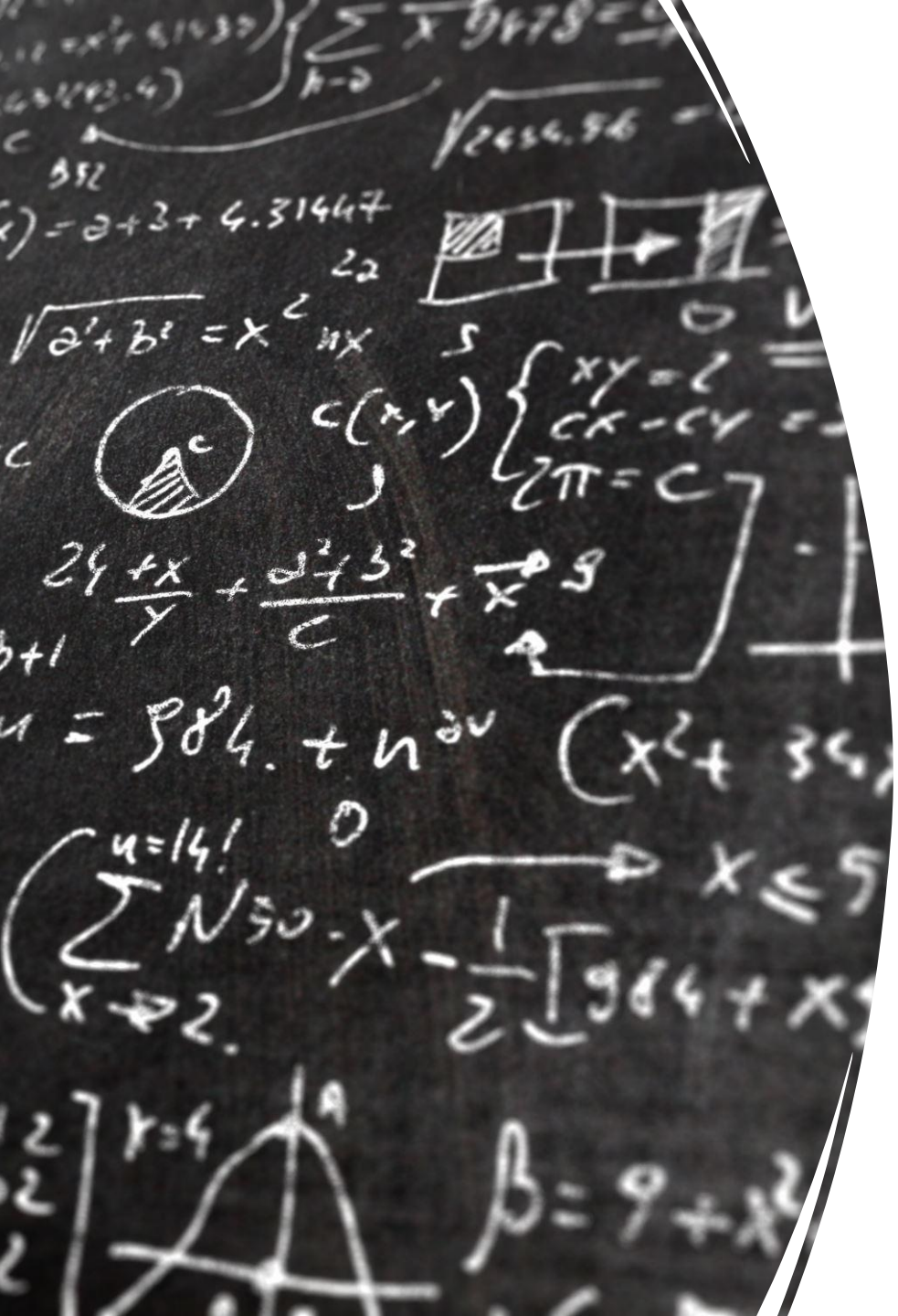

CS 113 – Computer Science I

Lecture 27 – Final Exam Review

5/2/2024

Announcements

Lab 9, Lab 10, Lab 11, HW9 due May 7th

Friday OH: 11-2pm Park 205

Exam Format

- Cumulative
- 180min
- 2 8.5/11in cheat sheets allowed (front and back)
- Format: 125 total points
  - 20 points T/F questions
  - 34 points short answer
  - 6 points reading and understanding code
  - 65 points programming

Searching, Sorting, and Runtime Complexity

Runtime Analysis: Big O Notation

- Mathematical notation used to describe the performance or complexity of an algorithm.
- Hardware independent
- Represents the upper bound of the time complexity in the **worst-case scenario.**
- Helps us understand how the runtime of an algorithm grows ***as the input size increases.***

Runtime Complexity

Sort these from fastest to slowest:

- O(n)
- O(n^2)
- O(logn)
- O(1)
- O(2^n)

Searching

- Linear Search
  - Best case?
  - Worst case?

- Binary Search
  - Best case?
  - Worst case?

Searching

[5, 10, 17, 22, 26, 40, 50, 100]

1. Perform a **linear search** for the element 50
   a. How many elements did we check?

1. Perform a **binary search** for the element 50 and and show each step
   b. how many elements did we check?

Searching

[5, 10, 17, 22, 26, 40, 50, 100]

1. Perform a **linear search** for the element 5
   a. How many elements did we check?

1. Perform a **binary search** for the element 5 and and show each step
   b. how many elements did we check?

Searching

Is binary search *always* faster than linear search?

No! Big-O notation is an analysis of the worst case.

In some cases, a linear search will be faster.

Sorting

Show each step of sorting the following list:

[12, 35, 78, 21, 93, 73, 8, 66]

1. Selection Sort
2. Bubble Sort

Sorting

- Selection Sort
  - runtime complexity?

- Bubble Sort
  - runtime complexity?

Big-O Example 1

```
int n = Integer.parseInt(args[0]);
int power = 1;
while (power < n) {
    System.out.print(power + " ");
    power *= 2;
}
```

How does the runtime grow as a function of the input size?

O(logn)

Big-O Example 2

```
int fetchFirstElement(int[] arr) {
    return arr[0];
}
```

How does the runtime grow as a function of the size of arr?

O(1)

Big-O Example 3

```
int n = Integer.parseInt(args[0]);
int tot = 0;
int i = 0;

while (i < n) {
   tot = tot * i;
   i++;

    for (int j=0; j<10000; j++) {
       System.out.println("hello");
    }
}
```

How does the runtime grow as a function of the input size?

Linearly!

O(n)

Big-O Example 4

```
int n = Integer.parseInt(args[0]);

for (int i = 0; i >(-1*n); i--) {
    for (int j = 0; j < n; j++) {
       System.out.println(i, j);
    }
}
```

How does the runtime grow as a function of the input size?

Quadratically!

O(n^2)

We do n operations n times

Big-O Example 5

```
String[] lst =
    {“19”, “12”, “20”, “15”};

for (int i=0; i<100; i++)  {
    System.out.println(getNum(lst));
}

int getNum(int[] arr) {
    return Integer.parseInt(arr[0]);
}
```

How does the runtime grow as a function of the size of lst?

Constant! The runtime is not affected by the number of elements in `lst`

O(1)

Big-O Example 6

```
int[] lst = {1,2,3,4,5,6,7};

for (int i=0; i<lst.length; i++)  {
    findMax(lst);
}

int findMax(int[] arr) {
    int max = Integer.MIN_VALUE;
    for (int i=0; i<arr.length; i++) {
        if (arr[i] > max) {
            max =  arr[i];
        }
    }
    return max;
}
```

How does the runtime grow as a function of the size of lst?

O(n^2)

Programming Questions

Q1 - Problem Solving, Recursion, and Loops

Write a function called "`numOccurs`(int[] a, int[] b)". The function should determine how many times the elements of a occur in b. You can assume that both arrays will not be empty.

1. Write it recursively
2. Write it with a loop

Q2 - Classes, OOP, Arrays of Objects

Testing you on:

1. How to initialize an array as an instance variable
   a. What size should I make it?

1. How to deal with dynamically sized arrays
   b. What if its full when I try to add to it?

1. Make sure to avoid NPEs

1. How and when to use inheritance

Q2 - Classes, OOP, Arrays of Objects

Design and Implement a class that represents a Team. The team should have Players each with a name. Players can either be Offense, Defense, or Coaches. The Team class should support the following operations:

1. add: takes a player and adds them to the team
   a. There is a max capacity of 2 players of each position (offense, defense)
   b. Only one coach is allowed
2. trade: remove the player from the team
3. getOffense: returns a list of offensive players
4. getCoach: returns the coach's name

Q2 - Classes, OOP, Arrays of Objects

Testing you on:

1. How to initialize an array as an instance variable
   a. What size should I make it?

1. How to deal with dynamically sized arrays
   b. What if its full when I try to add to it?

1. Make sure to avoid NPEs

1. How and when to use inheritance

Q3 - Problem Solving, Arrays of Arrays

Write a function called "`getPerim`(int[][] a)". The function should return the an int[] of the perimeter values of a.

1	7
35	1

[1,7,35,1]

1	7	4
3	15	2
9	-1	6
35	1	3

[1,7,4,3,2,9,6,35,1,3]

Q4 -Problem Solving, Runtime Complexity, Loops

Write a method called `uniqueElements()` t**hat takes in an array of integers and returns the number of unique elements from the original array**.

To receive full credit, your solution's **runtime must be O(n)** Partial credit will be given for less efficient solutions. **You may use additional data structures if needed.**

Q5 - dynamic array size

Write a method called `maxBoard()` **that takes in a filename** and generates a 2D array filled with ‘O’s.

The file contents will contain two row. The size of the 2D array should be [x][y] where x is the max value in the first row and y is the max value in the second row