Summary of:

**Data Structures and Algorithms**

Stacks

-> use arrays (dynamic, static arrays)

Queues

-> use linked lists (doubly, singly linked lists)

Ordered/Tree Sets and Ordered/Tree Maps

-> use binary search trees

Recursion

Single branch vs double branch

Insertion sort (is a stable sorting algorithm which is preferred) O(n^2)

Stable vs unstable sorting

Stable preserves original position

Merge Sort (divide and conquer) -> recursive, most common

Time complexity: O(n log n)

Space complexity: O(n)

Start at beginning of split array

Quick sort (pivot value) -> recursive

No additional memory, unstable sorting algorithm

Choose a pivot value (usually a middle item) and split the array in half

Time complexity O(n log n) but in worst case O(n^2)

Bucket sort

Constraints -> Small range of elements

Time complexity: O(n)

Space complexity: O(1)

Binary Search

Time Complexity: O(log n)

Space Complexity: O(1)

Binary Trees

Guaranteed to have leaf nodes, no cycles are allowed

Sibling nodes

Two pointers (left, right)

Root node

Parent node, child node

Leaf node = node that does not have child

Height of a tree, Depth of a node

Ancestor and descendent

Binary search tree

No duplicate values

Everything of the left subtree is less than the node

Everything of the right subtree is greater than the node

Use recursion

Removing and adding is better vs sorted arrays

Easier to add leaf nodes

Case 1: 0 or 1 child

Case 2: 2 children -> use smallest node on right sub tree or largest value on left sub tree

Traversal

Binary tree organization O(n log n) -> because there have to be n number of insertions which each of them take log n time

Preorder traversal (printing before getting to the leaf node)

Depth first search (DFS)

Breath first search (BFS)

Level order traversal

Time complexity O(n)

Sets

Search and remove values in O(log n)

Maps

Key - value pairing

Backtracking (brute force)

Like a maze go back -> recursive algorithm

Poping elements

Time complexity: O(n)

Heap / Priority Queue Min/Max -> implemented as complete binary tree

Compared to a binary search tree the min or max value can be retrieved at constant time O(1)

Generally Min heap are used (root node is the smallest value in the entire heap)

1. Structure property - complete binary tree: every single level is full except leaf level, adding items left to right across levels
2. Order property - allowed to have duplicated, every child node is larger than the parent node recursively

Binary heaps are implemented as arrays and the root node starts at index 1 (for the math to work out)

Left child = 2\*i

Right child = 2\*i + 1

Parent node = i/2

Pushing into a min heap leaf node

Put in proper place then if smaller than parent, swap with parent recursively percolate upward till the order property is satisfied

Time complexity: O(log n)

Popping a root node for a min heap

Replace the root node with the last leaf node and swap with child recursively percolate downward till the order property is satisfied

Heapify

Input array -> heap

Move index 0 to last spot in the array, then divide the (length of the array - 1) by 2 and start the process

Traverse the array in reverse, compare and swap if either of the child node is smaller than the parent

Time complexity: O(n)

Build the heap: O(n)

Poping: O(log n)

For n elements: O(n log n)

For coding interviews

When using heaps are more common than binary search trees

Hash sets/map

Keys are used to sort and access values

Tree maps

Insert: O(log n), remove: O(log n), search: O(log n), traversal: O(n)

Overall time complexity: O(n log n)

Hash maps

Insert: O(1), remove: O(1), search: O(1) -> technically average case time complexity in worst case: O(n)

traversal: O(n log n)

Overall time complexity: O(n)

Duplicates are not allowed in hash maps

Hash map collisions

Resize the array (double it) -> when resizing the value cannot remain in the same place (rehashing the array)

Collision handling -> chaining, open addressing (adding +1 / changing the index)

Size of the array should be a prime number to minimize collisions

/////////////////////////////

Graphs

Linked list, trees = subset of graphs

Made of nodes(vertices - vertex) and pointers(edges) that are connected together

E <= V^2 (node can point to itself and loop)

Directed graph vs indirective graph

* Matrix
* Adjacency Matrix
* Adjacency List (most common way implemented)

Common ways that graphs are implemented

Rows and columns

Depth first search

Visit hash set

O(4^n\*m)

Breadth first search

Shortest path

O(n\*m)

Value and list

dfs O(n^v)

bfs time complexity O(v+e), space complexity O(v)