CS 137

Big-Oh Notation

Fall 2025

Victoria Sakhnini

Table of Contents

Big-Oh Notation

Up to this point, we've been writing code without considering optimization. There are two major ways we try to optimize code - one is for memory storage, and the one we will focus on is time complexity. One major issue is quantifying and measuring time complexity. For example, how fast it runs is machine-dependent and depends on everything from the processor to the operating system type. To level the playing field, we use Big-Oh Notation.

Although I am providing lots of mathematical notations and definitions, the main goal is to assess a program's worst-case/best-case running time using big-Oh Notation without showing proof. Therefore, you do not need to memorize the proofs or the definitions.

Definition

Big-Oh Notation

Let $f(x)$ be a function from the real numbers to the real numbers. Define $O(f(x))$ to be the set of real functions $g(x)$ such that there exists a real number $C > 0$ and a real X such that $|g(x)| \leq C|f(x)|$ for all $x \geq X$.

or more symbolically

Big-Oh Notation

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

or from another perspective, in natural language:

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

For example, $3x^2 + 2 \in O(x^2)$ since for all $x \geq 1$, we have

$$|3x^2 + 2| = 3x^2 + 2 \leq 5x^2 = 5|x^2|.$$

Note in the definitions, $X = 1$ and $C = 5$.

As another example $6\sin(x) \in O(1)$ since for all $x \geq 0$, we have that

$$|6\sin(x)| \leq 6|1|.$$

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. However, this concept is used in other areas of mathematics (most notably Analytic Number Theory), so I'm presenting the general definition.

Usually, instead of `g(x) ∈ O(f(x))`, we write `g(x) = O(f(x))`.

Note that we only care about behaviour as `x` tends to infinity. Indeed, we could consider `x` approaching some fixed `a`, which is often done in analysis (calculus).

In fact, in computer science, our functions are almost always

`f: N → R` or even more likely, `f: N → N`

Useful Theorem

The following helps classify functions in terms of Big-Oh notation.

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.

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

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) What is the runtime of the following code?

```
1. void printAllElementOfArray(int arr[], int size) {
2.     for (int i = 0; i < size; i++)
3.     {
4.         printf("%d\n", arr[i]);
5.     }
6. }
```

Answer: `O(n)`, where n is the array's length. [`n* O(1)` is `O(n)`]

2) What is the runtime of the following code?

```
1. void printAllPossibleOrderedPairs(int arr[], int size) {
2.     for (int i = 0; i < size; i++) {
3.         for (int j = 0; j < size; j++) {
4.             printf("%d = %d\n", arr[i], arr[j]);
5.         }
6.      }
7. }
```

Answer: $O(n^2)$, where n is the array's length. [$n*n*O(1)$ is $O(n^2)$]

3) What is the runtime of the following code?

```
1. int fibonacci(int num){
2.     if (num <= 1) return num;
3.     return fibonacci(num - 2) + fibonacci(num - 1);
4. }
```

Answer: $O(2^n)$. The function doubles the number of function calls for each addition to the input.

4) What is the runtime of the following code?

```
1. void printStuff(int arr[], int size){
2.
3.      printf("First element of array = %d\n",arr[0]);
4.
5.      for (int i = 0; i < size/2; i++) {
6.          printf("%d\n", arr[i]);
7.      }
8.
9.      for (int i = 0; i < 100; i++){
10.           printf("Hi\n");
11.       }
12.  }
```

Answer: `O(n)` [first loop `O(n)`, second loop `O(1)`]

Extra Practice Problems

[**You can discuss together answers on Piazza. I won't post solutions**]

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?

```
#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?

```
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) Winter has arrived, and we want to build a snowman, but we want it to last as long as possible. You can access historical data regarding the snowfall amounts for each day last year. Write a C program in the snowman.c file to find the most extended period of consecutive days in which the snowfall was increasing (i.e., the snowfall amount on each day is greater than the previous day) and output the starting index for this period in the array (assume zero indexing).

For example:

```
snowfall[] = {5, 6, 7, 2, 3, 4, 8, 9, 5}
```

- The longest consecutive increasing sequence is `{2, 3, 4, 8, 9}` as it has a length of 5.
- This sequence started in the 3rd index so the function would return 3.

Write this function in O(n) time complexity and do not use any additional arrays.