NOTES INTRO TO ALGORITHMS / Big-O Notation

Programming Logic and Design (book) - Gaddis

TWO MAJOR METHODS OF PROGRAMMING:

Procedural programming - a method of writing software; centered on the procedures or actions that take place in a program (calculation)

Object-oriented programming - method of programming centered on objects, which are created from abstract data types and encapsulate data and functions together

Early programming languages were procedural

procedure - a module or function (method) that performs a specific task (ex. get input from user, perform calculations, read/write files, display output, etc.)

---

http://cs.lmu.edu/~ray/notes/pltypes/

Different languages have different purposes, so it makes sense to talk about different kinds, or types, of languages. Some types are:

- **Machine languages** — interpreted directly in hardware
- **Assembly languages** — thin wrappers over a corresponding machine language
- **High-level languages** — anything machine-independent
- **System languages** — designed for writing low-level tasks, like memory and process management
- **Scripting languages** — generally extremely high-level and powerful
- **Domain-specific languages** — used in highly special-purpose areas only
- **Visual languages** — non-text based
- **Esoteric languages** — not really intended to be used
- *These types are not mutually exclusive*: Perl is both high-level and scripting; C is considered both high-level and system.

https://en.wikipedia.org/wiki/List_of_programming_languages_by_type

---

Object-oriented programming (OOP) - focuses on creating objects

objects - software entity that contains both data and procedures

fields = data

(variables, arrays, other data structures stored in the object)

procedures = methods

(also known as modules or functions)

Encapsulation - combining of data and code into a single object

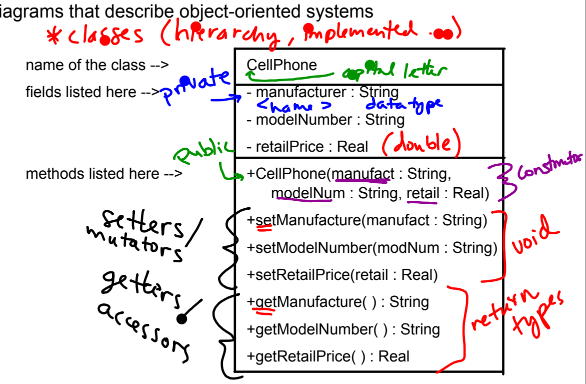
Data hiding - refers to the object's ability to hide its data from code that is outside the object; only the object's methods may access and make changes to the object's data

~protects data from accidental corruption

~outside code does not need to know the format/internal structure of an object (the programmer knows, and that is fine)

After this part in the book, lots of talk on what is a "class" and how to create them (private/public, Javadoc info, etc.)

Unified Modeling Language (UML) - standard way of drawing diagrams that describe object-oriented systems



Data Structures & Other Objects Using Java (book) - Main

data structure - a collection of data, organized so that items can be stored and retrieved by some fixed techniques (ex. arrays) (search + sort)

algorithm - a procedure or sequence of instructions for solving a problem

Data Structures & Algorithm Analysis in Java (book) - Weiss

"Suppose you have a group of *N* numbers and would like to determine the *k*th largest. This is known as the **selection problem**."

How would you solve this problem?

... if you had file full of 10 million random numbers, you can't solve this problem with any type of sorting algorithm we have learned in a reasonable amount of time... you would need to give the computer several days to solve this problem... how do we deal with this impracticality? Is there an algorithm to solve this reasonably? quickly?

Sample word puzzle:

	1	2	3	4
1	t	h	i	s
2	w	a	t	s
3	o	a	h	g
4	f	g	d	t

How do you find the words in the puzzle, moving in any direction?

How about with a program?

Now, what if the puzzle was 16 rows, 16 columns, and the word list used in it essentially the English dictionary?

Would take a LONG TIME for a computer to solve this...

how do we optimize our algorithms?

Intro to Algorithm Analysis

1) How to estimate time required for a program

2) How to reduce the running time of a program from days/years to a fraction of a second

3) How careless recursion can give you unwanted results

4)Very efficient algorithms

Math we will need...

Exponents

Logarithms

Series

Modular Arithmetic

PROOFS (induction, counterexample, contradiction)

Recursion (base case, making progress/the recursion)

Building Java Programs (book) - Reges, Stepp

complexity - a measure of the computing resources that are used by a piece of code, such as time, memory, or disk space

time complexity - how long the program takes to run

(this is what your reading from DD&OS 1.2 was all about, Big-O)

empirical analysis - run the program and measure how long it takes to run (example, comparing two sorts by running both of them)

~not a reliable measure... different computers with different processor speeds with different amounts of memory... doesn't work well

algorithm analysis - applying techniques to mathematically approximate the performance of various computer algorithms

complexity class - a set of algorithms that have a similar relationship between input data size and resource consumption

~determined by looking at the most frequently executed line of code

example: if the most frequent line executes ($2N^3 + 4N$) times, the algorithm is in the "order N^3" complexity class, or $O(N^3)$ for short

We use Big-O notation to compare complexity / rates of growth of algorithms...

The list shows the order from slowest to fastest growth (lowest to highest complexity)

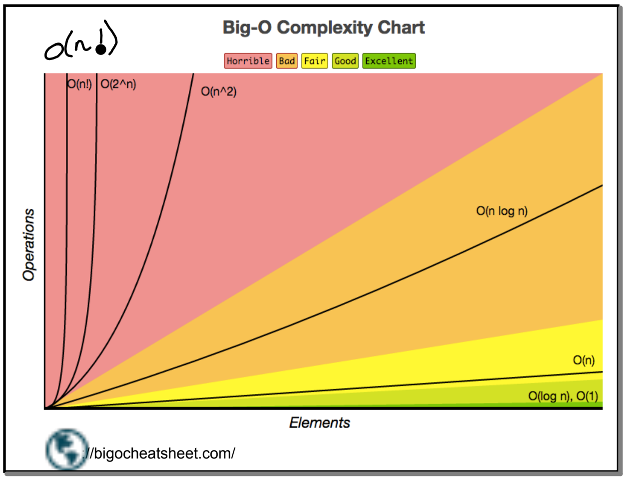
slowest growth = lowest complexity ↓ take the least amount of time to run

Big-Oh notation

relative rates of growth - how quickly a function is increasing compared to other functions

Function	Name	Label	Description
c (y = #)	Constant-time	$O(1)$	algorithms have runtimes that don't depend on input size; ex: converting C to F degrees, numerical functions (Math.abs)
log*N*	Logarithmic (base 2)	$O(\log N)$	algorithms typically divide a problem space in half repeatedly until it is solved; ex: binary search
$\log^2 N$	Log-squared		
N	Linear	$O(N)$	algorithms have runtimes that are directly proportional to *N*; ex: algorithms that compute the count, sum, average, maximum, or range of a lists of numbers

Function	Name	Label	Description
$N\log N$	Log-linear	$O(N\log N)$	algorithms that perform a combination of logarithmic and linear operations, such executing a logarithmic algorithm over every element of a dataset of size *N*
N^2	Quadratic	$O(N^2)$	algorithms have runtimes that are proportional to the square of the input size
N^3	Cubic	$O(N^3)$	algorithms have runtimes that are proportional to the cube of the input size; ex: code to multiply to *N* x *N* matrices
2^N	Exponential	$O(2^N)$	algorithms have runtimes that are proportional to 2 raised to the power of the input size; if the input size increases by 1, it takes twice as long to execute (THEY'RE SO SLOW, only use on small input datasets)



description	order of growth	example	framework
constant	1	`count++;`	*statement (increment an integer)*
logarithmic	$\log n$	`for (int i = n; i > 0; i /= 2) count++;`	*divide in half (bits in binary representation)*
linear	n	`for (int i = 0; i < n; i++) if (a[i] == 0) count++;`	*single loop (check each element)*
linearithmic	$n \log n$	[see *mergesort* (PROGRAM 4.2.6)]	*divide-and-conquer (mergesort)*
quadratic	n^2	`for (int i = 0; i < n; i++) for (int j = i+1; j < n; j++) if (a[i] + a[j] == 0) count++;`	*double nested loop (check all pairs)*
cubic	n^3	`for (int i = 0; i < n; i++) for (int j = i+1; j < n; j++) for (int k = j+1; k < n; k++) if (a[i] + a[j] + a[k] == 0) count++;`	*triple nested loop (check all triples)*
exponential	2^n	[see *Gray code* (PROGRAM 2.3.3)]	*exhaustive search (check all subsets)*

/introcs.cs.princeton.edu/java/41analysis/

<u>Homework:</u>

BJP Chapter 13 - page 816, #4 - 5 all

(will post online)

Math we will need...

Exponents

Logarithms

Series

Modular Arithmetic

PROOFS (induction, counterexample, contradiction)

Recursion (base case, making progress/the recursion)