Week 1 - Intro to DS&A

AD 325 - Brenden West

Contents

Learning Outcomes

- Class overview
- Java programming refresh
- Recursion
- Data structures
- Algorithmic analysis
- Linked lists

Reading & Videos

- Carrano & Henry, Chapters 1 - 4
- https://www.coursera.org/learn/algorithms-part1/home/week/1
- https://algs4.cs.princeton.edu/10fundamentals/ - Ch. 1.1 - 1.4 (review)
- https://www.geeksforgeeks.org/recursion/ (review)
- https://www.geeksforgeeks.org/data-structures/linked-list/ (review)
- https://www.geeksforgeeks.org/assertions-in-java/

Java Programming

DS&A concepts are language independent, but this class teaches them via Java which students should know from previous classes.

This course will make use of several Java features that may be new to students.

- ***Generics*** - A Java mechanism that allows a class to work for any data type. Classes are defined with a symbolic placeholder for some concrete type to be used in practice.
- ***Autoboxing*** - Java automatically converts (*casts*) between a primitive type (e.g. *int*) and the corresponding wrapper type (*Integer*). Automatically casting a wrapper type to a primitive type is known as *unboxing*.
- ***Assertions*** - Java uses the *assert* command to test if a boolean expression is *true*. Such assertions test the correctness of program assumptions and are the basis of *unit testing*.

Data Structures & Algorithms

Data structures & Algorithms (DS&A) describes the concepts for solving programming problems efficiently. ‘Correct’ solutions can differ by orders of magnitude in terms of speed & memory usage.

DS&A are frequently concerned with operations on collections of data such as - search, add, update, delete, count, and sort.

Data structures describe ways in which program data is stored and accessed. The choice of data structure can simplify or complicate common program operations such as - search, add, update, delete, count, and sort. Choice of data structure also affect computer memory requirements for running a program

Collections may use structures built-into a programming language, such as arrays or linked lists, or an abstract data types such as bag, stack, or queue.

Algorithms are the logical methods for solving a problem. Some common algorithms are independent of data structure, while others are closely bound to abstract data types (e.g. trees & graphs).

Algorithmic Analysis

Efficient algorithms can greatly speed calculations & even solve previously unsolvable problems.

The study of efficient algorithms is hard to separate from data structures, so these are usually taught together. **Algorithmic analysis** uses the scientific method to answer two key questions*:*

- How long will a program run?
- How much memory will a program consume?

Programmers can observe program running time or build a mathematical model for total running time based on*:*

- Cost of execution of each statement
- Frequency of execution of each statement

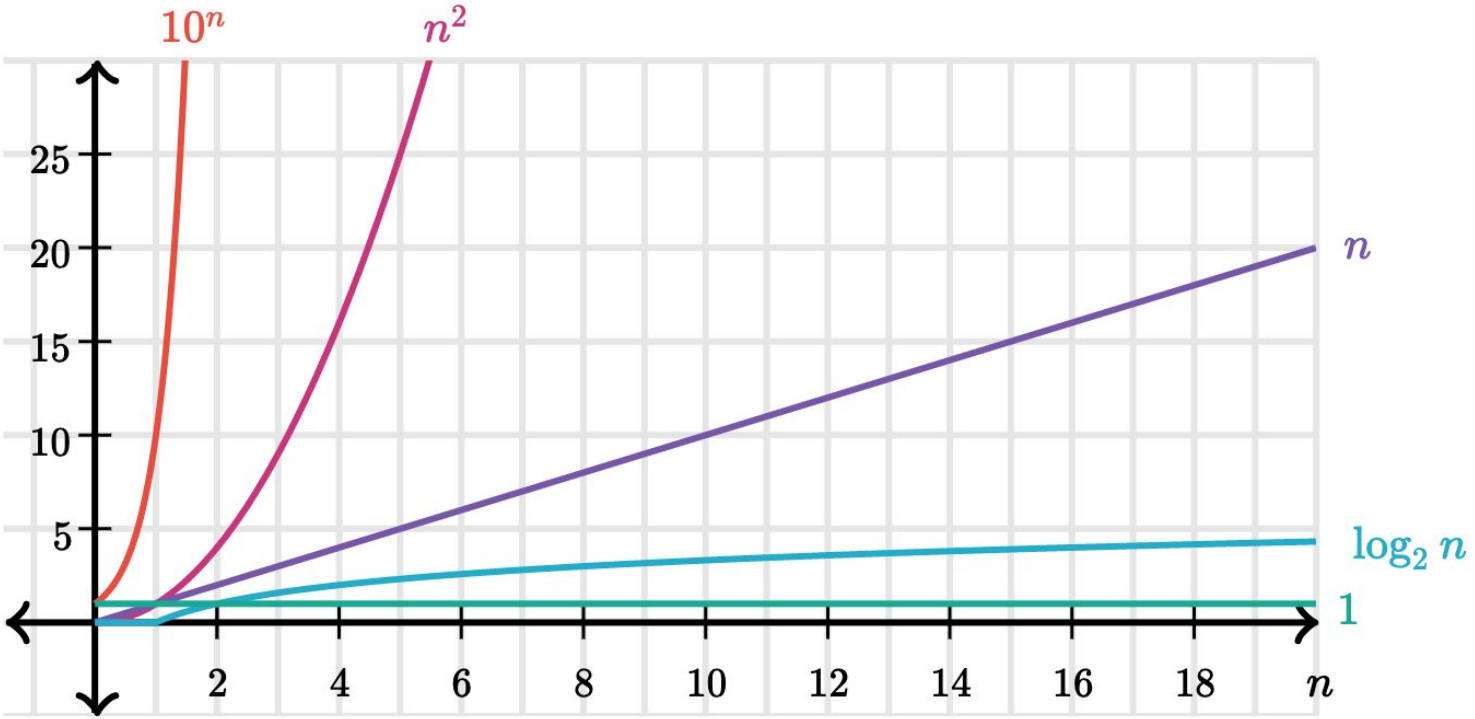
Cost models are functions that describe the program's 'order of growth' and allow evaluation of program efficiency independent of programming language or run-time environment.

Order of Growth - Big-O Notation

Cost models are usually shown in simplified *Big-O notation* that ignores low-order mathematical terms to represent program behavior at extreme scale:

Description	**Big-O**	**Example**
constant	1	add 2 numbers
logarithmic	logN	binary search
linear	N	find max in a list
linearithmic	NlogN	mergesort
quadratic	N^2	check all pairs in a list
cubic	N^3	check all triples in a list
exponential	2^N	check all subsets in a list

Order of Growth visualized



Source: https://cs61a.org/study-guide/orders-of-growth/

Order of Growth calculating

- Identify the operations performed for each input (n)
- Discard lower-order operations, as these become inconsequential as N grows large

Abstract Data Types

Programmers commonly use ADT's to hide implementation details from clients and provide a stable public interface to the data and methods.

Using an ADT allows the programmer to change the underlying algorithm or data storage without affecting clients.

ADT's typically store data in arrays, which are native to most languages, or in a linked list.

ADT's can be a composed of arrays, linked lists, and other ADT's.

Linked List

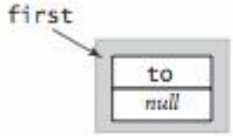
Linked Lists are a fundamental alternative to arrays for structuring a collection of items.

- Not native to Java
- Recursive structure that’s either null or a node w/ data & reference to a linked list
- Sequence of items, where each item links to next item in list (single linked list)
- In double linked list, items also have a link to ‘previous’ item
- Does not require contiguous memory
- Does not require advance sizing
- Can be used for any type of data,
- Items can’t be accessed by index
- Items can be added/removed more easily than for arrays but can require ‘traversing’ the list

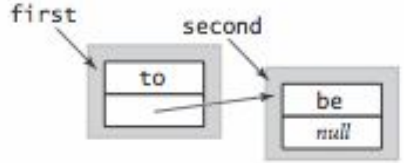
Linked Lists

In a linked list
To build a linked list, we create a `Node` for each item, set the item field in each of the nodes to the desired value, and set the **`next`** fields to the next node in the linked list.

```
Node first  = new Node();
first.item  = "to";
```




```
Node second = new Node();
second.item = "be";
first.next  = second;
```




```
Node third  = new Node();
third.item  = "or";
second.next = third;
```