Data Structures and Algorithms

Introduction

This is a little knowledge sheet that I have made to learn about the underlying background around each data – structure.

What are Data Structures

Coding is essentially manipulating data in a way in which we can accomplish something. Data structures are really just a way to organise and manage data. A **data structure** is a specialized format for organizing, processing, retrieving and storing **data**. While there are several basic and advanced **structure** types, any **data structure** is designed to arrange **data** to suit a specific purpose so that it can be accessed and worked with in appropriate ways.

Complexity Analysis

The **complexity** of an **algorithm** is a function describing the efficiency of the **algorithm** in terms of the amount of data the **algorithm** must process. ... Time **complexity** is a function describing the amount of time an **algorithm** takes in terms of the amount of input to the **algorithm**

Big O notation

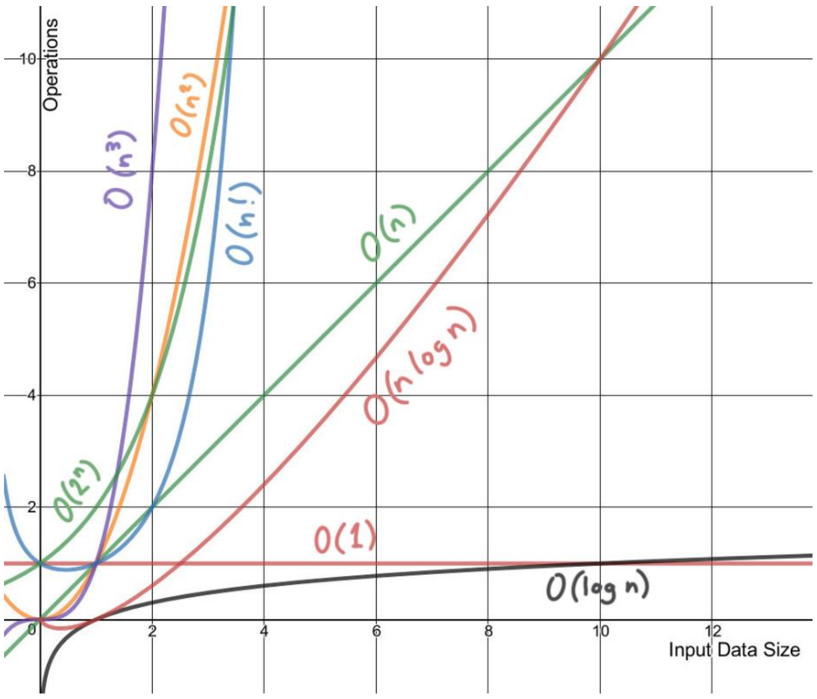
This is how we donate the time complexity of an algorithm, below shows an example for simple algorithms and their associated big O notation.

B_1 = 1 + a[0] – O(1)

B_2 = sum(a) – O(n)

B_3 = pair(a) – O(n^2)

If we were to run all the algorithms in the same method the overall time complexity would be simplified to that of B_3.



Logarithm

What it really means….. The base is generally always assumed to be 2. (binary logarithm)
$Log_b(x) = y$ if and only if $b^y = x$
$Log_2(N) = y$ if and only if $2^y = N$

When you double N, you're only increasing the increment by 1.
$2^4 = 2^3 * 2$.

Algorithms that eliminate half of the input at every step tend to be logarithmic in time complexity

Arrays

When arrays select memory they always ensure that they get back to back memory slots (every piece of memory store 8 bits, so if we had to store an array of 4 integers (32 bit ints) we would need 4 * 4 (16 slots of memory)

- Indexing – it knows the width of one element, and the index yours specifying – O(1) Time
- Searching – O(N) Time , O(1) Space (Static)
- Insert – O(N) Time, O(1) Space (Static)

<u>Two types of arrays: - Amortized Analysis</u>

Static: fixed number of memory slots (as described above)
Dynamic: is an array that can change in size (arraylist), normally allocates twice the amount of space subconsciously. If you fill this, it will copy itself to another array of double the size. Example an array of two would exactually have 4 slots, if we fill this then it will become 8 slots.

- Worst Case Insertion is O(N)

Linked Lists

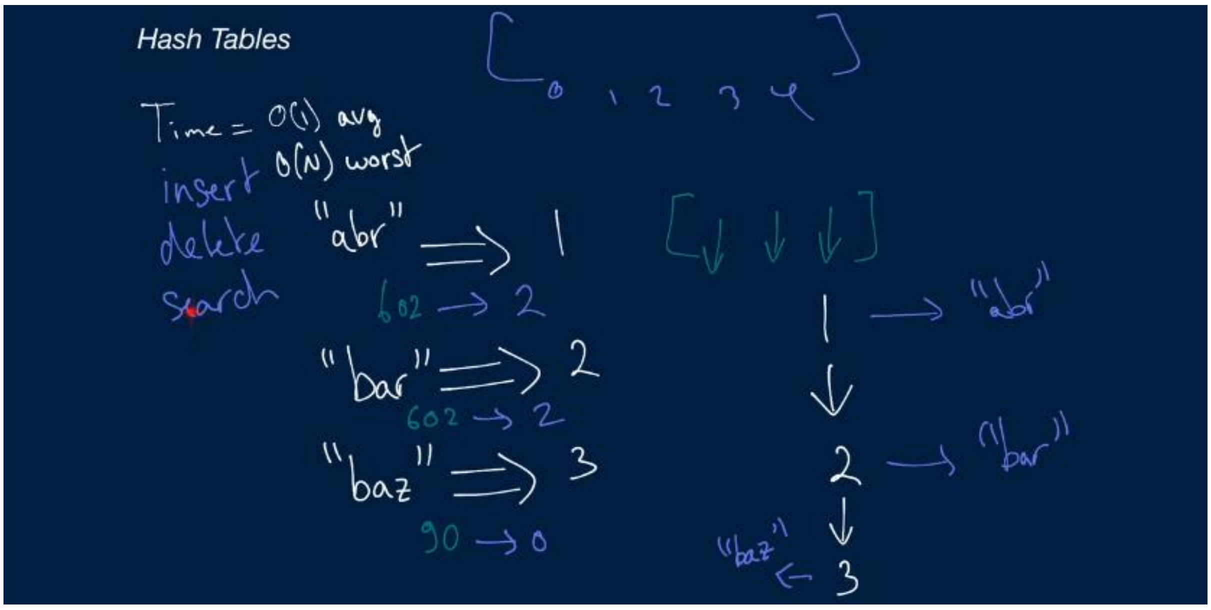
Linked Lists tend to read from left to right, they have a pointer to their next node. The last node points to null. The difference between this and arrays is how they're stored in memory, in the case of linked lists, memory allocation is not done in a consecutive manner. They can be stored anywhere in memory and are connected via their pointers (references the next memory slots address)

- Indexing – O(i) Time as you have to move through I elements to find the one you're looking for
- Init – O(n) ST.
- Searching O(N) time, O(1) space.
- Insert – O(1)

They can be doubly-linked, that is that each value has a pointer that points to the node before and another pointer that points to the value in front. (Head and Tail)

Hash Tables

Most questions involve using hash-tables, they're very important to learn! Behind the scenes when you're inserting a values you transform the key using a hash-function into an index that fits in an underlying array. You then transform in back when retrieving it. Sometimes we have indexes that are the same value (collision) – To prevent this our array points to linked lists instead of indexes. See example below. In average however, the values are almost always separated in their own linked list.



- Insertion – O(1) Time, O(N) worst case
- Deletion – O(1) Time
- Searching – O(1) Time

If you're underlying array does have enough space, you can implement a hash-table that resizes itself.

Stacks and Queues

A stack is a data structure that allow inserting and removing LIFO (Last in first out). A Queue is basically the opposite of a stack FIFO (First in, first out). They're essentially a list of elements.

Stack:

- Insertion / Delete – O(1) ST
  - Push (add)
  - Pop (remove)

Queue:

- Typically implemented with a linked list
  - Keep track of head and tail
  - Enqueue (Insertion): replace the head - O(1)
  - Dequeue (remove): remove the tail – O(1) as long as we have reference to the tail

Searching – O(N) Time, O(1) Space

Strings

A string is typically stored in memory as an array of integers. The way this is done is through some sort of character encoding standard (Ascii , A -> 65, a ->97 ETC). When we're dealing with a single character in a string it is all constant time.

Traversing O(N) Time, O(1) Space

Copy – O(N) ST

Get -O(1)

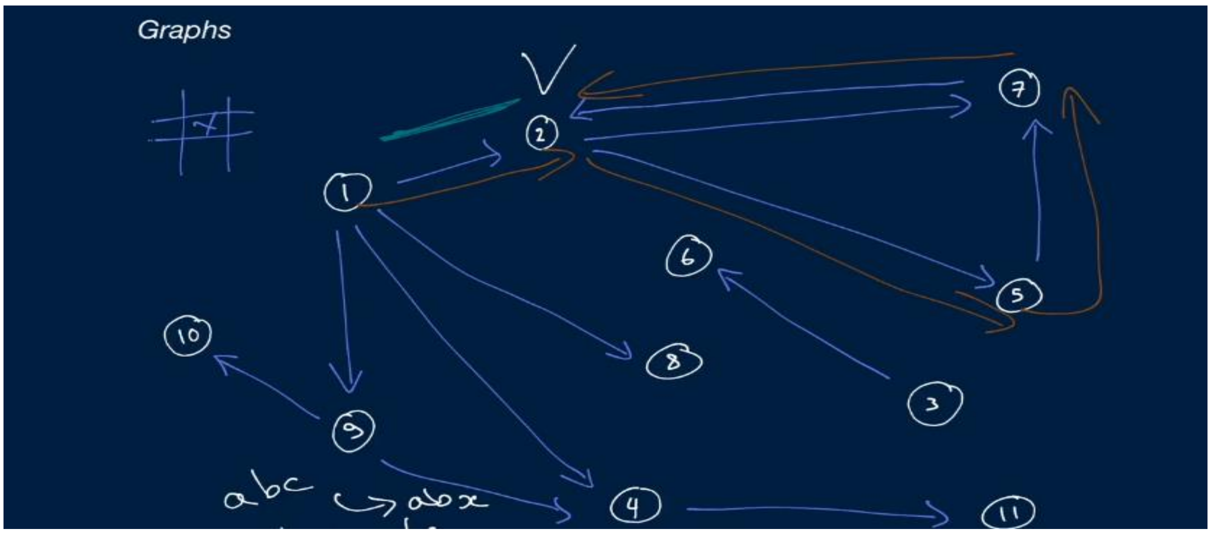
In java, String are Immutable. This means you cannot alter them after you create them, essentially you create a brand-new string every time. If you use a stringbuilder it is O(1) (imagine an array list), otherwise if you were to += on a string it would O(N). If you were adding two strings together for example, "ab" + "cd" would be O(MN).

Graphs

A graph is a collection nodes that may or may not be connected to each other (collection of edges connections (Arrows) and vertices(nodes(vertex))). Sometimes some nodes are disconnected from each other which means the graph would be denoted as "disconnected" or not connected. Arrows imply direction, for example if you had an arrow pointing from 1 to 2, you could go to 2 from 1 but you couldn't go from 1 to 2. Some graphs are "directed", an undirected graph would not have these directions (just a straight-line). if a graph does not have cycles (if at any area in the graph where you have 3 or more nodes that go in an infinite loop you have a cycle) "cyclic graph" - "acyclic graph".

- If you hit a node you've already visited you know you're in a cycle
- Cyclic graphs are really important, you want to make sure that if you're stuck In a loop you'd want to find a way of marking nodes you've visited (look out for them in questions)

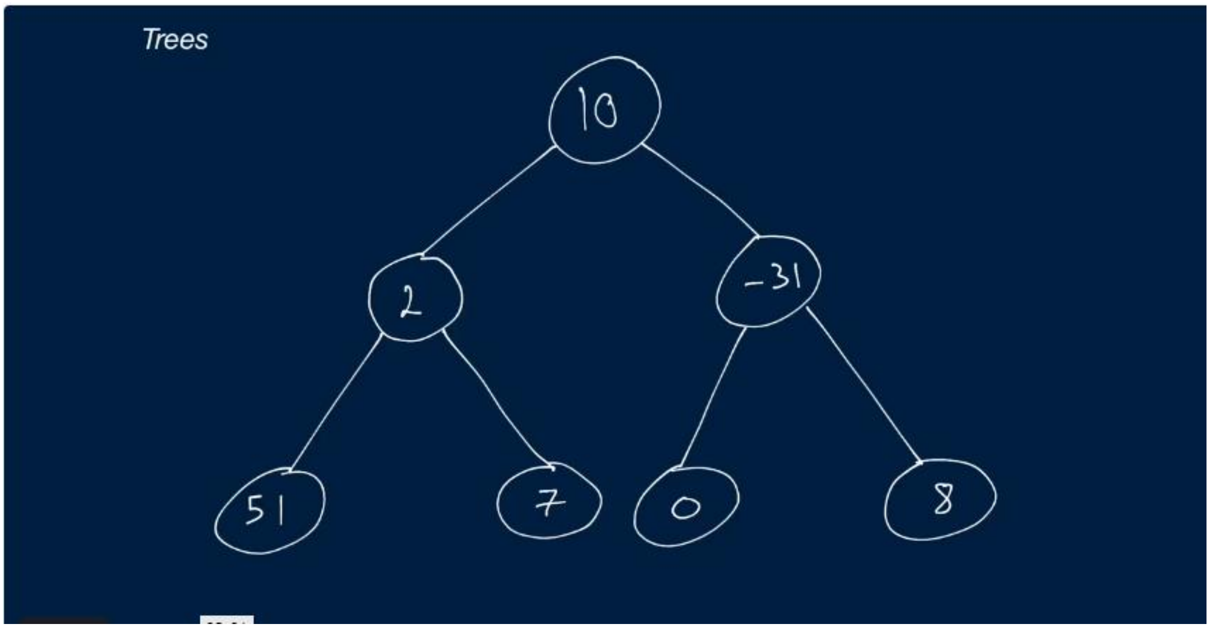
To represent a graph in code we typically use an adjacency-list. Essentially we could has a list of nodes or hash table where every key points to its relevant node and every node has a list of its edges



So the nodes 1 would have a list of (2,4,8,9). Its very easy to do, edges are essentially pointers under the hood. Space -> O(V+E). Search Methods Breadth (each level at a time), Depth first search (root, left,right). Check out the questions (Know them very well) – Time -> O(V+E).

Trees

A tree is a type of graph, in the context of coding interviews they're quite simple. More specifically when we talk about a tree, we're referring to a graph structure that is rooted and has child nodes.



For example, 10 is the root node and has 2/-31 as its child nodes. (does not have cycles). K-ary trees are trees where every node has at most k -children. Other examples (Min heaps, Max heaps, Binary Trees, Binary Search Trees, Tries (tree-like data structure that stores characters in a string)) – Lots of different types of trees.

Traversing through all nodes – O(N) ST.

One subtree – O(log(N) ST