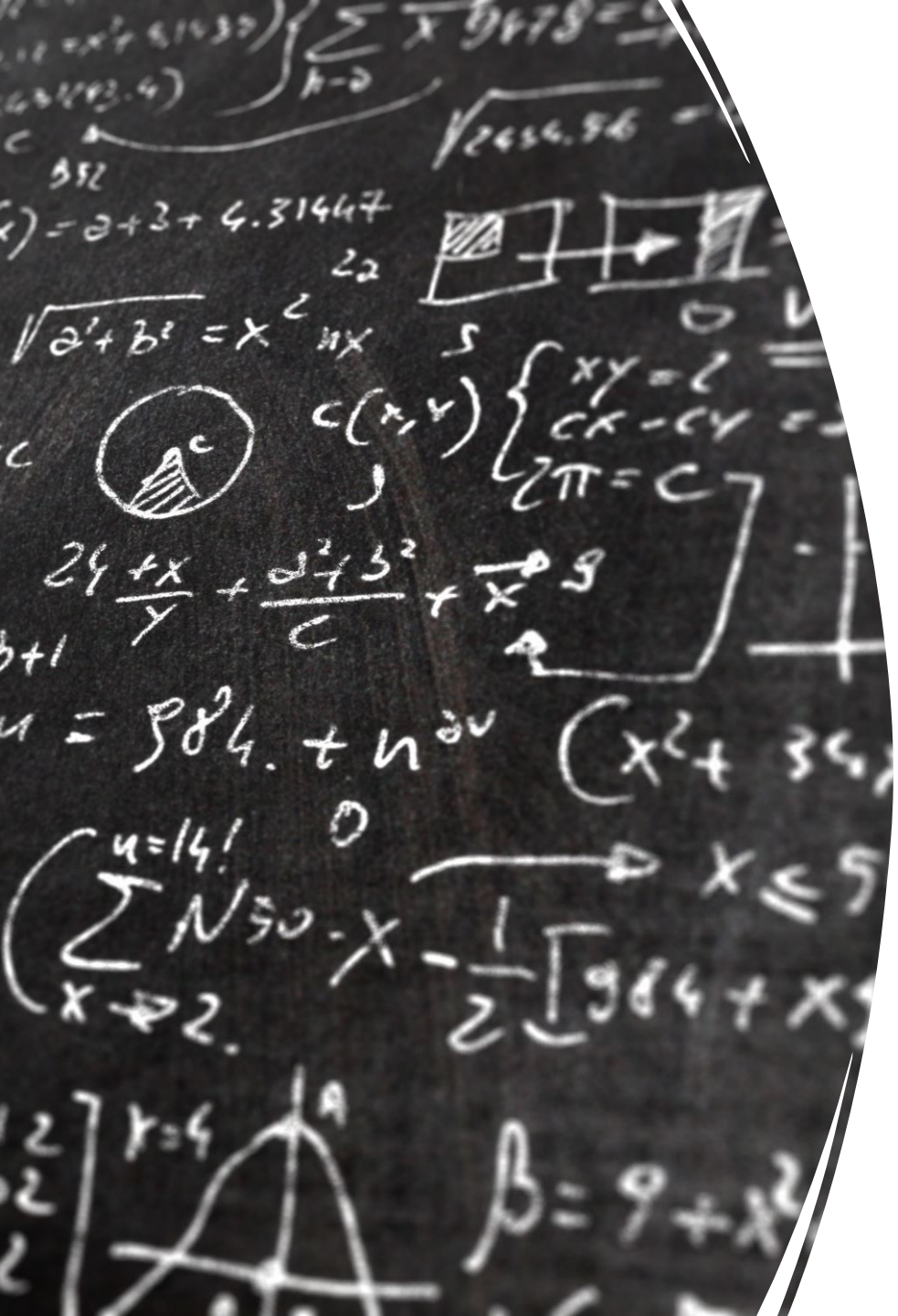

CS 113 – Computer Science I

Lecture 27 – Review

Adam Poliak

12/14/2023

Announcements

HW10:

- Due tonight

Adam: Office Hours

- Next week, TBA

Updated HW grading policy:

- Dropping lowest **2** homeworks

Topics

Terminal commands, vim, directory structure
variables (int, double, char, bool, string, array), expressions
Methods
Conditionals
Recursion
Loops
Strings
Arrays
Frame diagrams
Object Oriented Programming
Searching
Sorting
Runtime Analysis

Variables & Expressions

Variables as containers. Variables have:

- A name
- Location
- Data

Expressions

- A combination of variables, operators, and values that represents a single value. Expressions also have types, as determined by their operators and operands.
- Operands vs Operators

Methods

Methods have:

- Signature
  - Name
  - Parameters
  - Return Type
- Body

Parameters vs arguments

Keep track of methods (and order of methods) on the Method Stack in Frame Diagrams

Static vs instance vs abstract

Conditionals

- Conditional Statements allow our code to react based on conditions
- Check conditions using Boolean expressions
- If/else if/else

Recursion

Recursion as breaking down problems into simple problems and punting the rest of the problem down to someone else

Base case

Recursive step

Loops

- Idea: block of code that executes repetitively
- Differences between while and for loops

Arrays and Strings

Arrays as single variable to contain a list of similar items

Accessing items from and inserting items into an array

Resizing an array

Strings as array of characters

- String methods

Frame Diagrams

- Keep track of code execution
- Function Stack
- Object Stack
  - If two variables point to the same object and we change the object, the change occurs in both variables

Object Oriented Programming

Classes vs objects

Designing classes

Mutable vs immutable objects

Instance vs static vs abstract methods

Relationship between classes

- Inheritance
- Interfaces

Classes vs Objects

Class:

- custom data types that contains
  - the data (**instance variables**)
  - the operations (**instance methods**)

Object:

- an instance of the class

Example:

- String vs “hello world”

Designing classes

All classes should have:

- Constructor:
  - Difference between value and empty constructor
- Getters/accessors
- Comparators (equal() or compareTo()) – zoom poll
- toString()
- Setters
  - We'll see an example later where we wont want to have setters

Access modifiers

Instance variables and methods can be

`private`

Can’t be accessed directly by anyone else

`protected`

Only subclasses can access these

`public`

Anyone that has access to the object can access these

Mutable vs immutable objects

Whether data stored inside an object can change **(mutable**) or cannot change (**immutable**) once the object is created

Strings are **immutable**

Arrays are **mutable**

How would we design an **immutable** object

make instance variables `private`

do not include any setters

Static vs instance methods

Static

- Do not require an object
- no access to `this` keyword
- Examples:
  - Integer.parseInt(“99”);
  - Math.random();

Instance

- Acts on an object -> requires an objects
- has access to `this` keyword
- Examples:
  - “hello,world”.split(“,”)

Abstract methods

Contains method signatures: name, arguments, and return type

Does not include an implementation

Specify what a method does, not how it does it

Often used in interfaces

Each subclass that implements the interface can choose how to implement the method

Class relationships - inheritance

A subclass is a class that `extends` an existing class; that is, it has the attributes and methods of the existing class, plus more.

- Refer to the existing class as a *parent* or *superclass*
- When a class `extends` another class, it *inherits* the attributes and methods from the parent class

All classes by default extend *java.lang.Object.*

- Consequence: Compiler knows to call *“toString()”*

Designing classes

Time class:

- Hour, minute, second

Date class:

- Day, month, and year
- Contains everything in Time as well

Whats the superclass and whats the subclass?
How could we make these ***immutable***?
How could we define the distance between two Time or two Date objects?

Linear Search

Check each item in a collection one by one

Why is this call linear search?

Time it takes to search increases *linearly* with the size of the list

If we have 100 items in a list, how many items do we have to check in the worstcase scenario?

All 100

Linear Search

What happens (in terms of speed) when the list is very large?

The search becomes slower

In what cases do we do the most work (i.e. perform the most comparisons)?

When the item is not in the list

In what cases do we do the least amount of work?

When the item is the first element in the list

Binary Search

If the list is sorted in ascending order, we don't need to consider every element.

Which element should we check?

The middle

If the middle element isnt what we are looking for, what should we do?

Chop the search space in half (this is why its called ***binary*** search)

Binary Search run time

As the size of our collection increases, the number of guesses/comparisons increases, but not *linearly*

The time increases by $\log n$ $(\text{we use base } 2)$. Why?

Because we cut our search space in half each time

If our collection contains 8 data points, how many comparisons in worst case do we make:

$\log_2 8 = 3$

If our collection contains 512 data points, how many comparisons in worst case do we make:

$\log_2 512 = 9$

O(log(n))

Sorting

- BubbleSort & SelectionSort

- Runtime - $O(n^2)$

Runtime Analysis

Difference between Runtime Analysis and timing performance

Big-Oh: as the size of the data increases, how does the amount of steps an algorithm perform also increase

$$O(log(n)), O(n), O\big(nlog(n)\big), O(n^2), O(n^3)$$

Success in your learning/the course

Thanks you & Congrats!