

Tech interview

Content

1. Arrays/Lists/Stacks
2. Heap, PQ
3. Sorting
4. BST
5. Hash maps
6. Math
7. Graphs
8. Bit Manipulation
9. Recursion
10. Assignments

1. Arrays/Lists/Stacks

Arrays/Lists/Stacks

- **List vs Set**
 - Set: unsorted, unique
 - List: ordered(?), allow dups
- **Array vs List(=Vector)**
 - List (+): Dynamic size, quick insert/delete
 - List (-): quick random access

Arrays/Lists/Stacks

- **ArrayList**
 - resizable array.
 - size is increased dynamically.
 - elements can be accessed directly, since ArrayList is essentially an array.
 - multiple threads can work on ArrayList at the same time
- **LinkedList**
 - implemented as a double linked list
 - better performance on add/remove
 - worse performance on access
- **Vector** = synchronized **ArrayList**
 - Single thread can work on at the same time

Arrays/Lists/Stacks

- **Hash Map vs Hash Set**
 - Set: unique
- **Hash map vs Hash Table**
 - Hash Table: only one thread can access, does not allow NULL
 - Hash Map : many threads can access, allows NULL

Arrays/Lists/Stacks

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Skip List</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n \log(n))$
<u>Hash Table</u>	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Binary Search Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Cartesian Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>B-Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Red-Black Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Splay Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>AVL Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>KD Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$

2. Heap

- [Heap Sort](#)
- [Kth largest element in a stream](#)
- [Merge K sorted linked lists](#): PQ

Heap

	Avg	Worst
• Build:	N	
• Search:	N	N
• Sort:	N logN	
• Min:	1	
• Kth:	K*Log N	
• Insert:	logN(?)	
• Delete:	logN	logN

3. Sorting

Heap sort:

- in place (internal)
- Worst case: $N \log N$
- Is stable?

Merge sort

- External
- Is stable?

Quick sort:

- In place (internal)
- Worst: NN
- Avg: $N \log N$
- Is stable?

Sorting

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\Theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\Theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\Theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\Theta(n \log(n))$	$O(n \log(n))$	$O(n)$

Sorting

- **Bubble Sort:** swap elements
- **Selection Sort:** Linear search for a smallest element => move it to the front.
- **Merge Sort:** split/sort/sort/merge
- **Quick Sort:** partition/sort/sort
- **Heap search**
- **Binary search:**
- **Bucket sort:**

The quick sort is internal sorting method where the data is sorted in main memory.

The merge sort is external sorting method in which the data that is to be sorted cannot be accommodated in the memory and needed auxiliary memory for sorting.

Sorting: k-th best element

- **Alg0**: sort and select
- $N \log N$
- **Min Heap – HeapSelect** ([link](#))
- $N + K * \log N$
- **Max-Heap**
- $K + (N-K) * \log K$
- **Quick select**
- Average: N

4. BST

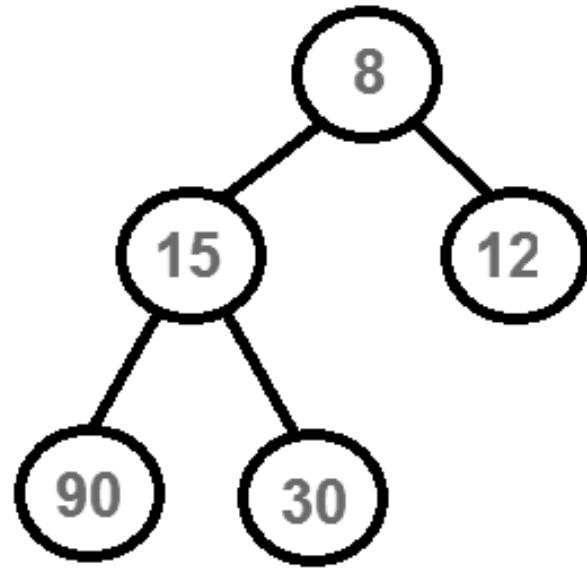
BST

- Inorder: Left, **Root**, Right
Preorder: **Root**, Left, Right
Postorder: Left, Right, **Root**

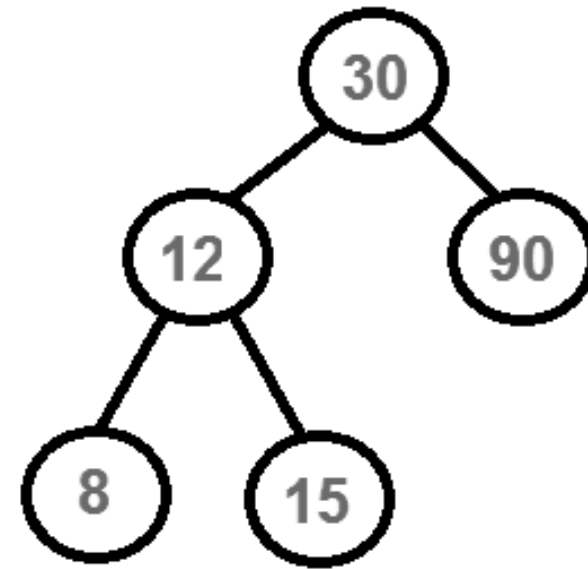
BST

	Avg	Worst
• Build:	N	
• Search:	$\log N$	
• Sort:	$N \log N$	
• Min	$\log N$	
• Kth:	$K \log N$	
• Insert:		
• Delete:		
• Prev/Next		

BST



Min Heap



Binary Search Tree

BST vs Heap

- **Binary Search Tree**

- search for any elements is $O(\log N)$

- **Heap**

- search Min/Max: $O(1)$

- <https://stackoverflow.com/questions/6147242/heap-vs-binary-search-tree-bst>
- If you only care about find Min/Max (e.g. priority-related), go with heap.
- If you want everything sorted, go with BST

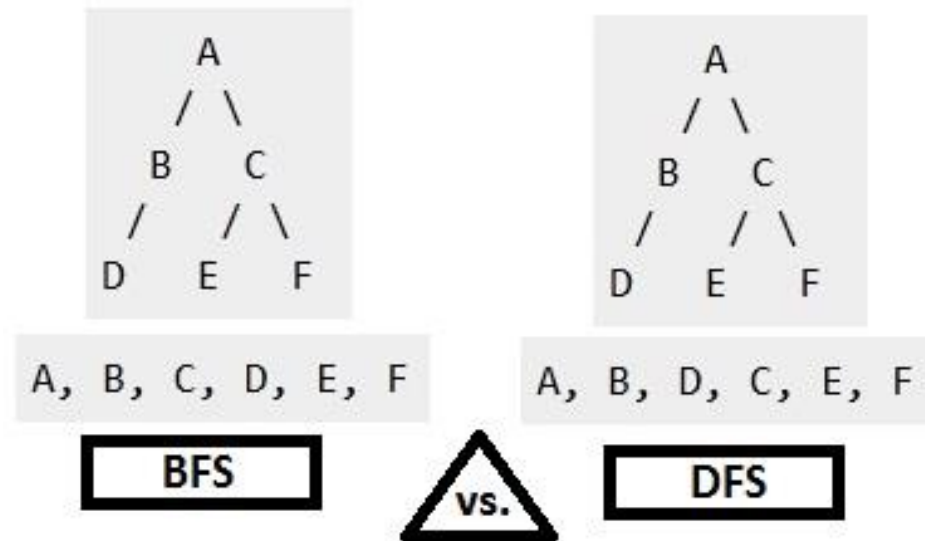
5. Hash maps

6. Graphs

- Problem by graph algorithms like distance, search, connectivity, cycle-detection, etc.
- 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list) - pros and cons.
- Basic graph traversal algorithms
- Breadth-first search, Depth-first search
- Binary vs Red-Black tree

Graphs: BFS vs DFS

	BFS	DFS
Basic	Vertex-based algorithm	Edge-based algorithm
Data structure	Queue	Stack
Structure of the constructed tree	Wide and short	Narrow and long



7. Math

- Essentials probability theory
- Combinatorics
- n -choose- k problems

8. Bit Manipulation

9. Recursion

- Staircase
- Fibonacci
- All permutations
- All subsets
- Has Sub-array
- All Subarrays

10. Assignments



Typical assignments



- [rec_01](#) [Subarray with given sum](#)
- [bits_02](#) [Finding 2 missing numbers:](#)
- [queu_01](#) [Longest valid Parentheses](#)
- [queue_02](#) [Jumping Numbers:](#)
- [recurs_02](#) [Find triplets with zero sum](#)
- [recurs_03](#) [Egg Dropping Puzzle](#) – recursive
- [dynamic_03](#) [Egg Dropping Puzzle](#) - dynamic
- [dynamic_01](#) [Word Break Problem](#)
- [Math_01](#) How to find if two given rectangles overlap?
- [Math_02](#) How to find angle between hour and minute hands at a given time?
- [Search_02](#) Given a big string of characters, how to efficiently find the first unique character in it?
- [rec_05](#) Coins game
- [dyn_01](#) Words break
- [Bits_00](#) Find one missing number an array [1 to n] of unique values.
- [search_00](#) Given a big array, how to efficiently find k'th largest element in it
- [dynamic_02](#) In array of sorted words find order (or precedence of characters) in the language

Algorithms: strings



recur_02 [Count triplets with sum smaller than a given value:](#)

sort_06 [Convert array into Zig-Zag fashion](#)

search_03 [Pythagorean Triplet in an array](#)

[Find the smallest positive integer value that cannot be represented as sum of any subset of a given array](#)

[Smallest subarray with sum greater than a given value](#)

[Stock Buy Sell to Maximize Profit](#)

[All Possible Palindromic Partitions](#)

[Generate all possible sorted arrays from alternate elements of two given sorted arrays](#)

[Reverse an array without affecting special characters](#)

[Length of the largest subarray with contiguous elements](#)

Algorithms: sorting



[Binary Search](#)

[Bubble Sort](#)

[Insertion Sort](#)

[Merge Sort](#)

[Heap Sort \(Binary Heap\)](#)

[Quick Sort](#)

[Find Kth Smallest/Largest Element In Unsorted Array](#)

[Search an element in a sorted and rotated array](#)

[Interpolation Search](#)

[Given a sorted array and a number x, find the pair in array whose sum is closest to x](#)

Algorithms: BST



1. [Find Minimum Depth of a Binary Tree](#)
2. [Maximum Path Sum in a Binary Tree](#)
3. [Check if a given array can represent Preorder Traversal of Binary Search Tree](#)
4. [Check whether a binary tree is a full binary tree or not](#)
5. [Bottom View Binary Tree](#)
6. [Print Nodes in Top View of Binary Tree](#)
7. [Remove nodes on root to leaf paths of length \$< K\$](#)
8. [Lowest Common Ancestor in a Binary Search Tree](#)
9. [Check if a binary tree is subtree of another binary tree](#)
10. [Reverse alternate levels of a perfect binary tree](#)
11. **Red-Black tree**

Algorithms: BST



- [Check for BST](#)
- [Lowest Common Ancestor in a BST](#)
- [Write Code to Determine if Two Trees are Identical or Not](#)
- [Height of Binary Tree](#)
- [Check if given Binary Tree is Height Balanced or Not](#)
- [Serialize and Deserialize a Binary Tree](#)

- [Print Left View of Binary Tree](#)
- [Print Bottom View of Binary Tree](#)
- [Print a Binary Tree in Vertical Order](#)
- [Level order traversal in spiral form](#)
- [Connect Nodes at Same Level](#)
- [Convert a given Binary Tree to Doubly Linked List](#)
- [Given a binary tree, check whether it is a mirror of itself](#)
- [Maximum Path Sum](#)
- [Diameter of a Binary Tree](#)
- [Number of leaf nodes](#)

Algorithms: graph



1. [Breadth First Search \(BFS\)](#)
2. [Depth First Search \(DFS\)](#)
3. [Shortest Path from source to all vertices **Dijkstra**](#)
4. [Shortest Path from every vertex to every other vertex **Floyd Warshall**](#)
5. [To detect cycle in a Graph **Union Find**](#)
6. [Minimum Spanning tree **Prim**](#)
7. [Minimum Spanning tree **Kruskal**](#)
8. [Topological Sort](#)
9. [Boggle \(Find all possible words in a board of characters\)](#)
10. [Bridges in a Graph](#)

Algorithms: Dynamic programming



[Longest Common Subsequence](#)

[Longest Increasing Subsequence](#)

[Edit Distance](#)

[Ways to Cover a Distance](#)

[Longest Path In Matrix](#)

[Optimal Strategy for a Game](#)

[0-1 Knapsack Problem](#)

[Boolean Parenthesization Problem](#)

[Flood fill Algorithm](#)

[Number of paths](#)

[Combination Sum – Part 2](#)

[Special Keyboard](#)

[Water Overflow](#)

[Josephus problem](#)

[Minimum Partition](#)

[Subset Sum Problem](#)



1. [Modular Exponentiation](#)
2. [Modular multiplicative inverse](#)
3. [Primality Test | Set 2 \(Fermat Method\)](#)
4. [Euler's Totient Function](#)
5. [Sieve of Eratosthenes](#)
6. [Convex Hull](#)
7. [Basic and Extended Euclidean algorithms](#)
8. [Segmented Sieve](#)
9. [Chinese remainder theorem](#)
10. [Lucas Theorem](#)

Algorithms: hashing



- [Largest subarray with 0 sum](#)
- [Swapping pairs make sum equal](#)
- [Count distinct elements in every window](#)
- [Array Pair Sum Divisibility Problem](#)
- [Longest consecutive subsequence](#)
- [Array Subset of another array](#)
- [Find all pairs with a given sum](#)
- [Find first repeated character](#)
- [Zero Sum Subarrays](#)
- [Minimum indexed character](#)
- [Check if two arrays are equal or not](#)
- [Uncommon characters](#)
- [Smallest window containing all the characters of another string](#)
- [First element to occur k times](#)
- [Check if frequencies can be equal](#)

Algorithms: bits



1. [Maximum Subarray XOR](#)
 2. [Magic Number](#)
 3. [Sum of bit differences among all pairs](#)
 4. [Swap All Odds And Even Bits](#)
 5. [Find the element that appears once](#)
 6. [Binary representation of a given number](#)
 7. [Count total set bits in all numbers from 1 to n](#)
 8. [Rotate bits of a number](#)
 9. [Count number of bits to be flipped to convert A to B](#)
 10. [Find Next Sparse Number](#)
- [Find first set bit](#)
 - [Rightmost different bit](#)
 - [Check whether K-th bit is set or not](#)
 - [Toggle bits given range](#)
 - [Set kth bit](#)
 - [Power of 2](#)
 - [Bit Difference](#)
 - [Rotate Bits](#)
 - [Swap all odd and even bits](#)
 - [Count total set bits](#)
 - [Longest Consecutive 1's](#)
 - [Sparse Number](#)
 - [Alone in a couple](#)
 - [Maximum subset XOR](#)



- **Recursion vs Dynamic,**
- **greedy algo for partition case**
- [binary-indexed-tree-or-fenwick-tree](#)
- [count-inversions-array-set-3-using-bit](#)

- [Connect Nodes at Same Level + link](#)
- [Count BST nodes that lie in a given range](#)
- [Implement LRU Cache](#)
- [Interleaved Strings](#)
- [Check if a Binary Tree contains duplicate subtrees of size 2 or more](#)
- [Find largest word in dictionary by deleting some characters of given string](#)
- [Modular Exponentiation \(Power in Modular Arithmetic\)](#)
- [Maximum Index](#)
- **How to write an efficient method to calculate x raise to the power n ?**
- **A sorted array is rotated at some unknown point, how to efficiently search an element in it.**