Lecture 4: Introduction to Asymptotic Analysis

CSE 373: Data Structures and Algorithms

5 Minutes

Warm Up

Read through the code on the worksheet given

Come up with a test case for each of the described test categories

Expected Behavior add(1)

Forbidden Input add(null)

Empty/Null Add into empty list

Boundary/Edge Add enough values to trigger internal array double and copy

Scale Add 1000 times in a row

Socrative:

www.socrative.com

Room Name: CSE373

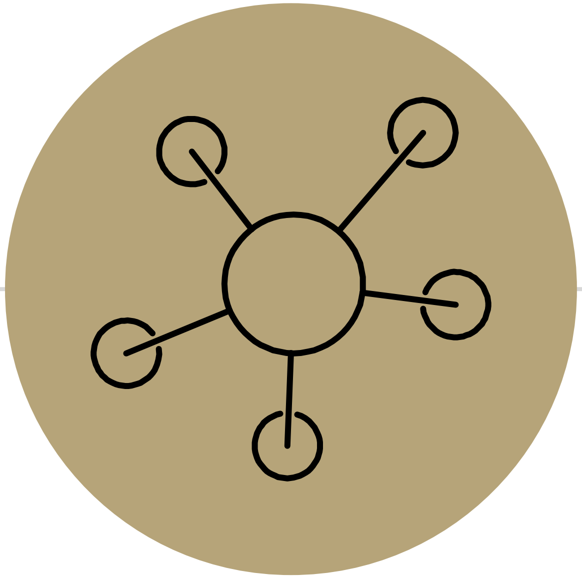
Please enter your name as: Last, First

Administrivia

Fill out class survey

Find a partner by Thursday!

143 Review TONIGHT - SIEG 134 6:00pm



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

Comparing Algorithms with Mathematical Models

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

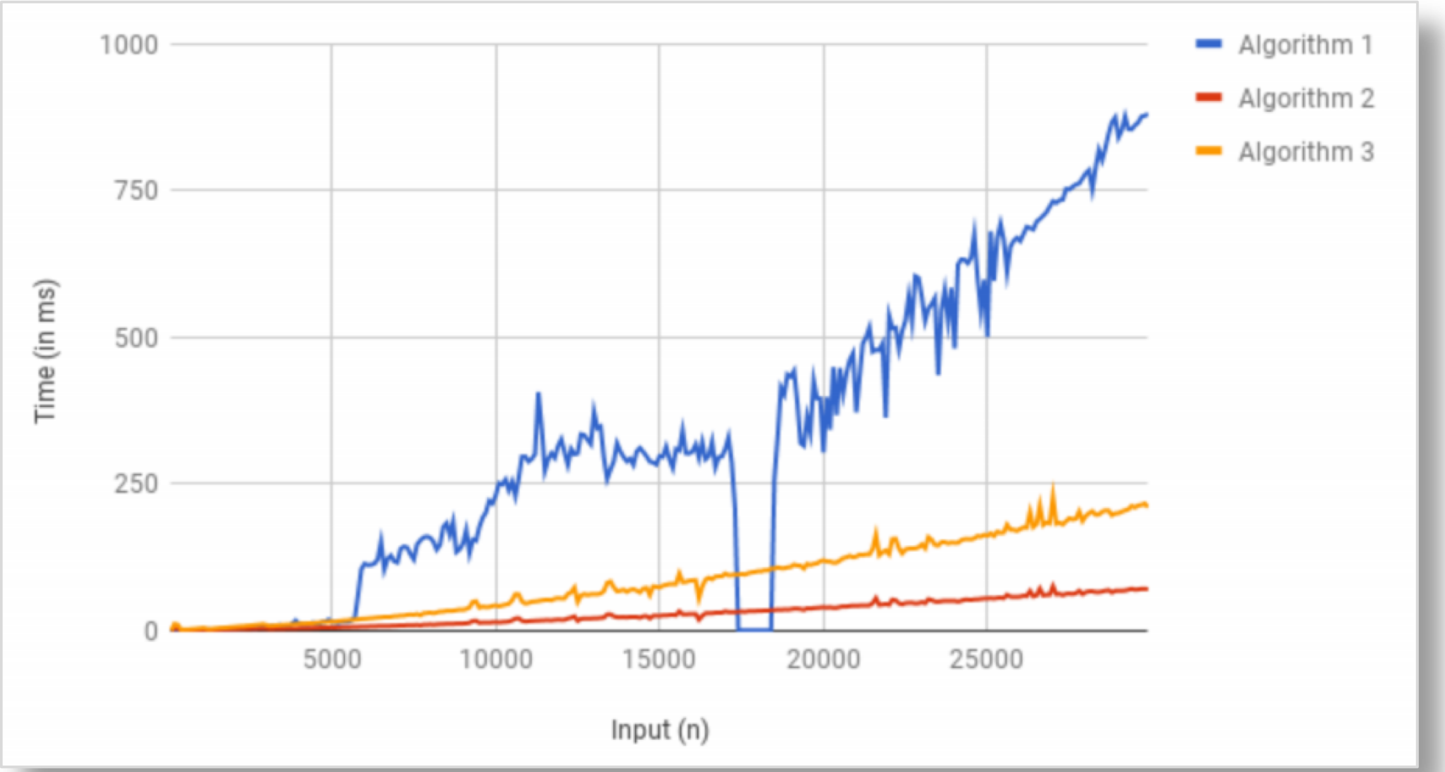
Model performance across multiple possible scenarios
- Worst case - what is the most expensive or least performant an operation can be
- Average case – what functions are most likely to come up?
- Best case – if we understand the ideal conditions can increase the likelihood of those conditions?

Identify trends without investing in testing

Which is the best algorithm?

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

- Does this apply to the same number of inputs?
- Are we going to pass in the same number of inputs on each run?



1 Minute

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	**10**	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	**42**	50	56	68	85	92	103

i

How many elements will be examined?

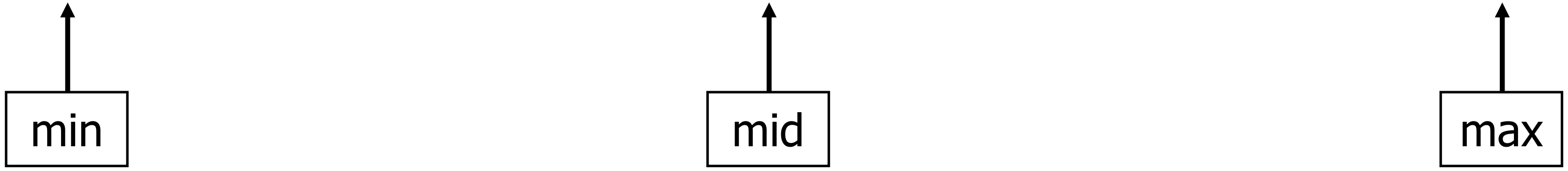
- What is the best case? First element examined, index 0
- What is the worst case? Last element examined, index 16
  Or item not in array
- What is the complexity class? O(n)

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	**10**	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	**42**	50	56	68	85	92	103



How many elements will be examined?

- What is the best case? First element examined, index 8
- What is the worst case? Last element examined, index 9
  Or item not array
- 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?

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

Finishes when $\frac{n}{2^k} = 1$

$$\frac{n}{2^k} = 1$$

-> multiply right side by 2^K

$N = 2^K$

-> isolate K exponent with logarithm

$\log_2 N = k$

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

Asymptotic Analysis

asymptotic analysis – the process of mathematically representing runtime of a algorithm in relation to the number of inputs and how that relationship changes as the number of inputs grow

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

Code Modeling

Code Modeling

code modeling – the process of mathematically representing how many operations a piece of code will run in relation to the number of inputs n

Examples:

- Sequential search $f(n) = n$
- Binary search $f(n) = log_2 n$

What counts as an "operation"?

Assume all operations run in equivalent time

Basic operations

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

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 x runtime of loop body

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

Goal: produce mathematical function representing runtime $f(n)$ where n = array.length

Solution 2: compare each consecutive pair of elements

```
public boolean hasDuplicate2(int[] array) {
    for (int i = 0; i < array.length - 1; i++) {   loop = (n – 1)(body)
        if (array[i] == array[i + 1]) { +4
            return true; +1                          If statement = 5
        }
    }
    return false;
}
                    +1
```

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

linear -> O(n)

Approach

-> start with basic operations, work inside out for control structures

- Each basic operation = +1
- Conditionals = worst case test operations + branch
- Loop = iterations (loop body)

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
}
```

6 | 6n | $6n^2$

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

quadratic -> $O(n^2)$

Approach

-> start with basic operations, work inside out for control structures

- Each basic operation = +1
- Conditionals = worst case test operations + branch
- Loop = iterations (loop body)

5 Minutes

Your turn!

Write the specific mathematical code model for the following code and indicate the big o runtime.

```
public void foobar (int k) {
    int j = 0; +1
    while (j < k) { +k/5 (body)
        for (int i = 0; i < k; i++) { +k(body)
            System.out.println("Hello world"); +1
        }
        j = j + 5; +2
    }
}
```

$$f(k) = \frac{(k+2)}{5}$$

linear -> O(n)

Approach

-> start with basic operations, work inside out for control structures

- Each basic operation = +1
- Conditionals = worst case test operations + branch
- Loop = iterations (loop body)