Introduction to Big O Notation & Algorithmic Analysis

Learning Objectives

- I will learn how to interpret and use Big O notation
- I will learn how to analyze code using Big O notation
- I will learn how to use these basic algorithms to solve problems:
  - Binary Search
  - Recursion
  - Divide and Conquer

What is an Algorithm?

A. a puzzle to test the IQ of software developers
B. a computer subroutine that abstracts performance optimization
C. a dance move popularized by former Vice President Al Gore
D. [✓ a step-by-step solution to a problem]

Algorithmic Programming – and Computer Science in general – is not just a body of knowledge to be learned, but a *set of skills to be acquired*.

Big O Notation

- How efficient is our algorithm?
- How well does it scale?
- What is the dominant factor in how the runtime increases when we add more data?

Design principle: “touch once” becomes “touch as little as possible”

Estimations of Time Complexity: O, Ω, ϴ

- Big O: what is the worst case?

Upper bound: for example, $f(n) = n^2$ is not any worse than n^3

- Big Ω: what is the best case?

Lower bound: for example, $f(n) = n^2$ is not any better than n

In practice, if you knew it was n^2 you would never estimate it as n or n^3. But sometimes you don't know, but you can estimate the bounds.

- Big ϴ: most precise: when O(f(n)) == Ω (f(n))

In industry today, "Big O" is loosely used to mean both O and ϴ *because they are risk adverse*. Let's be as precise as possible, but we <u>really want to avoid disaster</u>.

Linear Name Search

Fred Williams

John Smith

Bob Allen

Roger Wilco

Max Headroom

Bugs Bunny

Sponge Bob Square Pants

$O(n)$

Print Welcome Screen

```
System.out.println("Welcome to the Class Roster Project!");
```

O (1)

Binary Search

Bob Allen

Bugs Bunny

Fred Williams

John Smith

Max Headroom

Roger Wilco

Sponge Bob Square Pants

Binary Search

Bob Allen

Bugs Bunny

Fred Williams

John Smith

Max Headroom

Roger Wilco ←

Sponge Bob Square Pants

Binary Search

Bob Allen

Bugs Bunny

Fred Williams

John Smith

Max Headroom

Roger Wilco

Sponge Bob Square Pants ←

O (log n)

Definition of Logarithm

$$c^n = x$$

$$\log_c x = n$$

Each step is an inverse power of 2:

$1/2, 1/4, 1/8, 1/16, \ldots 1/2^n$

if n=8 items, I will get there in 3 steps or less:

$$\log_2 8 = 3$$

O (log n)

Logarithmic Math

Logarithms in computer science are base-2, but your calculator function is log base-10. You can easily convert with this formula:

$$\log_a(x) \div \log_a(b) = \log_b(x) \quad \Rightarrow$$

$$\log_{10}(x) \div \log_{10}(2) = \log_2(x)$$

This follows from the definition of logarithm and the logarithm of powers rule, see proof here:

https://proofwiki.org/wiki/Change_of_Base_of_Logarithm

Fist of Five

I'm completely lost

I need this explained again

I sorta get it

I understand but have questions

I'm confident

I could teach this!

Recursion

Recursion is when a thing is used to define itself. This can be very succinct and powerful. It is not circular reasoning as long as the definition builds from a base case.

Example: Definition of ***Ancestors***

Without recursion: My ancestors are my parents, my parents' parents, their parents' parents, the parents of my parents' parents, and so on…

With recursion: My ancestors are my parents and their ancestors.

Base case: my parents

Recursion is just a fancy way of saying "and so on."

Recursion

Java Code Example: ***Factorial***

Base case: n = 0 or 1

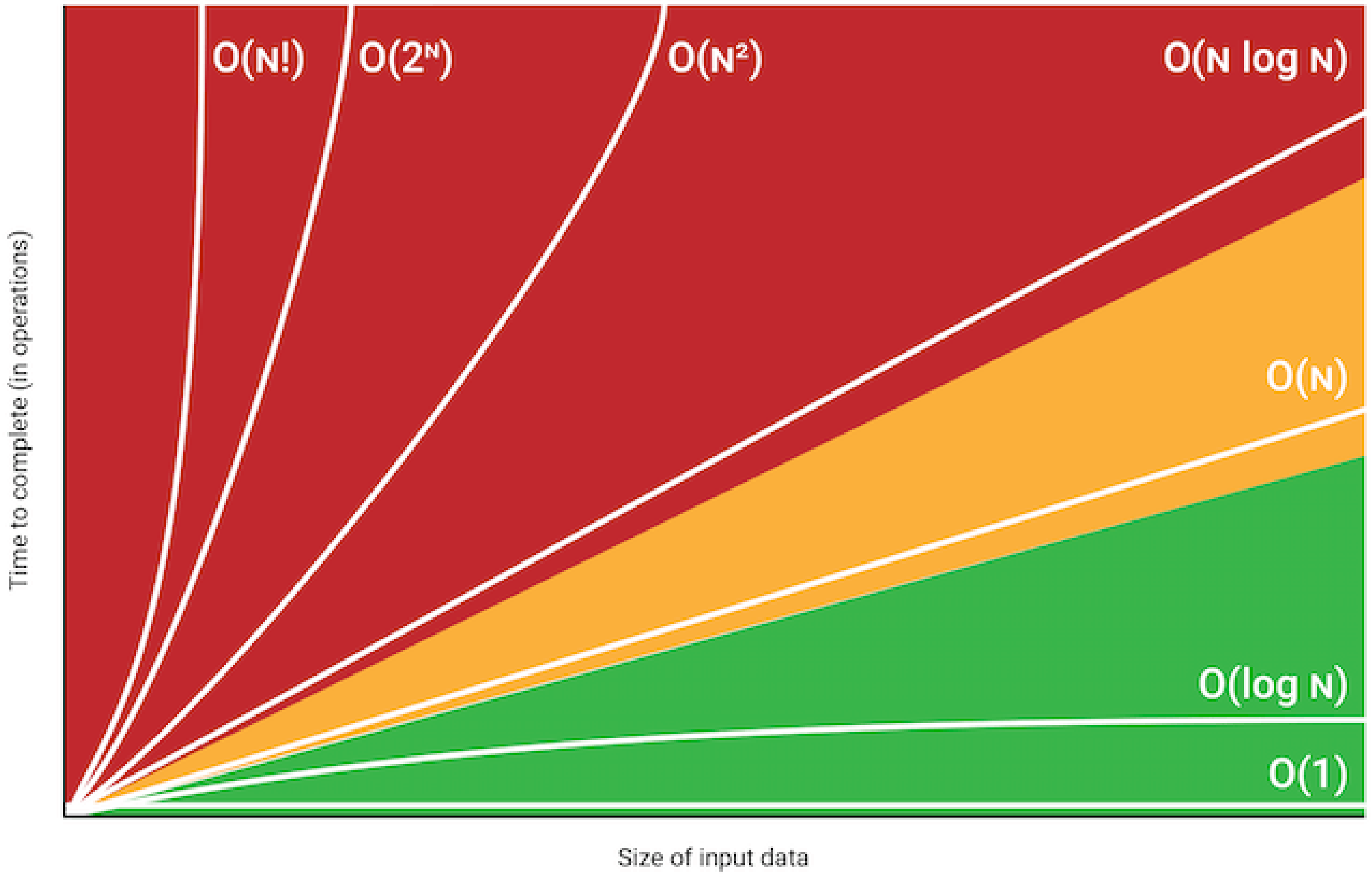
```
public static long factorialProgram(long n) {
  if (n <= 1) {
    return 1;
  } else {
    return n * factorialProgram(n - 1);
  }
}
```

Recursion: Divide & Conquer

- Recursion is often used in “Divide and Conquer” strategies
- If I cut my list in half recursively before doing any work, then similar to Binary Search, the number of iterations, or steps will be proportional to O(log n)
- Many sorting algorithms use Divide and Conquer to divide a list down to one item, then return that item as a sorted list (of one) – which becomes your base case.
- Merging two sorted lists is easy to do O(n)
- Number of recursive iterations: log n, Merging work each iteration: n
- This is how the Merge Sort algorithm works

O (n log n)

Big O Time Complexity Comparison



Which is Greater?

N^2

N log N

$N^2 > N \log N$

$N > \log N$

$N * N > N * \log N$

$N^2 > N \log N$

Which is Greater?

N

N log N

N log N > N ...for all N > 2

log N > 1

N * log N > 1 * N

N log N > N

We don't care about N <= 2

If you only have two customers, you don't have a scalability problem

Remember, in Computer Science when we talk about log we mean **$\log_2$**

Which is Greater?

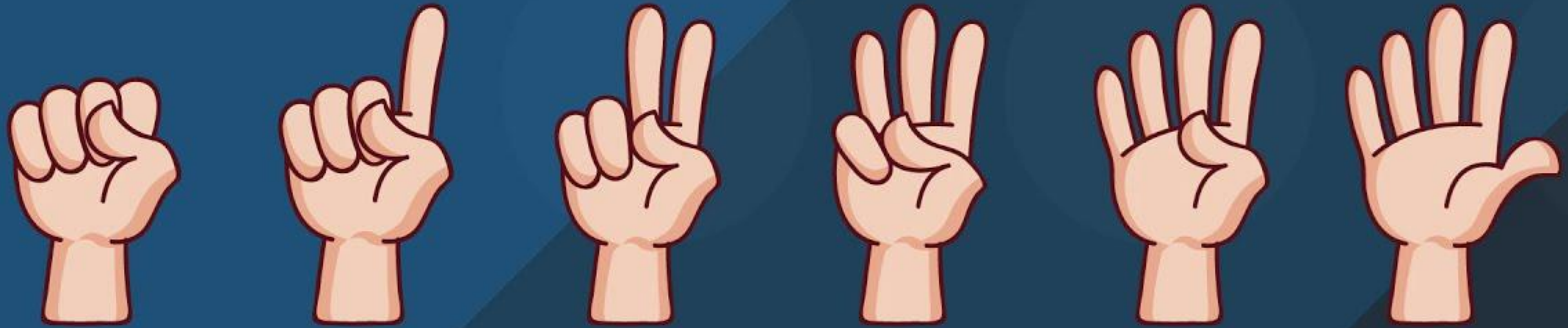
$\sqrt{N}$

log N

Big O Time Complexity Comparison

- If you can't remember which is bigger, $\sqrt{n}$ or $\log_2(n)$, try putting in some numbers:
  - $\sqrt{64} = 8$
  - $\log_2(64) = 6$
  - $\sqrt{4} = \log_2(4) = 2$ **=>** for $n > 4$, *log is smaller than sq. root*
- Be able to analyze code to determine its Big O

Fist of Five


I'm completely lost
I need this explained again
I sorta get it
I understand but have questions
I'm confident
I could teach this!

Modulus

- In computer programming, the % is used to mean *modulus*
- The modulus is the remainder after integer division
- For example, in normal math: 5 ÷ 2 = 2 remainder 1
- In integer math we drop the fractional part but can use modulus to give us the remainder. In computer programming:
- 5 / 2 == 2 ⬅ integer math
- 5 % 2 == 1 ⬅ the remainder
- *Don't let the percent sign confuse you. Modulus has nothing to do with percentages*.

Modulus

- The modulus will always be less than the divisor
- For example: n % 5 will always be 0, 1, 2, 3, or 4
- If the remainder is zero, that means the divisor went into the dividend evenly

```
public static boolean isMultipleOfFive(int n, int divisor){
  if (n % divisor == 0) return true;
    return false;
  }
```

Filtering on modulus == 0 is like dividing

- Given a large set of random numbers, how often will `isMultipleOfFive` return `true`?

Multiple Factors: *Don't Sweat the Small Stuff*

$2n^2 + n + C$

$\Rightarrow O(n^2)$

Ignore constants, coefficients, and non-dominant terms

In-Class Assignment

The Two Sum Problem

Given an array of integers “nums” and an integer “target”, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

Example 1:
Input: nums = [2,7,11,15], target = 9
Output: [0,1]

Example 2:
Input: nums = [3,2,4], target = 6
Output: [1,2]

Example 3:
Input: nums = [3,3], target = 6
Output: [0,1]

Constraints:

2 <= nums.length <= 104
-109 <= nums[i] <= 109
-109 <= target <= 109
Only one valid answer exists

Brute Force

Example: nums = [6, 3, 2, 11, 7, 15], target = 10

```
static int[] BruteForce(int[] nums, int target)
{
    for (int j = 0; j < nums.length; j++) {

        for (int k = j + 1; k < nums.length; k++){

            if (nums[j] + nums[k] == target) {

                int[] solution = { nums[j], nums[k] };
                return solution;
            }
        }
    }

    return null;
}
```

$O(n^2)$

Hash Map / Hash Table / Dictionary

Uses a hash function to compute an index or *hash code* based on the key and stores the value into a corresponding bucket.

```
Java
HashMap<Integer, Integer> map = new HashMap<>();

map.put(key, value);
map.get(key);
map.containsKey(key);

.NET C#
Dictionary<int, int> dictionary = new Dictionary<int, int>();

dictionary.Add(key, value);
dictionary[key];
dictionary.ContainsKey(key);
```

O (1)

Hash Map

Example: nums = [6, 3, 2, 11, 7, 15], target = 10

```
public static int[] GetPair(int[] nums, int target)
{
    int[] pair = { -1, -1 };
    HashMap<Integer, Integer> map = new HashMap<>();
    for (int n = 0; n < nums.length; n++) {
        int val = nums[n];
        int diff = target - val;
        if (map.containsKey(diff)) {
            pair[0] = map.get(diff);
            pair[1] = n;
            break;
        }
        else if (!map.containsKey(val)) {
            map.put(val, n);
        }
    }

    return pair;
}
```

O (n)

Two Pointer Method

[0, 1, 2, 3, 4,   5]

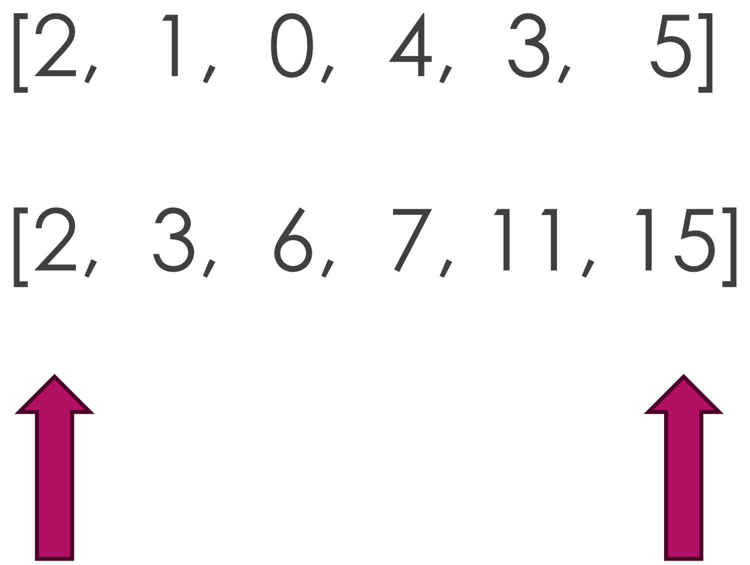
[6, 3, 2, 11, 7, 15]

Two Pointer Method

[2, 1, 0, 4, 3,   5]

[2, 3, 6, 7, 11, 15]

Two Pointer Method



[2, 1, 0, 4, 3, 5]

[2, 3, 6, 7, 11, 15]

Target: 10

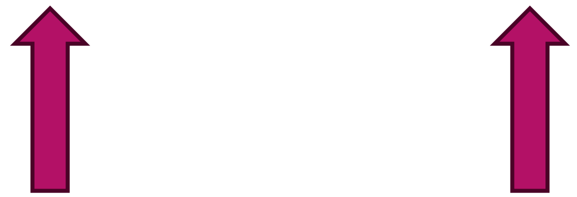
Sum: 17

Two Pointer Method

[2, 1, 0, 4, 3, 5]

[2, 3, 6, 7, 11, 15]

Target: 10

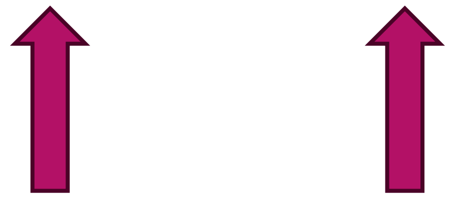


Sum: 13

Two Pointer Method

[2, 1, 0, 4, 3,   5]

[2, 3, 6, 7, 11, 15]



Target: 10

Sum: 9

Two Pointer Method

[2, 1, 0, 4, 3, 5]

[2, 3, 6, 7, 11, 15]

Target: 10

Sum: 10

O (n)

Indexes: 1, 4

Preferred Method

- Lowest Big O: both Hash Map and Two Pointer methods were O(n)

But I also care about:

- More Intuitive
- Easier to Code
- Quicker to Understand

=> More Maintainable

My Choice: using a Hash Map