**Questions:-**

* What is an algorithm
* Asymptotic Notations
* When does the worst case of quick sort occur?
* A sorted array is rotated at some unknown point, how to efficiently search an element in it. (Binary Search)
* How to count inversions in an unsorted array?
* Given a big array, how to efficiently find kth largest element in it?
* Given an array of size n with range of numbers from 1 to n+1. The array doesn’t contain any duplicate, one number is missing, find the missing number.
* How to write an efficient method to calculate x raise to the power n?
* Breadth First Search (BFS)
* Depth First Search (DFS)
* Shortest Path from source to all vertices \*\*Dijkstra\*\*
* Shortest Path from every vertex to every other vertex \*\*Floyd Warshall\*\*
* To detect cycle in a Graph \*\*Union Find\*\*
* Minimum Spanning tree \*\*Prim\*\*
* Minimum Spanning tree \*\*Kruskal\*\*
* Topological Sort
* Boggle (Find all possible words in a board of characters)
* Bridges in a Graph
* Insertion of a node in Linked List (On the basis of some constraints)
* Delete a given node in Linked List (under given constraints)
* Compare two strings represented as linked lists
* Add Two Numbers Represented By Linked Lists
* Merge A Linked List Into Another Linked List At Alternate Positions
* Reverse A List In Groups Of Given Size
* Union And Intersection Of 2 Linked Lists
* Detect And Remove Loop In A Linked List
* Merge Sort For Linked Lists
* Select A Random Node from A Singly Linked List
* Binary Search
* 2. Search an element in a sorted and rotated array
* 3. Bubble Sort
* 4. Insertion Sort
* 5. Merge Sort
* 6. Heap Sort (Binary Heap)
* 7. Quick Sort
* 8. Interpolation Search
* 9. Find Kth Smallest/Largest Element In Unsorted Array
* 10. Given a sorted array and a number x, find the pair in array whose sum is closest to x
* 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
* 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
* 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
* Reverse an array without affecting special characters
* 2. All Possible Palindromic Partitions
* 3. Count triplets with sum smaller than a given value
* 4. Convert array into Zig-Zag fashion
* 5. Generate all possible sorted arrays from alternate elements of two given sorted arrays
* 6. Pythagorean Triplet in an array
* 7. Length of the largest subarray with contiguous elements
* 8. Find the smallest positive integer value that cannot be represented as sum of any subset of a given array
* 9. Smallest subarray with sum greater than a given value
* 10. Stock Buy Sell to Maximize Profit
* Next greater element
* Balanced parenthesis
* Stock Span Problem
* Implement a stack using two queues
* Implement a queue using two stacks
* Implement 2 stacks in an array
* Implement k stacks in an array
* Implement a special stack that supports getMin() in O(1) time
* Implement LRU Cache
* Reverse a Stack
* Check if a Binary Tree is BST
* Convert a given Binary Tree to Doubly Linked List
* Inorder Tree Traversal without recursion and without stack
* Level order traversal line by line
* Construct Tree from given Inorder and Preorder traversals
* Construct Full Binary Tree from given preorder and postorder traversals
* Find distance between two nodes of a Binary Tree
* Two nodes of a BST are swapped, correct the BST
* Print Left View of a Binary Tree
* Flatten a binary tree into linked list
* Equilibrium Index
* Subarray with 0 sum
* Subarray with same number of 1s and 0s in a binary array
* Maximum sum of a subarray of size k
* Distinct elements in every window of size k
* Subarray with given sum in an array of positive numbers
* Minimum element in every window of size k
* N-bonacci Numbers
* Longest subsequence of the form 0\*1\*0\* in a binary string
* Longest Span with same Sum in two Binary arrays
* Median in a stream of integers (running integers)
* K’th largest element in a stream
* Sort a nearly sorted (or K sorted) array
* k largest(or smallest) elements in an array
* Merge k sorted arrays
* External Sorting
* Huffman Decoding
* Minimum sum of two numbers formed from digits of an array
* Find smallest range containing elements from k lists
* Check if a given Binary Tree is Heap
* Minimum Number of Platforms Required for a Railway/Bus Station
* Job Scheduling with two jobs allowed at a time
* Prim’s Minimum Spanning Tree (MST)
* Dijkstra’s shortest path algorithm
* Efficient Huffman Coding for Sorted Input
* Fractional Knapsack Problem
* Graph Coloring
* Greedy Algorithm to find Minimum number of Coins
* Minimize Cash Flow among a given set of friends who have borrowed money from each other
* Connect n ropes with minimum cost

**TRIE**

* Find duplicate rows in a binary matrix
* Word Break Problem
* Longest Common Prefix in given set of strings
* Find the k most frequent words from a file
* Find shortest unique prefix for every word in a given list
* Longest prefix matching
* Find maximum XOR of given integer in a stream of integers
* Auto-complete feature using Trie
* Implement a Phone Directory
* Count of distinct substrings of a string using Suffix Trie

**What is Algorithm?**

* An algorithm can be defined as well – defined computational procedure that takes some value or set of values as input and produce some values or set of values as output in order to solve any particular problem.
* For example:- We want to sort any given array.
  + In this case, First of all we will take set of values as input.
  + Then Sort them with the help of any sorting algorithm.
  + Then produce or print the desired result.

**Algorithms must have these properties:-**

* Correctness
* Finiteness
* Feasibility
* Flexibility
* Efficient
* Independent

**Need of Algorithms:-**

* To improve the efficiency of existing techniques.
* Algorithm gives a clear description of requirements and goal of the problem.
* Help us to understand the flow of problem.
* To identify requirement of resource such as memory.

**Complexity of Algorithm**:-

Complexity of algorithm specifies the growth of time/space requirement as a function of input size.

We basically use two complexities

* **Time Complexity: -** Specifies Running time of a program as a function of the input size.
* **Space Complexity: -** Specifies growth of space requirement as a function of input size.

Time Complexity can be further classified into three categories:-

* Worst Case: - Defined by maximum number of steps taken on any instance of size n.
* Best Case: - Define by minimum number of steps taken on any instance of size n.
* Average Case: - Defined by average number of steps taken on any instance of size n.

**Algorithm Design Techniques:-**

* Divide and Conquer approach
* Greedy Technique
* Dynamic Programming
* Branch and Bound
* Randomized Algorithms
* Backtracking Algorithm

**Asymptotic Notations:-**

As we know that write time requirement to execute any algorithm is represented as function of input size. In, Asymptotic Notations, we consider only the important part of this function, important part is the term which affect growth of the function most when size of problem is increased significantly.

For Example:-

Suppose we have a input function f(n) = 3n^2 + 4n + 5.

When value of n is increased by significant amount, n^2 term will be the one which affect function value most. That’s why we consider this term as value of function and all the asymptotic notations work on it.

F(n) = n^2.

**Why we use Asymptotic Notations?**

* They give simple characteristics of an algorithm’s efficiency.
* They allow the comparisons of the performances of various algorithms.

**Various Asymptotic Notations:-**

* Big-oh Notation: - represents the upper bound of an algorithm’s running time. **Expression**
* Omega Notation: - represents the lower bound of an algorithm’s running time. **Expression**
* Theta Notation: - represents the tight bound of an algorithm’s running time. **Expression.**

There are four methods for solving Recurrence:

* Substitution Method
* Iteration Method
* Recursion Tree Method
* Master Method