Week 1 Notes

Prof Bill - Mar 2018

Week 1 notes on:

A. Lightning Lecture - CSC 210 in 15 minutes or less

B. Java/OOP review - CSC 160/161 in 15 minutes or less (sort of)

thanks… yow, bill

A. Lightning Lecture

CSC 210 in 15 minutes…

Array

Fixed number of cells, adjacent in memory.

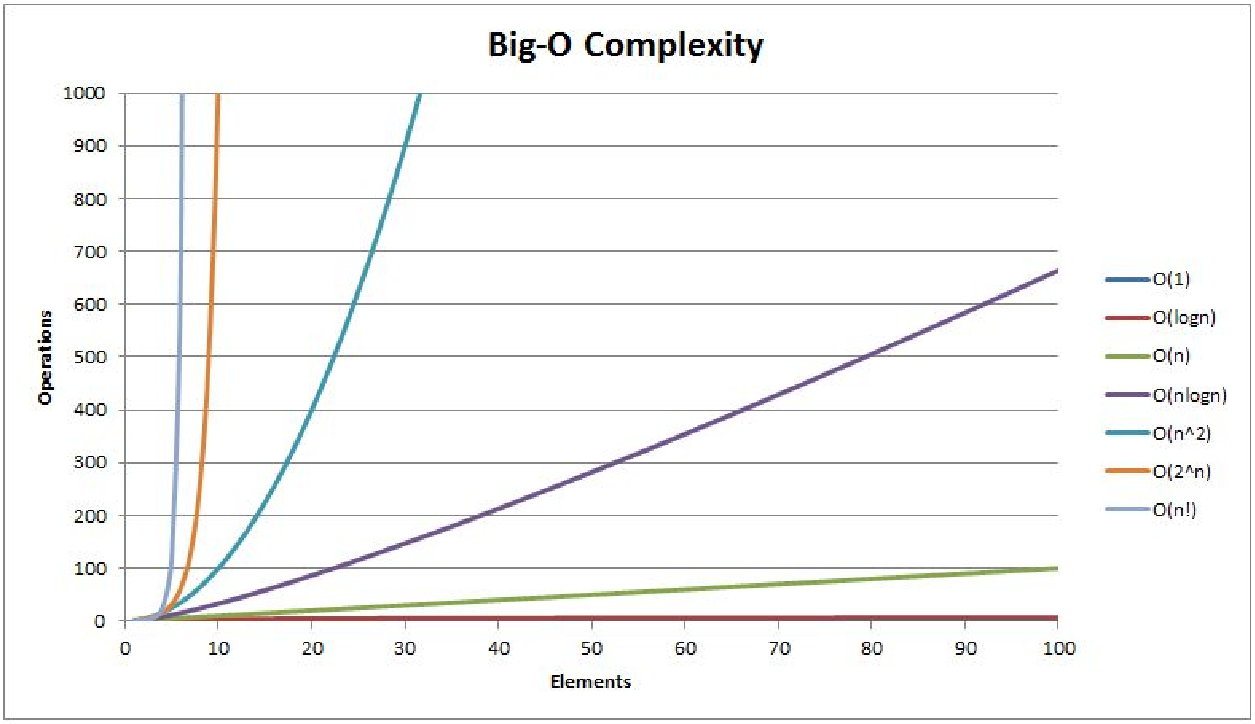
```
int[] example = new int[10];
```

Operations: Add to end; Add to beginning; Insert; Search; Remove

Advantage: easy, fast. Disadvantage: max size restriction

Big-O analysis

- Put your stopwatch away. This is not performance benchmarking.
- Theoretical worst case (upper bound) performance
- On data where problem size N = LARGE!
  - Use to estimate: CPU (time) usage, memory usage, disk usage, network
  - Don't worry about constants (startup time) or multipliers because our very large N dominates
- We will "do the math" later. The concept/i is more important.
- Seven performance categories are most common, for a problem of size = n:
  - O(1) - constant time
  - O(log(n)) - logarithmic time
  - O(n) - linear time
  - O(n log(n)) - quasi-linear or "n log n" time
  - O(n^2) - polynomial time
  - O(2^n) - exponential time
  - O(n!) - factorial time



Source:www.hackerearth.com/practice/notes/big-o-cheatsheet-series-data-structures-and-algorithms-with-thier-complexities-1/

Do you see why constants and multipliers don’t matter? They are insignificant compared to the performance function for large N. Try each function for (piddly) N=100.

Here’s another fun summary: bigocheatsheet.com/

Try - What Big-O are the array operations?

Linked List

Self-referential node.

Flavors: singly-linked, doubly-linked, head, tail

Operations: Add to end, Add to beginning, Insert, Search, Remove

Try again - What are the Big-O functions for these operations?

Advantage: No max size. Intuitive.

Disadvantage: No O(1) indexing into the list, garbage collecting nodes.

ArrayList

EZ rule: If we blowout our array size, then make it bigger.

Removes the max size disadvantage of an array.

Hash table

Goal: I'd like to get the array O(1) search by index performance for everything

Problem: But not everything is an integer/indexable. Like a name: “Prof Bill”

Solution: Create a “hash function” that turns “Prof Bill” into an integer.

B. Java/OOP Review

You should know this stuff from CSC 160/161. This is mostly terms and concepts that 210 students should be familiar with.

**** Muganda Ch 1-6, 8, 10**

Ch1 Intro

CPU, ALU, main memory, secondary storage
Von Neumann architecture: en.wikipedia.org/wiki/Von_Neumann_architecture
Are you older than Java? Java 1.0 in 1996, en.wikipedia.org/wiki/Java_version_history
keywords - reserved words in a programming language
compiler, Java Virtual Machine, executable code
IDE = Integrated Development Environment
The Programming Process… today is Agile, agilemanifesto.org
OOP = Object-Oriented Programming, goal = manage complexity

Ch 2 Java Fundamentals

console output, System.out.println
API = Application Programming Interface
variable, literal, primitive data types (int, float, char…)
Unicode for char representation
final keyword to create a constant
String class - part of standard Java API/library
comments

```
/* comments ignored by the compiler */
// end of line comment
/**
 * Javadoc comment!
 **/
```

javadoc - used by all JDK code, must use!
programming style - Java has strong idioms: camel notation, indentation, etc
Strong console idioms:

```
Scanner keyboard = new Scanner(System.in);   // input
System.out.println( “This is fun.”);   // output
```

Ch 3 Decision Structures

Style hint: always use curly braces with if and loops, even with only 1 stmt

logical operators: and (&&), or (||), not (!)
String comparison methods: equals(), equalsIgnoreCase(), compareTo()

Ch 4 Loops and Files

Increment (x++) and decrement (y--)
loops: while, for, do-while
nested loop
break, continue stmts within a loop
Random class, Java.Util.Random

Ch 5 Methods

method arguments, parameters
javadoc - @param, @return

Ch 6 A First Look at Classes

class, object
UML class diagram - attributes + methods + relationships (p 327)
data hiding - private attributes, public methods (setter, getter)
/* don't put data types in UML, p 342 */
instance methods, class methods
constructors (ctors), default ctor
overloading methods - same name, diff parameters
method signature

Ch 8 A Second Look at Classes

static fields and methods - class fields/methods
toString() - overload!
equals() method
copy ctor
aggregation vs. inheritance, has-a vs. is-a
this variable
garbage collection   /* not in C */

Ch 10 Inheritance

inheritance, is-a relationship, ex: Bee is-a Insect (and btw, Bee has-a Wing)
superclass, subclass, ctor interaction
override superclass methods (easy to confuse overload and override)
public, private, protected

Object class, everything is-a Object
polymorphism, dynamic binding
abstract class, abstract method, interface

**** Goodrich Chapter 2 OOP**

Java

Objects + **base types** = {boolean, char, byte, short, int, long, float, double}
In class, instance variables are private; **accessor**/getter and **mutator**/setter methods are public

Modifiers:

- **public**, **private**, **protected** - controls visibility to class variables and methods
- **abstract** - defines an interface, but no body/code
- **static** - makes a class variable or method (rather than instance)
- **final** - for variable, an initial value can never be changed; for method, it cannot be overridden

String class variables are immutable. Use **StringBuilder** to manipulate strings.

Simple I/O via console

- **System.out** is a **PrintStream** object, includes print() and println() methods
- Read from input stream using **Scanner** class with **System.in**

Section 1.7 An Example Program - review this

- ❖ **Notice (and copy) the structure!!!** instance variables; ctors; getters; update methods; main()
- ❖ private variables, public methods (why?)
- ❖ getter methods; no setters because variables are set in ctor and can't be changed after that
- ❖ printSummary is static, a class method (what's a better answer here?!?)

Just use **default package** for class
UML class diagram - a quick way to communicate class variables and methods

Javadoc - commenting standard used to produce documentation automagically (must use!); see page 51 example; the official Java API documentation is created using Javadoc, docs.oracle.com/javase/8/docs/api/

Consistent naming and indentation is part of quality code
Debugging = print statements or debugger

new operator “returns a reference to a newly-created object”; what’s a “reference”?
method signature - the name parameters and return value of a method; this is the interface, not the body/code
What’s the difference between an **instance variable** and a **class variable**? Method? How are these specified in Java?
ctor rules are complex: default ctor, ctor overloading, super, this, etc

Using Java from the **command line**: javac to compile, java to run your program
Scanner is nice for simple console input; see the 160/161 Muganda text for good examples
Just use **default package** for class; in larger projects, you’ll use packages

OO Design

Terms!
design pattern - a common or “typical” solution to a design problem

polymorphism means “many forms” (example: Pet p = new Dog(“Brownie”);)

inheritance = is-a relationship
composition = has-a relationship

interface - code describing an API (methods)
abstract class - in between concrete class and interface, some methods are abstract

> Interface is usually the starting point; sometimes you’ll do an abstract class to share snippets of code

exceptions - try, catch, throw, throws; exception hierarchy
generics - replace Object because “code became rampant with such explicit casts”

What is the **UML** representation for class, attributes, is-a relation, has-a relation? (see p 65) The relations between classes is a critical design decision.
Some nice text/examples in Wikipedia: en.wikipedia.org/wiki/Class_diagram

For OOP, use public methods and private variables. Why?
Java only supports **single inheritance**. But not multiple inheritance. Why?