Interview Questions

1. Can you explain the difference between file structure and storage structure?

- File Structure: Representation of data into secondary or auxiliary memory say any device such as hard disk or pen drives that stores data which remains intact until manually deleted is known as a file structure representation.
- Storage Structure: In this type, data is stored in the main memory i.e RAM, and is deleted once the function that uses this data gets completely executed.
- The difference is that storage structure has data stored in the memory of the computer system, whereas file structure has the data stored in the auxiliary memory.

2. Can you tell how linear data structures differ from non-linear data structures?

- If the elements of a data structure result in a sequence or a linear list then it is called a linear data structure. Whereas, traversal of nodes happens in a non-linear fashion in non-linear data structures.
- Lists, stacks, and queues are examples of linear data structures whereas graphs and trees are the examples of non-linear data structures.

3. What is an array?

- Arrays are the collection of similar types of data stored at contiguous memory locations.
- It is the simplest data structure where the data element can be accessed randomly just by using its index number.

4. What is a multidimensional array?

- Multi-dimensional arrays are those data structures that span across more than one dimension.
- This indicates that there will be more than one index variable for every point of storage. This type of data structure is primarily used in cases where data cannot be represented or stored using only one dimension. Most commonly used multidimensional arrays are 2D arrays.
- 2D arrays emulates the tabular form structure which provides ease of holding the bulk of data that are accessed using row and column pointers.

5. What is a linked list?

A linked list is a data structure that has sequence of nodes where every node is connected to the next node by means of a reference pointer. The elements are not stored in adjacent memory locations. They are linked using pointers to form a chain. This forms a chain-like link for data storage.

- Each node element has two parts:
- a data field
- a reference (or pointer) to the next node.
- The first node in a linked list is called the head and the last node in the list has the pointer to NULL. Null in the reference field indicates that the node is the last node. When the list is empty, the head is a null reference.

6. Are linked lists of linear or non-linear type?

Linked lists can be considered both linear and non-linear data structures. This depends upon the application that they are used for.

- When linked list is used for access strategies, it is considered as a linear data-structure. When they are used for data storage, it can be considered as a non-linear data structure.

7. How are linked lists more efficient than arrays?

1. Insertion and Deletion

- Insertion and deletion process is expensive in an array as the room has to be created for the new elements and existing elements must be shifted.
- But in a linked list, the same operation is an easier process, as we only update the address present in the next pointer of a node.

2. Dynamic Data Structure

  - Linked list is a dynamic data structure that means there is no need to give an initial size at the time of creation as it can grow and shrink at runtime by allocating and deallocating memory.
  - Whereas, the size of an array is limited as the number of items is statically stored in the main memory.

3. No wastage of memory

  - As the size of a linked list can grow or shrink based on the needs of the program, there is no memory wasted because it is allocated in runtime.

- In arrays, if we declare an array of size 10 and store only 3 elements in it, then the space for 3 elements is wasted. Hence, chances of memory wastage is more in arrays.

8. Explain the scenarios where you can use linked lists and arrays.

Following are the scenarios where we use linked list over array:

- When we do not know the exact number of elements beforehand.
- When we know that there would be large number of add or remove operations.
- Less number of random access operations.
- When we want to insert items anywhere in the middle of the list, such as when implementing a priority queue, linked list is more suitable.

Below are the cases where we use arrays over the linked list:

- When we need to index or randomly access elements more frequently.
- When we know the number of elements in the array beforehand in order to allocate the right amount of memory.
- When we need speed while iterating over the elements in the sequence.
- When memory is a concern:

1. Due to the nature of arrays and linked list, it is safe to say that filled arrays use less memory than linked lists.
2. Each element in the array indicates just the data whereas each linked list node represents the data as well as one or more pointers or references to the other elements in the linked list.

- To summarize, requirements of space, time, and ease of implementation are considered while deciding which data structure has to be used over what.

9. What is a doubly-linked list (DLL)? What are its applications.

This is a complex type of a linked list wherein a node has two references:

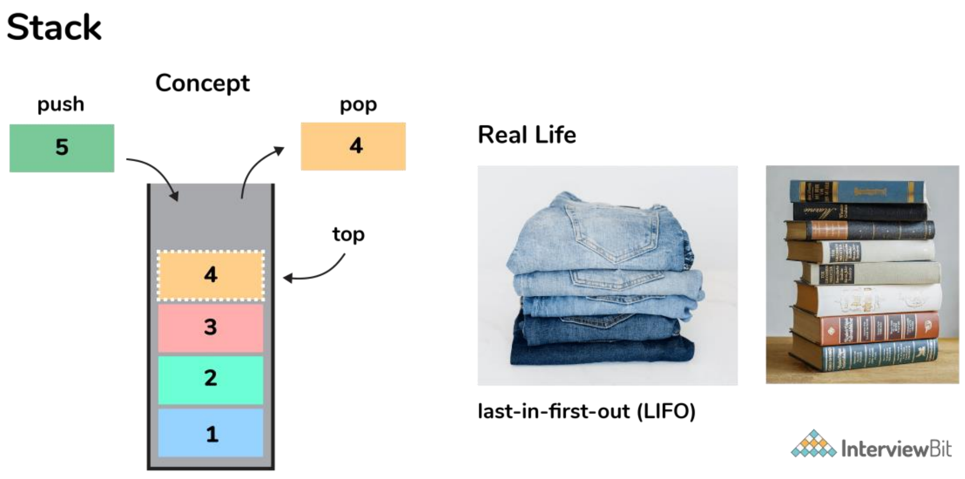
- One that connects to the next node in the sequence
- Another that connects to the previous node.
- This structure allows traversal of the data elements in both directions (left to right and vice versa).

Applications of DLL are:

- A music playlist with next song and previous song navigation options.
- The browser cache with BACK-FORWARD visited pages

10. What is a stack? What are the applications of stack?

- Stack is a linear data structure that follows LIFO (Last In First Out) approach for accessing elements.
- Push, pop, and top (or peek) are the basic operations of a stack.

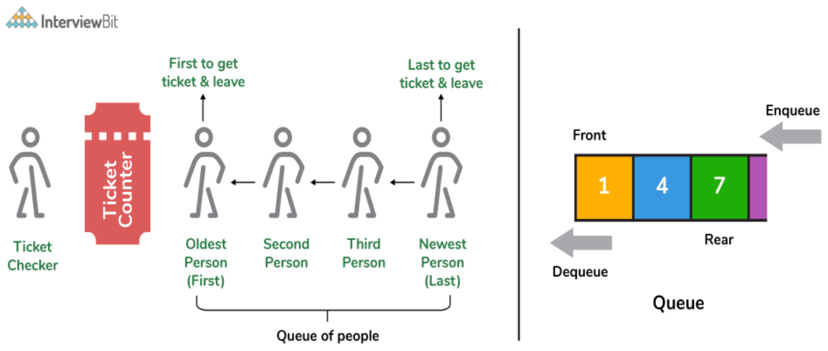


Following are some of the applications of a stack:

- Check for balanced parentheses in an expression
- Evaluation of a postfix expression
- Problem of Infix to postfix conversion
- Reverse a string

11. What is a queue? What are the applications of queue?

- A queue is a linear data structure that follows the FIFO (First In First Out) approach for accessing elements.
- Dequeue from the queue, enqueue element to the queue, get front element of queue, and get rear element of queue are basic operations that can be performed.

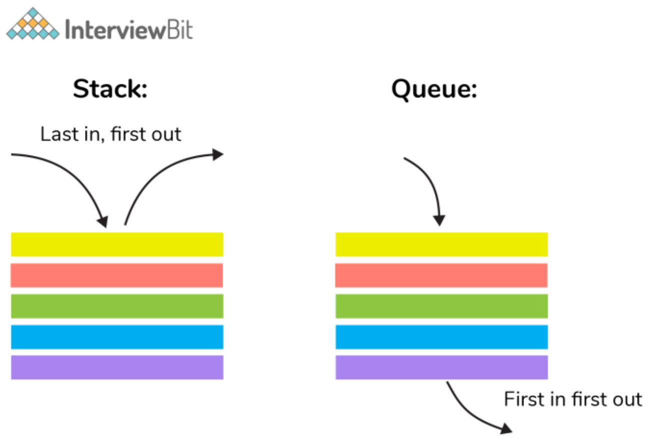


Some of the applications of queue are:

- CPU Task scheduling
- BFS algorithm to find shortest distance between two nodes in a graph.
- Website request processing
- Used as buffers in applications like MP3 media player, CD player, etc.
- Managing an Input stream

12. How is a stack different from a queue?

- In a stack, the item that is most recently added is removed first whereas in queue, the item least recently added is removed first.



13. Explain the process behind storing a variable in memory.

- A variable is stored in memory based on the amount of memory that is needed. Following are the steps followed to store a variable:
- The required amount of memory is assigned first.

- Then, it is stored based on the data structure being used.
- Using concepts like dynamic allocation ensures high efficiency and that the storage units can be accessed based on requirements in real time.

14. How to implement a queue using stack?

- A queue can be implemented using two stacks. Let q be the queue andstack1 and stack2 be the 2 stacks for implementing q. We know that stack supports push, pop, peek operations and using these operations, we need to emulate the operations of queue - enqueue and dequeue. Hence, queue q can be implemented in two methods (Both the methods use auxillary space complexity of O(n)):
  1. **By making enqueue operation costly:**

Here, the oldest element is always at the top of stack1 which ensures dequeue operation to occur in O(1) time complexity.

To place element at top of stack1, stack2 is used.

Pseudocode:

Enqueue: Here time complexity will be O(n)

```
enqueue(q, data):
  While stack1 is not empty:
      Push everything from stack1 to stack2.
      Push data to stack1
      Push everything back to stack1.
```

Dequeue: Here time complexity will be O(1)

```
deQueue(q):
  If stack1 is empty then error
  else
      Pop an item from stack1 and return it
```

2. By making dequeue operation costly:

Here, for enqueue operation, the new element is pushed at the top of stack1. Here, the enqueue operation time complexity is O(1).

In dequeue, if stack2 is empty, all elements from stack1 are moved to stack2 and top of stack2 is the result. Basically, reversing the list by pushing to a stack and returning the first enqueued element. This operation of pushing all elements to new stack takes O(n) complexity.

Pseudocode:

Enqueue: Time complexity: O(1)

```
enqueue(q, data):
  Push data to stack1
```

Dequeue: Time complexity: O(n)

```
dequeue(q):
  If both stacks are empty then raise error.
  If stack2 is empty:
     While stack1 is not empty:
        push everything from stack1 to stack2.
  Pop the element from stack2 and return it.
```

15. How do you implement stack using queues?

A stack can be implemented using two queues. We know that a queue supports enqueue and dequeue operations. Using these operations, we need to develop push, pop operations.

Let stack be ‘s’ and queues used to implement be ‘q1’ and ‘q2’. Then, stack ‘s’ can be implemented in two ways:

By making push operation costly:

This method ensures that newly entered element is always at the front of ‘q1’, so that pop operation just dequeues from ‘q1’.

‘q2’ is used as auxillary queue to put every new element at front of ‘q1’ while ensuring pop happens in O(1) complexity.

Pseudocode:

Push element to stack s : Here push takes O(n) time complexity.

```
push(s, data):
   Enqueue data to q2
   Dequeue elements one by one from q1 and enqueue to q2.
   Swap the names of q1 and q2
```

Pop element from stack s: Takes O(1) time complexity.

```
pop(s):
   dequeue from q1 and return it.
```

By making pop operation costly:

In push operation, the element is enqueued to q1.

In pop operation, all the elements from q1 except the last remaining element, are pushed to q2 if it is empty. That last element remaining of q1 is dequeued and returned.

Pseudocode:

Push element to stack s : Here push takes O(1) time complexity.

```
push(s,data):
   Enqueue data to q1
```

Pop element from stack s: Takes O(n) time complexity.

```
pop(s):
   Step1: Dequeue every elements except the last element from q1 and enqueue to q2.
   Step2: Dequeue the last item of q1, the dequeued item is stored in result variable.
   Step3: Swap the names of q1 and q2 (for getting updated data after dequeue)
   Step4: Return the result.
```

16. What is hashmap in data structure?

Hashmap is a data structure that uses implementation of hash table data structure which allows access of data in constant time (O(1)) complexity if you have the key.

17. What is the requirement for an object to be used as key or value in HashMap?

The key or value object that gets used in hashmap must implement equals() and hashcode() method.

The hash code is used when inserting the key object into the map and equals method is used when trying to retrieve a value from the map.

18. How does HashMap handle collisions in Java?

The java.util.HashMap class in Java uses the approach of chaining to handle collisions. In chaining, if the new values with same key are attempted to be pushed, then these values are stored in a linked list stored in bucket of the key as a chain along with the existing value.

In the worst case scenario, it can happen that all key might have the same hashcode, which will result in the hash table turning into a linked list. In this case, searching a value will take O(n) complexity as opposed to O(1) time due to the nature of the linked list. Hence, care has to be taken while selecting hashing algorithm.

19. What is the time complexity of basic operations get() and put() in HashMap class?

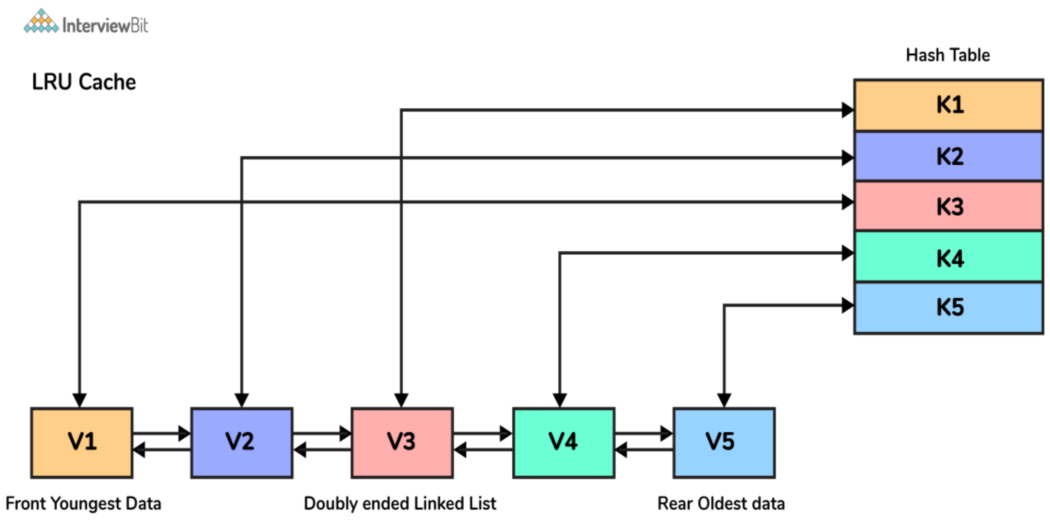
The time complexity is O(1) assuming that the hash function used in hash map distributes elements uniformly among the buckets.

20. Which data structures are used for implementing LRU cache?

LRU cache or Least Recently Used cache allows quick identification of an element that hasn't been put to use for the longest time by organizing items in order of use. In order to achieve this, two data structures are used:

Queue – This is implemented using a doubly-linked list. The maximum size of the queue is determined by the cache size, i.e by the total number of available frames. The least recently used pages will be near the front end of the queue whereas the most recently used pages will be towards the rear end of the queue.

Hashmap – Hashmap stores the page number as the key along with the address of the corresponding queue node as the value.



21. What is a priority queue?

A priority queue is an abstract data type that is like a normal queue but has priority assigned to elements.

Elements with higher priority are processed before the elements with a lower priority.

In order to implement this, a minimum of two queues are required - one for the data and the other to store the priority.

22. Can we store a duplicate key in HashMap?

No, duplicate keys cannot be inserted in HashMap. If you try to insert any entry with an existing key, then the old value would be overridden with the new value. Doing this will not change the size of HashMap.

This is why the keySet() method returns all keys as a SET in Java since it doesn't allow duplicates.

23. What is a tree data structure?

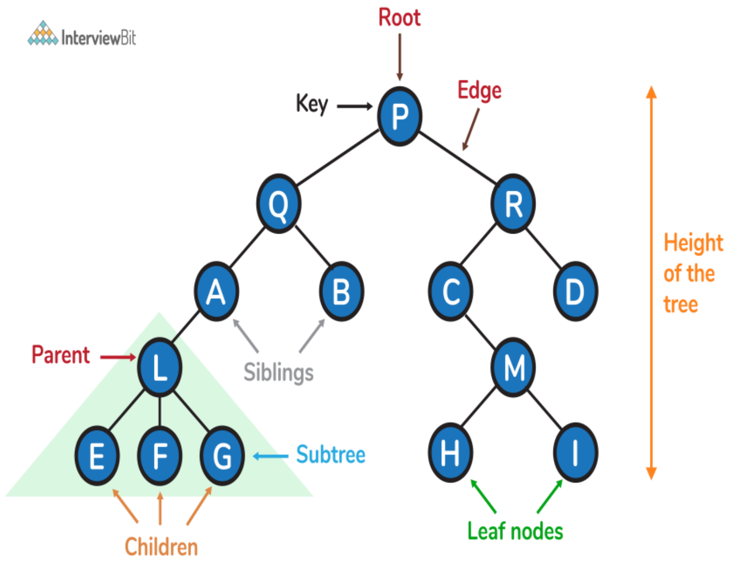
Tree is a recursive, non-linear data structure consisting of the set of one or more data nodes where one node is designated as the root and the remaining nodes are called as the children of the root.

Tree organizes data into hierarchial manner.

The most commonly used tree data structure is a binary tree and its variants.

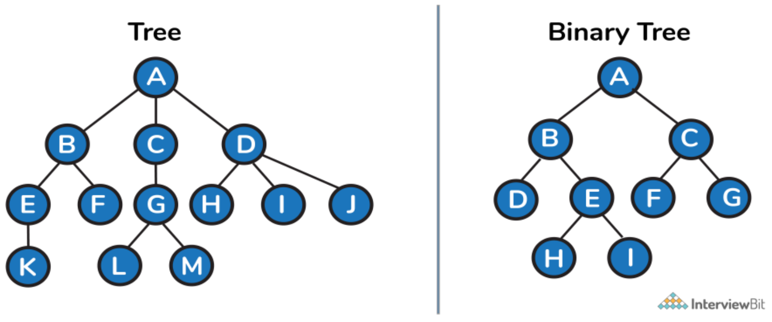
Some of the applications of trees are:

- **Filesystems** —files inside folders that are inturn inside other folders.
- **Comments on social media** — comments, replies to comments, replies to replies etc form a tree representation.
- **Family trees** — parents, grandparents, children, and grandchildren etc that represents the family hierarchy.



24. What are Binary trees?

A binary Tree is a special type of tree where each node can have at most two children. Binary tree is generally partitioned into three disjoint subsets, i.e. the root of the tree, left sub-tree and right sub-tree.



25. What is the maximum number of nodes in a binary tree of height k?

The maximum nodes are : 2k+1-1 where k >= 1

26. Write a recursive function to calculate the height of a binary tree in Java.

Consider that every node of a tree represents a class called Node as given below:

```
 public class Node{
    int data;
    Node left;
    Node right;
 }
```

Then the height of the binary tree can be found as follows:

```
    int heightOfBinaryTree(Node node)
    {
       if (node == null)
          return 0; // If node is null then height is 0 for that node.
       else
       {
          // compute the height of each subtree
          int leftHeight = heightOfBinaryTree(node.left);
          int rightHeight = heightOfBinaryTree(node.right);

          //use the larger among the left and right height and plus 1 (for the root)
          return Math.max(leftHeight, rightHeight) + 1;
       }
    }
```

27. Write Java code to count number of nodes in a binary tree.

```
int countNodes(Node root)
```

```
{
   int count =  1;            //Root itself should be counted
   if (root ==null)
      return 0;
   else
   {
      count += count(root.left);
      count += count(root.right);
      return count;
   }
}
```

28. What are tree traversals?

Tree traversal is a process of visiting all the nodes of a tree. Since root (head) is the first node and all nodes are connected via edges (or links) we always start with that node. There are three ways which we use to traverse a tree −

Inorder Traversal:

Algorithm:

Step 1. Traverse the left subtree, i.e., call Inorder(root.left)

Step 2. Visit the root.

Step 3. Traverse the right subtree, i.e., call Inorder(root.right)

Inorder traversal in Java:

```
    // Print inorder traversal of given tree.
    void printInorderTraversal(Node root)
    {
        if (root == null)

            return;



        //first traverse to the left subtree

        printInorderTraversal(root.left);



        //then print the data of node

        System.out.print(root.data + " ");



        //then traverse to the right subtree

        printInorderTraversal(root.right);

    }
```

Uses: In binary search trees (BST), inorder traversal gives nodes in ascending order.

Preorder Traversal:

Algorithm:

```
Step 1. Visit the root.

Step 2. Traverse the left subtree, i.e., call Preorder(root.left)

Step 3. Traverse the right subtree, i.e., call Preorder(root.right)
```

Preorder traversal in Java:

```
    // Print preorder traversal of given tree.

    void printPreorderTraversal(Node root)

    {

        if (root == null)
```

```
            return;

        //first print the data of node

        System.out.print(root.data + " ");



        //then traverse to the left subtree

        printPreorderTraversal(root.left);



        //then traverse to the right subtree

        printPreorderTraversal(root.right);

    }
```

Uses:

- Preorder traversal is commonly used to create a copy of the tree.
- It is also used to get prefix expression of an expression tree.

Postorder Traversal:

Algorithm:

Step 1. Traverse the left subtree, i.e., call Postorder(root.left)

Step 2. Traverse the right subtree, i.e., call Postorder(root.right)

Step 3. Visit the root.

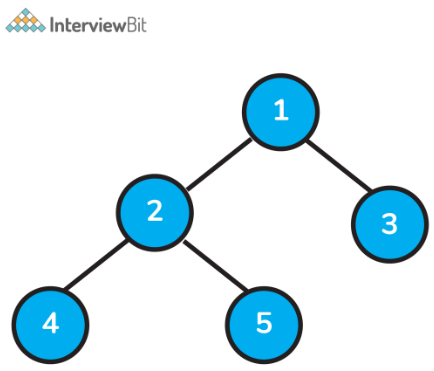
Postorder traversal in Java:

```
// Print postorder traversal of given tree.

void printPostorderTraversal(Node root)

{

  if (root == null)

      return;
  //first traverse to the left subtree
  printPostorderTraversal(root.left);
  //then traverse to the right subtree
  printPostorderTraversal(root.right);
  //then print the data of node
  System.out.print(root.data + " ");
}
```

Uses:

- Postorder traversal is commonly used to delete the tree.
- It is also useful to get the postfix expression of an expression tree.

Consider the following tree as an example, then:



Inorder Traversal => Left, Root, Right : [4, 2, 5, 1, 3]

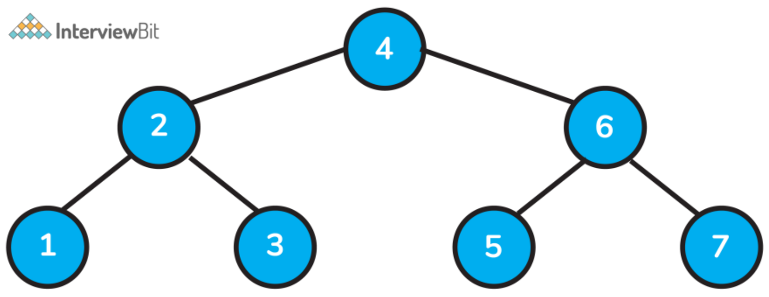
Preorder Traversal => Root, Left, Right : [1, 2, 4, 5, 3]

Postorder Traversal => Left, Right, Root : [4, 5, 2, 3, 1]

29. What is a Binary Search Tree?

A binary search tree (BST) is a variant of binary tree data structure that stores data in a very efficient manner such that the values of the nodes in the left sub-tree are less than the value of the root node, and the values of the nodes on the right of the root node are correspondingly higher than the root.

Also, individually the left and right sub-trees are their own binary search trees at all instances of time.



Example of Binary Search Tree

30. What is an AVL Tree?

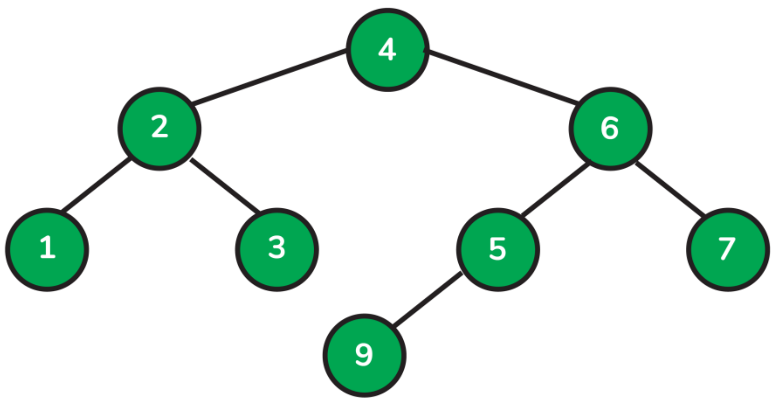
AVL trees are height balancing BST. AVL tree checks the height of left and right sub-trees and assures that the difference is not more than 1. This difference is called Balance Factor and is calculates as. BalanceFactor = height(left subtree) − height(right subtree)

31. Print Left view of any binary trees.

The main idea to solve this problem is to traverse the tree in pre order manner and pass the level information along with it. If the level is visited for the first time, then we store the information of the current node and the current level in the hashmap. Basically, we are getting the left view by noting the first node of every level.

At the end of traversal, we can get the solution by just traversing the map.

Consider the following tree as example for finding the left view:



Left View of this tree: 4,2,1,9

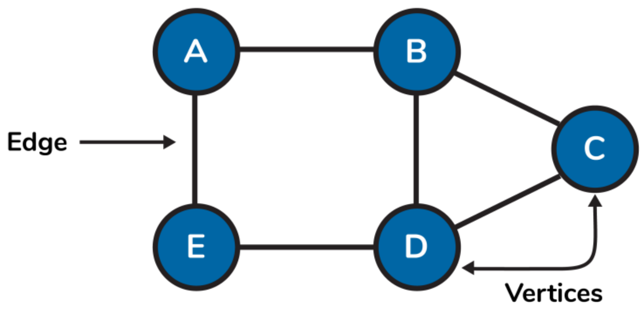
Left view of a binary tree in Java:

```
 import java.util.HashMap;

 //to store a Binary Tree node

 class Node

 {

    int data;

    Node left = null, right = null;

    Node(int data) {

       this.data = data;

    }

 }

 public class InterviewBit

 {

    // traverse nodes in pre-order way

    public static void leftViewUtil(Node root, int level, HashMap<Integer, Integer> map)

    {

       if (root == null) {

          return;
       }
       // if you are visiting the level for the first time
       // insert the current node and level info to the map
       if (!map.containsKey(level)) {
          map.put(level, root.data);
       }
       leftViewUtil(root.left, level + 1, map);
       leftViewUtil(root.right, level + 1, map);
    }
    // to print left view of binary tree
    public static void leftView(Node root)
    {
       // create an empty HashMap to store first node of each level
       HashMap<Integer, Integer> map = new HashMap<>();
       // traverse the tree and find out the first nodes of each level
       leftViewUtil(root, 1, map);
       // iterate through the HashMap and print the left view
       for (int i = 0; i <map.size(); i++) {
          System.out.print(map.get(i) + " ");
       }
    }
    public static void main(String[] args)
    {
       Node root = new Node(4);
       root.left = new Node(2);
       root.right = new Node(6);
       root.left.left = new Node(1);
```

```
        root.left.left = new Node(3);

        root.right.left = new Node(5);

        root.right.right = new Node(7);

        root.right.left.left = new Node(9);

        leftView(root);

    }

 }
```

32. What is a graph data structure?

Graph is a type of non-linear data structure that consists of vertices or nodes connected by edges or links for storing data. Edges connecting the nodes may be directed or undirected.



33. What are the applications of graph data structure?

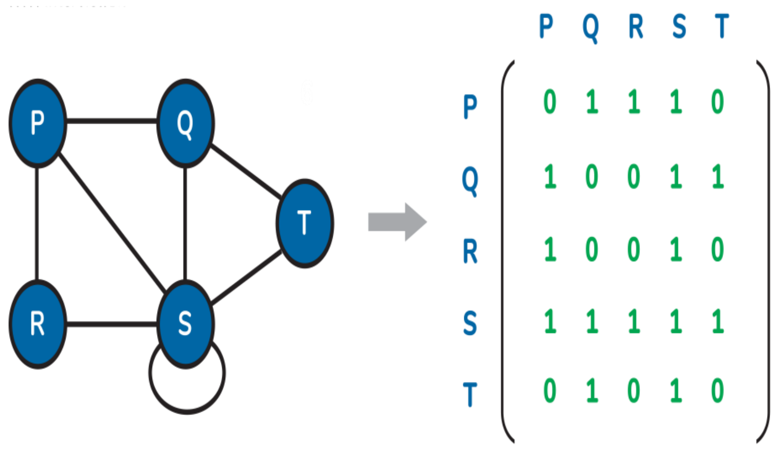
Graphs are used in wide varieties of applications. Some of them are as follows:

- Social network graphs to determine the flow of information in social networking websites like facebook, linkedin etc.
- Neural networks graphs where nodes represent neurons and edge represent the synapses between them
- Transport grids where stations are the nodes and routes are the edges of the graph.
- Power or water utility graphs where vertices are connection points and edge the wires or pipes connecting them.
- Shortest distance between two end points algorithms.

34. How do you represent a graph?

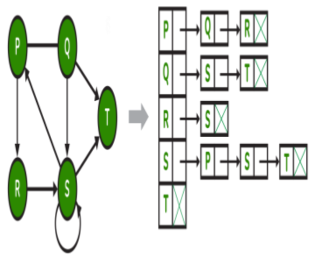
We can represent a graph in 2 ways:

Adjacency matrix: Used for sequential data representation



$$\begin{array}{c|ccccc} & P & Q & R & S & T \\ \hline P & 0 & 1 & 1 & 1 & 0 \\ Q & 1 & 0 & 0 & 1 & 1 \\ R & 1 & 0 & 0 & 1 & 0 \\ S & 1 & 1 & 1 & 1 & 1 \\ T & 0 & 1 & 0 & 1 & 0 \end{array}$$

Adjacency list: Used to represent linked data



35. What is the difference between tree and graph data structure?

Tree and graph are differentiated by the fact that a tree structure must be connected and can never have loops whereas in the graph there are no restrictions.

Tree provides insights on relationship between nodes in a hierarchical manner and graph follows a network model.

36. What is the difference between the Breadth First Search (BFS) and Depth First Search (DFS)?

BFS and DFS both are the traversing methods for a graph. Graph traversal is nothing but the process of visiting all the nodes of the graph.

The main difference between BFS and DFS is that BFS traverses level by level whereas DFS follows first a path from the starting to the end node, then another path from the start to end, and so on until all nodes are visited.

Furthermore, BFS uses queue data structure for storing the nodes whereas DFS uses the stack for traversal of the nodes for implementation.

DFS yields deeper solutions that are not optimal, but it works well when the solution is dense whereas the solutions of BFS are optimal.

37. How do you know when to use DFS over BFS?

The usage of DFS heavily depends on the structure of the search tree/graph and the number and location of solutions needed. Following are the best cases where we can use DFS:

If it is known that the solution is not far from the root of the tree, a breadth first search (BFS) might be better.

If the tree is very deep and solutions are rare, depth first search (DFS) might take an extremely long time, but BFS could be faster.

If the tree is very wide, a BFS might need too much memory, so it might be completely impractical. We go for DFS in such cases.

If solutions are frequent but located deep in the tree we opt for DFS.

38. What is topological sorting in a graph?

Topological sorting is a linear ordering of vertices such that for every directed edge ij, vertex i comes before j in the ordering.

Topological sorting is only possible for Directed Acyclic Graph (DAG).

Applications:

- jobs scheduling from the given dependencies among jobs.
- ordering of formula cell evaluation in spreadsheets
- ordering of compilation tasks to be performed in make files,
- data serialization
- resolving symbol dependencies in linkers.

Topological Sort Code in Java:

```
 // V - total vertices

 // visited - boolean array to keep track of visited nodes

 // graph - adjacency list.

// Main Topological Sort Function.

void topologicalSort()

{

    Stack<Integer> stack = new Stack<Integer>();

      // Mark all the vertices as not visited

    boolean visited[] = new boolean[V];

    for (int j = 0; j < V; j++){

        visited[j] = false;

    }
    // Call the util function starting from all vertices one by one
    for (int i = 0; i < V; i++)
        if (visited[i] == false)
            topologicalSortUtil(i, visited, stack);
     // Print contents of stack -> result of topological sort
    while (stack.empty() == false)
        System.out.print(stack.pop() + " ");
  }
    // A helper function used by topologicalSort
  void topologicalSortUtil(int v, boolean visited[], Stack<Integer> stack)
  {
    // Mark the current node as visited.
    visited[v] = true;
    Integer i;
    // Recur for all the vertices adjacent to the current vertex
    Iterator<Integer> it = graph.get(v).iterator();
    while (it.hasNext()) {
        i = it.next();
        if (!visited[i])
            topologicalSortUtil(i, visited, stack);
    }
    // Push current vertex to stack that saves result
    stack.push(new Integer(v));
  }
```

39. Given an m x n 2D grid map of '1's which represents land and '0's that represents water, return the number of islands (surrounded by water and formed by connecting adjacent lands in 2 directions - vertically or horizontally). Assume that the boundary cases - which is all four edges of the grid are surrounded by water.

Constraints are:

- m == grid.length
- n == grid[i].length
- 1 <= m, n <= 300

grid[i][j] can only be ‘0’ or ‘1’.

Example:
Input: grid = [
[“1” , “1” , “1” , “0” , “0”],
[“1” , “1” , “0” , “0” , “0”],
[“0” , “0” , “1” , “0” , “1”],
[“0” , “0” , “0” , “1” , “1”]
]
Output: 3

Solution:

```
class InterviewBit {

  public int numberOfIslands(char[][] grid) {

    if(grid==null || grid.length==0||grid[0].length==0)

      return 0;

    int m = grid.length;

    int n = grid[0].length;

    int count=0;

    for(int i=0; i<m; i++){

      for(int j=0; j<n; j++){

        if(grid[i][j]=='1'){

          count++;

          mergeIslands(grid, i, j);

        }

      }

    }

    return count;

  }

  public void mergeIslands(char[][] grid, int i, int j){
    int m=grid.length;

    int n=grid[0].length;

    if(i<0||i>=m||j<0||j>=n||grid[i][j]!='1')

      return;

    grid[i][j]='X';

    mergeIslands(grid, i-1, j);

    mergeIslands(grid, i+1, j);

    mergeIslands(grid, i, j-1);

    mergeIslands(grid, i, j+1);

  }

}
```

40. What is a heap data structure?

Heap is a special tree-based non-linear data structure in which the tree is a complete binary tree. A binary tree is said to be complete if all levels are completely filled except possibly the last level and the last level has all elements towards as left as possible. Heaps are of two types:

Max-Heap:

In a Max-Heap the data element present at the root node must be greatest among all the data elements present in the tree.

This property should be recursively true for all sub-trees of that binary tree.

Min-Heap:

In a Min-Heap the data element present at the root node must be the smallest (or minimum) among all the data elements present in the tree.

This property should be recursively true for all sub-trees of that binary tree.

MCQ Questions

1. Which of the following data structure can't store the non-homogeneous data elements?

Arrays Records Pointers Stacks

2. A directed graph is _______ if there is a path from each vertex to every other vertex in the graph.

Weakly connected Strongly connected

Tightly connected Linearly connected

3. In what traversal we process all of a vertex's descendants before we move to an adjacent vertex?

BFS DFS Level order Width first

4. In circular queue, the value of REAR would be?

REAR = REAR + 1 REAR = (REAR + 1) % (QUEUE_SIZE+1)

REAR = (REAR + 1) % (QUEUE_SIZE) REAR = (REAR - 1) % (QUEUE_SIZE-1)

5. Which of the following statement is true?

Statement i) Using singly linked lists and circular list, it is not possible to traverse the list backwards.

Statement ii) To find the predecessor, it is required to traverse the list from the first node in case of singly linked list.

i only ii only both i and ii none of the above

6. The binary search method needs no more than ________ comparisons.

(log2n) + 1 logn (logn) + 1 log2n

7. Which of the following are the properties of a binary tree?

The first subset is called left subtree

The second subtree is called right subtree

The root cannot contain NULL

The right subtree can be empty

8. Which of the following scenario is true for the statement - "Arrays are best data structures"?

a. For the size of the structure and the data in the structure are constantly changing

b. For relatively permanent collections of data

both a and b none of the above

9. Which of the following code snippet is used to convert decimal to binary numbers?

```
public void convertBinary(int num)
{
int bin[] = new int[50];
int index = 0;
while(num > 0)
{
bin[index++] = num%2;
num = num/2;
}
for(int i = index-1;i >= 0;i--)
{
System.out.print(bin[i]);
}
}
public void convertBinary(int num)
 {
    int bin[] = new int[50];
    int index = 0;
    while(num > 0)
    {
       bin[++index] = num/2;
       num = num%2;
    }
    for(int i = index-1;i >= 0;i--)
    {
       System.out.print(bin[i]);
    }
  }
public void convertBinary(int num)
{
    int bin[] = new int[50];
    int index = 0;
    while(num > 0)
    {
       bin[index++] = num/2;
       num = num%2;
    }
    for(int i = index-1;i >= 0;i--)
    {
       System.out.print(bin[i]);
    }
}
public void convertBinary(int num)
{
    int bin[] = new int[50];
    int index = 0;
    while(num > 0)
    {
     bin[++index] = num%2;
     num = num/2;
    }
    for(int i = index-1;i >= 0;i--)
    {
     System.out.print(bin[i]);
    }
}
```

10. What will be the final elements on the stack if the following sequence of operations are

executed?

* Push(a,s);   * Push(b,s);   * Pop(s);   * Push(c,s);

- where a, b, c are the data elements and s is the stack.

Abc   ac   acb   b

11. Dijkstra's Algorithm cannot be applied on which of the following?

Directed and weighted graphs
Graphs having negative weight function
Unweighted graphs
Undirected and unweighted graphs

What is a Data Structure?
A data structure is a way of organizing the data so that the data can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for the implementation of databases, while compiler implementations usually use hash tables to look up identifiers.

What are linear and non-linear data Structures?

Linear: A data structure is said to be linear if its elements form a sequence or a linear list. Examples: Array. Linked List, Stacks and Queues

Non-Linear: A data structure is said to be non-linear if the traversal of nodes is nonlinear in nature. Example: Graph and Trees.

What are the various operations that can be performed on different Data Structures?

Insertion : Add a new data item in the given collection of data items.

Deletion : Delete an existing data item from the given collection of data items.

Traversal : Access each data item exactly once so that it can be processed.

Searching : Find out the location of the data item if it exists in the given collection of data items.

Sorting : Arranging the data items in some order i.e. in ascending or descending order in case of numerical data and in dictionary order in case of alphanumeric data.

How is an Array different from Linked List?

The size of the arrays is fixed, Linked Lists are Dynamic in size.

Inserting and deleting a new element in an array of elements is expensive, Whereas both insertion and deletion can easily be done in Linked Lists.

Random access is not allowed in Linked Listed.

Extra memory space for a pointer is required with each element of the Linked list.

Arrays have better cache locality that can make a pretty big difference in performance.

What is Stack and where it can be used?

Stack is a linear data structure which the order LIFO(Last In First Out) or FILO(First In Last Out) for accessing elements. Basic operations of the stack are: Push, Pop, Peek

Applications of Stack:

- Infix to Postfix Conversion using Stack
- Evaluation of Postfix Expression
- Reverse a String using Stack
- Implement two stacks in an array
- Check for balanced parentheses in an expression

What is a Queue, how it is different from the stack and how is it implemented?

Queue is a linear structure that follows the order is First In First Out (FIFO) to access elements. Mainly the following are basic operations on queue: Enqueue, Dequeue, Front, Rear The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added. Both Queues and Stacks can be implemented using Arrays and Linked Lists.

What are Infix, prefix, Postfix notations?

Infix notation: X + Y – Operators are written in-between their operands. This is the usual way we write expressions. An expression such as

A * (B + C) / D

Postfix notation (also known as "Reverse Polish notation"): X Y + Operators are written after their operands. The infix expression given above is equivalent to

A B C + * D/

Prefix notation (also known as "Polish notation"): + X Y Operators are written before their operands. The expressions given above are equivalent to

/ A * + B C D

What is a Linked List and What are its types?

A linked list is a linear data structure (like arrays) where each element is a separate object. Each element (that is node) of a list is comprising of two items – the data and a reference to the next node.Types of Linked List :

- **Singly Linked List :** In this type of linked list, every node stores address or reference of next node in list and the last node has next address or reference as NULL. For example 1->2->3->4->NULL
- **Doubly Linked List :** Here, here are two references associated with each node, One of the reference points to the next node and one to the previous node. Eg. NULL<-1<->2<->3->NULL
- **Circular Linked List :** Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list. Eg. 1->2->3->1

Which data structures are used for BFS and DFS of a graph?

Queue is used for BFS

Stack is used for DFS. DFS can also be implemented using recursion (Note that recursion also uses function call stack).

Can doubly linked be implemented using a single pointer variable in every node?
Doubly linked list can be implemented using a single pointer.

How to implement a stack using queue?

A stack can be implemented using two queues. Let stack to be implemented be ‘s’ and queues used to implement be ‘q1’ and ‘q2’. Stack ‘s’ can be implemented in two ways:

Method 1 (By making push operation costly)

Method 2 (By making pop operation costly)

How to implement a queue using stack?

A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

Method 1 (By making enQueue operation costly)

Method 2 (By making deQueue operation costly)

Which Data Structure Should be used for implementing LRU cache?

We use two data structures to implement an LRU Cache.

Queue which is implemented using a doubly linked list. The maximum size of the queue will be equal to the total number of frames available (cache size). The most recently used pages will be near rear end and least recently pages will be near front end.

A Hash with page number as key and address of the corresponding queue node as value.

How to check if a given Binary Tree is BST or not?

If inorder traversal of a binary tree is sorted, then the binary tree is BST. The idea is to simply do inorder traversal and while traversing keep track of previous key value. If current key value is greater, then continue, else return false.

Linked List Questions

- Linked List Insertion
- Linked List Deletion
- middle of a given linked list
- Nth node from the end of a Linked List

Tree Traversal Questions

- Inorder
- Preorder and Postoder Traversals
- Level order traversal
- Height of Binary Tree

Convert a DLL to Binary Tree in-place

- Convert Binary Tree to DLL in-place
- Delete a given node in a singly linked list
- Reverse a Linked List
- Detect Loop in a Linked List

Which data structure is used for dictionary and spell checker?

Arrays

An array is collection of items stored at continuous memory locations. The idea is to declare multiple items of same type together. Array declaration: In C, we can declare an array by specifying its and size or by initializing it or by both.

// Array declaration by specifying size

```
int arr[10];

// Array declaration by initializing elements

int arr[] = {10, 20, 30, 40};

// Array declaration by specifying size and

// initializing elements

int arr[6] = {10, 20, 30, 40}
```

Formulas:

Length of Array = UB - LB + 1

Given the address of first element, address of any other element is calculated using the formula:-

Loc (arr [k]) = base (arr) + w * k

w = number of bytes per storage location

of for one element

k = index of array whose address we want

to calculate

Elements of two-dimensional arrays (mXn) are stored in two ways:-

Column major order: Elements are stored column by column, i.e. all elements of first column are stored, and then all elements of second column stored and so on.

Loc(arr[i][j]) = base(arr) + w (m *j + i)

Row major order: Elements are stored row by row, i.e. all elements of first row are stored, and then all elements of second row stored and so on.

Loc(arr[i][j]) = base(arr) + w (n*i + j)

Stacks

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

Basic operations :

Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition. (Top=Top+1) Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.(Top=Top-

1)

Peek: Get the topmost item.

Infix, prefix, Postfix notations

Infix notation: X + Y – Operators are written in-between their operands. This is the usual way we write expressions. An expression such as

A * (B + C) / D

Postfix notation (also known as “Reverse Polish notation”): X Y + Operators are written after their operands. The infix expression given above is equivalent to

A B C + * D/

Prefix notation (also known as “Polish notation”): + X Y Operators are written before their operands. The expressions given above are equivalent to

/ * A + B C D

Converting between these notations:

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

1) Only one disk can be moved at a time.

2) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

3) No disk may be placed on top of a smaller disk.

For n disks, total 2n – 1 moves are required

Time complexity : O(2n) [exponential time]

Queues

Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first. Stack : Remove the item the most recently added Queue: Remove the item the least recently added Operations on Queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue.

Linked Lists

Linked List is a linear data structure. Unlike arrays, linked list elements are not stored at contiguous location; the elements are linked using pointers.

Advantages over arrays

1) Dynamic size

2) Ease of insertion/deletion

Drawbacks:

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do binary search with linked lists.

2) Extra memory space for a pointer is required with each element of the list. Representation in C: A linked list is represented by a pointer to the first node of the linked list. The first node is called head. If the linked list is empty, then value of head is NULL.

Each node in a list consists of at least two parts:

1) data

2) pointer to the next node In C, we can represent a node using structures. Below is an example of a linked list node with an integer data.

```
// A linked list node

struct node
{
  int data;
  struct node *next;
};
```

1. What is a Data Structure?

The Data Structure is the way data is organized (stored) and manipulated for retrieval and access. It also defines the way different sets of data relate to one another, establishing relationships and forming algorithms.

2. What is a Linear Data Structure? Name a few examples.

A data structure is linear if all its elements or data items are arranged in a sequence or a linear order. The elements are stored in a non-hierarchical way so that each item has successors and predecessors except the first and last element in the list.

Examples of linear data structures are Arrays, Stack, Strings, Queue, and Linked List.

3. What are some applications of Data Structures?

Numerical analysis, operating system, AI, compiler design, database management, graphics, statistical analysis, and simulation.

4. What is the difference between file structure and storage structure?

The difference lies in the memory area accessed. Storage structure refers to the data structure in the memory of the computer system, whereas file structure represents the storage structure in the auxiliary memory.

5. What is a linked list Data Structure?

This is one of the most frequently asked data structure interview questions where the interviewer expects you to give a thorough answer. Try to explain as much as possible rather than finishing your answer in a sentence!

It’s a linear Data Structure or a sequence of data objects where elements are not stored in adjacent memory locations. The elements are linked using pointers to form a chain. Each element is a separate object, called a node. Each node has two items: a data field and a reference to the next node. The entry point in a linked list is called the head. Where the list is empty, the head is a null reference and the last node has a reference to null.

A linked list is a dynamic data structure, where the number of nodes is not fixed, and the list has the ability to grow and shrink on demand.

It is applied in cases where:

We deal with an unknown number of objects or don’t know how many items are in the list

We need constant-time insertions/deletions from the list, as in real-time computing where time predictability is critical

Random access to any elements is not needed

The algorithm requires a data structure where objects need to be stored irrespective of their physical address in memory

We need to insert items in the middle of the list as in a priority queue

Some implementations are stacks and queues, graphs, directory of names, dynamic memory allocation, and performing arithmetic operations on long integers.

6. Are linked lists considered linear or non-linear Data Structures?

Linked lists are considered both linear and non-linear data structures depending upon the application they are used for. When used for access strategies, it is considered as a linear data-structure. When used for data storage, it is considered a non-linear data structure.

7. What are the advantages of a linked list over an array? In which scenarios do we use Linked List and when Array?

This is another frequently asked data structure interview question! Advantages of a linked list over an array are:

1. Insertion and Deletion

Insertion and deletion of nodes is an easier process, as we only update the address present in the next pointer of a node. It’s expensive to do the same in an array as the room has to be created for the new elements and existing elements must be shifted.

2. Dynamic Data Structure

As a linked list is a dynamic data structure, there is no need to give an initial size as it can grow and shrink at runtime by allocating and deallocating memory. However, the size is limited in an array as the number of elements is statically stored in the main memory.

3. No Wastage of Memory

As the size of a linked list can increase or decrease depending on the demands of the program, and memory is allocated only when required, there is no memory wasted. In the case of an array, there is memory wastage. For instance, if we declare an array of size 10 and store only five elements in it, then the space for five elements is wasted.

4. Implementation

Data structures like stack and queues are more easily implemented using a linked list than an array.

Some scenarios where we use linked list over array are:

- When we know the upper limit on the number of elements in advance
- When there are a large number of add or remove operations
- When there are no large number of random access to elements
- When we want to insert items in the middle of the list, such as when implementing a priority queue

Some scenarios in which we use array over the linked list are:

- When we need to index or randomly access elements
- When we know the number of elements in the array beforehand, so we can allocate the correct amount of memory
- When we need speed when iterating through all the elements in the sequence
- When memory is a concern; filled arrays use less memory than linked lists, as each element in the array is the data but each linked list node requires the data as well as one or more pointers to the other elements in the linked list

8. What is a doubly-linked list? Give some examples.

It is a complex type (double-ended LL) of a linked list in which a node has two links, one that connects to the next node in the sequence and another that connects to the previous node. This allows traversal across the data elements in both directions.

Examples include:

A music playlist with next and previous navigation buttons

The browser cache with BACK-FORWARD visited pages

The undo and redo functionality on a browser, where you can reverse the node to get to the previous page

9. How do you reference all of the elements in a one-dimension array?

All of the elements in a one-dimension array can be referenced using an indexed loop as the array subscript so that the counter runs from 0 to the array size minus one.

10. What are dynamic Data Structures? Name a few.

They are collections of data in memory that expand and contract to grow or shrink in size as a program runs. This enables the programmer to control exactly how much memory is to be utilized.

Examples are the dynamic array, linked list, stack, queue, and heap.

11. What is an algorithm?

An algorithm is a step by step method of solving a problem or manipulating data. It defines a set of instructions to be executed in a certain order to get the desired output.

12. Why do we need to do an algorithm analysis?

A problem can be solved in more than one way using several solution algorithms. Algorithm analysis provides an estimation of the required resources of an algorithm to solve a specific computational problem. The amount of time and space resources required to execute is also determined.

The time complexity of an algorithm quantifies the amount of time taken for an algorithm to run as a function of the length of the input. The space complexity quantifies the amount of space or memory taken by an algorithm, to run as a function of the length of the input.

13. What is a stack?

A stack is an abstract data type that specifies a linear data structure, as in a real physical stack or piles where you can only take the top item off the stack in order to remove things. Thus, insertion (push) and deletion (pop) of items take place only at one end called top of the stack, with a particular order: LIFO (Last In First Out) or FILO (First In Last Out).

14. Where are stacks used?

Expression, evaluation, or conversion of evaluating prefix, postfix, and infix expressions

- Syntax parsing
- String reversal
- Parenthesis checking
- Backtracking

15. What is a queue Data Structure?

In this data structure interview question, you can also discuss your experience and situations using queue. A queue is an abstract data type that specifies a linear data structure or an ordered list, using the First In First Out (FIFO) operation to access elements. Insert operations can be performed only at one end called REAR and delete operations can be performed only at the other end called FRONT.

16. List some applications of queue Data Structure.

To prioritize jobs as in the following scenarios:

As waiting lists for a single shared resource in a printer, CPU, call center systems, or image uploads; where the first one entered is the first to be processed

In the asynchronous transfer of data; or example pipes, file IO, and sockets

As buffers in applications like MP3 media players and CD players

To maintain the playlist in media players (to add or remove the songs)

17. What is a Dequeue?

It is a double-ended queue, or a data structure, where the elements can be inserted or deleted at both ends (FRONT and REAR).

18. What operations can be performed on queues?

- enqueue() adds an element to the end of the queue
- dequeue() removes an element from the front of the queue
- init() is used for initializing the queue
- isEmpty tests for whether or not the queue is empty

The front is used to get the value of the first data item but does not remove it

The rear is used to get the last item from a queue

19. What are the advantages of the heap over a stack?

In this data structure interview questions, try giving various advantages, along with examples, if possible. It will show the interviewer your domain expertise. Generally, both heap and stack are part of memory and used in Java for different needs:

Heap is more flexible than the stack because memory space can be dynamically allocated and de-allocated as needed

Heap memory is used to store objects in Java, whereas stack memory is used to store local variables and function call

Objects created in the heap are visible to all threads, whereas variables stored in stacks are only visible to the owner as private memory

When using recursion, the size of heap memory is more whereas it quickly fill-ups stack memory

20. Where can stack Data Structure be used?

- Expression evaluation
- Backtracking
- Memory management
- Function calling and return

21. What is the difference between a PUSH and a POP?

The acronyms stand for Pushing and Popping operations performed on a stack. These are ways data is stored and retrieved.

PUSH is used to add an item to a stack, while POP is used to remove an item.

PUSH takes two arguments, the name of the stack to add the data to and the value of the entry to be added. POP only needs the name of the stack.

When the stack is filled and another PUSH command is issued, you get a stack overflow error, which means that the stack can no longer accommodate the last PUSH. In POP, a stack underflow error occurs when you're trying to POP an already empty stack.

22. Which sorting algorithm is considered the fastest? Why?

A single sorting algorithm can’t be considered best, as each algorithm is designed for a particular data structure and data set. However, the QuickSort algorithm is generally considered the fastest because it has the best performance for most inputs.

Its advantages over other sorting algorithms include the following:

Cache-efficient: It linearly scans and linearly partitions the input. This means we can make the most of every cache load.

Can skip some swaps: As QuickSort is slightly sensitive to input that is in the right order, it can skip some swaps.

Efficient even in worst-case input sets, as the order is generally random.

Easy adaption to already- or mostly-sorted inputs.

When speed takes priority over stability.

23. What is the merge sort? How does it work?

Merge sort is a divide-and-conquer algorithm for sorting the data. It works by merging and sorting adjacent data to create bigger sorted lists, which are then merged recursively to form even bigger sorted lists until you have one single sorted list.

24. How does the Selection sort work?

Selection sort works by repeatedly picking the smallest number in ascending order from the list and placing it at the beginning. This process is repeated moving toward the end of the list or sorted subarray.

Scan all items and find the smallest. Switch over the position as the first item. Repeat the selection sort on the remaining N-1 items. We always iterate forward (i from 0 to N-1) and swap with the smallest element (always i).

Time complexity: best case O(n2); worst O(n2)

Space complexity: worst O(1)

25. Define the graph Data Structure?

It is a type of non-linear data structure that consists of vertices or nodes connected by edges or arcs to enable storage or retrieval of data. Edges may be directed or undirected.

26. What are the applications of graph Data Structure?

Transport grids where stations are represented as vertices and routes as the edges of the graph

Utility graphs of power or water, where vertices are connection points and edge the wires or pipes connecting them

Social network graphs to determine the flow of information and hotspots (edges and vertices)

Neural networks where vertices represent neurons and edge the synapses between them

27. What are the advantages of binary search over a linear search?

In a sorted list:

A binary search is more efficient than a linear search because we perform fewer comparisons. With linear search, we can only eliminate one element per comparison each time we fail to find the value we are looking for, but with the binary search, we eliminate half the set with each comparison.

Binary search runs in O(log n) time compared to linear search's O(n) time. This means that the more of the elements present in the search array, the faster is binary search compared to a linear search.

28. What is an AVL tree?

An AVL (Adelson, Velskii, and Landi) tree is a height balancing binary search tree in which the difference of heights of the left and right subtrees of any node is less than or equal to one. This controls the height of the binary search tree by not letting it get skewed. This is used when working with a large data set, with continual pruning through insertion and deletion of data.

29. Differentiate NULL and VOID

- Null is a value, whereas Void is a data type identifier
- Null indicates an empty value for a variable, whereas void indicates pointers that have no initial size
- Null means it never existed; Void means it existed but is not in effect

30. Do dynamic memory allocations help in managing data? How?

Dynamic memory allocation stores simple structured data types at runtime. It has the ability to combine separately allocated structured blocks to form composite structures that expand and contract as needed, thus helping manage data of data blocks of arbitrary size, in arbitrary order.

Know where you stand in your data science preparation and work toward filling the knowledge gap with the Data Science with R Practice Test.

31. Name the ways to determine whether a linked list has a loop.

- Using hashing
- Using the visited nodes method (with or without modifying the basic linked list data structure)
- Floyd's cycle-finding algorithm

32. List some applications of multilinked structures?

- Sparse matrix
- Index generation

33. Explain the jagged array.

It is an array whose elements themselves are arrays and may be of different dimensions and sizes.

34. Explain the max heap Data Structure.

It is a type of heap data structure where the value of the root node is greater than or equal to either of its child nodes.

35. How do you find the height of a node in a tree?

The height of the node equals the number of edges in the longest path to the leaf from the node, where the depth of a leaf node is 0.

Top Data Structures and Algorithms Interview Q&As

What do you understand about 'Data Structures'?

Data Structures can be defined as techniques used to define, store, and access data systematically. They form the most important component of any algorithm. Depending on the type of Data Structures, they store different kinds of data and are accessible in different ways. For an algorithm to return a result, it needs to operate on and manipulate a set of data structures in an organised and efficient manner to come to the final result.

How can you differentiate between a File Structure and a Data Structure?

In File Structures, the data is stored on disks following standard file storage policies and is not compatible with external, third-party applications. In Data Structures, on the other hand, the data is stored both on the disk as well as RAM in customised storage policies, and these are highly compatible with external apps.

What are the broad types of data structures?

Data Structures can be broadly divided into two categories:

- Linear: In this, all the elements are stored sequentially, and retrieval takes place linearly. The arrangement is non-hierarchical, and each element has one successor and one predecessor. Example – Arrays, Linked Lists, Stacks, Queues, etc.
- Non-linear: Here, the storage does not happen in a linear sequence – i.e., all elements don't necessarily have just one successor and predecessor. Instead, elements in non-linear Data Structures are connected to two or more items in a non-linear manner. Example – Trees, Graphs, Heaps.

4. What are some key usage areas of Data Structures?

Data Structures are pretty much required in all the fields of computing that you can think of, especially Algorithms and Algorithm Optimization. Here are some other areas where Data Structures are extensively used:

- Operating system design
- Numerical analysis
- Machine Learning and AI
- Compiler design and development
- Database management
- Lexical analysis
- Graphical programming
- Searching and sorting algorithms, and more.

Explain the Stack Data Structure and mention its usage areas.

Stack is simply an ordered list that allows insertion and deletion only from one of the ends – which is known as the 'top'. It is a recursive Data Structure that has a pointer to its 'top' elements which lets us know about the topmost element of the Stack. Based on the element retrieval strategy, Stack is also known as Last-In-First-Out, since the last element pushed into the stack will be at the top, and will be the first to be popped out. Here are some uses of the Stack Data Structure:

- Backtracking
- Memory Management
- Function return and calling
- Expression evaluation

What are the operations that can be performed on a stack?

The Stack Data Structure supports the following three operations:

- push() — to insert an element into the top position of the Stack.
- pop() — to bring one element out from the top of the Stack.
- peek() — to just check the element present on the top of the Stack without bringing it out of the Stack.

What do you understand about Postfix Expressions?

Postfix Expression is an expression where operators follow the operands. This is extremely beneficial in computing terms since it doesn't require any grouping of sub-expressions into parenthesis or even consider operator precedence. The expression a+b is simply represented as ab+ in postfix.

How are 2D array elements stored in the memory?

The elements of a 2-D array can be stored in the memory in either of the two ways:

- **Row-Major**: In this method, first all the rows of the array are contiguously stored in the memory. First, the 1st row is stored completely, then the 2nd row, and so on till the last one.
- **Column-Major:** In this, all the columns of the array are continuously stored in the memory. First, the 1st column is stored completely, then the 2nd column, and so on.

Define Linked List Data Structure.

Linked Lists are collections of nodes – which are randomly stored objects. Each node has two **internal elements** – a Data field and a Link field. The Data field holds the value that the particular node has, while the Link field has a pointer to the next node that this one is linked to. Depending on the situation, a Linked List can be considered both as a linear as well as a non-linear Data Structure.

In what ways are Linked Lists better than arrays?

Linked Lists are better than arrays in the following ways:

- Array sizes are fixed at run-time and can't be modified later, but Linked Lists can be expanded in real-time, as per the requirements.
- Linked Lists are not stored contiguously in the memory, as a result, they are a lot more memory efficient than arrays that are statically stored.
- The number of elements that can be stored in any Linked List are limited to only the available memory space, while the number of elements is bound by the size of the array.

In C programming language, which pointer would you use to implement a heterogeneous linked list?

Heterogeneous linked lists, as the name suggests, hold different data types. As a result, it is not possible to use ordinary pointers here. So, Void pointers are normally used in such a scenario since they are capable of pointing to any type of value.

What is a Doubly Linked List?

As the name suggests, a Doubly Linked List is a Linked List which has nodes linked to both the succeeding and preceding nodes. As a result, the nodes of Doubly Linked List have three, not two, fields:

- The Data Field
- Next pointer (for pointing the next node)
- Previous pointer (for pointing the previous node)

Explain the Queue Data Structure with some of its applications.

A Queue is an ordered list that allows for insertion and deletion of elements from not one but two ends – called REAR and FRONT. The insertion takes place from the FRONT end while the deletion from the REAR end. As a result of this, Queue is often called First-In-First-Out (FIFO). Here are some widespread applications of Queues as a Data Structure:

- For waiting lists for singly shared resources like CPU, printer, disk, etc.
- For asynchronous transfer of data, example file IO, sockets, pipes.
- As buffers in most of the media player applications.
- In Operating Systems for handling interruptions.

Can you list some drawbacks of implementing Queues using arrays?

There are mainly two drawbacks that occur when implementing Queues with arrays:

Mismanagement of memory, since arrays are static data structures so implementing Queues with arrays removes a lot of functionalities of Queues.

Problem with size, since array sizes are defined during array definition. So, if we want to add more elements to our queue than the size of the array, it won’t be possible.

What conditions should be fulfilled in order for an element to be inserted into a circular queue?

Here are some relevant conditions regarding insertion into circular queues:

If (rear + 1)%maxsize == front -> this means the queue is full -> no more insertion possible.

If rear != max – 1, the value of rear gets incremented to maxsize and a new value will be inserted at the rear end.

If front != 0 and rear == max -1 –> this means that the queue is not full. So, the value of rear gets set to 0, and a new element is inserted into the rear end of the circular queue.

16. What is a dequeue?

Double-Ended Queue or deque is an ordered set of elements that facilitates insertion and deletion from both the ends – rear and front. As a result of which, this is even more flexible than the queue data structure.

Define the Tree Data Structure and list some types of trees.

Tree is a non-linear, recursive data structure that contains various nodes. One particular node is designated as the root of the tree from where all other nodes emerge. Apart from root, all the other nodes are called child nodes. There are broadly the following types of Tree Data Structures:

- General Trees
- Binary Trees
- Binary Search Trees
- Forests
- Expression Tree
- Tournament Tree

How does bubble sort work?

Bubble Sort is one of the most used sorting algorithms, and is used with arrays by comparing adjacent elements and exchanging their positions based on their values. It’s called bubble sort because the visualization of how it works is like bubbles floating from top of the water and larger entities sinking down.

Which is the fastest sorting algorithm available?

There are many different sorting algorithms available, like merge sort, quick sort, bubble sort, and more. Out of these, we can’t pick one specific algorithm which is objectively the fastest since their performance varies greatly based on the input data, the reaction after the algorithm has processed the data, and how it’s stored.

What are Binary Trees?

Binary Trees are special types of trees in which each node can have AT MOST two children. To make things easier, Binary Trees are generally divided into three disjoint sets — root node, right subtree, and left subtree.

How can AVL Trees be used in various operations as compared to BST?

AVL trees are height-balanced trees, so they don’t allow for the tree to get skewed from any one side. The time taken for all the operations performed on BST of height h is O(h). However, this can go on to be O(n) in the worst case scenario – where BST becomes skewed. AVL helps in eliminating this limitation by restricting the height of the tree. In doing so, it imposes an upper bound on all the operations to be maximum of O(log n) where n = number of nodes.

What are the properties of a B-Tree?

A B-Tree of an order m contains the following properties:

- All the properties of an M-way tree.
- Every node of the B_tree will have maximum m children.
- Every node except root and leaf will have at least m / 2 children.
- The root node must have at least 2 child nodes.
- All leaf nodes must lie on the same horizontal level.

What do you understand about the Graph Data Structure?

The Graph (G) Data Structure can be defined as an ordered set G(V,E) where V represents the set of vertices and E is the edges that form the connections. Basically, a Graph can be thought of as a cyclic tree where nodes can maintain complex relationships between them and not just parent-child relationships.

Differentiate between cycle, path, and circuit with reference to the Graph Data Structure.

The differences between the cycle, path, and circuit are as follows:

- A patch is an order of neighbouring vertices connected by edges without any restrictions.
- A cycle is a closed path wherein the initial vertex is the same as the end vertex. In a cycle, no particular verte can be visited twice.
- A circuit, like a cycle, is a closed path with the initial vertex the same as the end vertex. However, any particular vertex in a circuit can be visited more than once.

How does Kruskal's algorithm work?

Kruskal's Algorithm considers the graph as a forest and each of its nodes as an individual tree. A tree is said to connect to another tree if and only if it has the least cost among all the options, and it violates no property of a Minimum Spanning Tree (MST).

How does Prim's algorithm find the spanning tree?

Prim's algorithm works by considering nodes as single trees. Then, it keeps on adding new nodes to the spanning tree from the given graph that has to be converted into the required spanning tree.

What is a minimum spanning tree (MST)?

MSTs, in weighted graphs, are trees that have minimum weight, but they span across all the vertices.

What is a recursive function?

By definition, a recursive function calls itself back or directly calls a function that calls it. Every recursive function has some base criteria, following which the function stops calling itself.

What is the interpolation search technique?

The interpolation search technique is a modification over the Binary Search method. The interpolation search algorithm works on the probing position of desired values.

What is hashing?

Hashing is a very useful technique used to convert a range of key-value pairs into indexes of an array. Hash tables come in handy while creating associative data storage in which data index can easily be found just by providing its key-value pair!

Interviews for which job role generally ask Data Structure and Algorithm questions?

If you're sitting for any software development or engineering role, you will definitely be checked on your DSA skills. Apart from that, if you're applying for Data Science jobs or want to venture into Machine Learning, you will be expected to know DSA.

Do I need to know programming in order to understand Data Structure and Algorithms?

No, not necessarily. DSA is mostly abstract, and it is all about the mathematics and representations and flow of what goes on behind the scenes of any application or program. While having experience with programming will come in handy when you implement Algorithms, it is by no means a prerequisite to studying DSA.

Are Data Structures always static?

No, Data Structures can be both dynamic and static, depending on the way memory allocation works for them. For example, Arrays are static data structures since an entire lot of contiguous memory is allocated to them when they are defined. On the other hand, Linked Lists are dynamic Data Structures as they don't have any fixed size and the number of nodes can increase depending on the programmer's requirements.

What is data-structure?

Data structure is a way of defining, storing & retriving of data in a structural & systemetic way. A data structure may contain different type of data items.

What are various data-structures available?

Data structure availability may vary by programming languages. Commonly available data structures are list, arrays, stack, queues, graph, tree etc.

What is algorithm?

Algorithm is a step by step procedure, which defines a set of instructions to be executed in certain order to get the desired output.

Why we need to do algorithm analysis?

A problem can be solved in more than one ways. So, many solution algorithms can be derived for a given problem. We analyze available algorithms to find and implement the best suitable algorithm.

What are the criteria of algorithm analysis?

An algorithm are generally analyzed on two factors − time and space. That is, how much execution time and how much extra space required by the algorithm.

What is asymptotic analysis of an algorithm?

Asymptotic analysis of an algorithm, refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case and worst case scenario of an algorithm.

What are asymptotic notations?

Asymptotic analysis can provide three levels of mathematical binding of execution time of an algorithm −

- Best case is represented by Ω(n) notation.
- Worst case is represented by Ο(n) notation.
- Average case is represented by Θ(n) notation.

What is linear data structure?

A linear data-structure has sequentially arranged data items. The next time can be located in the next memory address. It is stored and accessed in a sequential manner. Array and list are example of linear data structure.

What are common operations that can be performed on a data-structure?

The following operations are commonly performed on any data-structure −

- Insertion − adding a data item
- Deletion − removing a data item
- Traversal − accessing and/or printing all data items
- Searching − finding a particular data item
- Sorting − arranging data items in a pre-defined sequence

Briefly explain the approaches to develop algorithms.

There are three commonly used approaches to develop algorithms −

- **Greedy Approach** − finding solution by choosing next best option
- **Divide and Conquer** − diving the problem to a minimum possible sub-problem and solving them independently

- **Dynamic Programming** − diving the problem to a minimum possible sub-problem and solving them combinedly

Give some examples greedy algorithms.

The below given problems find their solution using greedy algorithm approach −

- Travelling Salesman Problem
- Prim's Minimal Spanning Tree Algorithm
- Kruskal's Minimal Spanning Tree Algorithm
- Dijkstra's Minimal Spanning Tree Algorithm
- Graph - Map Coloring
- Graph - Vertex Cover
- Knapsack Problem
- Job Scheduling Problem

What are some examples of divide and conquer algorithms?

The below given problems find their solution using divide and conquer algorithm approach −

- Merge Sort
- Quick Sort
- Binary Search
- Strassen's Matrix Multiplication
- Closest pair (points)

What are some examples of dynamic programming algorithms?

The below given problems find their solution using divide and conquer algorithm approach −

- Fibonacci number series
- Knapsack problem
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall
- Shortest path by Dijkstra
- Project scheduling

What is a linked-list?

A linked-list is a list of data-items connected with links i.e. pointers or references. Most modern high-level programming language does not provide the feature of directly accessing memory location, therefore, linked-list are not supported in them or available in form of inbuilt functions.

What is stack?

In data-structure, stack is an Abstract Data Type (ADT) used to store and retrieve values in Last In First Out method.

Why do we use stacks?

Stacks follows LIFO method and addition and retrieval of a data item takes only O(n) time. Stacks are used where we need to access data in the reverse order or their arrival. Stacks are used commonly in recursive function calls, expression parsing, depth first traversal of graphs etc.

What operations can be performed on stacks?

The below operations can be performed on a stack −

- push() − adds an item to stack
- pop() − removes the top stack item
- peek() − gives value of top item without removing it
- isempty() − checks if stack is empty
- isfull() − checks if stack is full

What is a queue in data-structure?

Queue is an abstract data structure, somewhat similar to stack. In contrast to stack, queue is opened at both end. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Why do we use queues?

As queues follows FIFO method, they are used when we need to work on data-items in exact sequence of their arrival. Every operating system maintains queues of various processes. Priority queues and breadth first traversal of graphs are some examples of queues.

What operations can be performed on Queues?

The below operations can be performed on a stack −

- enqueue() − adds an item to rear of the queue
- dequeue() − removes the item from front of the queue
- peek() − gives value of front item without removing it
- isempty() − checks if stack is empty
- isfull() − checks if stack is full

What is linear searching?

Linear search tries to find an item in a sequentially arranged data type. These sequentially arranged data items known as array or list, are accessible in incrementing memory location. Linear search compares expected data item with each of data items in list or array. The average case time complexity of linear search is O(n) and worst case complexity is O(n2). Data in target arrays/lists need not to be sorted.

What is binary search?

A binary search works only on sorted lists or arrays. This search selects the middle which splits the entire list into two parts. First the middle is compared.

This search first compares the target value to the mid of the list. If it is not found, then it takes decision on whether.

What is bubble sort and how bubble sort works?

Bubble sort is comparison based algorithm in which each pair of adjacent elements is compared and elements are swapped if they are not in order. Because the time complexity is O(n2), it is not suitable for large set of data.

Tell me something about 'insertion sort'?

Insertion sort divides the list into two sub-list, sorted and unsorted. It takes one element at time and finds it appropriate location in sorted sub-list and insert there. The output after insertion is a sorted sub-list. It iteratively works on all the elements of unsorted sub-list and inserts them to sorted sub-list in order.

What is selection sort?

Selection sort is in-place sorting technique. It divides the data set into two sub-lists: sorted and unsorted. Then it selects the minimum element from unsorted sub-list and places it into the sorted list. This iterates unless all the elements from unsorted sub-list are consumed into sorted sub-list.

How insertion sort and selection sorts are different?

Both sorting techniques maintains two sub-lists, sorted and unsorted and both take one element at a time and places it into sorted sub-list. Insertion sort works on the current element in hand and places it in the sorted array at appropriate location maintaining the properties of insertion sort. Whereas, selection sort searches the minimum from the unsorted sub-list and replaces it with the current element in hand.

What is merge sort and how it works?

Merge sort is sorting algorithm based on divide and conquer programming approach. It keeps on dividing the list into smaller sub-list until all sub-list has only 1 element. And then it merges them in a sorted way until all sub-lists are consumed. It has run-time complexity of O(n log n) and it needs O(n) auxiliary space.

What is shell sort?

Shell sort can be said a variant of insertion sort. Shell sort divides the list into smaller sublist based on some gap variable and then each sub-list is sorted using insertion sort. In best cases, it can perform upto O(n log n).

How quick sort works?

Quick sort uses divide and conquer approach. It divides the list in smaller 'partitions' using 'pivot'. The values which are smaller than the pivot are arranged in the left partition and greater values are arranged in the right partition. Each partition is recursively sorted using quick sort.

What is a graph?

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.

How depth first traversal works?

Depth First Search algorithm(DFS) traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration.

How breadth first traversal works?

Breadth First Search algorithm(BFS) traverses a graph in a breadthwards motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

What is a tree?

A tree is a minimally connected graph having no loops and circuits.

What is a binary tree?

A binary tree has a special condition that each node can have two children at maximum.

What is a binary search tree?

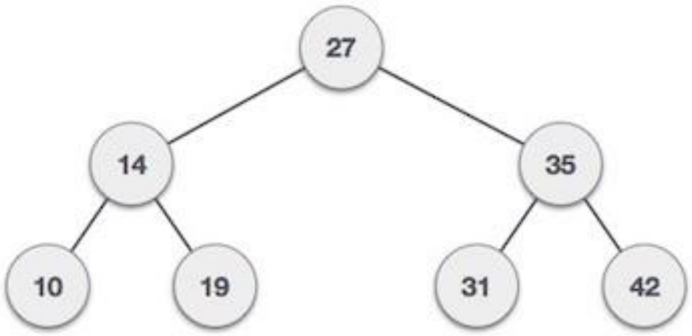
A binary search tree is a binary tree with a special provision where a node's left child must have value less than its parent's value and node's right child must have value greater than it's parent value.

What is tree traversal?

Tree traversal is a process to visit all the nodes of a tree. Because, all nodes are connected via edges (links) we always start from the root (head) node. There are three ways which we use to traverse a tree −

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

See the below image of a binary search tree, and traverse it using all available methods −



What is an AVL Tree?

AVL trees are height balancing binary search tree. AVL tree checks the height of left and right sub-trees and assures that the difference is not more than 1. This difference is called Balance Factor.

BalanceFactor = height(left-sutree) − height(right-sutree)

What is a spanning tree?

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. A spanning tree does not have cycles and it can not be disconnected.

How many spanning trees can a graph has?

It depends on how connected the graph is. A complete undirected graph can have maximum nn-1 number of spanning trees, where n is number of nodes.

How Kruskal's algorithm works?

This algorithm treats the graph as a forest and every node it as an individual tree. A tree connects to another only and only if it has least cost among all available options and does not violate MST properties.

How Prim's algorithm finds spanning tree?

Prim's algorithm treats the nodes as a single tree and keeps on adding new nodes to the spanning tree from the given graph.

What is a minimum spanning tree (MST)?

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight that all other spanning trees of the same graph.

What is a heap in data structure?

Heap is a special balanced binary tree data structure where root-node key is compared with its children and arranged accordingly. A min-heap, a parent node has key value less than its childs and a max-heap parent node has value greater than its childs.

What is a recursive function?

A recursive function is one which calls itself, directly or calls a function that in turn calls it. Every recursive function follows the recursive properties − base criteria where functions stops calling itself and progressive approach where the functions tries to meet the base criteria in each iteration.

What is tower of hanoi?

Tower of Hanoi, is a mathematical puzzle which consists of three tower (pegs) and more than one rings. All rings are of different size and stacked upon each other where the large disk is always below the small disk. The aim is to move the tower of disk from one peg to another, without breaking its properties.

What is fibonacci series?

Fibonacci Series generates subsequent number by adding two previous numbers. For example − 0 1 1 2 3 5 8 13.

What is hashing?

Hashing is a technique to convert a range of key values into a range of indexes of an array. By using hash tables, we can create an associative data storage where data index can be find by providing its key values.

What is interpolation search technique?

Interpolation search is an improved variant of binary search. This search algorithm works on the probing position of required value.

What is the prefix and post fix notation of (a + b) * (c + d) ?

- Prefix Notation − * + a b + c d
- Postfix Notation − a b + c d + *

What is Next ?

Further you can go through your past assignments you have done with the subject and make sure you are able to speak confidently on them. If you are fresher then interviewer does not expect you will answer very complex questions, rather you have to make your basics concepts very strong.

1) What is Data Structure? Explain.

The data structure is a way that specifies how to organize and manipulate the data. It also defines the relationship between them. Some examples of Data Structures are arrays, Linked List, Stack, Queue, etc. Data Structures are the central part of many computer science algorithms as they enable the programmers to handle the data in an efficient way

2) Describe the types of Data Structures?

Data Structures are mainly classified into two types:

Linear Data Structure: A data structure is called linear if all of its elements are arranged in the sequential order. In linear data structures, the elements are stored in a non-hierarchical way where each item has the successors and predecessors except the first and last element.

Non-Linear Data Structure: The Non-linear data structure does not form a sequence i.e. each item or element is connected with two or more other items in a non-linear arrangement. The data elements are not arranged in the sequential structure.

3) List the area of applications of Data Structure.

Data structures are applied extensively in the following areas of computer science:

- Compiler Design,
- Operating System,
- Database Management System,
- Statistical analysis package,
- Numerical Analysis,
- Graphics,
- Artificial Intelligence,
- Simulation

4) What is the difference between file structure and storage structure?

Difference between file structure and storage structure:

- The main difference between file structure and storage structure is based on memory area that is being accessed.
- Storage structure: It is the representation of the data structure in the computer memory.
- File structure: It is the representation of the storage structure in the auxiliary memory.

5) List the data structures which are used in RDBMS, Network Data Modal, and Hierarchical Data Model.

- RDBMS uses Array data structure
- Network data model uses Graph
- Hierarchal data model uses Trees

6) Which data structure is used to perform recursion?

Stack data structure is used in recursion due to its last in first out nature. Operating system maintains the stack in order to save the iteration variables at each function call

7) What is a Stack?

Stack is an ordered list in which, insertion and deletion can be performed only at one end that is called the top. It is a recursive data structure having pointer to its top element. The stack is sometimes called as Last-In-First-Out (LIFO) list i.e. the element which is inserted first in the stack will be deleted last from the stack.

8) List the area of applications where stack data structure can be used?

- Expression evaluation
- Backtracking
- Memory Management
- Function calling and return

9) What are the operations that can be performed on a stack?

- Push Operations
- Pop Operations
- Peek Operations

10) Write the stack overflow condition.

Overflow occurs when top = Maxsize -1

11) What is the difference between PUSH and POP?

- PUSH and POP operations specify how data is stored and retrieved in a stack.
- PUSH: PUSH specifies that data is being "inserted" into the stack.
- POP: POP specifies data retrieval. It means that data is being deleted from the stack.

12) Write the steps involved in the insertion and deletion of an element in the stack.

Push:

- Increment the variable top so that it can refer to the next memory allocation
- Copy the item to the at the array index value equal to the top
- Repeat step 1 and 2 until stack overflows

Pop:

- Store the topmost element into the an another variable
- Decrement the value of the top
- Return the topmost element

13) What is a postfix expression?

An expression in which operators follow the operands is known as postfix expression. The main benefit of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

The expression "a + b" will be represented as "ab+" in postfix notation.

14)Write the postfix form of the expression: (A + B) * (C - D)

AB+CD-*

15) Which notations are used in Evaluation of Arithmetic Expressions using prefix and postfix forms?

Polish and Reverse Polish notations.

16)What is an array?

Arrays are defined as the collection of similar types of data items stored at contiguous memory locations. It is the simplest data structure in which each data element can be randomly accessed by using its index number.

17) How to reference all the elements in a one-dimension array?

It can be done by using an indexed loop such that the counter runs from 0 to the array size minus one. In this manner, you can reference all the elements in sequence by using the loop counter as the array subscript.

18) What is a multidimensional array?

The multidimensional array can be defined as the array of arrays in which, the data is stored in tabular form consists of rows and columns. 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

19) How are the elements of a 2D array are stored in the memory?

There are two techniques by using which, the elements of a 2D array can be stored in the memory.

- Row-Major Order: In row-major ordering, all the rows of the 2D array are stored into the memory contiguously. First, the 1st row of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last row.
- Column-Major Order: In column-major ordering, all the columns of the 2D array are stored into the memory contiguously. first, the 1st column of the array is stored into the memory completely, then the 2nd row of the array is stored into the memory completely and so on till the last column of the array.

20) Calculate the address of a random element present in a 2D array, given base address as BA.

Row-Major Order: If array is declared as a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in row major order is calculated as,

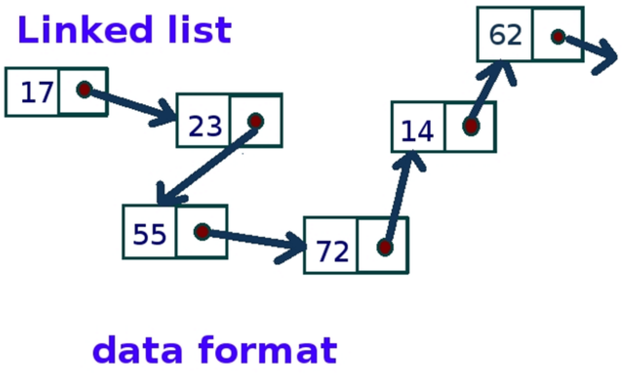
Address(a[i][j]) = B. A. + (i * n + j) * size

Column-Major Order: If array is declared as a[m][n] where m is the number of rows while n is the number of columns, then address of an element a[i][j] of the array stored in column major order is calculated as

Address(a[i][j]) = ((j*m)+i)*Size + BA.

21) Define Linked List Data structure.

Linked List is the collection of randomly stored data objects called nodes. In Linked List, each node is linked to its adjacent node through a pointer. A node contains two fields, i.e. Data Field and Link Field.



22) Are linked lists considered linear or non-linear data structures?

A linked list is considered both linear and non-linear data structure depending upon the situation.

o On the basis of data storage, it is considered as a non-linear data structure.

o On the basis of the access strategy, it is considered as a linear data-structure.

____

23) What are the advantages of Linked List over an array?

o The size of a linked list can be incremented at runtime which is impossible in the case of the array.

o The List is not required to be contiguously present in the main memory, if the contiguous space is not available, the nodes can be stored anywhere in the memory connected through the links.

o The List is dynamically stored in the main memory and grows as per the program demand while the array is statically stored in the main memory, size of which must be declared at compile time.

o The number of elements in the linked list are limited to the available memory space while the number of elements in the array is limited to the size of an array.

____

24) Write the syntax in C to create a node in the singly linked list.

```
struct node
{
    int data;
    struct node *next;
};
struct node *head, *ptr;
ptr = (struct node *)malloc(sizeof(struct node));
```

____

25) If you are using C language to implement the heterogeneous linked list, what pointer type should be used?

The heterogeneous linked list contains different data types, so it is not possible to use ordinary pointers for this. For this purpose, you have to use a generic pointer type like void pointer because the void pointer is capable of storing a pointer to any type.

____

26) What is doubly linked list?

The doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. In a doubly linked list, a node consists of three parts:

- node data
- pointer to the next node in sequence (next pointer)
- pointer to the previous node (previous pointer).

____

27) Write the C program to insert a node in circular singly list at the beginning.

```
#include<stdio.h>
#include<stdlib.h>
void beg_insert(int);
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void main ()
{
    int choice,item;
    do
    {
        printf("\nEnter the item which you want to insert?\n");
        scanf("%d",&item);
        beg_insert(item);
        printf("\nPress 0 to insert more ?\n");
        scanf("%d",&choice);
    }while(choice == 0);
}
void beg_insert(int item)
{

    struct node *ptr = (struct node *)malloc(sizeof(struct node));
    struct node *temp;
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        ptr -> data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
```

```
                temp = temp->next;
            ptr->next = head;
            temp -> next = ptr;
            head = ptr;
        }
    printf("\nNode Inserted\n");
    }

}

```

____

28) Define the queue data structure.

A queue can be defined as an ordered list which enables insert operations to be performed at one end called REAR and delete operations to be performed at another end called FRONT.

____

29) List some applications of queue data structure.

The Applications of the queue is given as follows:

o Queues are widely used as waiting lists for a single shared resource like a printer, disk, CPU.

o Queues are used in the asynchronous transfer of data (where data is not being transferred at the same rate between two processes) for eg. pipes, file IO, sockets.

o Queues are used as buffers in most of the applications like MP3 media player, CD player, etc.

o Queues are used to maintain the playlist in media players to add and remove the songs from the play-list.

o Queues are used in operating systems for handling interrupts.

____

30) What are the drawbacks of array implementation of Queue?

o Memory Wastage: The space of the array, which is used to store queue elements, can never be reused to store the elements of that queue because the elements can only be inserted at front end and the value of front might be so high so that, all the space before that, can never be filled.

o Array Size: There might be situations in which, we may need to extend the queue to insert more elements if we use an array to implement queue, It will almost be impossible to extend the array size, therefore deciding the correct array size is always a problem in array implementation of queue.

____

31) What are the scenarios in which an element can be inserted into the circular queue?

o If (rear + 1)%maxsize = front, the queue is full. In that case, overflow occurs and therefore, insertion can not be performed in the queue.

o If rear != max - 1, the rear will be incremented to the mod(maxsize) and the new value will be inserted at the rear end of the queue.

o If front != 0 and rear = max - 1, it means that queue is not full therefore, set the value of rear to 0 and insert the new element there.

____

32) What is a dequeue?

Dequeue (also known as double-ended queue) can be defined as an ordered set of elements in which the insertion and deletion can be performed at both the ends, i.e. front and rear.

____

33) What is the minimum number of queues that can be used to implement a priority queue?

Two queues are needed. One queue is used to store the data elements, and another is used for storing priorities.

____

34) Define the tree data structure.

The Tree is a recursive data structure containing the set of one or more data nodes where one node is designated as the root of the tree while the remaining nodes are called as the children of the root. The nodes other than the root node are partitioned into the nonempty sets where each one of them is to be called sub-tree.

____

35) List the types of tree.

There are six types of tree given as follows.

o General Tree

o Forests

o Binary Tree

o Binary Search Tree

o Expression Tree

o Tournament Tree

____

36) What are Binary trees?

A binary Tree is a special type of generic tree in which, each node can have at most two children. Binary tree is generally partitioned into three disjoint subsets, i.e. the root of the node, left sub-tree and Right binary sub-tree.

____

37) Write the C code to perform in-order traversal on a binary tree.

```
void in-order(struct treenode *tree)
```

```
    {
        if(tree != NULL)
        {
            in-order(tree→ left);
            printf("%d",tree→ root);
            in-order(tree→ right);
        }
    }
```

____

38) What is the maximum number of nodes in a binary tree of height k?

2k+1-1 where k >= 1

____

39) Which data structure suits the most in the tree construction?

Queue data structure

____

40) Which data structure suits the most in the tree construction?

Queue data structure

____

41) Write the recursive C function to count the number of nodes present in a binary tree.

```
int count (struct node* t)
{
   if(t)
   {
      int l, r;
      l = count(t->left);
      r=count(t->right);
      return (1+l+r);
   }
   else
   {
      return 0;
   }
}
```

____

42) Write a recursive C function to calculate the height of a binary tree.

```
int countHeight(struct node* t)
{
   int l,r;
```

```
    if(!t)
        return 0;
    if((!(t->left)) && (!(t->right)))
        return 0;
    l=countHeight(t->left);
    r=countHeight(t->right);
    return (1+((l>r)?l:r));
}
```

____

43) How can AVL Tree be useful in all the operations as compared to Binary search tree?

AVL tree controls the height of the binary search tree by not letting it be skewed. The time taken for all operations in a binary search tree of height h is O(h). However, it can be extended to O(n) if the BST becomes skewed (i.e. worst case). By limiting this height to log n, AVL tree imposes an upper bound on each operation to be O(log n) where n is the number of nodes.

____

44) State the properties of B Tree.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

o Every node in a B-Tree contains at most m children.

o Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.

o The root nodes must have at least 2 nodes.

o All leaf nodes must be at the same level.

____

45) What are the differences between B tree and B+ tree?

SN B Tree B+ Tree

1	Search keys cannot repeatedly be stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes is so complicated and time-consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes cannot be linked together.	Leaf nodes are linked together to make the search operations more efficient.

46) List some applications of Tree-data structure?

Applications of Tree- data structure:

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis
- Hierarchal data model

____

47) Define the graph data structure?

A graph G can be defined as an ordered set G(V, E) where V(G) represents the set of vertices and E(G) represents the set of edges which are used to connect these vertices. A graph can be seen as a cyclic tree, where the vertices (Nodes) maintain any complex relationship among them instead of having parent-child relations.

____

48) Differentiate among cycle, path, and circuit?

- Path: A Path is the sequence of adjacent vertices connected by the edges with no restrictions.

o Cycle: A Cycle can be defined as the closed path where the initial vertex is identical to the end vertex. Any vertex in the path can not be visited twice

o Circuit: A Circuit can be defined as the closed path where the intial vertex is identical to the end vertex. Any vertex may be repeated.

____

49) Mention the data structures which are used in graph implementation.

For the graph implementation, following data structures are used.

o In sequential representation, Adjacency matrix is used.

o In Linked representation, Adjacency list is used.

____

50) Which data structures are used in BFS and DFS algorithm?

o In BFS algorithm, Queue data structure is used.

o In DFS algorithm, Stack data structure is used.

____

51) What are the applications of Graph data structure?

The graph has the following applications:

o Graphs are used in circuit networks where points of connection are drawn as vertices and component wires become the edges of the graph.

o Graphs are used in transport networks where stations are drawn as vertices and routes become the edges of the graph.

o Graphs are used in maps that draw cities/states/regions as vertices and adjacency relations as edges.

o Graphs are used in program flow analysis where procedures or modules are treated as vertices and calls to these procedures are drawn as edges of the graph.

____

54) In what scenario, Binary Search can be used?

Binary Search algorithm is used to search an already sorted list. The algorithm follows divide and conqer approach

Example:

____

52) What are the advantages of Binary search over linear search?

There are relatively less number of comparisons in binary search than that in linear search. In average case, linear search takes O(n) time to search a list of n elements while Binary search takes O(log n) time to search a list of n elements.

____

53) What are the advantages of Selecetion Sort?

o It is simple and easy to implement.

o It can be used for small data sets.

o It is 60 per cent more efficient than bubble sort.

____

55) List Some Applications of Multilinked Structures?

o Sparse matrix,

o Index generation.

____

56) What is the difference between NULL and VOID?

o Null is actually a value, whereas Void is a data type identifier.

o A null variable simply indicates an empty value, whereas void is used to identify pointers as having no initial size.

Data Structures and Algorithms (DSA) is an important part of Computer Science that deals with the organization of data values and code optimization. To understand more about DSA, let’s break down the word DSA into two parts:

- Data Structures
- Algorithms

A data structure is used for the storage and organization of data values for further use in an application. Talking about an algorithm, it is a stepwise process to solve a problem.

Walmart, Myntra, and 20+ companies are hiring at package up to 24 LPA via D2C Recruitables. Apply now.

Now, let’s look at some of the most commonly used data structures:

Data Structure	Meaning
Array	It is one of the basic data structures that is like a linear list of data points of the same type.
Linked List	This is similar to an array but here it has two fields: Data Field: For data values Reference Field: For storing address
Trees	A collection of nodes is a tree. A node is a data point connected with other points with the help of edges. The top of a tree is the root node.
Stacks	Imagine a pile of chairs. If you want to add another chair, for that, you will have to add the chair at the top. That’s a stack. Insertion or deletion takes place at the top of the stack of data values. It follows Last-In-First-Out (LIFO) Principle.
Queues	A queue in English is a line. Similarly, in DSA, it is a line of data values that follows the FIFO (First in First out) Principle.
Graphs	A graph is a collection of nodes connected to one another via edges. It forms a network of nodes like in the case of a journey from one source to a destination.

Applications of Data Structures and Algorithms

Data Structures are used in a wide variety of fields. Some of the important applications include:

Data Structure	Application
Array	Mobile phone contacts
Linked List	Next feature of Music Player
Trees	Indexing in databases
Stacks	Undo and Redo tasks in editors
Queues	In Operating Systems for FCFS scheduling
Graphs	Google Maps, Facebook, and LinkedIn

Importance of DSA for interviews

DSA is paramount as in today's world, companies deal with a lot of data. As coders, you should know about code optimization to ensure space and time resources are utilized efficiently. Also, for cracking product-based company interviews which offer a high-paying job, DSA is a must.

Let's look at some important product-based companies that test DSA:

Company	Median Pay*
Adobe	USD 91,900
Amazon	USD 81,900
Facebook	USD 91,300
Google	USD 106,900
Microsoft	USD 102,500

*Median pay for workers with 0-5 years of experience

Source: Payscale

Useful resources to learn DSA

Here is a list of some of the best resources that can help you learn the inside-out of the DSA:

Books	Data Structures and Algorithms Made Easy Narshima Karumanchi.
	Introduction to Algorithms (Cormen)

Youtube Channels	Abdul Bari
	Tushar Roy
	Apni Kaksha
	Neso Academy
	Jenny’s Lectures
Online Sites	Udemy
	Coursera
	GeeksforGeeks

Now, let’s look at some of the important DSA questions.

1. What are data structures?

A data structure is a way of storing and organization of data values for further use in an application. Examples include Graph, Trees, Arrays, Linked List, etc.

2. Name different types of data structures?

Data structures are of two types:

- Linear Array, Stack, Queue, Linked-List
- Non-Linear Graphs, Trees

3. What is the use of dynamic Data Structures?

Dynamic data structures are flexible and size changes can occur during insertion or deletion operations. Dynamic data structures play a key role in programming because they provide the programmer with the flexibility to adjust the memory consumption of programs.

4. Name various operations that can be performed in DSA.

- Insert: Here, we add a new data item in a data structure.
- Search: In this, we find an element in the data structure.
- Sort: Simple sorting like arranging values in ascending or descending order. For example, arranging values in an array.
- Delete: Here, we delete unwanted data points in a data structure.

- Traversal: We access each data item exactly once so that it can be processed for further use.

5. What are Infix, prefix, Postfix notations?

- Infix: We write expressions in infix notation, e.g. x+y - z, where operators are used in-between operands. For humans, it goes well, but for computing devices, it's not preferred.
- Prefix: Here, the operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +xy instead of x+y.
- Postfix: Here, we use the operator after the operands. For example, X*Y is written as XY*.

6. What are the types of searching used in Data Structures?

- Linear Search.
- Binary Search.
- Jump Search.
- Interpolation Search.
- Exponential Search.

7. What is an array?

An array is a collection of data points of the same type stored at contiguous memory locations.

For example, an array of integers like 1,2,3,4,5. This array has 5 elements.

The index of an array usually starts with 0.

8. What are dynamic arrays?

A dynamic array is an array with a modification that is automatic resizing. This means that the array expands when we add more elements. This helps to provide flexibility as size is not fixed. So, there's no need to know the size in advance.

9. Name some characteristics of Array Data Structure.

- Array elements are stored in contiguous memory blocks in the primary memory.
- Array name represents its base address.
- The base address is the address of the first element of the array.
- Array's index starts with 0 and ends with size-1.

10. What is a linked list?

This is similar to an array but it consists of a node that has two fields:

- Data Field: For storing data values.
- Reference Field: For storing address.
- A linked list is used in the next feature of a music player.

11. What are the different types of linked lists?

Ans: The different types of linked lists are:

- Singly Linked list
- Doubly Linked list
- Circular Linked list
- Doubly Circular Linked list

12. What are trees in DSA?

A collection of nodes is a tree. A node is a data point connected with other points with the help of edges. The top of a tree is the root node. Applications of trees include indexing in databases.

13. What are graphs and their uses?

A graph is a collection of nodes connected to one another via edges. It forms a network of nodes like in the case of a journey from one source to a destination.

Uses:

- Google Maps
- Linkedin
- Facebook

14. What's the difference between the data structure Tree and Graph?

Ans: A tree structure is connected and can never have loops whereas in the graph there are no such constraints. A tree provides information about the relationship between nodes in a hierarchical manner and the graph follows a network model.

15. What is the LIFO and FIFO principle?

- LIFO: (Last-In-First-Out) Last inserted element is removed first in the LIFO principle. Example: Stack follows LIFO.

- FIFO: (First-In-First-Out) First Element (first to be inserted) is taken out first.

Example: Queue follows FIFO

16. What is a queue?

A queue in English is a line. Similarly, in DSA, it is a line of data values that follows the FIFO (First in First out) principle. Insertion is done at the rear end and deletion at the front end.

17. What is a priority queue?

A priority queue is like a normal queue of elements but here each element has some priority. The elements in the priority queue occur in this order only.

Say, we have some values like 1, 2, 7, 8, 14, 31 inserted in a priority queue with an ordering based on least values to the greatest value. Therefore, the 1 number will have the highest priority while 31 will have the lowest priority.

18. What is a stack?

A stack is a linear data structure having values that are inserted as per the LIFO principle. To make the point more clear, imagine a pile of chairs. You want to add another chair, for that, you will have to add the chair at the top. That's a stack. Insertion or deletion takes place at the top of the stack of data values.

19. State the difference between stack and queue?

Stack	**Queue**
Insertion and deletion in stacks take place only from one end of the list called the top.	Insertion and deletion in queues take place from the opposite ends of the list. The insertion takes place at the rear end of the queue and the deletion takes place from the front end.
Insert operation is called push in a stack.	Insert operation is called enqueue operation.
Delete operation is called pop in a stack.	Delete operation is called dequeue operation

.

20. What is a heap in DSA?

A heap data structure is a complete binary tree that follows a specific order. Heaps are of two types:

- Max Heap

- Min Heap

21. What is a binary heap?

- A binary heap is a complete binary tree that satisfies the heap ordering property.
- A Binary Heap can either be Min Heap or Max Heap.
- It’s a complete tree, thus it is suitable for being stored in an array.

22. What is the meaning of the AVL tree?

An AVL tree is a type of binary search tree. It is named after its inventors Adelson, Velskii, and Landis. An AVL tree is a balanced binary search tree where the height of the two subtrees of a node differs by a maximum of one unit.

23. What do you mean by Hash Table?

A Hash table is a data structure used to store data points in an associative manner. The values are stored in an array format. Hash tables are used to store keys/value pairs

24. What is the complexity of a Hash Table?

Hash tables provide constant-time O(1) lookup on average, irrespective of the number of items(n) in the table.

25. What Data Structures make use of pointers?

- Stack
- Queue
- Linked List
- Binary Tree.

26. What is a dequeue?

A dequeue is a double-ended queue. This queue data structure includes elements that can be inserted or removed from either end.

27. What is the meaning of the stack overflow condition?

When a stack is completely full and we try to insert more elements onto the stack then this condition is called stack overflow condition. Here, top=maxsize-1, and no further elements can be inserted.

28. What is the postfix form of (X + Y) * (Z - C)

The postfix form of the given expression is XY+ZC-*

29. What is a Balanced Tree and why is that important?

A tree is perfectly height-balanced if the left and right subtrees of any node are of the same height. We can also say that a tree is height-balanced if the heights of the left and right subtrees of each node differ by a maximum of one unit.

30. What are the Data Structures that are used to represent graphs?

- Adjacency matrix
- Adjacency list
- And adjacency set.

31. What operations can be performed on stacks?

- push() - Adds an element at the top of the stack.
- pop() - Deletes an element from the top of the stack.
- peek()- Displays the topmost element.

32. Why use queues?

Queues are important in computer science. They are useful in transport, and operations research. They are also useful in a case where resource sharing takes place among a myriad of consumers. For example, FCFS scheduling.

33. What is linear search?

Linear search is a technique in which we traverse a list in a sequential manner to find an element. When we find the element that is required, we return the index or position of that element.

34. What is binary search?

In this type of search technique, we divide the sorted array or list into two halves. This is done to save time in searching. Here, we consider arrays in sorted form only.

35. What are the time complexities of linear search and binary search?

Linear Search-O(N) Binary Search- O(log 2 N)

36. Tell me about tree traversal.

Tree traversal is a process that goes through the entire tree in a particular manner. Depending upon the order of traversal, we have different types:

- Inorder Traversal (Left, Root, Right)
- Preorder Traversal (Root, Left, Right)
- Postorder Traversal (Left, Right, Root)

37. How does Kruskal's algorithm work?

Kruskal algorithm treats a graph as a forest and every node as an individual tree. A tree connects to another only and only if it has the least cost among all available choices without violating Minimum Spanning Tree (MST) properties.

38. How does Prim's algorithm find a spanning tree?

Prim's algorithm treats each node as a single tree and continues to add new nodes to the spanning tree from the given graph.

39. What is a minimum spanning tree (MST)?

It is a spanning tree that has the minimum weight among all the spanning trees of the same graph.

40. Explain the Tower of Hanoi Problem.

- The Tower of Hanoi is a problem that comprises three rods and multiple disks.
- At the start, all the disks are placed on one rod, one over the other in increasing order.
- The aim of this problem is to move the stack of disks from the starting rod to another rod, following these rules as below:
    - A disk cannot be placed on top of a smaller disk
    - No disk can be placed on top of the smaller disk.

41. What are recursive algorithms?

Recursion in general is a function calling itself. Extending the same logic for Algorithms, we can say that an algorithm that calls itself is a recursive algorithm. One good example of their use would be searching through a file system. Usually, they are used when the iterative approach is not useful for complex problems.

42. Explain why stack is a recursive data structure?

A stack is a recursive data structure because:

- A stack can either be empty or
- It will have a top pointer and the rest part apart from the top is also a stack by itself, thus it's recursive.

43. What is merge sort time complexity?

The time complexity of MergeSort is O(n*log n)

44. What is shell sort?

Shell sort is a sorting algorithm based on the insertion sort algorithm.

This algorithm is highly efficient in sorting. This algorithm tries to avoid large shifts as in the case of insertion sort if the smaller value is to the far right and has to be moved to the far left.

45. What is quicksort time complexity?

Worst-case time complexity is O(n^2)

46. What is a Red-Black Tree?

A red-black tree is a binary tree that has nodes represented by two colors: red and black. The tree follows specific properties.

These include:

- The root node of the tree is always black.
- Every path from the root to any of the leaf nodes should have the same number of black nodes.
- No two red nodes can be adjacent to each other.

47. When does the worst case of QuickSort occur?

It occurs when the picked pivot is an extreme (smallest or largest) element. Usually, when the input array is sorted or reverse sorted, it also leads to the worst case.

48. Which data structures are used for the BFS and DFS of a graph?

- For BFS - Queue Data Structure

- For DFS - Stack Data Structure

49. What do you mean by BFS and DFS?

BFS Algorithm stands for Breadth-First Search. It is a vertex-based technique for finding the shortest path in the graph. For BFS, we use a Queue data structure.

DFS Algorithm stands for Depth First Search. It is an edge-based technique. In DFS, traversal can be started from any node. For DFS, we use a stack data structure.

50. What is the maximum number of nodes in a binary tree of height k?

Ans: The maximum nodes in a binary tree are: 2k+1-1 where k >= 1.

String Interview Questions

- Print duplicate characters from String?
- Check if two Strings are anagrams of each other?
- Print first non-repeated character from String?
- Reverse a given String using recursion?
- Check if a String contains only digits?
- Find duplicate characters in a String?
- Count many vowels and consonants in a given String?
- Count the occurrence of a given character in String?
- Find all permutations of String?
- Reverse words in a given sentence without using any library method?
- Check if two String is a rotation of each other?
- Check if the given String is Palindrome?

Array and Matrix Interview Questions

In Java, an array is again an object, which provides just the length method.

You can only access the array using the index, and Java also doesn't valid index check, and if you try to access a display with an invalid index, you will get Java. lang.ArrayIndexOutOfBoundsException, so beware of that.

Here is a list of some of the frequently asked Array and Matrix-based Programming questions:

- Find a missing number in a given integer array of 1 to 100?
- Find the duplicate number on a given integer array?
- The largest and smallest number in an unsorted integer array?

- Find all pairs of integer arrays whose sum is equal to a given number?
- Find duplicate numbers in an array if it contains multiple duplicates?
- Remove duplicates from the given array in Java?
- Sort an integer array in place using the QuickSort algorithm?
- Remove duplicates from an array in place?
- Reverse an array in place in Java?
- Find multiple missing numbers in a given integer array with duplicates?
- Perform a binary search in a given array?
- Transpose a Matrix?
- Add or subtract two Matrices?
- Multiply two Matrices in Java?
- Calculate the average of all numbers in a given array?

Linked List Interview Questions

A linked list is another vital data structure from an interview point of view; here are some of the frequently asked related list questions from programming interviews:

Here is a list of some of the standard linked list data structure questions from interviews:

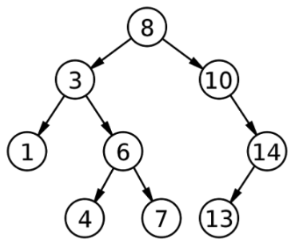
- Find the middle element of a singly linked list in one pass?
- Find the 3rd node from the end in a singly linked list?
- Check if a given linked list contains a cycle?
- How to find the starting node of the cycle?
- Find the length of a singly linked list?
- Reverse a linked list? (solution)
- Reverse a singly linked list without recursion?
- Remove duplicate nodes in an unsorted linked list?
- Find the sum of two linked lists using Stack?

Binary Tree Interview Questions

The tree data structure is another famous data structure in programming interviews. It has several variants, e.g., a binary tree, search tree, and even binary heaps. It's almost guaranteed to see a couple of binary tree questions in programming job interviews.

Here is a list of some of the popular binary tree interview questions from programming job interviews:

- Implement a binary search tree?
- Pre-order traversal in a given binary tree?
- Traverse a given binary tree in Pre-order without recursion
- Implement a Post-order traversal algorithm?
- Traverse a binary tree in Postorder traversal without recursion
- Print all leaves of a binary search tree?
- Count many leaf nodes in a given binary tree?
- In order traversal in given binary tree?
- Print all nodes of given binary tree using inorder traversal without recursion
- Check if a given binary tree is a binary search tree?
- Check if a binary tree is balanced or not?
- Given a binary search tree, how do you check whether there are two nodes in it whose sum equals a given value?
- Convert a binary search tree to a sorted doubly-linked list. you are only allowed to change the target of pointers but cannot create any new nodes.
- Given a binary search tree and a value k, How do you find a node in the binary search tree whose value is closest to k.



Stack and Queue Interview Questions

Stack and Queue are derived data structures, i.e., they are implemented either using an array or linked list, but they have unique features. A queue is also known as FIFO data structure, which means First In First Out, i.e., the element added first will also be retrieved first. The Queue allows you to add an element at the tail and retrieve a piece from the head, thus giving FIFO ordering. On the other hand, Stack is a LIFO data structure, Last In First out, i.e., the element added first will be the last one to go. This property is often used to convert a recursive algorithm into an iterative one.

1) How do you implement a Queue using two Stacks?

2) Write a Java program to implement Stack using an array and linked list?

3) How do you implement Stack using Queues?

4) Given a binary tree, return the postorder traversal of its nodes' values, using Stack?

5) Difference between Stack and Queue data structure

Search and Sort Algorithmic Interview Questions

Search and Sort based questions are the most popular algorithmic questions on any programming job interview. The interviewer often asks to implement various sorting algorithms, like the Bubble sort, Quicksort, merge sort, and asking to perform a binary search, etc. Other algorithms questions, like collision detection, are not so popular, but they are exciting to solve and develop your grasp on creating your algorithms.

- Implement the Bubble Sort algorithm?
- Implement Iterative QuickSort Algorithm?
- Implement the Bucket Sort Algorithm?
- Implement the Counting Sort Algorithm?
- Implement the Insertion Sort Algorithm?
- Implement a Merge Sort Algorithm?
- Implement the Radix Sort Algorithm?
- Implement Sieve of Eratosthenes Algorithm to find Prime numbers?
- Find GCD of two numbers using Euclid's Algorithm?

Bit Manipulation Interview Questions

- Check if a number is the power of two?
- Check if a number is even or odd without using a modulo operator?
- Subtract two binary numbers?
- Find the number of 1s (the Set bit) in a given Bit Sequence?

Problem-Solving Coding Questions

- Swap two numbers without using the third variable?
- Check if two rectangles overlap with each other?
- Design a Vending Machine?
- Implement an LRU Cache in your favorite programming language?

- Check if a given number is a Palindrome?
- Check if a given number is an Armstrong number?
- Find all prime factors of a given number?
- Check if a given number is positive or negative in Java?
- Find the most significant prime factor of a given integral number?
- Print all prime numbers up to a given number?
- Print Floyd's triangle?
- Print Pascal's triangle?
- Calculate the square root of a given number?
- Check if the given number is a prime number?
- Add two numbers without using the plus operator in Java?
- Check if a given number is even/odd without using an Arithmetic operator?
- Print a given Pyramid structure?
- Find the highest repeating world from a given file in Java?
- Reverse a given Integer in Java?
- Convert a decimal number to binary in Java?
- Check if a given year is a leap year in Java?
- How do you reverse a given string in place?
- How do you print duplicate characters from a string?
- How do you check if two strings are anagrams of each other?
- How do you find all the permutations of a string?
- How can a given string be reversed using recursion?
- How do you check if a string contains only digits?
- How do you find duplicate characters in a given string?
- How do you count a number of vowels and consonants in a given string?
- How do you count the occurrence of a given character in a string?
- How do you print the first non-repeated character from a string?
- How do you convert a given String into int like the atoi()?
- How do you reverse words in a given sentence without using any library method?
- How do you check if two strings are a rotation of each other?
- How do you check if a given string is a palindrome?
- How do you find the length of the longest substring without repeating characters?
- Given string str, How do you find the longest palindromic substring in str?

- How to convert a byte array to String?
- how to remove the duplicate character from String?
- How to find the maximum occurring character in given String?
- How do you remove a given character from String?
- How do you find the missing number in a given integer array of 1 to 100?
- How do you find the duplicate number on a given integer array?
- How do you find the largest and smallest number in an unsorted integer array?
- How do you find all pairs of an integer array whose sum is equal to a given number?
- How do you find duplicate numbers in an array if it contains multiple duplicates?
- How to remove duplicates from a given array in Java?
- How do you search a target value in a rotated array?
- Given an unsorted array of integers, find the length of the longest consecutive elements sequence?
- How is an integer array sorted in place using the quicksort algorithm?
- How do you remove duplicates from an array in place?
- How do you reverse an array in place in Java?
- How are duplicates removed from an array without using any library?
- How to convert a byte array to String?
- What is the difference between an array and a linked list?
- How do you perform a binary search in a given array?
- How to find a median of two sorts arrays?
- How to rotate an array left and right by a given number K?
- How do you find duplicates from an unsorted array?
- Given an array of integers sorted in ascending order, find the starting and ending position of a given value?
- Given an integer array, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum?
- How do you find the missing number in a given integer array of 1 to 100?
- How do you find the duplicate number on a given integer array?
- How do you find the largest and smallest number in an unsorted integer array?
- How do you find all pairs of an integer array whose sum is equal to a given number?
- How do you find duplicate numbers in an array if it contains multiple duplicates?
- How are duplicates removed from a given array in Java?
- How is an integer array sorted in place using the quicksort algorithm?

- How do you remove duplicates from an array in place?
- How do you reverse an array in place in Java?
- How are duplicates removed from an array without using any library?
- How do you find the middle element of a singly linked list in one pass?
- How do you check if a given linked list contains a cycle? How do you find the starting node of the cycle?
- How do you reverse a linked list?
- How do you reverse a singly linked list without recursion?
- How are duplicate nodes removed in an unsorted linked list?
- How do you find the length of a singly linked list?
- How do you find the third node from the end in a singly linked list?
- How do you find the sum of two linked lists using Stack?
- How do you print duplicate characters from a string?
- How do you check if two strings are anagrams of each other?
- How do you print the first non-repeated character from a string?
- How can a given string be reversed using recursion?
- How do you check if a string contains only digits?
- How are duplicate characters found in a string?
- How do you count a number of vowels and consonants in a given string?
- How do you count the occurrence of a given character in a string?
- How do you find all permutations of a string?
- How do you reverse words in a given sentence without using any library method?
- How do you check if two strings are a rotation of each other?
- How do you check if a given string is a palindrome?
- How is a binary search tree implemented?
- How do you perform preorder traversal in a given binary tree?
- How do you traverse a given binary tree in preorder without recursion?
- How do you perform an inorder traversal in a given binary tree?
- How do you print all nodes of a given binary tree using inorder traversal without recursion?
- How do you implement a postorder traversal algorithm?
- How do you traverse a binary tree in postorder traversal without recursion?
- How are all leaves of a binary search tree printed?
- How do you count a number of leaf nodes in a given binary tree?
- How do you perform a binary search in a given array?

- How is a bubble sort algorithm implemented?
- How is an iterative quicksort algorithm implemented?
- How do you implement an insertion sort algorithm?
- How is a merge sort algorithm implemented?
- How do you implement a bucket sort algorithm?
- How do you implement a counting sort algorithm?
- How is a radix sort algorithm implemented?
- How do you swap two numbers without using the third variable?
- How do you check if two rectangles overlap with each other?
- How do you design a vending machine?

1) What is data structure?

Data structure refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with the data structure, we not only focus on one piece of data but the different set of data and how they can relate to one another in an organized manner.

2) Differentiate between file and structure storage structure.

The key difference between both the data structure is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

3) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is searched starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

4) What is a linked list?

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link for data storage.

5) How do you reference all the elements in a one-dimension array?

To reference all the elements in a one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.

6) In what areas do data structures are applied?

Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

7) What is LIFO?

LIFO is a short form of Last In First Out. It refers how data is accessed, stored and retrieved. Using this scheme, data that was stored last should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

8) What is a queue?

A queue is a data structure that can simulate a list or stream of data. In this structure, new elements are inserted at one end, and existing elements are removed from the other end.

9) What are binary trees?

A binary tree is one type of data structure that has two nodes, a left node, and a right node. In programming, binary trees are an extension of the linked list structures.

10) Which data structures are applied when dealing with a recursive function?

Recursion, is a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

11) What is a stack?

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

12) Explain Binary Search Tree

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

13) What are multidimensional arrays?

Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

14) Are linked lists considered linear or non-linear data structures?

It depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

15) How does dynamic memory allocation help in managing data?

Apart from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

16) What is FIFO?

FIFO stands for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

17) What is an ordered list?

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

18) What is merge sort?

Merge sort, is a divide-and-conquer approach for sorting the data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continues until you have one single sorted list.

19) Differentiate NULL and VOID

Null is a value, whereas Void is a data type identifier. A variable that is given a Null value indicates an empty value. The void is used to identify pointers as having no initial size.

20) What is the primary advantage of a linked list?

A linked list is an ideal data structure because it can be modified easily. This means that editing a linked list works regardless of how many elements are in the list.

21) What is the difference between a PUSH and a POP?

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being “pushed” into the stack. On the other hand, a pop denotes data retrieval, and in particular, refers to the topmost data being accessed.

22) What is a linear search?

A linear search refers to the way a target key is being searched in a sequential data structure. In this method, each element in the list is checked and compared against the target key. The process is repeated until found or if the end of the file has been reached.

23) How does variable declaration affect memory allocation?

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

24) What is the advantage of the heap over a stack?

The heap is more flexible than the stack. That’s because memory space for the heap can be dynamically allocated and de-allocated as needed. However, the memory of the heap can at times be slower when compared to that stack.

25) What is a postfix expression?

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

26) What is Data abstraction?

Data abstraction is a powerful tool for breaking down complex data problems into manageable chunks. This is applied by initially specifying the data objects involved and the operations to be performed on these data objects without being overly concerned with how the data objects will be represented and stored in memory.

27) How do you insert a new item in a binary search tree?

Assuming that the data to be inserted is a unique value (that is, not an existing entry in the tree), check first if the tree is empty. If it's empty, just insert the new item in the root node. If it's not empty, refer to the new item's key. If it's smaller than the root's key, insert it into the root's left subtree, otherwise, insert it into the root's right subtree.

28) How does a selection sort work for an array?

The selection sort is a fairly intuitive sorting algorithm, though not necessarily efficient. In this process, the smallest element is first located and switched with the element at subscript zero, thereby placing the smallest element in the first position.

The smallest element remaining in the subarray is then located next to subscripts 1 through n-1 and switched with the element at subscript 1, thereby placing the second smallest element in the second position. The steps are repeated in the same manner till the last element.

29) How do signed and unsigned numbers affect memory?

In the case of signed numbers, the first bit is used to indicate whether positive or negative, which leaves you with one bit short. With unsigned numbers, you have all bits available for that number. The effect is best seen in the number range (an unsigned 8-bit number has a range 0-255, while the 8-bit signed number has a range -128 to +127.

30) What is the minimum number of nodes that a binary tree can have?

A binary tree can have a minimum of zero nodes, which occurs when the nodes have NULL values. Furthermore, a binary tree can also have 1 or 2 nodes.

31) What are dynamic data structures?

Dynamic data structures are structures that expand and contract as a program runs. It provides a flexible means of manipulating data because it can adjust according to the size of the data.

32) In what data structures are pointers applied?

Pointers that are used in linked list have various applications in the data structure. Data structures that make use of this concept include the Stack, Queue, Linked List and Binary Tree.

33) Do all declaration statements result in a fixed reservation in memory?

Most declarations do, with the exemption of pointers. Pointer declaration does not allocate memory for data, but for the address of the pointer variable. Actual memory allocation for the data comes during run-time.

34) What are ARRAYs?

When dealing with arrays, data is stored and retrieved using an index that refers to the element number in the data sequence. This means that data can be accessed in any order. In programming, an array is declared as a variable having a number of indexed elements.

35) What is the minimum number of queues needed when implementing a priority queue?

The minimum number of queues needed in this case is two. One queue is intended for sorting priorities while the other queue is used for actual storage of data.

36) Which sorting algorithm is considered the fastest?

There are many types of sorting algorithms: quick sort, bubble sort, balloon sort, radix sort, merge sort, etc. Not one can be considered the fastest because each algorithm is designed for a particular data structure and data set. It would depend on the data set that you would want to sort.

37) Differentiate STACK from ARRAY.

Stack follows a LIFO pattern. It means that data access follows a sequence wherein the last data to be stored when the first one to be extracted. Arrays, on the other hand, does not follow a particular order and instead can be accessed by referring to the indexed element within the array.

38) Give a basic algorithm for searching a binary search tree.

- if the tree is empty, then the target is not in the tree, end search
- if the tree is not empty, the target is in the tree
- check if the target is in the root item
- if a target is not in the root item, check if a target is smaller than the root's value
- if a target is smaller than the root's value, search the left subtree
- else, search the right subtree

39) What is a dequeue?

A dequeue is a double-ended queue. This is a structure wherein elements can be inserted or removed from either end.

40) What is a bubble sort and how do you perform it?

A bubble sort is one sorting technique that can be applied to data structures such as an array. It works by comparing adjacent elements and exchanges their values if they are out of order. This method lets the smaller values “bubble” to the top of the list, while the larger value sinks to the bottom.

41) What are the parts of a linked list?

A linked list typically has two parts: the head and the tail. Between the head and tail lie the actual nodes. All these nodes are linked sequentially.

42) How does selection sort work?

Selection sort works by picking the smallest number from the list and placing it at the front. This process is repeated for the second position towards the end of the list. It is the simplest sort algorithm.

43) What is a graph?

A graph is one type of data structure that contains a set of ordered pairs. These ordered pairs are also referred to as edges or arcs and are used to connect nodes where data can be stored and retrieved.

44) Differentiate linear from a nonlinear data structure.

The linear data structure is a structure wherein data elements are adjacent to each other. Examples of linear data structure include arrays, linked lists, stacks, and queues. On the other hand, a non-linear data structure is a structure wherein each data element can connect to more than two adjacent data elements. Examples of nonlinear data structure include trees and graphs.

45) What is an AVL tree?

An AVL tree is a type of binary search tree that is always in a state of partially balanced. The balance is measured as a difference between the heights of the subtrees from the root. This self-balancing tree was known to be the first data structure to be designed as such.

46) What are doubly linked lists?

Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

47) What is Huffman's algorithm?

Huffman's algorithm is used for creating extended binary trees that have minimum weighted path lengths from the given weights. It makes use of a table that contains the frequency of occurrence for each data element.

48) What is Fibonacci search?

Fibonacci search is a search algorithm that applies to a sorted array. It makes use of a divide-and-conquer approach that can significantly reduce the time needed in order to reach the target element.

49) Briefly explain recursive algorithm.

Recursive algorithm targets a problem by dividing it into smaller, manageable sub-problems. The output of one recursion after processing one sub-problem becomes the input to the next recursive process.

50) How do you search for a target key in a linked list?

To find the target key in a linked list, you have to apply sequential search. Each node is traversed and compared with the target key, and if it is different, then it follows the link to the next node. This traversal continues until either the target key is found or if the last node is reached.

Write Programs for the following

- Reverse a linked list
- Find the middle element of a linked list
- Find if the linked list is circular
- Reverse a linked list
- Find the middle element of a linked list
- Find if the linked list is circular
- Find if two elements in the list sum to a target value
- Group Anagrams
- Longest substring without repeating characters

- Binary Tree Right Side View
- Task Scheduler
- Shortest Subarray with Sum at least K
- Valid Parentheses
- Trapping Rain Water
- Exclusive Time of Functions
- Validate Binary Search Tree
- Binary Tree Level Order Traversal
- Lowest Common Ancestor of a Binary Tree
- Clone Graph
- Is Graph Bipartite?
- Alien Dictionary
- LeetCode.com
- BigOCheatSheet

Arrays

- Subarray with given sum
- Count the triplets
- Kadane’s Algorithm
- Missing number in array
- Merge two sorted arrays
- Rearrange array alternatively
- Number of pairs
- Inversion of Array
- Sort an array of 0s, 1s and 2s
- Equilibrium point
- Leaders in an array
- Minimum Platforms
- Reverse array in groups
- K’th smallest element
- Trapping Rain Water
- Pythagorean Triplet
- Chocolate Distribution Problem
- Stock buy and sell

- Element with left side smaller and right side greater
- Convert array into Zig-Zag fashion
- Last Index of 1
- Spirally traversing a matrix
- Largest Number formed from an Array

String

- Reverse words in a given string
- Permutations of a given string
- Longest Palindrome in a String
- Recursively remove all adjacent duplicates
- Check if string is rotated by two places
- Roman Number to Integer
- Anagram
- Remove Duplicates
- Form a Palindrome
- Longest Distinct Characters in the string
- Implement Atoi
- Implement strstr
- Longest Common Prefix

Linked List

- Finding middle element in a linked list
- Reverse a linked list
- Rotate a Linked List
- Reverse a Linked List in groups of given size
- Intersection point in Y shaped linked lists
- Detect Loop in linked list
- Remove loop in Linked List
- n’th node from end of linked list
- Flattening a Linked List
- Merge two sorted linked lists
- Intersection point of two Linked Lists
- Pairwise swap of a linked list

- Add two numbers represented by linked lists
- Check if Linked List is Palindrome
- Implement Queue using Linked List
- Implement Stack using Linked List
- Given a linked list of 0s, 1s and 2s, sort it
- Delete without head pointer

Stack and Queue

- Parenthesis Checker
- Next larger element
- Queue using two Stacks
- Stack using two queues
- Get minimum element from stack
- LRU Cache
- Circular tour
- First non-repeating character in a stream
- Rotten Oranges
- Maximum of all subarrays of size k

Tree

- Print Left View of Binary Tree
- Check for BST
- Print Bottom View of Binary Tree
- Print a Binary Tree in Vertical Order
- Level order traversal in spiral form
- Connect Nodes at Same Level
- Lowest Common Ancestor in a BST
- Convert a given Binary Tree to Doubly Linked List
- Write Code to Determine if Two Trees are Identical or Not
- Given a binary tree, check whether it is a mirror of itself
- Height of Binary Tree
- Maximum Path Sum
- Diameter of a Binary Tree
- Number of leaf nodes

- Check if given Binary Tree is Height Balanced or Not
- Serialize and Deserialize a Binary Tree

Heap

- Find median in a stream
- Heap Sort
- Operations on Binary Min Heap
- Rearrange characters
- Merge K sorted linked lists
- Kth largest element in a stream

Recursion

- Flood fill Algorithm
- Number of paths
- Combination Sum – Part 2
- Special Keyboard
- Josephus problem

Hashing

- Relative Sorting
- Sorting Elements of an Array by Frequency
- Largest subarray with 0 sum
- Common elements
- Find all four sum numbers
- 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 in a string containing all the characters of another string
- First element to occur k times
- Check if frequencies can be equal

Graph

- Depth First Traversal
- Breadth First Traversal
- Detect cycle in undirected graph
- Detect cycle in a directed graph
- Topological sort
- Find the number of islands
- Implementing Dijkstra
- Minimum Swaps
- Strongly Connected Components
- Shortest Source to Destination Path
- Find whether path exist
- Minimum Cost Path
- Circle of Strings
- Floyd Warshall
- Alien Dictionary
- Snake and Ladder Problem

Greedy

- Activity Selection
- N meetings in one room
- Coin Piles
- Maximize Toys
- Page Faults in LRU
- Largest number possible
- Minimize the heights
- Minimize the sum of product
- Huffman Decoding
- Minimum Spanning Tree

- Shop in Candy Store
- Geek collects the balls

Dynamic Programming

- Minimum Operations
- Max length chain
- Minimum number of Coins
- Longest Common Substring
- Longest Increasing Subsequence
- Longest Common Subsequence
- 0 – 1 Knapsack Problem
- Maximum sum increasing subsequence
- Minimum number of jumps
- Edit Distance
- Coin Change Problem
- Subset Sum Problem
- Box Stacking
- Rod Cutting
- Path in Matrix
- Minimum sum partition
- Count number of ways to cover a distance
- Egg Dropping Puzzle
- Optimal Strategy for a Game
- Shortest Common Supersequence

Divide and Conquer

- Find the element that appears once in sorted array
- Search in a Rotated Array
- Binary Search
- Sum of Middle Elements of two sorted arrays
- Quick Sort
- Merge Sort
- K-th element of two sorted Arrays

Backtracking

- N-Queen Problem
- Solve the Sudoku
- Rat in a Maze Problem
- Word Boggle
- Generate IP Addresses

Bit Magic

- 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

Arrays

- Find Missing And Repeating
- Maximum Index
- Consecutive 1's not allowed
- Majority Element
- Two numbers with sum closest to zero
- Nuts and Bolts Problem
- Boolean Matrix Problem
- Smallest Positive missing number
- Jumping Caterpillars

Strings

- Most frequent word in an array of strings
- CamelCase Pattern Matching
- String Ignorance
- Smallest window in a string containing all the characters of another string
- Design a tiny URL or URL shortener
- Permutations of a given string
- Non Repeating Character
- Check if strings are rotations of each other or not
- Save Ironman
- Repeated Character
- Remove common characters and concatenate
- Geek and its Colored Strings
- Second most repeated string in a sequence

Trees

- Mirror Tree
- Longest consecutive sequence in Binary tree
- Bottom View of Binary Tree
- Lowest Common Ancestor in a Binary Tree
- Binary to DLL

Arrays

1-D Arrays

- **Basic array operations (Insert, delete and search an element)**
- **Smallest and largest element in an array**
- **Sum of elements in an array**
- **Check if two arrays are the same or not**
- **Finding the array type**
- **Sum of positive square elements in an array**
- **Second smallest element in an array**
- **Sorting the elements of an array**
- **Reversing an array**

- **Longest palindrome in an array**
- **Count distinct elements of an array**
- **Print all distinct elements in an array**
- **Non-repeating elements of an array**
- **Repeating elements in an array**
- **Remove duplicate elements in an array**
- **Minimum scalar product of two vectors**
- **Maximum scalar product of two vectors**
- **Can the numbers of an array be made equal?**
- **Missing elements of a range**
- **Triplets with a given sum**
- **Frequency of each element of an array**
- **Symmetric pairs in an array**
- **Maximum product subarray in a given array**
- **Arrays are disjoint or not**
- **Array is a subset of another array or not**
- **Can all numbers of an array be made equal**
- **Minimum sum of absolute difference of given array**
- **Sorting elements of an array by frequency**
- **Sort an array according to the order defined by another array**
- **Replace each element of the array by its rank in the array**
- **Equilibrium index of an array**
- **Array rotation Left and right**
- **Block swap algorithm for array rotation**
- **Juggling algorithm for array rotation**
- **Circular rotation of an array by K positions**
- **Convert an array into a zig-zag fashion**
- **Merge two sorted arrays**
- **Longest subarray having an average greater than or equal to k**
- **Rearrange positive and negative numbers in an array**
- **Sum of all odd frequency elements in an array**
- **Median of two sorted arrays**
- **0-1 Knapsack problem**

- **K'th smallest element in an unsorted array**

2-D Arrays

- Matrix operations (Addition and subtraction)
- Transpose of a matrix
- Upper triangular matrix or not
- Lower triangular matrix or not
- The maximum element in a row
- The maximum element in a column
- Sum of each row and column of a matrix
- Saddle point coordinates of a given matrix
- Sum of elements in the zig-zag sequence of a matrix
- Sum of boundary elements of a matrix
- Matrix printing in a spiral form
- Rotate the matrix by K times
- Matrix rotation by 90 degrees clockwise and anticlockwise
- Maximum size of square submatrix with all 1s in a binary matrix

Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Shell Sort
- Heap Sort
- Comb Sort

Stacks, Queues and Lists

- Stack implementation using arrays
- Stack implementation using a Linked list
- Balanced parenthesis checker
- Redundant braces
- Sorting a stack using a temporary stack

- Infix to postfix using a stack
- Infix to prefix using a stack
- Queue using arrays
- Queue using a linked list
- Circular queue using arrays and Linked list
- Implement stack using a queue
- Implement queue using a stack
- Reversing a queue
- Reverse first k elements of a queue
- Program to return the nth node from the end in a linked list
- Remove duplicates from a linked list
- Reverse a linked list
- Detect a loop in a linked list

Trees and Graphs

- Height of a binary tree
- Product of all leaf nodes of binary tree
- Find kth maximum value in a binary search tree
- Graph is a tree or not
- Nodes at k distance from the root
- Ancestors of a given node in a binary tree
- The vertical sum of a tree
- Implement Depth First Search (DFS)
- Implement Breadth First Search (BFS)
- Number of edges in an undirected graph
- Shortest path between two vertices
- Trace all paths of a directed graph

Miscellaneous problems

- Open and closed gates problem
- Number of Islands
- N digit numbers with digits in non-decreasing order
- Maximum number of As using four keys
- Minimum number of jumps to reach the end of an array

- Subset sum problem
- Rat in a maze problem
- Minimum and maximum values of a given expression
- Minimum sum partition problem
- Number of ways to reach the n'th stair
- Word break problem
- Two player coin game
- Sudoku problem
- Maximum sum in a array such that no two elements are adjacent
- Treasure and cities problem

Question asked company wise.

1. Roman To Integer->Asked in: Amazon, Microsoft, Facebook

2. Reverse Bits->Asked in: Amazon

3. Square Root of Integer->Asked in: Amazon, Microsoft, Facebook

4. Calculate power function->Asked in: Google, LinkedIn, Amazon

5. Greatest Common Divisor->Asked in: Google

6. Find the Closest Palindrome->Asked in: Microsoft, Amazon

7. Rotate matrix->Asked in: Google

8. Spiral Matrix->Asked in: Microsoft, Amazon

9. Wave Array->Asked in: Amazon, Google, Adobe

10. Set Matrix Zeroes->Asked in: Amazon, Google

11. maximum j — i such that A[j] > A[i]->Asked in: Google, Amazon, Adobe

12. Move zeroes to an end->Asked in: Facebook, Uber

13. Merge two sorted arrays->Asked in: Microsoft, Adobe

14. Container with Most Water->Asked in: Amazon, Google, Facebook, Adobe

15. Remove duplicates from sorted array->Asked in: Amazon, Microsoft, Google

16. Find an element in Bitonic array->Asked in: Amazon

17. Find minimum element in sorted and rotated array->Asked in: Facebook

18. Median of two sorted array of same size->Asked in: Amazon, Microsoft, Google

19. Inversion count in an array->Asked in: Amazon, Google

20. Search for a Range in a sorted array->Asked in: Microsoft, Google

21. Longest Common Prefix->Asked in: Amazon, Google

22. Median in row wise sorted matrix->Asked in: Amazon

23. Swap List Nodes in pairs->Asked in: Amazon, Microsoft

24. Add Two Numbers as Lists->Asked in: Amazon, Microsoft, Facebook

25. Check if a singly linked list is palindrome->Asked in: Amazon, Microsoft

26. Reverse a linked list from position m to n->Asked in: Amazon, Microsoft, Facebook

27.Detect and Remove Loop in a Linked List->Asked in: Amazon, Microsoft

28. Merge Two Sorted Lists->Asked in: Amazon, Microsoft, Yahoo

29. Remove Nth Node from List End->Asked in: Amazon

30. Sort a linked list using insertion sort->Asked in: Microsoft, Google

31. Find next greater element in an array->Asked in: Amazon, Microsoft

32. Trapping rain water->Asked in: Amazon, Google

33. Merge overlapping intervals->Asked in: Amazon, Google

34. Largest Rectangle in Histogram->Asked in: Amazon, Google, Facebook

35. Check for balanced parentheses in an expression->Asked in: Amazon, Microsoft

36. Min Stack Problem->Asked in: Amazon, Microsoft, Yahoo, Adobe

37. LRU Cache implementation->Asked in: Amazon, Microsoft, Adobe, Google

38. Sort a stack using another stack->Asked in: Amazon, Microsoft

39. Lowest Common Ancestor of a Binary tree->Asked in: Amazon, Google, Microsoft, Facebook, Adobe

40. Path sum in binary tree->Asked in: Amazon, Microsoft, Yahoo

41. Min Depth of Binary Tree->Asked in: Amazon, Facebook

42. Binary Tree Zigzag Level Order Traversal->Asked in: Amazon, Microsoft

43. Invert Binary Tree->Asked in: Amazon, Google

44. Flatten Binary Tree to Linked List->Asked in: Amazon, Microsoft, Yahoo, Adobe

45. Find diameter of binary tree->Asked in: Facebook, Google, Amazon

46. All Nodes Distance K in Binary Tree->Asked in: Microsoft

47. Merge two binary tree->Asked in: Amazon, Microsoft

48. Shortest Unique Prefix->Asked in: Google

49. Sorted Array To Balanced BST->Asked in: Amazon

50. K-th largest element in BST->Asked in: Amazon

51. Minimum absolute difference in BST->Asked in: Google

52. Recover Binary Search Tree->Asked in: Amazon, Microsoft

53. Merge Two BST->Asked in: Google, Amazon, Microsoft

54. Lowest Common Ancestor of a BST->Asked in: Amazon, Microsoft

55. K Pairs with Smallest Sums->Asked in: Google

56. Sliding window maximum->Asked in: Amazon, Google

57. Merge K sorted list->Asked in: Amazon, Google

58. Convert a min heap to max heap->Asked in: Google

59. Check if two arrays are equal or not->Asked in: Amazon, Goldman Sachs

60. Intersection of two unsorted array->Asked in: Google, Facebook

61. Longest Consecutive Sequence->Asked in: Amazon, Google

62. Valid Anagram->Asked in: Google, Amazon, Microsoft

63. Majority Element->Asked in: Amazon, Microsoft, Yahoo, Google

64. Sort Characters by Frequency->Asked in: Facebook, Google

65. First Unique Character in a String->Asked in: Amazon, Microsoft

66. Triplet with zero sum->Asked in: Facebook, Amazon, Microsoft

67. First missing positive->Asked in: Amazon

68. Largest subarray with 0 sum->Asked in: Microsoft

69. Max points on the straight line->Asked in: Amazon, Google

70. Climbing Stairs Problem->Asked in: Amazon, Google

71. Matrix Chain Multiplication->Asked in: Amazon, Microsoft

72. Longest Increasing subsequence->Asked in: Amazon, Microsoft, Facebook

73. Partition Equal Subset Sum->Asked in: Amazon, Adobe

74. Minimum number of jumps to reach end->Asked in: Amazon, Google, eBay

75. Interleaving String->Asked in: Google, Microsoft, Yahoo

76. Coin change problem->Asked in: Microsoft

77. Edit distance Problem->Asked in: Amazon, Google, Microsoft

78. Min Cost Path->Asked in: Amazon

79. Maximal Square->Asked in: Google, Microsoft

80. Longest Arithmetic Progression->Asked in: Google, Microsoft

81. Word break problem->Asked in: Facebook, Google

82. Maximum Subarray Sum->Asked in: Amazon, Microsoft, Yahoo, Facebook

83. Palindrome Partitioning->Asked in: Amazon, Google

84. Max Product Subarray->Asked in: Amazon, Microsoft

85. Maximum Product of Three Numbers->Asked in: Amazon

86. Gas station Problem->Asked in: Amazon, Google

87. Distribute Candy Problem->Asked in: Amazon, Microsoft

88. Fractional Knapsack problem->Asked in: Amazon

89. Sudoku Solver->Asked in: Amazon, Google, Microsoft

90. Generate Parentheses->Asked in: Facebook, Microsoft

91. Print all subset->Asked in: Amazon, Microsoft, Facebook

92. Combination Sum->Asked in: Amazon, Facebook, Adobe

93. Combinations->Asked in: Amazon, Adobe

94. Letter Combinations of a Phone Number->Asked in: Google

95. Word ladder problem->Asked in: Google, Microsoft, eBay

96. Smallest Multiple With 0 and 1->Asked in: Amazon

97. Check loop in array according to given constraints->Asked in: Google, Amazon

98. Course Schedule->Asked in: Amazon

99. Knight on chessboard->Asked in: Amazon, Goldman Sachs

100. Surrounded regions ->Asked in: Google

Array

1. Find pair with given sum in the array

2. Check if subarray with 0 sum is exists or not

3. Print all sub-arrays with 0 sum

4. Sort binary array in linear time

5. Find a duplicate element in a limited range array

6. Find maximum length sub-array having given sum

7. Find maximum length sub-array having equal number of 0’s and 1’s

8. Find maximum product of two integers in an array

9. Sort an array containing 0’s, 1’s and 2’s (Dutch National Flag Problem)

10. In place merge two sorted arrays

11. Merge two arrays by satisfying given constraints

12. Find index of 0 to replace to get maximum length sequence of continuous ones

13. Shuffle a given array of elements (Fisher–Yates shuffle)

14. Rearrange the array with alternate high and low elements

15. Find equilibrium index of an array

16. Find largest sub-array formed by consecutive integers

17. Find majority element (Boyer–Moore Majority Vote Algorithm)

18. Move all zeros present in the array to the end

19. Replace each element of array with product of every other element without using / operator

20. Find Longest Bitonic Subarray in an array

21. Longest Increasing Subsequence

22. Find maximum difference between two elements in the array by satisfying given constraints

23. Maximum Sum Subarray Problem (Kadane’s Algorithm)

24. Print continuous subarray with maximum sum

25. Maximum Sum Circular Subarray

26. Find all distinct combinations of given length — I

27. Find all distinct combinations of given length with repetition allowed

28. Find maximum sequence of continuous 1’s formed by replacing at-most k zeroes by ones

29. Find minimum sum subarray of given size k

30. Find maximum product subarray in a given array

31. Find subarray having given sum in given array of integers

32. Find the length of smallest subarray whose sum of elements is greater than the given number

33. Find largest number possible from set of given numbers

34. Find the smallest window in array sorting which will make the entire array sorted

35. Find maximum sum path involving elements of given arrays

36. Maximum profit earned by buying and selling shares any number of times

37. Trapping Rain Water within given set of bars

38. Find minimum platforms needed in the station so to avoid any delay in arrival of any train

39. Decode the array constructed from another array

40. Sort an array using one swap

41. Find Triplet with given sum in an array

42. Length of longest continuous sequence with same sum in given binary arrays

43. Reverse every consecutive m elements of the given subarray

44. Maximum Product Subset Problem

45. Find pairs with given difference k in the array

46. Find pairs with given difference k in the array | Constant space solution

47. 4 sum problem | Quadruplets with given sum

48. Print all quadruplets with given sum | 4-sum problem extended

49. Quickselect Algorithm

50. Rearrange array such that A[A[i]] is set to i for every element A[i]

51. Print all Triplets that forms Arithmetic Progression

52. Print all Triplets that forms Geometric Progression

53. Print all combination of numbers from 1 to n having sum n

54. Replace each element of the array by its corresponding rank in the array

55. Print all Triplets in an array with sum less than or equal to given number

56. Group elements of an array based on their first occurrence

57. Find minimum difference between index of two given elements present in the array

58. Find maximum absolute difference between sum of two non-overlapping sub-arrays

59. Find all Symmetric Pairs in an Array of Pairs

60. Partition an array into two sub-arrays with the same sum

61. Find count of distinct elements in every sub-array of size k

62. Find two numbers with maximum sum formed by array digits

63. Print all sub-arrays of an array having distinct elements

64. Find a Triplet having Maximum Product in an Array

65. Find Minimum Index of Repeating Element in an Array

66. Generate random input from an array according to given probabilities

67. Find pair in an array having minimum absolute sum

68. Find Index of Maximum Occurring Element with Equal Probability

69. Check if an Array is Formed by Consecutive Integers

70. Find two non-overlapping pairs having same sum in an array

71. Add elements of two arrays into a new array

72. Find Minimum Product among all Combinations of Triplets in an Array

73. Replace every element of an array with the least greater element on its right

74. Find all odd occurring elements in an array having limited range of elements

75. Count the distinct absolute values in the sorted array

76. Print all combinations of positive integers in increasing order that sum to a given number

77. Find all distinct combinations of given length — II

78. Find subarrays with given sum in an array

79. Find the surpasser count for each element of an array

80. Find maximum length sequence of continuous ones (Using Sliding Window)

81. Find maximum length sequence of continuous ones

82. Find index that divides an array into two non-empty subarrays of equal sum

83. Calculate frequency of all elements present in an array of specified range

84. Rearrange the array such that it contains positive and negative numbers at alternate positions

85. Find a sorted triplet in the given array

86. Shuffle an array according to the given order of elements

87. Count number of strictly increasing sub-arrays in an array

88. Find duplicates within given range k in an array

89. Longest Alternating Subarray Problem

90. Find minimum range with at-least one element from each of the given arrays

91. Find longest subsequence formed by consecutive integers

92. Find all elements in an array that are greater than all elements present to their right

93. Find missing number in array without using extra space

94. Determine index of an element in given array which satisfies given constraints

95. Find minimum moves required for converting a given array to an array of zeroes

96. Left rotate an array

97. Right rotate an array k times

98. Find maximum profit earned from at most two stock transactions

99. Find Frequency of each element in a sorted array containing duplicates

100. Find Minimum and Maximum element in an array using minimum comparisons

101. Difference between Subarray, Subsequence and Subset

102. Find odd occurring element in an array in single traversal

103. Find odd occurring element in logarithmic time

104. Find two odd occurring elements in an array without using any extra space

105. Check if given array represents min heap or not

106. Find K’th smallest element in an array

107. Find K’th largest element in an array

108. Sort a K-Sorted Array

109. Merge M sorted lists of variable length

110. Find smallest range with at-least one element from each of the given lists

111. Merge M sorted lists each containing N elements

112. Find maximum sum of subsequence with no adjacent elements

113. Find ways to calculate a target from elements of specified array

114. Sort elements by their frequency and Index

115. Sort an array based on order defined by another array

116. Inversion Count of an array

117. Segregate positive and negative integers in linear time

118. Find number of rotations in a circularly sorted array

119. Search an element in a circular sorted array

120. Find first or last occurrence of a given number in a sorted array

121. Count occurrences of a number in a sorted array with duplicates

122. Find smallest missing element from a sorted array

123. Find Floor and Ceil of a number in a sorted array

124. Search in a nearly sorted array in logarithmic time

125. Find number of 1’s in a sorted binary array

126. Find Missing Term in a Sequence in Logarithmic time

127. Find missing number and duplicate elements in an array

128. Find the peak element in an array

129. Find Floor and Ceil of a number in a sorted array (Recursive solution)

130. Print all distinct subsets of a given set

131. Find two duplicate elements in a limited range array (using XOR)

132. Combinations of words formed by replacing given numbers with corresponding alphabets

133. 0–1 Knapsack Problem

134. Subset sum Problem

135. Partition Problem

136. 3-Partition Problem

137. 3-partition problem extended | Print all partitions

138. K-Partition Problem | Printing all Partitions

139. Minimum Sum Partition Problem

140. Rod Cutting

141. Longest Alternating Subsequence Problem

142. Coin change-making problem (unlimited supply of coins)

143. Coin Change Problem — Find total number of ways to get the denomination of coins

144. Find maximum profit earned from at most K stock transactions

Backtracking

- Print all possible solutions to N Queens Problem
- Print all Possible Knight's Tours in a chessboard
- Find Shortest Path in Maze
- Find Longest Possible Route in a Matrix
- Find path from source to destination in a matrix that satisfies given constraints
- Find total number of unique paths in a maze from source to destination
- Print All Hamiltonian Path present in a graph
- Print all k-colorable configurations of the graph (Vertex coloring of graph)
- Find all Permutations of a given string
- All combinations of elements satisfying given constraints
- Find all binary strings that can be formed from given wildcard pattern
- K-Partition Problem | Printing all Partitions
- Magnet Puzzle
- Find ways to calculate a target from elements of specified array
- Find minimum number possible by doing at-most K swaps
- Determine if a pattern matches with a string or not
- Generate list of possible words from a character matrix
- Find the path between given vertices in a directed graph
- Find all Possible Topological Orderings of a DAG
- Print all shortest routes in a rectangular grid

Binary Tree

- Check if two given binary trees are identical or not
- Calculate height of a binary tree
- Delete given Binary Tree
- Inorder Tree Traversal (Iterative & Recursive Implementation)
- Preorder Tree Traversal (Iterative & Recursive Implementation)
- Postorder Tree Traversal (Iterative & Recursive Implementation)
- Level Order Traversal of Binary Tree
- Spiral Order Traversal of Binary Tree
- Reverse Level Order Traversal of Binary Tree
- Print all nodes of a given binary tree in specific order

- Print left view of binary tree
- Print Bottom View of Binary Tree
- Print Top View of Binary Tree
- Find next node in same level for given node in a binary tree
- Check if given binary tree is complete binary tree or not
- In-place convert given binary tree to its sum tree
- Determine if given two nodes are cousins of each other
- Print cousins of given node in a binary tree
- Check if given binary tree is a sum tree or not
- Combinations of words formed by replacing given numbers with corresponding alphabets
- Determine if given binary tree is a subtree of another binary tree or not
- Find diameter of a binary tree
- Check if given binary Tree has symmetric structure or not
- Convert binary tree to its mirror
- Check if binary tree can be converted to another by doing any no. of swaps of left & right child
- Find Lowest Common Ancestor (LCA) of two nodes in a binary tree
- Print all paths from root to leaf nodes in a binary tree
- Find ancestors of given node in a Binary Tree
- Find the distance between given pairs of nodes in a binary tree
- Find Vertical Sum in a given Binary Tree
- Perform vertical traversal of a binary tree — I
- Perform vertical traversal of a binary tree — II
- Print corner nodes of every level in binary tree
- Find the diagonal sum of given binary tree
- Print Diagonal Traversal of Binary Tree
- In-place convert Binary Tree to Doubly Linked List
- Sink nodes containing zero to the bottom of the binary tree
- Convert given binary tree to full tree by removing half nodes
- Truncate given binary tree to remove nodes which lie on a path having sum less than K
- Find maximum sum root-to-leaf path in a binary tree
- Check if given binary tree is height balanced or not
- Find maximum width of given binary tree
- Convert normal binary tree to Left-child right-sibling binary tree
- Determine if given Binary Tree is a BST or not

- Convert a Binary Tree to BST by maintaining its original structure
- Invert a Binary Tree
- Print Right View of a Binary Tree
- Print all paths from leaf to root node in given binary tree
- Iteratively print leaf to root path for every leaf node in a binary tree
- Build Binary Tree from given Parent array
- Find all nodes at given distance from leaf nodes in a binary tree
- Count all subtrees having same value of nodes in a binary tree
- Find Maximum Difference Between a Node and its Descendants in a Binary Tree
- Construct a Binary Tree from Ancestor Matrix
- Calculate height of a binary tree with leaf nodes forming a circular doubly linked list
- Find maximum sum path between two leaves in a binary tree
- Fix a binary tree that is only one swap away from becoming a BST
- Construct a binary tree from inorder and preorder traversal
- Construct a binary tree from inorder and postorder traversals
- Construct a binary tree from inorder and level order sequence
- Construct a full binary tree from preorder sequence with leaf node information
- Construct a full binary tree from a preorder and postorder sequence
- Set next pointer to inorder successor of all nodes in binary tree
- Efficiently print all nodes between two given levels in a binary tree
- Find preorder traversal of a binary tree from its inorder and postorder sequence
- Find the difference between sum of all nodes present at odd and even levels in a binary tree
- Find the size of the largest BST in a Binary Tree
- Link nodes present in each level of a binary tree in the form of a linked list
- Construct a Cartesian Tree from In-order Traversal
- Implementation of Treap Data Structure (Insert, Search and Delete)
- Clone a binary tree with random pointers
- Threaded Binary Tree: Overview and Implementation
- Invert alternate levels of a perfect binary tree
- Convert a Binary Tree into a Doubly Linked List in Spiral Order
- Check if a binary tree is a min-heap or not
- Determine if a binary tree satisfy the height-balanced property of red–black tree
- Depth first search (DFS) vs Breadth first search (BFS)

BST

- Insertion in BST
- Search given key in BST
- Deletion from BST
- Construct balanced BST from given keys
- Determine if given Binary Tree is a BST or not
- Check if given keys represents same BSTs or not without building the BST
- Find inorder predecessor for given key in a BST
- Find Lowest Common Ancestor (LCA) of two nodes in a Binary Search Tree
- Find K’th smallest and K’th largest element in BST
- Floor and Ceil in a Binary Search Tree
- Find optimal cost to construct binary search tree
- Convert a Binary Tree to BST by maintaining its original structure
- Remove nodes from BST that have keys outside the valid range
- Find a pair with given sum in a BST
- Find inorder successor for given key in a BST
- Replace every element of an array with the least greater element on its right
- Fix a binary tree that is only one swap away from becoming a BST
- Update every key in BST to contain sum of all greater keys
- Check if a given sequence represents preorder traversal of a BST
- Build a Binary Search Tree from a Postorder Sequence
- Build a Binary Search Tree from a Preorder Sequence
- Find a triplet with given sum in a BST
- Count subtrees in a BST whose nodes lies within a given range
- Merge two BSTs into a doubly linked list in sorted order
- Construct a height-balanced BST from an unbalanced BST
- Find the size of the largest BST in a Binary Tree
- Convert a Binary Search Tree into a Min Heap
- Construct a Height-Balanced BST from a Sorted Doubly Linked List

Divide & Conquer

- Binary Search Algorithm
- Find number of rotations in a circularly sorted array
- Search an element in a circular sorted array
- Find first or last occurrence of a given number in a sorted array

- Count occurrences of a number in a sorted array with duplicates
- Find smallest missing element from a sorted array
- Find Floor and Ceil of a number in a sorted array
- Search in a nearly sorted array in logarithmic time
- Find number of 1's in a sorted binary array
- Find the peak element in an array
- Maximum Sum Subarray using Divide & Conquer
- Efficiently implement a power function
- Find Missing Term in a Sequence in Logarithmic time
- Division of Two Numbers using Binary Search Algorithm
- Find Floor and Ceil of a number in a sorted array (Recursive solution)
- Find Frequency of each element in a sorted array containing duplicates
- Find odd occurring element in logarithmic time
- Ternary Search vs Binary search
- Exponential search
- Unbounded Binary Search
- Interpolation search
- Merge Sort Algorithm
- QuickSort Algorithm

Dynamic Programming

- Introduction to Dynamic Programming
- Longest Common Subsequence Problem
- Longest Common Subsequence | Space optimized version
- Longest Common Subsequence of K-sequences
- Longest Common Subsequence | Finding all LCS
- Longest Common Substring Problem
- Longest Palindromic Subsequence Problem
- Longest Repeated Subsequence Problem
- Implement Diff Utility
- Shortest Common Supersequence Problem
- Shortest Common Supersequence | Finding all SCS
- Shortest Common Supersequence Problem using LCS
- Longest Increasing Subsequence Problem

- Longest Decreasing Subsequence Problem
- Longest Bitonic Subsequence
- Increasing Subsequence with Maximum Sum
- The Levenshtein Distance (Edit Distance) Problem
- Find size of largest square sub-matrix of 1’s present in given binary matrix
- Matrix Chain Multiplication
- Find the minimum cost to reach last cell of the matrix from its first cell
- Find longest sequence formed by adjacent numbers in the matrix
- Count number of paths in a matrix with given cost to reach destination cell
- 0–1 Knapsack Problem
- Maximize value of the expression
- Partition Problem
- Subset sum Problem
- 3-Partition Problem
- Minimum Sum Partition Problem
- Rod Cutting
- Maximum Product Rod Cutting
- Coin change-making problem (unlimited supply of coins)
- Coin Change Problem — Find total number of ways to get the denomination of coins
- Total possible solutions to linear equation of k variables
- Longest Alternating Subsequence Problem
- Count number of times a pattern appears in given string as a subsequence
- Collect maximum points in a matrix by satisfying given constraints
- Find all N-digit binary strings without any consecutive 1’s
- Count total possible combinations of N-digit numbers in a mobile keypad
- Word Break Problem
- Determine Minimal Adjustment Cost of an Array
- Check if a string is K-Palindrome or not
- Find total ways to achieve given sum with n throws of dice having k faces
- Wildcard Pattern Matching
- Find number of ways to fill a N x 4 matrix with 1 x 4 tiles
- Ways to reach the bottom-right corner of a matrix with exactly k turns allowed
- Weighted Interval Scheduling Problem
- Box Stacking Problem

- Find total ways to reach the n’th stair with at-most m steps
- Find total ways to reach the n’th stair from the bottom
- Activity Selection Problem
- Find minimum number of deletions required to convert a string into palindrome
- Calculate minimum cost to reach destination city from source city
- Pots of Gold Game Problem
- Find minimum cuts needed for palindromic partition of a string
- Weighted Interval Scheduling using LIS algorithm
- Find minimum jumps required to reach the destination
- Find probability that a person is alive after taking N steps on the island
- Find maximum sum of subsequence with no adjacent elements
- Maximum Length Snake Sequence
- Calculate size of the largest plus of 1’s in binary matrix
- Longest Increasing Subsequence using LCS
- Find maximum profit earned from at most K stock transactions
- Count all paths in a matrix from first cell to last cell
- Check if a string matches with a given wildcard pattern
- Check if given string is interleaving of two other given strings
- Find all employees who directly or indirectly reports to a manager
- Find optimal cost to construct binary search tree
- Find maximum sum of subsequence with no adjacent elements
- Maximum Sum Subarray Problem (Kadane’s Algorithm)
- Longest Alternating Subarray Problem
- Collect maximum value of coins in a matrix
- Find length of longest path in the matrix with consecutive characters
- Find ways to calculate a target from elements of specified array
- Calculate sum of all elements in a sub-matrix in constant time
- Find maximum sum K x K sub-matrix in a given M x N matrix
- Find maximum sum submatrix present in a given matrix
- Single-Source Shortest Paths — Bellman Ford Algorithm
- All-Pairs Shortest Paths — Floyd Warshall Algorithm

Graph

- Terminology and Representations of Graphs

- Graph Implementation — C, C++, C++ (STL), Java (Collections), Python
- Breadth First Search (BFS) Algorithm
- Depth First Search (DFS) Algorithm
- Depth first search (DFS) vs Breadth first search (BFS)
- Arrival and Departure Time of Vertices in DFS
- Types of edges involved in DFS and relation between them
- Bipartite Graph
- Determine if a given graph is Bipartite Graph using DFS
- Snake and Ladder Problem
- Topological Sorting in a DAG
- Kahn’s Topological Sort Algorithm
- Transitive Closure of a Graph
- Check if an undirected graph contains cycle or not
- Total paths in given digraph from given source to destination having exactly m edges
- Determine if an undirected graph is a Tree (Acyclic Connected Graph)
- 2-Edge Connectivity in the graph
- 2-Vertex Connectivity in the graph
- Check if given digraph is a DAG (Directed Acyclic Graph) or not
- Disjoint-Set Data Structure (Union-Find Algorithm)
- Chess Knight Problem — Find Shortest path from source to destination
- Check if given Graph is Strongly Connected or not
- Check if given Graph is Strongly Connected or not using one DFS Traversal
- Union-Find Algorithm for Cycle Detection in undirected graph
- Kruskal’s Algorithm for finding Minimum Spanning Tree
- Single-Source Shortest Paths — Dijkstra’s Algorithm
- Single-Source Shortest Paths — Bellman Ford Algorithm
- All-Pairs Shortest Paths — Floyd Warshall Algorithm
- Find Cost of Shortest Path in DAG using one pass of Bellman-Ford
- Least Cost Path in Weighted Digraph using BFS
- Find maximum cost path in graph from given source to destination
- Determine negative-weight cycle in a graph
- Least cost path in given digraph from given source to destination having exactly m edges
- Find the path between given vertices in a directed graph

- Find all Possible Topological Orderings of a DAG
- Find the correct order of alphabets in a given dictionary of ancient origin
- Find longest path in a Directed Acyclic Graph (DAG)
- Construct a directed graph from undirected graph that satisfies given constraints
- Print all k-colorable configurations of the graph (Vertex coloring of graph)
- Print All Hamiltonian Path present in a graph
- Graph Coloring Problem

Greedy

- Activity Selection Problem
- Huffman Coding
- Job Sequencing Problem with Deadlines
- Graph Coloring Problem
- Kruskal’s Algorithm for finding Minimum Spanning Tree
- Single-Source Shortest Paths — Dijkstra’s Algorithm
- Shortest Superstring Problem

Heap

- Introduction to Priority Queues using Binary Heaps
- Min Heap and Max Heap Implementation — C++, Java
- Heap Sort Algorithm
- Check if given array represents min heap or not
- Convert Max Heap to Min Heap in linear time
- Find K’th largest element in an array
- Sort a K-Sorted Array
- Merge M sorted lists of variable length
- Merge K sorted linked lists
- Find K’th smallest element in an array
- Find smallest range with at-least one element from each of the given lists
- Merge M sorted lists each containing N elements
- Find first k non-repeating characters in a string in single traversal
- Find first k maximum occurring words in given set of strings
- Implementation of Treap Data Structure (Insert, Search and Delete)
- Convert a Binary Search Tree into a Min Heap

- Check if a binary tree is a min-heap or not
- Huffman Coding
- External Merge Sort Algorithm

Linked List

- Introduction to Linked Lists
- Linked List Implementation — C, C++, Java, Python
- Linked List | Insertion at Tail
- Static Linked List
- Clone given Linked List
- Delete Linked List
- Pop operation in linked list
- Insert given node into the correct sorted position in the given sorted linked list
- Rearrange linked list in increasing order (Sort linked list)
- Split the nodes of the given linked list into front and back halves
- Remove duplicates from a sorted linked list
- Move front node of the given list to the front of the another list
- Move even nodes to the end of the list in reverse order
- Split given linked list into two lists where each list containing alternating elements from it
- Construct a linked list by merging alternate nodes of two given lists
- Merge Sort Algorithm for Singly Linked List
- Merge two sorted linked lists into one
- Merge K sorted linked lists
- Intersection of two given sorted linked lists
- Reverse Linked List (Iterative Solution)
- Reverse Linked List (Recursive Solution)
- Reverse every group of k nodes in given linked list
- Find K’th node from the end in a linked list
- Merge alternate nodes of two linked lists into the first list
- Merge two sorted linked lists from their end
- Delete every N nodes in a linked list after skipping M nodes
- Rearrange linked list in specific manner in linear time
- Check if linked list is palindrome or not
- Move last node to front in a given Linked List

- Rearrange the linked list in specific manner
- Detect Cycle in a linked list (Floyd’s Cycle Detection Algorithm)
- Sort linked list containing 0’s, 1’s and 2’s
- Implement Stack using Linked List
- Implement Queue using Linked List
- Remove duplicates from a linked list
- Rearrange the linked list so that it has alternating high, low values
- Rearrange a Linked List by Separating Odd Nodes from the Even Ones
- Calculate height of a binary tree with leaf nodes forming a circular doubly linked list
- XOR Linked List: Overview and Implementation
- Convert a multilevel linked list to a singly linked list
- Recursively check if linked list of characters is palindrome or not
- Merge two BSTs into a doubly linked list in sorted order
- Remove redundant nodes from a path formed by a linked list
- Add a single-digit number to a linked list representing a number
- Reverse every alternate group of k nodes in a linked list
- Determine if a given linked list is a palindrome or not
- Sort a Doubly Linked List using Merge Sort
- Reverse a Doubly Linked List
- Pairwise swap adjacent nodes of a linked list
- Flatten a linked list
- Check if a Linked List of String is Palindromic
- Flatten a multilevel linked list
- Construct a height-balanced BST from an unbalanced BST
- Swap K’th node from beginning with K’th node from end in a Linked List
- Add two linked lists without using any extra space
- Clone a Linked List with Random Pointers
- Update random pointer for each linked list node to point to the maximum node
- Link nodes present in each level of a binary tree in the form of a linked list
- Convert a Ternary Tree to a Doubly Linked List
- Print nodes of a given binary tree in vertical order
- Convert a Binary Tree into a Doubly Linked List in Spiral Order
- Construct a Height-Balanced BST from a Sorted Doubly Linked List
- In-place merge two sorted linked lists without modifying links of the first list

- Reverse specified portion of a Linked List

Matrix

- Print Matrix in Spiral Order
- Create Spiral Matrix from given array
- Shift all matrix elements by 1 in Spiral Order
- Find Shortest path from source to destination in a matrix that satisfies given constraints
- Change all elements of row i and column j in a matrix to 0 if cell (i, j) has value 0
- Print diagonal elements of the matrix having positive slope
- Find all paths from first cell to last cell of a matrix
- Replace all occurrences of 0 that are not surrounded by 1 in a binary matrix
- In-place rotate the matrix by 90 degrees in clock-wise direction
- Count negative elements present in sorted matrix in linear time
- Report all occurrences of an element in row wise and column wise sorted matrix in linear time
- Calculate sum of all elements in a sub-matrix in constant time
- Find maximum sum K x K sub-matrix in a given M x N matrix
- Find maximum sum submatrix present in a given matrix
- Count the number of islands
- Flood Fill Algorithm
- Find shortest safe route in a field with sensors present
- Find all occurrences of given string in a character matrix
- Shortest path in a Maze | Lee Algorithm
- Check if given matrix is Toeplitz matrix or not
- In-place rotate the matrix by 180 degrees
- Fill Binary Matrix with Alternating Rectangles of 0 and 1
- Find all common elements present in every row of given matrix
- Construct a Binary Tree from Ancestor Matrix
- Find common elements present in all rows of a matrix
- Find index of the row containing maximum number of 1’s in a binary matrix
- Find the largest square sub-matrix which is surrounded by all 1's
- Find minimum passes required to convert all negative values in the matrix
- Print a spiral square matrix without using any extra space
- Print all shortest routes in a rectangular grid
- Find length of longest path in the matrix with consecutive characters

- Collect maximum value of coins in a matrix
- Young Tableau | Insert, Search, Extract-Min, Delete, Replace
- Sort an array using Young tableau
- Find path from source to destination in a matrix that satisfies given constraints
- Generate list of possible words from a character matrix
- Find probability that a person is alive after taking N steps on the island
- Collect maximum points in a matrix by satisfying given constraints
- Count number of paths in a matrix with given cost to reach destination cell
- Find longest sequence formed by adjacent numbers in the matrix
- Find the minimum cost to reach last cell of the matrix from its first cell
- Ways to reach the bottom-right corner of a matrix with exactly k turns allowed
- Matrix Chain Multiplication
- Find size of largest square sub-matrix of 1's present in given binary matrix
- Chess Knight Problem — Find Shortest path from source to destination
- Find Duplicate rows in a binary matrix
- Print all possible solutions to N Queens Problem
- Print all Possible Knight's Tours in a chessboard
- Find Shortest Path in Maze
- Find Longest Possible Route in a Matrix
- Find total number of unique paths in a maze from source to destination
- Calculate size of the largest plus of 1's in binary matrix
- Find the maximum value of M[c][d] — M[a][b] over all choices of indexes
- Find shortest distance of every cell from landmine in a Maze
- Find shortest route in a device to construct the given string
- Calculate minimum cost to reach destination city from source city
- Count all paths in a matrix from first cell to last cell
- Merge M sorted lists each containing N elements
- Travelling Salesman Problem using Branch and Bound

Puzzles

- Clock Angle Problem — Find angle between hour and minute hand
- Add two numbers without using addition operator
- Generate power set of a given set
- Implement power function without using multiplication and division operators

- Print all numbers between 1 to N without using semicolon
- Swap two numbers without using third variable
- Determine the if condition to print specific output
- Find maximum & minimum of triplet without using conditional statement and ternary operator
- Find numbers represented as sum of two cubes for two different pairs
- Print “Hello World” with empty main() function
- Tower of Hanoi Problem
- Print all numbers between 1 to N without using any loop
- Print a semicolon without using semicolon anywhere in the program
- Multiply two numbers without using multiplication operator or loops
- Find square of a number without using multiplication and division operator
- Find if a number is even or odd without using any conditional statement
- Set both elements of a binary array to 0 in single line
- Find minimum number without using conditional statement or ternary operator
- Perform Division of two numbers without using division operator (/)
- Generate 0 and 1 with 75% and 25% Probability
- Generate Desired Random Numbers With Equal Probability
- Return 0, 1 and 2 with equal Probability using the specified function
- Generate Fair Results from a Biased Coin
- Generate numbers from 1 to 7 with equal probability using specified function
- Implement Ternary Operator Without Using Conditional Expressions
- Determine if two integers are equal without using comparison and arithmetic operators
- Return 0 and 1 with equal Probability using the specified function
- Generate random input from an array according to given probabilities
- Compute modulus division without division and modulo operator

Queue

- Queue Implementation using Array/List — C, C++, Java, Python
- Queue Implementation using Linked List
- Implement Stack using Queue Data Structure
- Implement a Queue using Stack Data Structure
- Efficiently print all nodes between two given levels in a binary tree
- Chess Knight Problem — Find Shortest path from source to destination
- Shortest path in a Maze | Lee Algorithm

- Find shortest safe route in a field with sensors present
- Flood Fill Algorithm
- Count the number of islands
- Find Shortest path from source to destination in a matrix that satisfies given constraints
- Generate binary numbers between 1 to N
- Calculate height of a binary tree
- Delete given Binary Tree
- Level Order Traversal of Binary Tree
- Spiral Order Traversal of Binary Tree
- Reverse Level Order Traversal of Binary Tree
- Print all nodes of a given binary tree in specific order
- Print left view of binary tree
- Find next node in same level for given node in a binary tree
- Check if given binary tree is complete binary tree or not
- Print Diagonal Traversal of Binary Tree
- Print corner nodes of every level in binary tree
- Invert a Binary Tree
- Find minimum passes required to convert all negative values in the matrix
- Convert a Binary Tree into a Doubly Linked List in Spiral Order
- Check if a binary tree is a min-heap or not
- Invert alternate levels of a perfect binary tree
- Convert a Binary Search Tree into a Min Heap
- Snake and Ladder Problem
- Find shortest distance of every cell from landmine in a Maze
- Convert a multilevel linked list to a singly linked list
- Breadth First Search (BFS) Algorithm
- Check if an undirected graph contains cycle or not
- Find maximum cost path in graph from given source to destination
- Total paths in given digraph from given source to destination having exactly m edges
- Least cost path in given digraph from given source to destination having exactly m edges

Sorting

- Insertion Sort Algorithm
- Selection Sort Algorithm

- Bubble Sort Algorithm
- Merge Sort Algorithm
- Iterative Merge Sort Algorithm (Bottom-up Merge Sort)
- QuickSort Algorithm
- Iterative Implementation of QuickSort
- Hybrid QuickSort
- QuickSort using Dutch National Flag Algorithm
- QuickSort using Hoare's Partitioning scheme
- Heap Sort Algorithm
- Introsort Algorithm
- External Merge Sort Algorithm
- Counting Sort Algorithm
- Inversion Count of an array
- Sort an array using Young tableau
- Merge Sort Algorithm for Singly Linked List
- Problems solved using partitioning logic of QuickSort
- Sort a Doubly Linked List using Merge Sort
- Sort elements by their frequency and Index
- Sort an array based on order defined by another array
- Efficiently sort an array with many duplicated values
- Find largest number possible from set of given numbers
- Find the surpasser count for each element of an array
- Segregate positive and negative integers using Merge Sort
- Group anagrams together from given list of words

Stack

- Stack Implementation using Array/List — C, C++, Java, Python
- Stack Implementation using Linked List
- Check if given expression is balanced expression or not
- Find duplicate parenthesis in an expression
- Evaluate given postfix expression
- Decode the given sequence to construct minimum number without repeated digits
- Design a stack which returns minimum element in constant time
- Design a stack which returns minimum element without using auxiliary stack

- Merging Overlapping Intervals
- Reverse String without using Recursion
- Implement Stack using Queue Data Structure
- Implement a Queue using Stack Data Structure
- Implement two stacks in a single array
- Recursive solution to sort a stack
- Find length of the longest balanced parenthesis in a string
- Reverse a string using stack data structure
- Find all elements in an array that are greater than all elements present to their right
- Inorder Tree Traversal
- Preorder Tree Traversal
- Postorder Tree Traversal
- Find preorder traversal of a binary tree from its inorder and postorder sequence
- Find ancestors of given node in a Binary Tree
- Check if two given binary trees are identical or not
- Reverse Level Order Traversal of Binary Tree
- Reverse given text without reversing the individual words
- Find all binary strings that can be formed from given wildcard pattern
- Iterative Implementation of QuickSort
- Depth First Search (DFS) Algorithm
- Invert a Binary Tree
- Print leaf to root path for every leaf node in a binary tree
- Longest Increasing Subsequence
- Invert alternate levels of a perfect binary tree

String

- Check if given string is a rotated palindrome or not
- Longest Palindromic Substring (Non-DP Space Optimized Solution)
- Check if repeated subsequence is present in the string or not
- Check if strings can be derived from each other by circularly rotating them
- Check if given set of moves is circular or not
- Convert given number into corresponding excel column name
- Determine if two strings are anagram or not
- Find all binary strings that can be formed from given wildcard pattern

- Find all interleaving of given strings
- Isomorphic Strings
- Find all possible palindromic substrings in a string
- Find all possible combinations of words formed from mobile keypad
- Find all possible combinations by replacing given digits with characters of the corresponding list
- Find all words from given list that follows same order of characters as given pattern
- Group anagrams together from given list of words
- Find minimum operations required to transform a string into another string
- Determine if a string can be transformed into another string with a single edit
- Find length of the longest balanced parenthesis in a string
- In place remove all occurrences of ‘AB’ and ‘C’ from the string
- Longest even length palindromic sum substring
- Print string in zig-zag form in k rows
- Reverse given text without reversing the individual words
- Run Length Encoding (RLE) Data Compression Algorithm
- Find the longest substring of given string containing k distinct characters
- Find all palindromic permutations of a string
- Find all substrings of a string that are permutation of a given string
- Find the longest substring of given string containing all distinct characters
- Iterative Approach to find Permutations of a String
- Generate all Permutations of a String in Java
- Find all lexicographically next permutations of a string sorted in ascending order
- Find Lexicographically minimal string rotation
- Find all strings of given length containing balanced parentheses
- Find all combinations of non-overlapping substrings of a string
- Determine if a given string is palindrome or not
- Find the minimum number of inversions needed to make the given expression balanced
- Construct the longest palindrome by shuffling or deleting characters from a string
- Print all combinations of phrases formed by picking words from each of the given lists
- Break a string into all possible combinations of non-overlapping substrings
- Remove all extra spaces from a string
- Remove adjacent duplicate characters from a string
- Find first non-repeating character in a string by doing only one traversal of it
- Find all N-digit strictly increasing numbers (Bottom-Up and Top-Down Approach)

- Find all N-digit binary numbers having more 1’s than 0’s for any prefix
- Find all N-digit numbers with given sum of digits
- Find all N-digit binary numbers with k-bits set where k ranges from 1 to N
- Find all N-digit binary numbers with equal sum of bits in its two halves
- Find all N-digit numbers with equal sum of digits at even and odd index
- Find all Lexicographic Permutations of a String
- Lexicographic Rank of a String
- Find all lexicographically previous permutations of a string sorted in descending order
- Replace all non-overlapping occurrences of the pattern
- Introduction to Pattern Matching
- Implementation of KMP Algorithm
- Reverse String without using Recursion
- Reverse given string using Recursion
- Determine if characters of a String follow a specified order or not
- In-place remove all adjacent duplicates from the given string
- Check if given sentence is syntactically correct or not
- Find all Permutations of a given string
- Find first k non-repeating characters in a string in single traversal
- Check if given string is interleaving of two other given strings
- Decode the given sequence to construct minimum number without repeated digits
- Combinations of words formed by replacing given numbers with corresponding alphabets
- Count number of times a pattern appears in given string as a subsequence
- Check if a string matches with a given wildcard pattern
- Find all words matching a pattern in the given dictionary
- The Levenshtein Distance (Edit Distance) Problem
- Longest Common Subsequence Problem
- Longest Repeated Subsequence Problem
- Longest Palindromic Subsequence using Dynamic Programming
- Longest Common Substring Problem
- Shortest Common Supersequence Problem
- Word Break Problem
- Wildcard Pattern Matching
- Find minimum cuts needed for palindromic partition of a string
- Check if a string is K-Palindrome or not

- Find shortest route in a device to construct the given string
- Find minimum number possible by doing at-most K swaps
- Determine if a pattern matches with a string or not
- Find minimum number of deletions required to convert a string into palindrome

Trie

- Trie Implementation — C, C++, Java, Python
- Memory Efficient Implementation of Trie | Insert, Search and Delete
- Longest Common Prefix in given set of strings (using Trie)
- Lexicographic sorting of given set of keys
- Find maximum occurring word in given set of strings
- Find first k maximum occurring words in given set of strings
- Find Duplicate rows in a binary matrix
- Word Break Problem | Using Trie
- Generate list of possible words from a character matrix
- Find all words matching a pattern in the given dictionary

Top Algorithms:

- Binary Search Algorithm
- Breadth First Search (BFS) Algorithm
- Depth First Search (DFS) Algorithm
- Inorder, Preorder, Postorder Tree Traversals
- Insertion Sort, Selection Sort, Merge Sort, Quicksort, Counting Sort, Heap Sort
- Kruskal’s Algorithm
- Floyd Warshall Algorithm
- Dijkstra’s Algorithm
- Bellman Ford Algorithm

Top 10 algorithms in Interview Questions

- Difficulty Level : Medium

- Last Updated : 21 Aug, 2017

Top 10 coding problems of important topics

Topics :

1. Graph
2. Linked List
3. Dynamic Programming
4. Sorting And Searching
5. Tree / Binary Search Tree
6. Number Theory
7. BIT Manipulation
8. String / Array

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

Linked List

1. Insertion of a node in Linked List (On the basis of some constraints)

2. Delete a given node in Linked List (under given constraints)

3. Compare two strings represented as linked lists

4. Add Two Numbers Represented By Linked Lists

5. Merge A Linked List Into Another Linked List At Alternate Positions

6. Reverse A List In Groups Of Given Size

7. Union And Intersection Of 2 Linked Lists

8. Detect And Remove Loop In A Linked List

9. Merge Sort For Linked Lists

10. Select A Random Node from A Singly Linked List

Dynamic Programming

1. Longest Common Subsequence

2. Longest Increasing Subsequence

3. Edit Distance

4. Minimum Partition

5. Ways to Cover a Distance

6. Longest Path In Matrix

7. Subset Sum Problem

8. Optimal Strategy for a Game

9. 0-1 Knapsack Problem

10. Boolean Parenthesization Problem

Sorting And Searching

1. 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

Tree / Binary Search Tree

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

Number Theory

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

BIT Manipulation

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

String / Array

1. 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

- Graph algorithms
- Dynamic programming
- Searching and Sorting:
- Number theory and Other Mathematical
- Geometrical and Network Flow Algorithms
- Data Structures

The links below cover most important algorithms and data structure topics:

Graph Algorithms

- 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**
- Minimum Spanning tree **Prim**
- Minimum Spanning tree **Kruskal**
- Topological Sort
- Johnson's algorithm
- Articulation Points (or Cut Vertices) in a Graph
- Bridges in a graph

All Graph Algorithms

Dynamic Programming

- Longest Common Subsequence
- Longest Increasing Subsequence
- Edit Distance
- Minimum Partition
- Ways to Cover a Distance
- Longest Path In Matrix
- Subset Sum Problem
- Optimal Strategy for a Game
- 0-1 Knapsack Problem
- Assembly Line Scheduling

All DP Algorithms

Searching And Sorting

- Binary Search
- Quick Sort
- Merge Sort
- Order Statistics
- KMP algorithm
- Rabin karp
- Z's algorithm
- Aho Corasick String Matching
- Counting Sort
- Manacher's algorithm: Part 1, Part 2 and Part 3
- All Articles on Searching, Sorting and Pattern Searching.
- Number theory and Other Mathematical
- Prime Numbers and Prime Factorization
- Primality Test | Set 1 (Introduction and School Method)
- Primality Test | Set 2 (Fermat Method)
- Primality Test | Set 3 (Miller–Rabin)
- Sieve of Eratosthenes
- Segmented Sieve
- Wilson's Theorem

- Prime Factorization
- Pollard’s rho algorithm

Modulo Arithmetic Algorithms

- Basic and Extended Euclidean algorithms
- Euler’s Totient Function
- Modular Exponentiation
- Modular Multiplicative Inverse
- Chinese remainder theorem Introduction
- Chinese remainder theorem and Modulo Inverse Implementation
- nCr%m and this.

Miscellaneous:

- Counting Inversions
- Counting Inversions using BIT
- logarithmic exponentiation
- Square root of an integer
- Heavy light Decomposition , this and this
- Matrix Rank
- Gaussian Elimination to Solve Linear Equations
- Hungarian algorithm
- Link cut
- Mo’s algorithm and this
- Factorial of a large number in C++
- Factorial of a large number in Java+
- Russian Peasant Multiplication
- Catalan Number

All Articles on Mathematical Algorithms

Geometrical and Network Flow Algorithms

- Convex Hull
- Graham Scan
- Line Intersection

- Interval Tree
- Matrix Exponentiation and this
- Maxflow Ford Furkerson Algo and Edmond Karp Implementation
- Min cut
- Stable Marriage Problem
- Hopcroft–Karp Algorithm for Maximum Matching
- Dinic’s algo and e-maxx

All Articles on Geometric Algorithms

Data Structures

- Binary Indexed Tree or Fenwick tree
- Segment Tree (RMQ, Range Sum and Lazy Propagation)
- K-D tree (See insert, minimum and delete)
- Union Find Disjoint Set (Cycle Detection and By Rank and Path Compression)
- Tries
- Suffix array (this, this and this)
- Sparse table
- Suffix automata
- Suffix automata II
- LCA and RMQ

Classic problems:

1) Rotate Array, Reverse Words in a String

2) Evaluate Reverse Polish Notation (Stack)

3) Isomorphic Strings

4) Word Ladder (BFS), Word Ladder II (BFS)

5) Median of Two Sorted Arrays

5) Kth Largest Element in an Array

6) Wildcard Matching, Regular Expression Matching

7) Merge Intervals, Insert Interval

9) Two Sum, Two Sum II, Two Sum III, 3Sum, 4Sum

10) 3Sum Closest

11) String to Integer

12) Merge Sorted Array

13) Valid Parentheses

13) Longest Valid Parentheses

14) Implement strStr()

15) Minimum Size Subarray Sum

16) Search Insert Position

17) Longest Consecutive Sequence

18) Valid Palindrome

19) ZigZag Conversion

20) Add Binary

21) Length of Last Word

22) Triangle

24) Contains Duplicate: I, II, III

25) Remove Duplicates from Sorted Array: I, II, Remove Element , Move Zeroes

27) Longest Substring Without Repeating Characters

28) Longest Substring that contains 2 unique characters [Google]

28) Substring with Concatenation of All Words

29) Minimum Window Substring

31) Find Minimum in Rotated Sorted Array: I, II

32) Search in Rotated Array:I, II

33) Min Stack

34) Majority Element: I, II

35) Bulls and Cows

36) Largest Rectangle in Histogram

37) Longest Common Prefix [Google]

38) Largest Number

39) Simplify Path

40) Compare Version Numbers

41) Gas Station

44) Pascal's Triangle: I, II

45) Container With Most Water

45) Candy [Google]

45) Trapping Rain Water

46) Count and Say

47) Search for a Range

48) Basic Calculator, Basic Calculator II

49) Group Anagrams

50) Shortest Palindrome

51) Rectangle Area

52) Summary Ranges

53) Increasing Triplet Subsequence

54) Get Target Using Number List And Arithmetic Operations

55) Reverse Vowels of a String

56) Flip Game, Flip Game II

57) Missing Number, Find the duplicate number, First Missing Positive

58) Valid Anagram, Group Shifted Strings

59) Top K Frequent Elements

60) Find Peak Element

61) Word Pattern, Word Pattern II

62) H-Index , H-Index II

63) Palindrome Pairs

64) One Edit Distance

65) Scramble String

66) First Bad Version

67) Integer to English Words

68) Text Justification

69) Remove Invalid Parentheses

70) Intersection of Two Arrays, Intersection of Two Arrays II

71) Sliding Window Maximum, Moving Average from Data Stream

72) Guess Number Higher or Lower

2. Matrix

Common methods to solve matrix related problem include DFS, BFS, dynamic programming, etc.

Classic Problems:

1) Set Matrix Zeroes

2) Spiral Matrix

2) Spiral Matrix II

3) Search a 2D Matrix

3) Search a 2D Matrix II

4) Rotate Image [Palantir]

5) Valid Sudoku

6) Minimum Path Sum (DP) [Google]

7) Unique Paths (DP) [Google]

7) Unique Paths II (DP)

8) Number of Islands (DFS/BFS), Number of Islands II (Disjoint Set), Number of Connected Components in an Undirected Graph

9) Surrounded Regions (BFS)

10) Maximal Rectangle

10) Maximal Square

11) Word Search (DFS)

11) Word Search II

13) Range Sum Query 2D – Immutable

14) Longest Increasing Path in a Matrix (DFS)

15) Shortest Distance from All Buildings

16) Game of Life

17) Paint House, Paint House II

18) Sudoku Solver (DFS)

19) Walls and Gates (DFS/BFS)

20) Tic-Tac-Toe

21) Best Meeting Point

3. Linked List

The implementation of a linked list is pretty simple in Java. Each node has a value and a link to next node.

```
class Node {
        int val;
        Node next;

        Node(int x) {
                val = x;
                next = null;
        }
}
```

Two popular applications of linked list are stack and queue.

Stack

```
class Stack{
        Node top;

        public Node peek(){
                if(top != null){
                        return top;
                }

                return null;
        }
        public Node pop(){

                if(top == null){

                        return null;

                }else{

                        Node temp = new Node(top.val);

                        top = top.next;

                        return temp;

                }

        }



        public void push(Node n){

                if(n != null){

                        n.next = top;

                        top = n;

                }

        }

}
```

Queue

```
class Queue{

        Node first, last;



        public void enqueue(Node n){

                if(first == null){
```

```
                first = n;

                last = first;

            }else{

                last.next = n;

                last = n;

            }

        }



        public Node dequeue(){

            if(first == null){

                return null;

            }else{

                Node temp = new Node(first.val);

                first = first.next;

                return temp;

            }

        }

}
```

The Java standard library contains a class called "Stack". Another class from Java SDK is LinkedList, which can be used as a Queue (add() and remove()). (LinkedList implements the Queue interface.) If a stack or queue is required to solve problems during your interview, they are ready to be used.

Classic Problems:

0) Implement a Stack Using an Array

1) Add Two Numbers

2) Reorder List

3) Linked List Cycle

4) Copy List with Random Pointer

5) Merge Two Sorted Lists

6) Odd Even Linked List

7) Remove Duplicates from Sorted List

7) Remove Duplicates from Sorted List II

8) Partition List

9) LRU Cache

10) Intersection of Two Linked Lists

11) Remove Linked List Elements

12) Swap Nodes in Pairs

13) Reverse Linked List, Reverse Linked List II, Print Linked List in Reversed Order

14) Remove Nth Node From End of List (Fast-Slow Pointers)

15) Implement Stack using Queues

15) Implement Queue using Stacks

16) Palindrome Linked List

17) Implement a Queue using an Array

18) Delete Node in a Linked List

19) Reverse Nodes in k-Group

4. Tree, Heap and Trie

A tree normally refers to a binary tree. Each node contains a left node and right node like the following:

```
class TreeNode{
        int value;

        TreeNode left;

        TreeNode right;

}
```

Here are some concepts related with trees:

- Binary Search Tree: for all nodes, left children <= current node <= right children
- Balanced vs. Unbalanced: In a balanced tree, the depth of the left and right subtrees of every node differ by 1 or less.
- Full Binary Tree: every node other than the leaves has two children.
- Perfect Binary Tree: a full binary tree in which all leaves are at the same depth or same level, and in which every parent has two children.
- Complete Binary Tree: a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

Heap is a specialized tree-based data structure that satisfies the heap property. The time complexity of its operations are important (e.g., find-min, delete-min, insert, etc). In Java, PriorityQueue is important to know.

4.1 Tree

1) Binary Tree Traversal: Preorder, Inorder, Postorder, Level Order, Level Order II, Vertical Order

2) Invert Binary Tree

3) Kth Smallest Element in a BST

4) Binary Tree Longest Consecutive Sequence

5) Validate Binary Search Tree

6) Flatten Binary Tree to Linked List

7) Path Sum (DFS or BFS)

7) Path Sum II (DFS)

8) Construct Binary Tree from Inorder and Postorder Traversal

8) Construct Binary Tree from Preorder and Inorder Traversal

9) Convert Sorted Array to Binary Search Tree [Google]

10) Convert Sorted List to Binary Search Tree [Google]

11) Minimum Depth of Binary Tree

12) Binary Tree Maximum Path Sum *

13) Balanced Binary Tree

14) Symmetric Tree

15) Binary Search Tree Iterator

16) Binary Tree Right Side View

17) Lowest Common Ancestor of a Binary Search Tree

18) Lowest Common Ancestor of a Binary Tree

19) Verify Preorder Serialization of a Binary Tree

20) Populating Next Right Pointers in Each Node

21) Populating Next Right Pointers in Each Node II

21) Unique Binary Search Trees (DP)

21) Unique Binary Search Trees II (DFS)

22) Sum Root to Leaf Numbers (DFS)

23) Count Complete Tree Nodes

24) Closest Binary Search Tree Value

25) Binary Tree Paths

26) Maximum Depth of Binary Tree

27 Recover Binary Search Tree

28) Same Tree

29) Serialize and Deserialize Binary Tree

30) Inorder Successor in BST

31) Find Leaves of Binary Tree

32) Largest BST Subtree

4.2 Heap

1) Merge k sorted arrays [Google]

2) Merge k Sorted Lists *

3) Find Median from Data Stream

4) Meeting Rooms II, Meeting Rooms

5) Range Addition

4.3 Trie

1) Implement Trie (Prefix Tree)

2) Add and Search Word - Data structure design (DFS)

4.4 Segment Tree

1) Range Sum Query - Mutable

2) The Skyline Problem

5. Graph

Graph related questions mainly focus on depth first search and breath first search. Depth first search is straightforward, you can just loop through neighbors starting from the root node.

Below is a simple implementation of a graph and breath first search. The key is using a queue to store nodes.

1) Define a GraphNode

```
class GraphNode{
	int val;

	GraphNode next;

	GraphNode[] neighbors;

	boolean visited;



	GraphNode(int x) {

		val = x;

	}



	GraphNode(int x, GraphNode[] n){

		val = x;

		neighbors = n;

	}



	public String toString(){

		return "value: "+ this.val;

	}

}
```

2) Define a Queue

```
class Queue{

	GraphNode first, last;



	public void enqueue(GraphNode n){
		if(first == null){
			first = n;
			last = first;
		}else{
			last.next = n;
			last = n;
		}
	}

	public GraphNode dequeue(){
		if(first == null){
			return null;
		}else{
			GraphNode temp = new GraphNode(first.val, first.neighbors);
			first = first.next;
			return temp;
		}
	}
}
```

3) Breath First Search uses a Queue

```
public class GraphTest {

	public static void main(String[] args) {
        GraphNode n1 = new GraphNode(1);

        GraphNode n2 = new GraphNode(2);

        GraphNode n3 = new GraphNode(3);

        GraphNode n4 = new GraphNode(4);

        GraphNode n5 = new GraphNode(5);



        n1.neighbors = new GraphNode[]{n2,n3,n5};

        n2.neighbors = new GraphNode[]{n1,n4};

        n3.neighbors = new GraphNode[]{n1,n4,n5};

        n4.neighbors = new GraphNode[]{n2,n3,n5};

        n5.neighbors = new GraphNode[]{n1,n3,n4};



        breathFirstSearch(n1, 5);

}



public static void breathFirstSearch(GraphNode root, int x){

        if(root.val == x)

                System.out.println("find in root");



        Queue queue = new Queue();

        root.visited = true;

        queue.enqueue(root);
        while(queue.first != null){
            GraphNode c = (GraphNode) queue.dequeue();
            for(GraphNode n: c.neighbors){

                if(!n.visited){
                    System.out.print(n + " ");
                    n.visited = true;
                    if(n.val == x)
                        System.out.println("Find "+n);
                    queue.enqueue(n);
                }
            }
        }
    }
}
```

Output:

```
value: 2 value: 3 value: 5 Find value: 5
value: 4
```

Classic Problems:

1) Clone Graph

2) Course Schedule , Course Schedule II , Minimum Height Trees

3) Reconstruct Itinerary

4) Graph Valid Tree

6. Sorting

Time complexity of different sorting algorithms. You can go to wiki to see basic idea of them.

Algorithm	Average Time	Worst Time	Space
Bubble sort	n^2	n^2	1
Selection sort	n^2	n^2	1
Insertion sort	n^2	n^2	
Quick sort	n log(n)	n^2	
Merge sort	n log(n)	n log(n)	depends

* BinSort, Radix Sort and CountSort use different set of assumptions than the rest, and so they are not "general" sorting methods. (Thanks to Fidel for pointing this out)

Here are some implementations/demos, and in addition, you may want to check out how Java developers sort in practice.

1) Mergesort

2) Quicksort

3) InsertionSort.

4) Maximum Gap (Bucket Sort)

5) Sort Colors (Counting Sort)

7. Dynamic Programming

Dynamic programming is a technique for solving problems with the following properties:

- An instance is solved using the solutions for smaller instances.
- The solution for a smaller instance might be needed multiple times.
- The solutions to smaller instances are stored in a table, so that each smaller instance is solved only once.
- Additional space is used to save time.

The problem of climbing steps perfectly fit those 4 properties. Therefore, it can be solve by using dynamic programming.

```
public static int[] A = new int[100];


public static int f3(int n) {

        if (n <= 2)

                A[n]= n;


        if(A[n] > 0)

                return A[n];

        else

                A[n] = f3(n-1) + f3(n-2);//store results so only calculate once!

        return A[n];

}
```

Classic problems:

1) Edit Distance

1) Distinct Subsequences Total

2) Longest Palindromic Substring

3) Word Break

3) Word Break II

4) Maximum Subarray

4) Maximum Product Subarray

5) Palindrome Partitioning

5) Palindrome Partitioning II

6) House Robber [Google]

6) House Robber II

6) House Robber III

7) Jump Game

7) Jump Game II

8) Best Time to Buy and Sell Stock

8) Best Time to Buy and Sell Stock II

8) Best Time to Buy and Sell Stock III

8) Best Time to Buy and Sell Stock IV

9) Dungeon Game

10) Minimum Path Sum

11) Unique Paths

12) Decode Ways

13) Longest Common Subsequence

14) Longest Common Substring

15) Longest Increasing Subsequence

16) Coin Change

17) Perfect Squares

8. Bit Manipulation

Bit operators:

OR (|)	AND (&)	XOR (^)	Left Shift (<<)	Right Shift (>>)	Not (~)
1|0=1	1&0=0	1^0=1	0010<<2=1000	1100>>2=0011	~1=0

Get bit i for a give number n. (i count from 0 and starts from right)

```
public static boolean getBit(int num, int i){
        int result = num & (1<<i);


        if(result == 0){
                return false;
        }else{
                return true;
        }
}
```

For example, get second bit of number 10.

i=1, n=10

1<<1= 10 1010&10=10 10 is not 0, so return true;

Classic Problems:

1) Single Number

1) Single Number II

2) Maximum Binary Gap

3) Number of 1 Bits

4) Reverse Bits

5) Repeated DNA Sequences

6) Bitwise AND of Numbers Range

7) Sum of Two Integers

8) Counting Bits

9) Maximum Product of Word Lengths

10) Gray Code

9. Combinations and Permutations

The difference between combination and permutation is whether order matters.

Example 1:

Given 5 numbers - 1, 2, 3, 4 and 5, print out different sequence of the 5 numbers. 4 can not be the third one, 3 and 5 can not be adjacent. How many different combinations?

Example 2:

Given 5 banaba, 4 pear, and 3 apple, assuming one kind of fruit are the same, how many different combinations?

Class Problems:

1) Permutations

2) Permutations II

3) Permutation Sequence

4) Generate Parentheses

5) Combination Sum (DFS), II (DFS), III (DFS), IV (DP)

6) Combinations (DFS)

7) Letter Combinations of a Phone Number (DFS)

8) Restore IP Addresses

9) Factor Combinations (DFS)

10. Math

Solving math problems usually require us to find regularities or repeated pattern from the observations. List the results for a small set of numbers first, if you do not have any ideas.

1) Reverse Integer

2) Palindrome Number

3) Pow(x,n), Power of Two, Power of Three, Power of Four

4) Subsets

5) Subsets II

6) Fraction to Recurring Decimal [Google]

7) Excel Sheet Column Number

8) Excel Sheet Column Title

9) Factorial Trailing Zeroes

10) Happy Number

11) Count Primes

12) Plus One

13) Divide Two Integers

14) Multiply Strings

15) Max Points on a Line

16) Product of Array Except Self

17) Integer Break

18) Add Digits

21) Ugly Number, 9Ugly Number II, Super Ugly Number, Find K Pairs with Smallest Sums

UPDATE: I decided to add more categories below.

11. HashMap

1) Shortest Word Distance II

Additional Problems:

1) Self Crossing

2) Patching Array

3) Nim Game

4) Bulb Switcher

5) Pain Fence

6) Nested List Weight Sum

Additional Resources

1. Share your code to Github/BitBucket

Related Posts:

1. LeetCode – Word Ladder II (Java)
2. How to answer coding questions for your interview?
3. LeetCode – LRU Cache (Java)
4. LeetCode – Count of Smaller Numbers After Self (Java)

Category >> Algorithms >> Interview

If you want someone to read your code, please put the code inside <pre><code> and </code></pre> tags. For example:

```
<pre><code>

String foo = "bar";

</code></pre>
```

Dynamic Programming

Dynamic programming is the strategy of optimizing recessive functions by eliminating the need for recursive calling. Whenever we see a recursive function in which a certain part of the code is called multiple times it can be greatly improved with the use of dynamic programming. The recursiveness is removed by storing the results of the previous sub-function so that they do not have to be called back multiple times. This reduces the time complexity from exponential to polynomial time. Example of algorithms which fall under the dynamic programming category are:

- Ugly Numbers
- Fibonacci Numbers
- Binomial Coefficient
- Permutation Coefficient

Binary Search

As the name suggests searching algorithms are used to search for an element from a given set known as the data structure. Binary search works when provided with a sorted array of elements and a search key. Binary search works by selecting the middle element and comparing it with the search key if the key is smaller than the left part of the middle element is traverse in the same way. If now than the right portion on searched for the key. The time complexity of a binary search is O(log n), where n is the number of elements in the array.

Sorting Algorithms

Sorting algorithms are used for sorting an array, the input includes a data type which needs sorting. The data set can be sorted in either ascending or descending order. The following are a few important sorting algorithms to remember.

Merge Sort

Merge sort works on the principles of divide and conquer algorithm. It refers to the practice of dividing a problem into smaller parts and solving them one by one and merging them together in the end. Merge sort divides the array in half and calls the sort function on the two halves, those two halves are sorted and then merged together using the merge function. The time complexity of merge sort is O(n log n).

Quick Sort

Like merge sort, quick sort is also based on divide and conquer algorithm, it varies from merge sort in terms of its sorting functionality. Quicksort works by selecting the last element as the pivot number and places it in the middle with smaller numbers on the left while the larger ones on the right. The left and right sides are again called with the sort function which in result sortes the whole array. The time complexity of quicksort is O(n^2).

Depth First Search

DFS is a searching algorithm that starts the search process from the node and goes all the way down to the last leaf of the leftmost branch. After it has reached the leftmost leaf the algorithm starts backtracking and traverses the right side of the tree and so on. The issue with this DFS is that if a cycle exists, a certain node can be visited more than once. The time complexity of DFS is O(V + E), where V and E represent the number of vertices and edges respectively in the graph.

Breadth-First Search

BFS is a searching algorithm that starts from the root just like DFS. But instead of traversing all the leaves on the left, it searches in the neighbourhood of the node on the same level. After a level is traversed the algorithm moves forward to the next level and keeps on traversing until the element is found. The time complexity of BFS is the same as DFS, which is O(V + E).

Custom Data structure

Sometimes the typical, pre-defined data structures do not get the job done and you need something better and more powerful. Custom data structures can be real or abstract objects depending on the uses of their data members. Data members can be considered as variables belonging to objects which are specified.

Hash Tables

Hash tables are a type of data structure that is used to store, access, and modify data with the time complexity of O(1). Hash data structures use the Hash function to map a given value to a specific key. This key is then used to access and retrieve those values quickly, the efficiency of how the Hash would perform depends upon the type of Hash function used.

Linked Lists

Usually, the components of an array or any linked data structures are stored in contiguous memory locations. This takes up spaces and certain chunks of memory are not accessible (that is if you have low memory). To overcome this, linked list data structures are used in which the data is not stored contiguously, instead every item in the list has a pointer that points to the next element's memory location. The first element is known as the head and the last is known as the tail.

Asking Questions

The most important thing that a software engineer should know is what to ask its client. Most clients are not able to get their point across and if the developer does not ask any questions then it could cause a problem due to miscommunication. This way you will be able to understand the core problem of what they are trying to achieve and not just the difficulties that they are facing.

Top 10 Algorithms to Crack Coding Interviews

If you’re from a computer science background, you might already know what algorithms are. But if you’re just starting out in the field of programming, you might want to pay close attention. Algorithms form the basis of all programming languages and give a clear definition of your program.

It may sound really intimidating, but an algorithm is just a set of statements that have a purpose and define what your program will do and how it will do that. That’s pretty much it. However, they do form the building blocks of a lot of programs making them extremely essential to learn.

In this article, let’s look at some of the most popular algorithms that you should have the basic knowledge of when going in for an interview. We have used Python to explain the workings, but the logic of each technique can be replicated in any other language.

1. Dynamic Programming

Dynamic programming is a type of algorithm that works on the principle of recursion where each problem is broken into smaller sub-problems and the solution of the final problem is dependent on the solutions of the smaller ones. This works by storing the solutions of each sub-problem and then using these states later to simplify complexities and reduce computation time.

To understand how it works, let’s consider the example of a Fibonacci sequence. A Fibonacci sequence is a series of numbers that starts with 0 & 1, and every subsequent number is the sum of the preceding two. So it would go something like 0,1,1,2,3,5,8,13,... and so on.

If we had to find the Nth number in the sequence, we could write the code using simple recursion as follows:

```
def fib(n):

        if n <= 0:

                raise Exception('Number must be greater than or equal to 1')

        elif n == 1:

                return 0

        elif n == 2:
```

```
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

As can be seen from the example above, if we had to find the 10th term in the sequence, the entire code would be executed multiple times over and over again and is thus inefficient. Let’s optimize this with dynamic programming. The code for the same is shown below.

```
fib_array=[0, 1]
def fib(n):
    if n <= 0:
        raise Exception('Number must be greater than or equal to 0')
    elif n <= len(fib_array):
        return fib_array[n - 1]
    else:
        temp = fib(n - 1) + fib(n - 2)
        fib_array.append(temp)
        return temp
```

Every time this piece of code is run, the array is updated with any new terms in the Fibonacci sequence, and the “else” part is run only for those specific cases when the term is not existing in the array. This greatly reduces the required computation power in terms of CPU cycles and is thus used to solve complex problems efficiently.

2. Tree Traversal Algorithms

Trees are a special form of data structure that include a root node connected to sub-trees in a linked node format. The most commonly used type is called the Binary Tree where each node can have a maximum of two children only. Being hierarchical in nature, a binary tree is extremely efficient to traverse and has the benefits of both — ordered arrays and linked lists.

Tree traversal is essentially the process of visiting each node of the tree while also performing some functions on all the values. Being hierarchical in nature where the root node is connected to its children via edges. There are many types of tree traversals possible, but these are the three most common ones to start with. These are:

- Pre-order traversal
- Post-order traversal
- In-order traversal

Here's a comparison of these three traversal techniques. For reference, consider a sample tree shown below.

The code for any of the tree traversal techniques involves the creation of a class to store the locations of the root node, as well as the left and right nodes. We then create an empty list to add the values of the root node, the left node, and the right node in the desired order recursively to obtain the final traversal of the binary tree. Here's the code for all three traversal techniques.

```
class Node:

        def __init__(self, data):

                self.left = None

                self.right = None

                self.data = data

n1 = Node(1)

n2 = Node(2)

n3 = Node(3)

n4 = Node(4)

n5 = Node(5)

n1.left = n2

n1.right = n3

n2.left = n4
```

```
n2.right = n5

#Pre-Order Traversal

#Root - Left Node - Right Node

pre_order_traversal_res = []

def pre_order_traversal(root):

        if root:

                pre_order_traversal_res.append(root.data)

                pre_order_traversal(root.left)

                pre_order_traversal(root.right)

pre_order_traversal(n1)

print(pre_order_traversal_res)

#Post-Order Traversal

#Left Node - Right Node - Root

post_order_traversal_res = []

def post_order_traversal(root):

        if root:

                post_order_traversal(root.left)

                post_order_traversal(root.right)

                post_order_traversal_res.append(root.data)

post_order_traversal(n1)

print(post_order_traversal_res)

#In-Order Traversal

#Left Node - Root - Right Node
```

```
in_order_traversal_res = []

def in_order_traversal(root):

        if root:

                in_order_traversal(root.left)

                in_order_traversal_res.append(root.data)

                in_order_traversal(root.right)

in_order_traversal(n1)

print(in_order_traversal_res)
```

3. Graph Traversal

Similar to trees, graphs are also a type of non-linear data structure that consists of nodes connected to each other through edges. But unlike trees that have a root node, graphs have no unique node called the root and can even be in the form of a cycle. Due to this, graphs are generally used to denote networks and solve complex real-life problems such as finding the shortest path in a large network.

To read and understand the data in a graph, we use a technique called the graph traversal. There are two ways a graph can be traversed — breadth-first or depth-first.

These algorithms are similar to that of a tree traversal, but the only catch here is the presence of cycles causing nodes to be counted multiple times.

- Depth-First Search: The depth-first algorithm is similar to the one used in tree traversal, but with the added visited[ ] array that keeps a track of all the visited nodes to avoid re-counting. It starts by selecting a random arbitrary node in the graph and traversing the entire extent of each branch before backtracking. In the example above, the Depth-first Search output would be: A-B-D-E-C or A-B-C-D-E.
- Breadth-First Search: In the breadth-first traversal algorithm, after each node is visited, it's children are put into a First-In-First-Out queue. It starts with a random arbitrary node, and then reads all the nodes connected to it at the same depth before moving on to the next depth level. In the example above, the Breadth-first Search output would be: A-B-C-D-E or A-B-D-C-E.

Search Algorithms

An essential part of using any data set, searching algorithms are used to search for an element stored in any form of data structure. These algorithms form the foundation of any search system and find use cases in essentially four areas — databases, virtual spaces, sub-structures, or quantum computers.

Search algorithms, based on how they function, can be divided into two classes — linear and interval. Let’s understand how each works and their code. Consider the list of numbers shown below as an example.

4. Linear Search

Linear search works by starting from the 0th element and comparing the user’s input element with each term and finally returning the position of the element. This generally works well for unstructured data as shown in List 2.

Let’s say the user wants to search for the number 32 in List 2. The program would compare the input to the 0th element, check if it’s the same, proceed to the second one, check again, and so on till a term in the list equals 32. It will then return the position of that element in the list. Here’s the code for a linear search algorithm.

```
#Linear Search Function

def linear_search(array, x):

    for i in range(len(array)):

        if array[i] == x:

            return i

    return -1


#Calling the function

arr = [7,3,4,26,17,10,32,45,38,63]

num = 10
```

```
print(linear_search(arr,num))
```

5. Binary Search

While linear search compares the input element with every element in the list starting from the first element, the binary search algorithm starts with the middle element. As it is a type of interval search algorithm, it is used only on sorted data structures as shown in List 1.

To understand, let’s consider that the user wants to find the position of the number 49 in List 2. To determine that, the binary search algorithm would start by comparing 49 with the middle term and then divide the entire list into two sub-sets一 one side with terms smaller than the middle term and another side that contains terms larger than the middle term.

As 49 is larger than the middle term, the algorithm then starts comparing it with the numbers in the right-hand-side subset following the same process until it finds the location of the input term. The Binary search algorithm works with a sorted list and decreases the time complexity of the problem making it a highly efficient one.

```
#Binary Search Function

def binary_search(arr, l, r, num):

    if r >= l:

        mid = l + (r - l) // 2

        if arr[mid] == num:

            return mid

        elif arr[mid] > num:

            return binary_search(arr, l, mid-1, num)

        else:

            return binary_search(arr, mid+1, r, num)

    else:

        return -1
```

```
#Calling the function

arr = [1, 3, 4, 6, 7, 10, 22, 45, 49, 63, 78]

num = 10

print(binary_search(arr, 0, len(arr)-1, num))
```

Sorting Algorithms

They take an array or list of values as input and based on user needs, they can be customized to sort data into an ascending or descending order. This order can either be set in the algorithm itself or taken as an input from the user.

Sorting algorithms are generally differentiated based on the method they use to sort the data and each technique has its own set of codes and uses. Let’s understand three popular sorting algorithms — bubble sort, insertion sort, and selection sort.

6. Bubble Sort

This is the easiest sorting algorithm and works by comparing adjacent values in the list and places them in the correct order. It then continues to do that throughout the list over and over again until all the values in the list have been arranged in the preferred order.

Let’s take the input set of values as [7,2,5,4,3,1], and let’s say we want to arrange this in ascending order. In the first round, it starts from the 0th position and compares the first 2 elements, arranges them if not in order, and then moves to the next 2 elements and so on. Let’s see how the code looks like for this.

```
def bubble_sort(arr):               #define the Bubble Sort function

        for i in range(len(arr)):   #length of the array = number of input values

                for j in range(len(arr)-1):

                        if arr[j] > arr[j+1]:

                                arr[j], arr[j+1] = arr[j+1], arr[j]     #swapping

        return arr
```

Now you can call the function by just defining the array and then calling the function as follows:

```
arr = [7, 2, 5, 4, 3, 1] #define the array

bubble_sort(arr) #sort the array

print(arr)
```

The output of this program would be a sorted array and shows as below:

```
[1, 2, 3, 4, 5, 7]
```

7. Insertion Sort

The second sorting algorithm that we'll look at is the Insertion Sort. Unlike bubble sort, where the program starts at the first element, sorts the adjacent elements, reaches the last element, and then starts at the first element again to repeatedly go through this process and sort the complete array, insertion sort starts at the first element and as it moves towards the last element, sorts all the elements that it passes.

To understand this, let's take the same example array as before: A[6]= [7,2,5,4,3,1]

Continues till the end of the array and the entire array is sorted

The code for this algorithm also uses a nested loop but there is a small difference in the way it's used here compared to the way it's used in Bubble Sort. Here's the code for sorting the example array above.

```
def insertion_sort(arr):  # define the Insertion Sort function

        for i in range(len(arr)):

                j = i  # start at the last element of the sub-array

                while j != 0 and arr[j] < arr[j - 1]:

                        arr[j - 1], arr[j] = arr[j], arr[j - 1]

                        j = j - 1  # go down from i to 0 in the sub-array while sorting it

                print(arr)  # we place it in the loop to display each step
```

Now we call the function to sort our example array as follows:

```
arr = [7, 2, 5, 4, 3, 1] #define the array
```

```
insertion_sort(arr) #sort the array
```

The output of this program would be each step of the sorting process like this:

```
[7, 2, 5, 4, 3, 1]

[2, 7, 5, 4, 3, 1]

[2, 5, 7, 4, 3, 1]

[2, 4, 5, 7, 3, 1]

[2, 3, 4, 5, 7, 1]

[1, 2, 3, 4, 5, 7]
```

8. Selection Sort

Another simple sorting algorithm, Selection Sort works by finding the smallest element (in case of ascending order) or the largest element (in case of descending order) and placing them in the first position. It then omits the first element and repeats the process with the leftover array and places the next smallest or largest element in the second position. This process continues until all elements in the array have been sorted into their proper positions.

The selection sort program works with a nested for loop to:

- Divide the array into 2 sub-arrays
- Find the smallest value in the sub-array and swap it with the first position element
- Re-create the sub-array excluding this minimum value
- Set the first element as the minimum
- Compare this minimum with other values to find the actual minimum and then swap positions
- Go back to step 3 and repeat until the entire array has been sorted in ascending order.

Here’s the code for you to understand it better and try for yourself.

```
def selection_sort(arr):          #define the function

        for i in range(len(arr)):

                min=i                         #setting the first element as the minimum
        for j in range(i+1, len(arr)):
            if arr[j] < arr[min]:          #compare to find smallest
                min = j                    #set the smallest as the min
        arr[i], arr[min] = arr[min], arr[i] #swap the first element and the min
        print(arr)  # display the array at each step
```

To test this out, we define our example array and call the SelectionSort function as below:

```
arr = [7, 2, 5, 4, 3, 1] #define the array
print(arr)
selection_sort(arr) #sort the array
```

The output consists of the array at each step of the sorting process and looks like this:

```
[7, 2, 5, 4, 3, 1]
[1, 2, 5, 4, 3, 7]
[1, 2, 5, 4, 3, 7]
[1, 2, 3, 4, 5, 7]
[1, 2, 3, 4, 5, 7]
[1, 2, 3, 4, 5, 7]
```

9. Hashing Algorithms

To understand what hashing algorithms are and how they work, we first need to understand what hashing is and what hash functions are. Hash functions are, essentially, mathematical functions that convert complex input numbers into compressed numerical values. With any arbitrary length of input values, the output is always of a fixed length and the final output value is called the hash value.

It is also important to understand the difference between a hash function and a hash algorithm. A hash function creates a hash value based on the input data blocks that have fixed-length data. On the other hand, a hashing algorithm describes how the hash function will be used and defines the complete process of breaking up the message and bringing it back together.

There are a lot of different hashing algorithms. A few commonly used hashing algorithms are as follows.

- Message Digest (MD5)
- Secure Hash Algorithm
- RACE Integrity Primitives Evaluation Message Digest (RIPEMD)
- Whirlpool
- RSA (Rivest–Shamir–Adleman)

It's important that you practice these Algorithms before your next tech interview. They may seem easy and obvious, but sometimes they become tricky to solve in an actual interview. Also, these algorithms are used to test the understanding of a software engineer on whether or not he knows the working of the code. Practising these problems before an interview will not only make you familiar with them but also give you confidence in explaining the solution to the interviewer.