LEC 07

CSE 123

Runtime Analysis

Questions during Class?

Raise hand or send here

sli.do #cse123A

BEFORE WE START

Talk to your neighbors:

What was the latest Youtube rabbit hole you went down?

Instructors: Nathan Brunelle

TAs:	Arohan	Ashar	Neha	Rohini	Rushil
	Ido	Zachary	Sebastian	Joshua	Sean
	Hayden	Caleb	Justin	Heon	Rashad
	Srihari	Benoit	Derek	Chris	Bhaumik
	Kuhu	Kavya	Cynthia	Shreya	Ashley
	Ziao	Kieran	Marcus	Crystal	Eeshani
	Prakshi	Packard	Cora	Dixon	Nichole
	Niyati	Trien	Lawrence	Evan	Cady

Announcements

- Resubmission Period 1 closes tonight at 11:59pm
- Programming Project 1 out now, due May 7 at 11:59pm
- Quiz 0 grades released sometime next week
  - After makeups are complete, before Quiz 1

Runtime Analysis

- What's the "best" way to write code?
  - Depends on how you define best: Code quality, memory usage, speed, etc.

- Runtime = most popular way of analyzing solutions
  - Slow code = bad for business
- How do we figure out how long execution takes?
  - Stopwatch = human error
  - Computers = computer error (artifacts, operating systems, language)
  - Need a way to formalize abstractly…

Runtime Analysis

- We'll count simple operations as 1 unit
  - variable initialize / update `int x = 0;`
  - array accessing `arr[0] = 10;`
  - conditional checks `if (x < 10) {`

- Goal: determine how the number of operations scales w/ input size
  - Don't care about the difference between 2 and 4
  - Find the appropriate **complexity class**

- Result: evaluation tactic independent of OS, language, compiler, etc.
  - Simple operation = constant regardless of if it is truly 1

Complexity Classes

- Input will always be an array `arr` of length *n*
- Constant (1)
  - # Ops doesn't relate to *n*

```
return arr[0];
```

- Linear (n)
  - # Ops proportional to *n*

```
for (int i = 0; i < arr.length; i++)
```

- Quadradic (n^2)
  - # Ops proportional to *n^2*

```
for (int j = 0; j < arr.length; j++)
    for (int j = 0; j < arr.length; j++)
```

- Lets say # Ops = n^2 + 100000n
  - If n was really, really, really big, which term matters more?
  - Only care about the **dominating term** for complexity!

Complexity Classes

What's the complexity class of the following?

```
public static void mystery(int[] arr) {
    if (arr.length == 0) {                    1
        throw new IllegalArgumentException();
    }                                         2
    return arr[arr.length - 1];               1
}
```

Constant Complexity (1)

Complexity Classes

What's the complexity class of the following?

```
public static int mystery(int[] arr) {
    int sum = 0;                          1
    for (int i = 0; i < arr.length; i++) {
        sum += arr[i];                    3
    }                                     3n
    return sum;                           1
}                                         3n + 2
```

Linear Complexity (n)

Complexity Classes

What's the complexity class of the following?

```
public static int mystery(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr.length; j++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

n(2n + 1)
= 2n^2 + n

2n

2

1

Quadratic Complexity (n^2)

Complexity Classes

What's the complexity class of the following?

```
public static int mystery(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        for (int j = i; j < arr.length; j++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
}
```

Quadratic Complexity (n^2)

Big-Oh Notation

- Programmers… are pessimists (or maybe realists)
  - Case in point: dominating term
- In the real world, best-case complexity isn't super useful
  - Want to make sure solutions work well in the worst possible situations

- We use Big-Oh notation to demonstrate worst-case complexity!

```
public static int indexOf(int[] arr, int x) {
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == x) return i;
    }
    return -1;
}
```

Worst-case linear

<u>O(n)</u>

ArrayList vs LinkedList

Operation	ArrayIntList	LinkedIntList
`size()`	O(1)	O(n)
`get(index)`	O(1)	O(n)
`add(val)`	O(1)	O(n)
`add(0, val)`	O(n)	O(1)
`add(index, val)`	O(n)	O(n)
`remove(0)`	O(n)	O(1)
`remove(n-1)`	O(1)	O(n)
`remove(index)`	O(n)	O(n)

How should we implement a stack?

- With an `ArrayIntList`?
  - push = what?
  - pop = what?

- With a `LinkedIntList`?
  - push = what?
  - pop = what?

Is running time an implementation detail?

- Yes
- No

- Does that help? :D