

# CSE 332

## Data Abstractions

# Algorithm Analysis 1



```
public void run() {  
    //for (int i = 0; i < 1000000; i++) {  
        //doLongCalculation();  
        //anotherAnalysis();  
        //solvePNP();  
    //}  
    System.out.println("Done!");  
}
```

# Outline

1 Comparing Algorithms

2 Asymptotic Analysis

In 143, we asked:

**What does it mean to have an “efficient program”?**

```
1 System.out.println("hello");      vs.      1 System.out.print("h");
                                           2 System.out.print("e");
                                           3 System.out.print("l");
                                           4 System.out.print("l");
                                           5 System.out.println("o");
```

OUTPUT

```
>> left average run time is 1000 ns.
>> right average run time is 5000 ns.
```

**We’re measuring in NANoseconds!**

Both of these run **very very** quickly. The first is definitely better style, but it’s not “more efficient.”

## hasDuplicate

Given a **sorted int array**, determine if the array has a duplicate.

### Algorithm 1

For each **pair of elements**, check if they're the same.

### Algorithm 2

For each **element**, check if it's equal to the one after it.

## Why Not Time Programs?

Timing programs is prone to error (not **reliable** or **portable**):

- Hardware: processor(s), memory, etc.
- OS, Java version, libraries, drivers
- Other programs running
- Implementation dependent
- Can we even time an algorithm?

## hasDuplicate

Given a **sorted int array**, determine if the array has a duplicate.

### Example

```
public int stepsHasDuplicate1(int[] array) {
    int steps = 0;
    for (int i=0; i < array.length; i++) {
        for (int j=0; j < array.length; j++) {
            steps++; // The if statement is a step
            if (i != j && array[i] == array[j]) {
                return steps;
            }
        }
    }
    return steps;
}
```

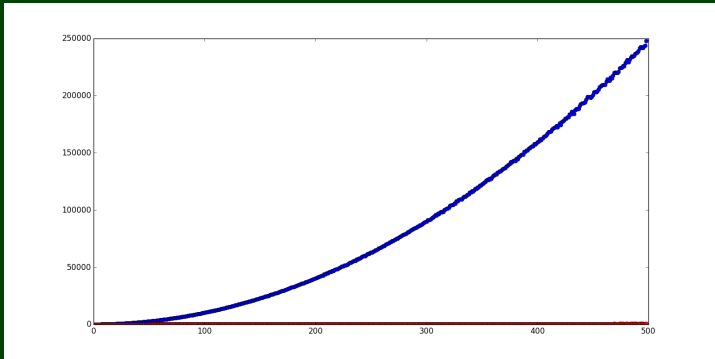
### OUTPUT

```
>> hasDuplicate1 average number of steps is 9758172 steps.
>> hasDuplicate2 average number of steps is 170 steps.
```

## Why Not Count Steps in Programs?

- Can we even count steps for an algorithm?
- We must do this via **testing**; so, we may miss worst-case input!
- We must do this via **testing**; so, we may miss best-case input!

Instead, let's try running on arrays of size 1, 2, 3, ..., 1000000, and plot:



## Why Not Plot Steps in Programs?

- Can we even count steps for an algorithm?
- We must do this via **testing**; so, we may miss worst-case input!
- We must do this via **testing**; so, we may miss best-case input!

We want to compare **algorithms**, not programs. In general, there are many answers (clarity, security, etc.). Performance (space, time, etc.) are generally among the most important.

- Only consider large inputs (any algorithm will work on 10)
- Answer will be independent of CPU speed, programming language, coding tricks, etc.
- Answer is general and rigorous, complementary to “coding it up and counting steps on some test cases”
- Can do analysis before coding!



## Basic Operations take “some amount of” Constant Time

- Arithmetic (fixed-width)
- Variable Assignment
- Access one Java field or array index
- etc.

(This is an approximation of reality: a very useful “lie”.)

## Complex Operations

**Consecutive Statements.** Sum of time of each statement

**Conditionals.** Time of condition +  $\max(\text{ifBranch}, \text{elseBranch})$

**Loops.** Number of iterations \* Time for Loop Body

**Function Calls.** Time of function’s body

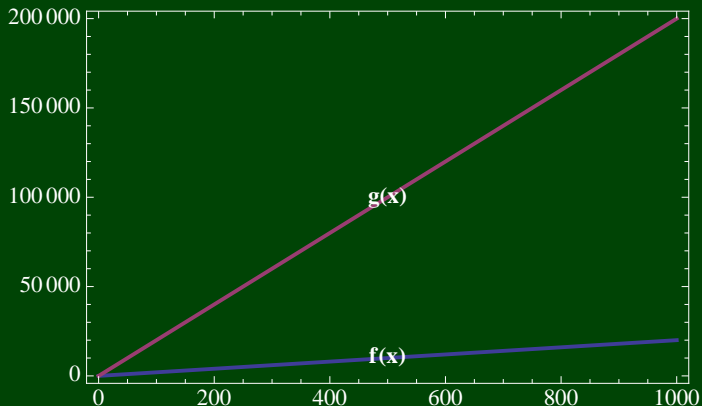
**Recursive Function Calls.** Solve Recurrence



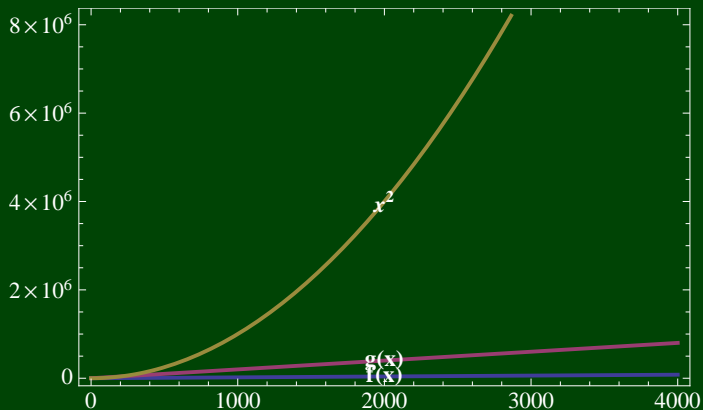
# Outline

1 Comparing Algorithms

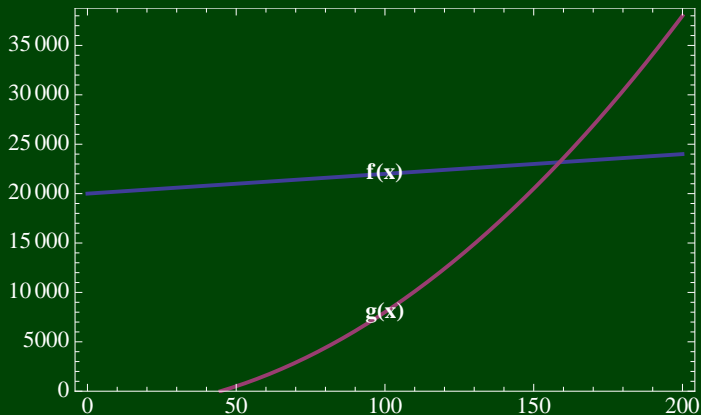
2 Asymptotic Analysis



Should we consider these “the same”?



Probably a good idea, since they seem to be growing at the same rate. For reference, the function that dwarfs them both is  $x^2$ .



Here's two functions,  $f(x)$  and  $g(x)$ . Ultimately,  $g(x)$  will grow much faster than  $f(x)$ , but at the beginning, it is smaller.

We'd like to be able to compare two functions. Intuitively, we want an operation like " $\leq$ " (e.g.  $4 \leq 5$ ), but for functions.

If we have  $f$  and  $4f$ , we should consider them the same:

$f \leq g$  when...

$f \leq cg$  where  $c$  is a constant and  $c \neq 0$ .

We also care about **all values of the function** that are **big enough**:

$f \leq g$  when...

For all  $n$  "large enough",  $f(n) \leq cg(n)$ , where  $c \neq 0$

For some  $n_0 \geq 0$ , for all  $n \geq n_0$ ,  $f(n) \leq cg(n)$ , where  $c \neq 0$

For some  $c \neq 0$ , for some  $n_0 \geq 0$ , for all  $n \geq n_0$ ,  $f(n) \leq cg(n)$

Definition (Big-Oh)

We say a function  $f : A \rightarrow B$  is **dominated by** a function  $g : A \rightarrow B$  when:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \leq cg(n)$$

Formally, we write this as  $f \in \mathcal{O}(g)$ .

True or False?

(1)  $4 + 3n \in \mathcal{O}(n)$

(2)  $4 + 3n = \mathcal{O}(1)$

(3)  $4 + 3n$  is  $\mathcal{O}(n^2)$

(4)  $n + 2\log n \in \mathcal{O}(\log n)$

(5)  $\log n \in \mathcal{O}(n + 2\log n)$



True or False?

(1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )

(2)  $4 + 3n = \mathcal{O}(1)$

(3)  $4 + 3n$  is  $\mathcal{O}(n^2)$

(4)  $n + 2\log n \in \mathcal{O}(\log n)$

(5)  $\log n \in \mathcal{O}(n + 2\log n)$

True or False?

(1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )

(2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )

(3)  $4 + 3n$  is  $\mathcal{O}(n^2)$

(4)  $n + 2\log n \in \mathcal{O}(\log n)$

(5)  $\log n \in \mathcal{O}(n + 2\log n)$

True or False?

(1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )

(2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )

(3)  $4 + 3n$  is  $\mathcal{O}(n^2)$  True: ( $n \leq n^2$ )

(4)  $n + 2\log n \in \mathcal{O}(\log n)$

(5)  $\log n \in \mathcal{O}(n + 2\log n)$

True or False?

- (1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )
- (2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )
- (3)  $4 + 3n$  is  $\mathcal{O}(n^2)$  True: ( $n \leq n^2$ )
- (4)  $n + 2\log n \in \mathcal{O}(\log n)$  False: ( $n \gg \log n$ )
- (5)  $\log n \in \mathcal{O}(n + 2\log n)$

True or False?

- (1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )
- (2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )
- (3)  $4 + 3n$  is  $\mathcal{O}(n^2)$  True: ( $n \leq n^2$ )
- (4)  $n + 2\log n \in \mathcal{O}(\log n)$  False: ( $n \gg \log n$ )
- (5)  $\log n \in \mathcal{O}(n + 2\log n)$  True: ( $\log n \leq n + 2\log n$ )

## True or False?

- (1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )
- (2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )
- (3)  $4 + 3n$  is  $\mathcal{O}(n^2)$  True: ( $n \leq n^2$ )
- (4)  $n + 2\log n \in \mathcal{O}(\log n)$  False: ( $n \gg \log n$ )
- (5)  $\log n \in \mathcal{O}(n + 2\log n)$  True: ( $\log n \leq n + 2\log n$ )

## Big-Oh Gotchas

- $\mathcal{O}(f)$  is a **set!** This means we should treat it as such.
- If we know  $f(n) \in \mathcal{O}(n)$ , then it is also the case that  $f(n) \in \mathcal{O}(n^2)$ , and  $f(n) \in \mathcal{O}(n^3)$ , etc.
- Remember that small cases, really don't matter. As long as it's **eventually** an upper bound, it fits the definition.

## True or False?

- (1)  $4 + 3n \in \mathcal{O}(n)$  True ( $n = n$ )
- (2)  $4 + 3n = \mathcal{O}(1)$  False: ( $n \gg 1$ )
- (3)  $4 + 3n$  is  $\mathcal{O}(n^2)$  True: ( $n \leq n^2$ )
- (4)  $n + 2\log n \in \mathcal{O}(\log n)$  False: ( $n \gg \log n$ )
- (5)  $\log n \in \mathcal{O}(n + 2\log n)$  True: ( $\log n \leq n + 2\log n$ )

## Big-Oh Gotchas

- $\mathcal{O}(f)$  is a **set!** This means we should treat it as such.
- If we know  $f(n) \in \mathcal{O}(n)$ , then it is also the case that  $f(n) \in \mathcal{O}(n^2)$ , and  $f(n) \in \mathcal{O}(n^3)$ , etc.
- Remember that small cases, really don't matter. As long as it's **eventually** an upper bound, it fits the definition.

Okay, but we haven't actually shown anything. Let's **prove** (1) and (2).

## Definition (Big-Oh)

We say a function  $f : A \rightarrow B$  is **dominated by** a function  $g : A \rightarrow B$  when:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \leq cg(n)$$

Formally, we write this as  $f \in \mathcal{O}(g)$ .

We want to prove  $4 + 3n \in \mathcal{O}(n)$ . That is, we want to prove:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). 4 + 3n \leq cn$$

## Proof Strategy

- Choose a  $c, n_0$  that work.
- Prove that they work for all  $n \geq n_0$ .

## Proof

Choose  $c = 5$  and  $n_0 = 5$ . Then, note that  $4 + 3n \leq 4n \leq 5n$ , because  $n \geq 5$ . It follows that  $4 + 3n \in \mathcal{O}(n)$ .



## Definition (Big-Oh)

We say a function  $f : A \rightarrow B$  is **dominated by** a function  $g : A \rightarrow B$  when:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \leq cg(n)$$

Formally, we write this as  $f \in \mathcal{O}(g)$ .

We want to prove  $4 + 3n + 4n^2 \in \mathcal{O}(n^3)$ .

## Scratch Work

We want to choose a  $c$  and  $n_0$  such that  $4 + 3n + 4n^2 \leq cn^3$ . So, manipulate the equation:

$$4 + 3n + 4n^2 \leq 4n^3 + 3n^3 + 4n^3 = 11n^3$$

For this to work, we need  $4 \leq 4n^3$  and  $3n \leq 3n^3$ .  $n \geq 1$  satisfies this.

## Proof

Choose  $c = 11$  and  $n_0 = 1$ . Then, note that  $4 + 3n + 4n^2 \leq 4n^3 + 3n^3 + 4n^3 = 11n^3$ , because  $n \geq 1$ . It follows that  $4 + 3n + 4n^2 \in \mathcal{O}(n^3)$ .

## Definition (Big-Oh)

We say a function  $f : A \rightarrow B$  is **dominated by** a function  $g : A \rightarrow B$  when:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \leq cg(n)$$

Formally, we write this as  $f \in \mathcal{O}(g)$ .

## Definition (Big-Omega)

We say a function  $f : A \rightarrow B$  **dominates** a function  $g : A \rightarrow B$  when:

$$\exists(c, n_0 > 0). \forall(n \geq n_0). f(n) \geq cg(n)$$

Formally we write this as  $f \in \Omega(g)$ .

## Definition (Big-Theta)

We say a function  $f : A \rightarrow B$  **grows at the same rate as** a function  $g : A \rightarrow B$  when:  $f \in \mathcal{O}(g)$  and  $f \in \Omega(g)$

Formally we write this as  $f \in \Theta(g)$ .

**Important:** You need not use the same  $c$  value for  $\mathcal{O}$  and  $\Omega$  to prove  $\Theta$ .

True or False?

(1)  $4 + 3n \in \Theta(n)$

(2)  $4 + 3n$  is  $\Theta(n^2)$

True or False?

(1)  $4 + 3n \in \Theta(n)$  True

(2)  $4 + 3n$  is  $\Theta(n^2)$

True or False?

(1)  $4 + 3n \in \Theta(n)$  True

(2)  $4 + 3n$  is  $\Theta(n^2)$  False

True or False?

(1)  $4 + 3n \in \Theta(n)$  True

(2)  $4 + 3n$  is  $\Theta(n^2)$  False

If you want to say “ $f$  is a tight bound for  $g$ ”, **do not use**  $\mathcal{O}$ —use  $\Theta$ .

Remember, we're analyzing the **worst case time!** What else can we analyze?

- Space?
- Average Case?
- Best Case?
- Time **over multiple operations?**

Which is Better?

$n^{1/10}$  or  $\log n$

- $\log n$  grows more slowly (Big-Oh)
- ... But the cross-over point is around  $5 \times 10^{17}$



- There are many ways to compare algorithms
- Understand formal Big-Oh, Big-Omega, Big-Theta
- Be able to prove any of these