CS 137

Programming Principles

Chapter 11

Big-Oh Notation

Victoria Sakhnini

Table of Contents

Big-Oh Notation

Up to this point, we have been writing code without considering optimization. There are two major ways to optimize code: one is for **memory storage**, and the one we will focus on is **time complexity**. One challenge is quantifying and measuring time complexity — how fast a program runs is machine-dependent and depends on everything from the processor to the operating system. To level the playing field, we use **Big-Oh Notation**.

> 📌 Although this chapter includes mathematical notation and definitions, the main goal is to assess a program's worst-case/best-case running time using Big-Oh notation without showing proof. You do not need to memorize the proofs or the formal definitions.

Definition

In natural language:

```
The functions in O(f(x)) are all functions that grow asymptotically
at an equal or slower rate than f(x).
```

```
More formally: g(x) ∈ O(f(x)) (also written g(x) = O(f(x))) if there exist
positive constants C and x₀ such that for all x > x₀:

|g(x)| ≤ C · |f(x)|
```

> 📌 In computer science, most of our f and g functions take non-negative integers to non-negative integers, so the absolute values are usually unnecessary. We only care about behaviour as x tends to infinity (asymptotic behaviour). In CS, our functions are almost always: $f: \mathbb{N} \to \mathbb{R}$ or even more likely $f: \mathbb{N} \to \mathbb{N}$

Useful Theorem

The following theorem helps classify functions in terms of Big-Oh notation. If the limit of g(x)/f(x) as $x \to \infty$ exists and is finite (and non-zero), then g(x) = O(f(x)) and f(x) = O(g(x)) — they are in the same complexity class.

We can also use L'Hôpital's Rule when evaluating limits of the form 0/0 or ∞/∞ to compare growth rates.

Limit Implication

For positive real valued functions f and g, if $\lim_{x\to\infty} \frac{g(x)}{f(x)} < \infty$, then $g(x) = O(f(x))$.

We can also use this.

Limit Implication

For positive real valued functions f and g, if $\lim_{x\to\infty} \frac{g(x)}{f(x)}$ diverges to infinity, then $f(x) = O(g(x))$.

More Examples

Big-Oh Notation

$$g(x) \in O(f(x))$$
$$\Leftrightarrow$$
$$\exists C \in \mathbb{R}_{>0}\ \exists X \in \mathbb{R}\ \forall x \in \mathbb{R}\ (x \geq X \Rightarrow |g(x)| \leq C|f(x)|)$$

Claim: For any polynomial $g(x) = \sum_{i=0}^{n} a_i x^i$ then $g(x) \in O(x^n)$.

Proof: For all $x \geq 1$, we have

$$\begin{aligned} |a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0| &\leq |a_n|x^n + \ldots + |a_1|x + |a_0| \\ &\leq |a_n|x^n + \ldots + |a_1|x^n + |a_0|x^n \\ &\leq (|a_n| + \ldots + |a_1| + |a_0|)x^n \end{aligned}$$

and the bracket above is a constant with respect to x and we are done.

Claim: $\log_a x \in O(\log_b x)$ for $a, b > 0$.

Proof: For all $x \geq 1$, we have

$$|\log_a x| = \left|\frac{\log_b x}{\log_b a}\right| = \left|\frac{1}{\log_b a}\right| |\log_b x|$$

and the bracket above is a constant with respect to x and we are done.

More Results

In what follows (for real valued functions), let $g_0(x) \in O(f_0(x))$ and $g_1(x) \in O(f_1(x))$. Then...

1. $g_0(x) + g_1(x) \in O(|f_0(x)| + |f_1(x)|)$
2. $g_0(x)g_1(x) \in O(f_0(x) \cdot f_1(x))$
3. $O(|f_0(x)| + |f_1(x)|) = O(\max\{|f_0(x)|, |f_1(x)|\})$

Note that the last bullet is actually a set equality! I'll prove 1 and 3 on the next slides.

Also note that if f_0 and f_1 are positive functions then bullets 1 and 3 can have their absolute values removed.

Prove that $g_0(x) + g_1(x) \in O(|f_0(x)| + |f_1(x)|)$.
Proof: Given $g_0(x) \in O(f_0(x))$ and $g_1(x) \in O(f_1(x))$, we know that there exists constants C_0, C_1, X_0, X_1 such that

$$|g_0(x)| \leq C_0|f_0(x)| \ \forall x > X_0 \qquad \text{and} \qquad |g_1(x)| \leq C_1|f_1(x)| \ \forall x > X_1$$

Now, let $X_2 = \max\{X_0, X_1\}$ and $C_2 = 2\max\{C_0, C_1\}$ to see that by the triangle inequality, for all $x > X_2$,

$$\begin{aligned} |g_0(x) + g_1(x)| &\leq |g_0(x)| + |g_1(x)| \\ &\leq C_0|f_0(x)| + C_1|f_1(x)| \\ &\leq \max\{C_0, C_1\}(|f_0(x)| + |f_1(x)|) \\ &\quad + \max\{C_0, C_1\}(|f_0(x)| + |f_1(x)|) \\ &\leq 2\max\{C_0, C_1\}(|f_0(x)| + |f_1(x)|) \\ &\leq C_2(|f_0(x)| + |f_1(x)|) \end{aligned}$$

and so $g_0(x) + g_1(x) \in O(|f_0(x)| + |f_1(x)|)$

Prove that $O(|f_0(x)| + |f_1(x)|) = O(\max\{|f_0(x)|, |f_1(x)|\})$.
Proof: Notice that

$$\begin{aligned} \max\{|f_0(x)|, |f_1(x)|\} &\leq |f_0(x)| + |f_1(x)| \\ &\leq \max\{|f_0(x)|, |f_1(x)|\} + \max\{|f_0(x)|, |f_1(x)|\} \\ &\leq 2\max\{|f_0(x)|, |f_1(x)|\} \end{aligned}$$

Prove that $O(|f_0(x)| + |f_1(x)|) = O(\max\{|f_0(x)|, |f_1(x)|\})$.
Proof Continued Using the inequalities on the previous slide, to show $\subseteq$, we have that if $g(x) \in O(|f_0(x)| + |f_1(x)|)$, then by definition there exists a positive real C and a real X such that for all $x \geq X$, we have that

$$|g(x)| \leq C(|f_0(x)| + |f_1(x)|) \leq 2C\max\{|f_0(x)|, |f_1(x)|\}$$

and so $g(x) \in O(\max\{|f_0(x)|, |f_1(x)|\})$.

Prove that $O(|f_0(x)| + |f_1(x)|) = O(\max\{|f_0(x)|, |f_1(x)|\})$.
Proof Continued Using the inequalities on the previous slide, to show $\supseteq$, we have that if $g(x) \in O(\max\{|f_0(x)|, |f_1(x)|\})$, then by definition there exists a positive real C and a real X such that for all $x \geq X$, we have that

$$|g(x)| \leq C \max\{|f_0(x)|, |f_1(x)|\} \leq C(|f_0(x)| + |f_1(x)|)$$

and so $g(x) \in O(|f_0(x)| + |f_1(x)|)$.

More Properties

📌 Useful properties for analysing programs: Sequential composition (one block after another): O(f) followed by O(g) => O(f + g) => O(max(f, g)) Nested loops: O(f) loop containing O(g) work => O(f · g) Dropping constants and lower-order terms: $O(5n^2 + 3n + 7) = O(n^2)$ Recursion: Analyse using recurrence relations (e.g., T(n) = 2T(n/2) + O(n) => O(n log n))

Transitivity

If $h(x) = O(g(x))$ and $g(x) = O(f(x))$ then $h(x) = O(f(x))$.
(where f, g and h are real valued functions).

Inequalities

We write $f \ll g$ if and only if $f(x) = O(g(x))$

Growth Rates of Functions

The following is true for ϵ and c fixed positive real constants

$$1 \ll \log n \ll n^{\epsilon} \ll c^n \ll n! \ll n^n$$

In order left to right: constants, logarithms, polynomial (if ϵ is an integer), exponential, factorials.

- You should be aware that there are many other notations here for runtime.
- Big-Oh notation $O(f(x))$ is an upper bound notation ($\leq$)
- Little-Oh notation $o(f(x))$ is a weak upper bound notation ($<$)
- Big-Omega notation $\Omega(f(x))$ is a lower bound notation ($\geq$)
- Little-Omega notation $\omega(f(x))$ is a weak lower bound notation ($>$)
- Big-Theta (or just Theta) notation $\Theta(f(x))$ is an exact bound notation ($=$)

Ideally, we want the Big-Oh notation to be as tight as possible (so really we want to use Θ notation but it involves far too large of a detour). In our class when you are asked for the runtime or anything related to Big-Oh notation, make it the best possible bound.

Let Us Practice

1. printAllElementOfArray:

```
void printAllElementOfArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
    {
        printf("%d\n", arr[i]);
    }
}
```

O(n), where n is the array's length. [n × O(1) = O(n)]

2. printAllPossibleOrderedPairs:

```
void printAllPossibleOrderedPairs(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            printf("%d = %d\n", arr[i], arr[j]);
        }
    }
}
```

$O(n^2)$, where n is the array's length. [n × n × O(1) = $O(n^2)$]

3. fibonacci (recursive):

```
int fibonacci(int num) {
    if (num <= 1) return num;
    return fibonacci(num - 2) + fibonacci(num - 1);
}
```

$O(2^n)$. The function roughly doubles the number of function calls for each increment of the input.

4. printStuff:

```
void printStuff(int arr[], int size) {
    printf("First element of array = %d\n", arr[0]);

    for (int i = 0; i < size / 2; i++) {
        printf("%d\n", arr[i]);
    }

    for (int i = 0; i < 100; i++) {
        printf("Hi\n");
    }
}
```

O(n). First loop: O(n/2) = O(n). Second loop: O(100) = O(1). Total: O(n) + O(1) = O(n).

Extra Practice Problems

Problems 1–10 below were code-image-only in the original slides. Refer to your lecture slides for the code snippets. You can discuss answers on Piazza. Solutions are not posted by the instructor.

1–10. Runtime analysis (code image-only in slides):

For each of the following programs, determine the Big-Oh runtime.
The code for problems 1–10 appears only as images in the original slides.
Refer to the lecture slides for the code snippets.

1) What is the runtime of the following code?

```
int alpha(char *s){
  char *t = s;
  char *u = malloc((strlen(s)+1)*sizeof(char));
  for(;*t;t++){
    strcpy(s,t);
  }
  return -1;
}
```

2) What is the runtime of the following code?

```
int beta(int n){
  int s=0;
  for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
      s++;
    }
  }
  return s;
}
```

3) What is the runtime of the following code?

```
int gamma(int n){
  int s=0;
  for(int i=0; i<1000000; i++){
    for(int j=0; j<n/10; j++){
      s++;
    }
  }
  return s;
}
```

4) What is the runtime of the following code?

```
int epsilon(int n){
  int s=0;
  for(int i=0; i<n; i++){
    for(int j=i; j<2*n; j++){
        for(int k=i; k<3*n; k++){
         s++;
       }
    }
  }
  return s;
}
```

5) What is the runtime of the following code?

```
int zeta(int a[], int n){
  int s=0;
  for(int i=1; i<n; i*=2){
    if (a[i] ==0) return -1;
    else{
      s++;
    }
  }
  return s;
}
```

6) What is the runtime of the following code?

```
int iota(int n){
  int s=0;
  for(int i=1; i<n; i+=2){
   for(int j=i; j<n*n; j++){
     s++;
   }
  }
  return s;
}
```

7) What is the runtime of the following code?

```
int delta(int n, int a[n][n] ){
  int s=1,i=0;
  while(s < n){
    if (a[0][i] < 0){
     s *=2;
    }else{
     s += 2;
    }
  }
  return s;
}
```

8) What is the runtime of the following code?

```c
#include <stdio.h>
#include <stdlib.h>

int * add_to_array(int * arr, int arr_length, int new_int) {
    int * new_arr = malloc(sizeof(int) * (arr_length + 1));

    for (int j = 0; j < arr_length; j++) {
        new_arr[j] = arr[j];
    }

    free(arr);
    new_arr[arr_length] = new_int;

    return new_arr;
}

void main() {
    // Express runtime in terms of n
    // Here, the exact value of n is arbitrary and only
    // only included for the code to compile
    int n = 10000000;

    int * arr = malloc(sizeof(int));
    arr[0] = 0;
    int arr_size = 1;

    for (int i = 0; i < n; i++) {
        arr = add_to_array(arr, arr_size, i);
        arr_size ++;
    }

    free(arr);
}
```

9) What is the runtime of the following code?

```c
int search(int * arr, int n, int value) {
    // Assume arr is sorted in ascending order
    // Assume arr has no duplicates
    // Returns the index where value is located, or -1 if not found

    if (n < 1)
        return -1;

    int mid_pt = n / 2;

    if (a[mid_pt] == value)
        return mid_pt;

    if (a[mid_pt] < value)
        return search(arr, mid_pt, value);

    if (a[mid_pt] > value) {
        int sub_i = search(arr[mid_pt], n - mid_pt, value);

        if (sub_i == -1)
            return -1;

        return mid_pt + sub_i;
    }
```

10) What is the runtime of the following code?

```
int i_cant_even(int n) {
    for (int i = 0; i < n; i++) {
        if (i % 2 != 0) {
            for (int j = 0; j < i; j++) {
                printf("%d\n", j);
            }
        }
}
```

11. Longest consecutive increasing snowfall period:

Write a C program in snowman.c to find the longest consecutive period of days in which the snowfall was strictly increasing (each day's snowfall is greater than the previous day's). Return the starting index (zero-indexed) of that period.

Example:
snowfall[] = {5, 6, 7, 2, 3, 4, 8, 9, 5}
Longest increasing sequence: {2, 3, 4, 8, 9} (length 5, starts at index 3)
Output: 3

Requirement: O(n) time complexity. Do not use any additional arrays.