a, b) = |a - b|$.

e. Find the best solution for the 2-dimensional post-office location problem, in which the points are (x, y) coordinate pairs and the distance between points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is the ***Manhattan distance*** given by $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$.

Challenging problems

[CLRS] ***4.4-5***

Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n-1) + T(n/2) + n$. Use the substitution method to verify your answer.