



# Lecture 4: Introduction to Asymptotic Analysis

CSE 373: Data Structures and Algorithms



# Warm Up

# Administrivia

Fill out class survey

Find a partner by Thursday!

Meet our lovely TAs





# Algorithm Analysis

---

# Code Analysis

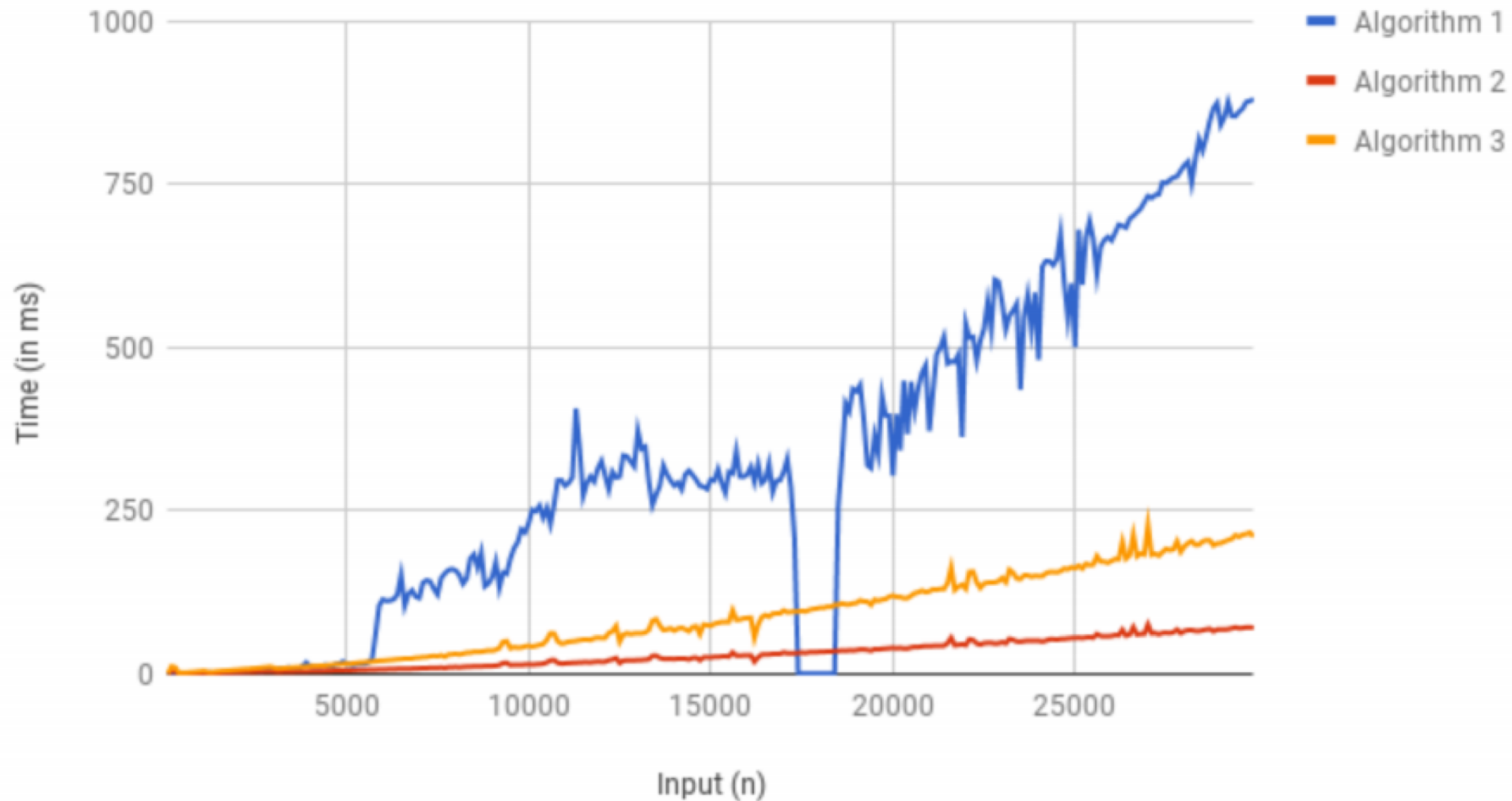
How do we compare two pieces of code?

- Time needed to run 
- Memory used 
- Number of network calls
- Amount of data saved to disk
- Specialized vs generalized
- Code reusability
- Security

# Which is the best algorithm?

Algorithm	Runtime (in ms)
Algorithm 1	1
Algorithm 2	30
Algorithm 3	100

# Which is the best algorithm?



# When choosing an algorithm

consider overall trends as inputs increase

- Computers are fast, small inputs don't differentiate
- Must consider what happens with large inputs

Estimate final result independent of incidental factors

CPU speed, programming language, available computer memory

Identify trends without investing in testing

Predict worst case scenarios



# How do we compare across scenarios?

Worst case analysis

What is the longest or most expensive this algorithm can run

Best case analysis

What happens if the stars align

Average case analysis

Runtime analysis across all possible inputs/states

Can be very difficult to not bias the answer

We usually consider worst case

# Review: Sequential Search

**sequential search:** Locates a target value in an array / list by examining each element from start to finish.

- Example: Searching the array below for the value **42**:

index	0	1	2	3	4	5	6	7	8	9	<b>10</b>	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	<b>42</b>	50	56	68	85	92	103

i

↑

How many elements will be examined?

- What is the best case?                      First element examined
- What is the worst case?                      Last element examined
- What is the complexity class?     $O(n)$

# Asymptotic Analysis

Definition - TODO

Two step process

1. Model – reduce code run time to a mathematical relationship with number of inputs
2. Analyze – compare runtime/input relationship across multiple algorithms

# Review: Binary Search

**binary search:** Locates a target value in a *sorted* array or list by successively eliminating half of the array from consideration.

- Example: Searching the array below for the value **42**:

index	0	1	2	3	4	5	6	7	8	9	<b>10</b>	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	<b>42</b>	50	56	68	85	92	103

↑  
min

↑  
mid

↑  
max

How many elements will be examined?

- What is the best case?      First element examined
- What is the worst case?      Last element examined
- What is the complexity class?    $\log_2(n)$

# Analyzing Binary Search

What is the pattern?

- At each iteration, we eliminate half of the remaining elements

How long does it take to finish?

- 1<sup>st</sup> iteration –  $N/2$  elements remain
- 2<sup>nd</sup> iteration –  $N/4$  elements remain
- Kth iteration -  $N/2^k$  elements remain

index	0	1	2	3	4	5	6	7	8	9	<b>10</b>	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	<b>42</b>	50	56	68	85	92	103

# Analyzing Binary Search

Finishes when  $N / 2^K = 1$

$$N / 2^K = 1$$

-> multiply right side by  $2^K$

$$N = 2^K$$

-> isolate K exponent with logarithm

$$\log_2 N = k$$

Is this exact?

- N can be things other than powers of 2
- If N is odd we can't technically use  $\log_2$
- When we have an odd number of elements we select the larger half
- Within a fair rounding error



# Code Modeling

Definition – todo

What counts?

Consecutive statements

- Sum time of each statement

Function calls

- Count runtime of function body

Conditionals

- Time of test + worst case scenario branch

Loops

- Number of iterations of loop body

Assumptions

Assume basic operations take same constant time

- Adding ints or doubles
- Variable assignment
- Variable update
- Return statement
- Accessing array index or object field

# Modeling Case Study

**Goal:** return 'true' if a sorted array of ints contains duplicates

**Solution 1:** compare each pair of elements

```
public boolean hasDuplicate1(int[] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array.length; j++) {
            if (i != j && array[i] == array[j]) {
                return true;
            }
        }
    }
    return false;
}
```

**Solution 2:** compare each consecutive pair of elements

```
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {
        if (array[i] == array[i + 1]) {
            return true;
        }
    }
    return false;
}
```

# Modeling Case Study: Solution 2

$T(n)$  where  $n = \text{array.length}$

-> work inside out

**Solution 2:** compare each consecutive pair of elements

```
public boolean hasDuplicate2(int[] array) {  
    for (int i = 0; i < array.length - 1; i++) {  
        if (array[i] == array[i + 1]) {  
            return true;  
        }  
    }  
    return false;  
}
```

**Annotations:**

- x n - 1**: Multiplier for the loop body.
- +4**: Cost of the `if` statement body.
- +1**: Cost of the `for` loop increment and condition check.
- If statement = 5**: Total cost of the `if` statement (body + overhead).
- +1**: Cost of the final `return` statement.

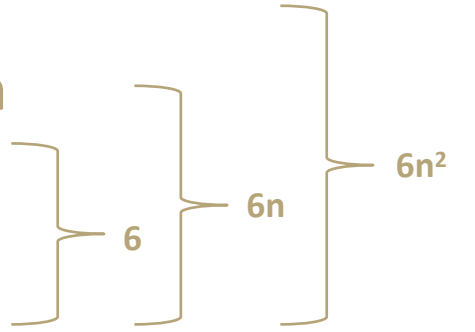
$$T(n) = 5(n-1) + 1$$

linear time complexity class  $O(n)$

# Modeling Case Study: Solution 1

**Solution 1:** compare each consecutive pair of elements

```
public boolean hasDuplicate1(int[] array) {  
    for (int i = 0; i < array.length; i++) { x n  
        for (int j = 0; j < array.length; j++) { x n  
            if (i != j && array[i] == array[j]) { +5  
                return true; +1  
            }  
        }  
    }  
    return false; +1  
}
```



$$T(n) = 6n^2 + 1$$

quadratic time complexity class  $O(n^2)$