Megha 230

Institute for Advanced Computing And Software Development (IACSD)
Akurdi, Pune

Algorithms and Data Structures

Dr. D.Y. Patil Educational Complex, Sector 29, Behind Akurdi Railway Station,

Nigdi Pradhikaran, Akurdi, Pune - 411044.

Data structures Interview Questions

Data structures:

- way of organizing the data in an efficient manner.
- It is a template using which we can ~~using~~ mention the different operations that must be defined, which are applied on a data.
- Every data structure is an ADT --- how to store the data, the list of operations is given, which rules to be applied on the data or the operations.

ADT : It is a mathematical model which is defined using a triplet D, F and A.

D : data objects
F : functions
A : axioms --- rules

Every data structures tells --- what to do, not how to do.

- It also contains different algorithms -- way of solving the problem.

Data structure = data object + algorithm + program

For e.g. **–Integer stack :**
D : integer domain
F : push, pop, peak, isEmpty --- basic (isfull -- used for static stack)
A : only one end is opened, LIFO manner, only one element can be inserted or removed from stack, only topmost element is visible

----------Set I----------

1) 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.

2) 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.

3) 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.

4) 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.

5) 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.

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

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

7) Define the queue data structure & List some applications of 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.

The Applications of the queue is given as follows:

- Queues are widely used as waiting lists for a single shared resource like a printer,disk, CPU.
- 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.
- Queues are used as buffers in most of the applications like MP3 media player, CDplayer, etc.
- Queues are used to maintain the playlist in media players to add and remove the songs from the play-list.
- Queues are used in operating systems for handling interrupts.

8) What is the difference between NULL and VOID?

- Null is actually a value, whereas Void is a data type identifier.
- A null variable simply indicates an empty value, whereas void is used to identify pointers as having no initial size.

9) What are the advantages of Selecetion Sort?

- It is simple and easy to implement.
- It can be used for small data sets.
- It ist more efficient than bubble sort.

10) What are the applications of Graph data structure?

The graph has the following applications:

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

- Graphs are used in transport networks where stations are drawn as vertices and routes become the edges of the graph.
- Graphs are used in maps that draw cities/states/regions as vertices and adjacency relations as edges.
- 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.

11) 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. It has one end open called top end from where all insertions and deletions are made.

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

12) 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. It has two ends open, called as front and rear, all insertions are made from one end called rear and all deletions are made from front end.

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

13) What are different ways of representing the graph?

We can represent a graph in 2 ways:

1. Adjacency matrix: Used for sequential data representation
2. Adjacency list: Used to represent linked data

14) What is meant by indegree and outdegree of vertex?

In case of directed graph, the number of incoming edges towards the vertex is called indegree of that vertex and number of outgoing edges from the vertex is called outdegree of that vertex.

15) 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.

16) 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.

17) 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).

18) 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:

1. 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.

2. 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.

19) 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
- The undo and redo functionality on platforms such as word, paint etc, where you can reverse the node to get to the previous page.

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

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

21) What is a deque?

Deque (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.

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

In 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.

23) 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.

24) What is Stack Along With Its Applications.

A stack is a data structure in which elements are placed one on top of the other and additions and deletions can occur only at the top. Stacks can be used in the following applications:

- A stack can be used in applications where users are enabled to backtrack on previous operations by one step or move forward with a new operation.
- Converting infix expressions into postfix expressions.
- The process of reversing the characters in a string can be completed by placing them in a stack and using the pop operation.

25) 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.

26) How graph represented in memory?

A graph can be represented using 3 data structures- adjacency matrix, adjacency list and adjacency set. An adjacency matrix can be thought of as a table with rows and columns. The row labels and column labels represent the nodes of a graph.

27)What are Categories of Data Structure?

The data structure can be sub divided into major types:

- Linear Data Structure
- Non-linear Data Structure

28) Explain Linear Data Structure.

A data structure is said to be linear if its elements combine to form any specific order. There are basically two

techniques of representing such linear structure within memory. First way is to provide the linear relationships among all the elements represented by means of linear memory location. These linear structures are termed as arrays.

The second technique is to provide the linear relationship among all the elements represented by using the concept of pointers or links. These linear structures are termed as linked lists. The common examples of linear data structure are:

□ Arrays □ Queues □ Stacks □ Linked lists

29)Explain Non linear Data Structure.

This structure is mostly used for representing data that contains a hierarchical relationship among various elements. Examples of Non Linear Data Structures are listed below:

□ Graphs □ family of trees and □ table of contents

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

30) What are Basic Operations supported by an array?

- Traverse – print all the array elements one by one
- Insertion – Adds an element at the given index.
- Deletion – Deletes an element at the given index.
- Search – Searches an element using the given index or by the value.
- Update – Updates an element at the given index.

31) what are Applications of Stack?

Direct applications • Balancing of symbols

- Infix-to-postfix conversion • Evaluation of postfix expression
- Implementing function calls (including recursion) • Finding of spans (finding spans in stock markets)
- Page-visited history in a Web browser [Back Buttons] • Undo sequence in a text editor
- Matching Tags in HTML and XML implemented Indirect applications
- Auxiliary data structure for other algorithms (Example: Tree traversal s) • Component of other data structures

Example: Simulating queues, Queues)

32) Explain Stack with example

--- linear data structure.
--- Having one end open called top end.
--- Only one element can be inserted or deleted at a time.
--- Insertions and deletions are made from top end only.
--- Works in LIFO (FILO) manner.
--- Visualized vertically.
--- has sequential access.
--- dynamic in nature --- no upper limit --- we can add as many elements as we want.

ADT :
For e.g. stack of integers
D : all integer values
F :
init() --- create()
push() --- insert
pop() ---- delete
peak() --- displaying the topmost element
isEmpty()
isFull() --- applicable to a static stack.

A : LIFO manner working
Insertion and deletion must be top end.

Static stack : implementing stack using array

1. init() : we wil set maximum size for a stack
   top = -1.
2. push() :
   a) check whether stack is full or not.
   b) If not, then increment top and place the new value at top index.
3. pop() :
   a) check whether stack is empty or not.
   b) If not, fetch / retrieve the top index value, and then decrement top.
4. isEmpty() :
   a) if top == -1 then stack is empty.
5. isFull() :
   a) if top reaches to size of array-1 then stack is full.

33) How to implement Stack dynamically?

Dynamic stack : Implementing a stack using SLL.

1. init() : top reference (pointer) assigned to null.
2. push() : mapped to the SLL operation - insert at front (insert at first position / insert at beginning).

3. pop() : mapped to the SLL operation - delete from front
4. isEmpty() : if top == null then stack is empty.

34)Applications of stack :

1. Recursion
2. Reverse a string
3. undo / redo
4. paranthesis matching
5. Expression conversions

Three ways to represent the expressions :

a) Infix : operators between operands
For e.g. a+b*c/d

b) postfix -- reverse polish notation
-- operators after operands

c) prefix -- polish notation
-- operator before operands

6. Expression evaluation
7. Backtracking

35) Explain Queue

--- It is a linear data structure.
--- Two ends of a queue are open -- front and rear.
--- Insertions are made at rear end and deletions are made from front.
--- Only one element can be inserted or deleted from a queue.
--- Works in a FIFO --- First in first out.
--- It is dynamic in nature.

36) What are Applications of queue?

1. OS uses it CPU scheduling purpose.
2. Resource allocation / sharing.
3. Synchronous data transfer.
4. BFT traversal of tree.

37) Types of Queue:

1. Linear Queue 2. Circular Queue 3. Priority Queue 4. Dequeue (Double Ended Queue)

Operations performed on Queue **ENQUEUE** operation 1. Check if the queue is full or not. 2. If the queue is full, then print overflow error and exit the program. 3. If the queue is not full, then increment the rear and add the element. **DEQUEUE** operation 1. Check if the queue is empty or not. 2. If the queue is empty, then print underflow error and exit the program. 3. If the queue is not empty, then print the element at the front and increment the front.

Applications of Queue –

1. When a resource is shared among multiple consumers.
2. Examples include CPU scheduling, Disk Scheduling.
3. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

38)Explain in detail types of queue :

1. Linear queue :

front and rear are always increased.

a) Static linear queue : implemented using array.

i)Initialize :

1. Create an array of some size.
2. Initialize front = rear = -1; // front and rear are indices --- int

ii) enqueue :

1. Check whether queue is full or not.
2. If not, increment rear.
3. Store element at rear index.

iii) dequeue :

1. Check whether queue is empty or not.
2. If not, increment front.
3. Retrieve element from front index.

return(arr[++front]);

iv) IsEmpty() : front == rear.
v) IsFull() : rear== size of array - 1.

b)Dynamic linear queue : implemented using SLL.

1. Initialize : front = rear = null.
2. enqueue : mapped to insert at end / append in SLL.
3. dequeue : mapped to delete first position node in SLL.
4. isEmpty() : front == null

---

2. circular queue :

In case of static linear queue, we cannot utilize the vacant spaces present at the beginning of the queue, as front and rear are not reset and they are always increased.
To overcome this drawback circular queue is used --- front and rear are reset.

A).Static circular queue : implemented using array

Different ways :

1. count the number of elements in a queue.
2. To give a circular nature to a queue without using counter, we reset the front and rear by keeping one space vacant in the array.

Second way :

1. Initialize : front = rear = array_size -1. / front = rear = -1

2. enqueue :

a) Check whether queue is full or not.
b) if not, increment rear as rear =(rear+1) % array_size
c) Store element at rear index.

3. dequeue :

a) Check whether queue is empty or not.
b) if not, increment front as front = (front+1)% array_size.
c) Retrieve the element from front index.

4.isEmpty() : front == rear
5.isFull() : (rear+1)% array_size == front
fornt index is the index of vacant space.

3.Priority queue : insertions are made in sequence, but while deleting element, element with higher priority

will be deleted first.After insertion or at the time insertion, sort the elements according to priority.

4.Deque -- double ended queue

Deque is a data structure that inherits the properties of both queues and stacks. Deque or Double Ended Queue is a type of queue in which insertion and removal of elements can either be performed from the front or the rear.

39) Difference between Stack and Queue:

Sr.No	Stack	Queue
1	A Stack Data Structure works on Last In First Out (LIFO) principle.	A Queue Data Structure works on First In First Out (FIFO) principle.
2.	Push and pop operations are done from same end i.e "top"	Push and pop operations are done from same end i.e "rear" and "front" respectively
3.	You can implement multi-stack approach	Threre are four types of Queue i.e linear, circular, priority and dequeue.
4.	Application: • Used in infix to postfix conversion, • scheduling algorithms • depth first search and evaluation of an expression	Application: • Printer mainatains queue of documents to be printed • OS uses queues for many functionalities : Ready Queue, Waiting Queue, Message Queue • To implements algo like Breadth first search.

40)Explain Linked List

Linked list : it is a collection of nodes having information (as a data) and the addresses or references of current node's successor and / or predecessor node.

It is a linear data structure --- order is important.

Nodes are stored in a random location.

Access is sequential.

Dynamic in nature --- we can add as many nodes as we want.

41)What are types of LL?

There are 5 types of LL :

1. SLL
2. DLL
3. CSLL
4. CDLL
5. Generized linked list

1. SLL : Every node contains information plus the address of next or successor node.

We need to maintain the address of first node

Only forward direction traversal is possible.

Last node next reference must be null.

2. DLL : Data + reference of next + reference of previous

Forward as well as backward traversal possible.

We can maintain the DLL by storing the reference of first node and/or tail node.

Last node next reference and first node prev reference must be null.

3. CSLL : Every node contains information plus the address of next or successor node.

We need to maintain the address of first node and / or last node (if we retain the address of last node then that can be efficient).

Only forward direction traversal is possible but we can visit all nodes from any node.

Last node next reference must be store reference of first node.

4. CDLL : Data + reference of next + reference of previous

Forward as well as backward traversal possible.

We can maintain the DLL by storing the reference of first node and/or tail node (efficient way maintain either head or tail).

Last node next reference stores the reference of first node and first node prev reference stores the reference of last node.

42) Explain Tree Data structure.

A tree consists of nodes connected by edges, which do not form cycle. For collection of nodes & edges to define as tree, there must be one & only one path from the root to any other node. A tree is a connected graph of N vertices with N-1 Edges.

43) Tree Terminologies:

1. Node: A node stands for the item of information plus the branches of other items. 2. Siblings: Children of the same parent are siblings. 3. Degree: The number of sub trees of a node is called degree. The degree of a tree is the maximum degree of the nodes in the tree. 4. Leaf Nodes: Nodes that have the degree as zero are called leaf nodes or terminal nodes. Other nodes are called non terminal nodes. 5. Ancestor: The ancestor of a node are all the nodes along the path from the root to that node. 6. Level: The level of a node is defined by first letting the root of the tree to be level = 1or level=0

7. Height/Depth: The height or depth of the tree is defined as the maximum level of any node in the tree.

44) Explain Anatomy of binary search tree

-subtree

-left link

-root

-Null link

-right child of roots

45) Explain tree traversal

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are

connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree.

There are three ways which we use to traverse a tree,

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

46) Explain Preorder traversal algorithm with e.g.

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Algorithm

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

47) Explain In order traversal algorithm with e.g.

Inorder: In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself. If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.

Algorithm

1. Traverse the left subtree, i.e., call Inorder (left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder (right-subtree)

48) Explain Postorder traversal alogorithm with e.g.

Postorder: In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.

Algorithm

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root

49)What are threaded binary tree.Explain with e.g.

Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its inorder successor.

J. Perlis and C. Thornton have proposed new binary tree called "Threaded Binary Tree", which makes use of NULL pointers to improve its traversal process. In a threaded binary tree, NULL pointers are replaced by references of other nodes in the tree. These extra references are called as threads.

50)Applications of graph?

- Modeling connectivity in computer and communications networks.
- Representing an abstract map as a set of locations with distances between locations. This can be used to compute shortest routes between locations such as in a GPS routefinder.
- Modeling flow capacities in transportation networks to find which links create the bottlenecks.
- Finding a path from a starting condition to a goal condition. This is a common way to model problems in artificial intelligence applications and computerized game players.

- Modeling computer algorithms, to show transitions from one program state to another.
- Finding an acceptable order for finishing subtasks in a complex activity, such as constructing large buildings.
- Modeling relationships such as family trees, business or military organizations, and scientific taxonomies.

51)Define graph.

Definition A Graph is a collection of nodes, which are called vertices 'V', connected in pairs by line segments, called Edges E. Sets of vertices are represented as V(G) and sets of edges are represented as E(G). So a graph is represented as G = (V, E).

52)What are Types of Graph?

- Undirected Graph - Directed Graph Directed Graph are usually referred as Digraph for Simplicity.

-A directed Graph is one, where each edge is represented by a specific direction or by a directed pair .Hence & represents two different edges.

53)Explain Terminology related to Graph.

- Out – degree: The number of Arc exiting from the node is called the out degree of the node.
- In- Degree: The number of Arcs entering the node is the In degree of the node.
- Sink node: A node whose out-degree is zero is called Sink node.
- Path: path is sequence of edges directly or indirectly connected between two nodes.
- Cycle: A directed path of length at least L which originates and terminates at the same node in the graph is a cycle

Terms used with graph :

1. Degree of vertex :

-- In case of undirected graph, degree of vertex is the total number of edges that vertex is connected to.

2. Indegree of vertex :

-- In case of directed graph, the indegree of a vertex is the number of edges for which that vertex is head i.e. the number of incoming edges to that vertex.

3. outdegree of vertex :

-- In case of directed graph, the outdegree of a vertex is the number of edges for which that vertex is tail.e. the number of outgoing edges from that vertex.

4. Path :

-- A path Vi to Vj exists if there exist vertices V1, V2, ...,Vn such that there exist edges (Vi,V1),(V1,V2), ...,(Vn-1,Vn),(Vn,Vj).

5. Length of a path :

--- No. of edges on the path.

6. Cycle :

-- a path having same start vertex and end vertex.

7. Subgraph : A graph G' is a subgraph of graph G=(V,E), if V(G') is a subset of V(G) and E(G') is also a subset of E(G).

8. Spanning tree :

-- It is a subset of graph with all vertices and n-1 edges such that they connect all the vertices.

54)How to represent graph in program

Two ways --- using array and using linked list

1. **Using array** : we use 2D array -- square matrix --- n by n where n is no. of vertices.
   This matrix is called adjacency matrix.
   For undirected graph, adjacency matrix is always symmetric i.e.
   adj[i][j]=adj[j][i]

   For undirected graph,
   if there is an edge (i,j) then,
   adj[i][j]=1,
   adj[j][i]=1.
   otherwise,
   adj[i][j]=0,
   adj[j][i]=0.

If there is no self edge for all vertices, then diagonal entries are zero.

2. Using linked list : we generate singly linked list for every vertex contain ng adjacen: vertex number as the value of a node in SLL. So if there are 'n' nodes then we have 'n' no. of singly linked list. all these SLL's first node addresses are maintained in either in an array or in another linked list. This is useful for directed graph.

55)What are algorithms to travers graph?

1.Depth first search
2.Breadth first search

Traversal of a graph : visiting every node.

1. Depth first search / traversal (DFS / DFT) :

a) Starting vertex is current vertex, visit (print) it first.
b) Visit (print) one adjacent vertex of the current vertex at at time and make it current.
c) If current vertex is not having any adjacent vertex, then backtrack.
d) Continue step b and c till all the vertices are not visited.

--- Backtracking is used.
--- We may get many DFS traversal for one starting vertex.

2. Breadth first search / traversal (BFS / BFT) :

--- Visit all the adjacent vertex of the current vertex and make any adjacen: vertex as current. Continue this till all the vertices are not visited.
--- We may get many BFS traversal for one starting vertex.

56)What is the meaning of searching and sorting?

Searching means finding out an element in array that meet some specified criteria. Sorting means rearranging all the items in array in increasing or decreasing order.

57)Analysis of sequential search .

The best case for sequential search is that it does one comparison, and matches X right away. In the worst case, sequential search does n comparisons, and either matches the last item in the list or doesn't match anything. The average case is harder to do. The Binary Searc

58)Explain algorithm for Binary search.

Binary Search Algorithm: The basic steps to perform Binary Search are:

- Sort the array in ascending order.
- Set the low index to the first element of the array and the high index to the last element.
- Set the middle index to the average of the low and high indices.
- If the element at the middle index is the target element, return the middle index.
- If the target element is less than the element at the middle index, set the high index to the middle index – 1.
- If the target element is greater than the element at the middle index,set the low index to the middle index + 1.
- Repeat steps 3-6 until the element is found or it is clear that the element is not present in the array.

59) Analysis of Binary Search.

In the base case, the algorithm will end up either finding the element or just failing and returning false. In both cases, the algorithm is going to take a constant time because only comparison and return statements are going to be executed.

60)Categories of sorting algorithms :

1. Stable Vs unstable
   Stable sorting algorithms preserve the relative order of equal elements.
   Unstable sorting algorithms don't.
2. In-place Vs out-place : no extra memory required for in-place - bubble sort, etc.
   Extra memory (as much as input data) is required for out-place ---- merge sort
3. Internal Vs External :
   Internal sorting -At the time of sorting, if sorting algorithm requires all the data must be present in the main memory .
   External sorting- At the time of sorting, if sorting algorithm can be applied on a a partial data which is present in main memory, and the remaining data may be present on secondary memory ---- --- merge sort.

61)What is stable and unstable sorting.

The **stability** of a sorting algorithm is concerned with how the algorithm treats equal (or repeated) elements.
Stable sorting algorithms preserve the relative order of equal elements.
Unstable sorting algorithms don't. In other words, **stable sorting** maintains the position of two equals elements relative to one another.

Some examples of stable algorithms are Merge Sort, Insertion Sort, Bubble Sort and Binary Tree Sort.
Unstable sorting algorithm- QuickSort, Heap Sort, and Selection sort .

62)What is the meaning of Best case,worst case and average case regarding to sorting?
Best case : the data to be sorted is already sorted.
Worst case : the data to be sorted is in a reverse order.
Average case : the data to be sorted is in random order.

63) Types of Sorting

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

64)Explain in short all sorts
1. Bubble sort :
In this adjacent elements are compared with each other.
In case of ascending order the largest element are taken to the bottom and placed at it's right place, in case of descending order, smallest elements are taken at the bottom and placed at it's right place.
2. Selection sort :
Select one place from an array, identify the correct element for that place,and place that element over there.
3. Insertion sort :
--- It is useful when our data is nearly sorted.
--- Given array is considered as two sections - sorted section and unsorted section. We take / pick one element from unsorted section and try to insert it in sorted section at its right place. We compare the picked value (key value) with every element of sorted section from right to left, if it is greater than key value then we will shift that element to the right side.
4.Quick Sort : partition sort / exchange sort
--- Technique used is divide and conquer
--- one element is taken as a pivot element at a time, and we place it in its correct position. While placing the pivot element we also rearrange the elements of an array in such a way that the elements to the left side are smaller than the pivot element and the elements to the right side of the pivot are greater than the pivot.
--- The place of the pivot is the partition point.
--- The element which is chosen as a pivot will also affect complexity of the quick sort.

5.Merge sort :
--- Divide and conquer technique
--- Uses the merging process - merge the two sorted list.
--- Divide the given array into two halves sub-arrays, divide again sub-arrays into two halves, continue this till every sub-array does not contain single element.
An array with one element is always sorted.

Then merge the sub-arrays to get the entire array back.

65) Explain Bubble Sort

- Bubble sorting is a simple sorting technique in which we arrange the elements of the list by forming pairs of adjacent elements
- Bubble Sort is an algorithm which is used to sort N elements that are given in a memory
- Bubble Sort compares all the element one by one and sort them based on their values.
- It is called Bubble sort, because with each iteration the smaller element in the list bubbles up towards the first place, just like a water bubble rises up to the water surface.
- Sorting takes place by stepping through all the data items one-by-one in pairs and comparing adjacent data items and swapping each pair that is out of order.

```
void bubbleSort(int[] arr)
{
int n = arr.length;
int temp = 0;
for(int i=0; i < n; i++){
for(int j=1; j < (n-i); j++){
if(arr[j-1] > arr[j]){
//swap elements
temp = arr[j-1];
arr[j-1] = arr[j];
arr[j] = temp;
}}
```

66)Does sorting happen in place in Bubble sort?
Yes, Bubble sort performs the swapping of adjacent pairs without the use of any major data structure. Hence Bubble sort algorithm is an in-place algorithm.

67)Is the Bubble sort algorithm stable?
Yes, the bubble sort algorithm is stable.

68)Advantages of bubble sort

- Bubble sort is easy to understand and implement.
- It does not require any additional memory space.
- It's adaptability to different types of data.

69)Disadvantages of bubble sort.

- Bubble sort has a time complexity of O(n^2) which makes it very slow for large data sets.
- It is not efficient for large data sets, because it requires multiple passes through the data.
- It is not a stable sorting algorithm, meaning that elements with the same key value may not maintain their relative order in the sorted output.

70)Explain Selection sort.

Selection sort is an algorithm that selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.

```
// Selection sort in Java
 void selectionSort(int array[]) {
 int size = array.length;
 for (int step = 0; step < size - 1; step++) {
 int min_idx = step;
 for (int i = step + 1; i < size; i++) {
 // To sort in descending order, change > to < in this line.
 // Select the minimum element in each loop.
 if (array[i] < array[min_idx]) {
 min_idx = i;
 }
 }

 // put min at the correct position
 int temp = array[step];
 array[step] = array[min_idx];
 array[min_idx] = temp;
 }
 }
```

71)Explain Selection Sort Applications.

The selection sort is used when:

-a small list is to be sorted.

-cost of swapping does not matter.

-checking of all the elements is compulsory.

-cost of writing to a memory matters like in flash memory (number of writes/swaps is O(n) as compared to O(n2) of bubble sort).

72) Explain Insertion sort.

Insertion sort is the sorting mechanism where the sorted array is built having one item at a time. The array elements are compared with each other sequentially and then arranged simultaneously in some particular order.

The analogy can be understood from the style we arrange a deck of cards. This sort works on the principle of inserting an element at a particular position, hence the name Insertion Sort.

```
 void insertionSort(int array[]) {
 int size = array.length;
 for (int step = 1; step < size; step++) {
 int key = array[step];
 int j = step - 1;
```



```
 // Compare key with each element on the left of it until an element
smaller than
 // it is found.
 // For descending order, change key<array[j] to key>array[j].
 while (j >= 0 && key < array[j]) {
 array[j + 1] = array[j];
 --j;
 }
 // Place key at after the element just smaller than it.
 array[j + 1] = key;
 }
 }
```

73) Insertion Sort Applications.

The insertion sort is used when:

- the array is has a small number of elements
- there are only a few elements left to be sorted

74)Explain Heap sort.

Heap Sort is one of the best sorting methods being in-place and with no quadratic worst-case running time. Heap sort involves building a Heap data structure from the given array and then utilizing the Heap to sort the array.

You must be wondering, how converting an array of numbers into a heap data structure will help in sorting the array.

Heap sort algorithm is divided into two basic parts:

- Creating a Heap of the unsorted list/array.
- Then a sorted array is created by repeatedly removing the largest/smallest element from the heap, and inserting it into the array. The heap is reconstructed after each removal.

```
public void sort(int arr[])
{
int n = arr.length;
// Build heap (rearrange array)
for (int i = n / 2 - 1; i >= 0; i--)
heapify(arr, n, i);
// One by one extract an element from heap
for (int i=n-1; i>=0; i--)
{
// Move current root to end
int temp = arr[0];
arr[0] = arr[i];
arr[i] = temp;
// call max heapify on the reduced heap
heapify(arr, i, 0);
}
}
```

```
// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
int largest = i; // Initialize largest as root
int l = 2*i + 1; // left = 2*i + 1
int r = 2*i + 2; // right = 2*i + 2
// If left child is larger than root
if (l < n && arr[l] > arr[largest])
largest = l;
// If right child is larger than largest so far
if (r < n && arr[r] > arr[largest])
largest = r;
// If largest is not root
if (largest != i)
{
int swap = arr[i];
arr[i] = arr[largest];
arr[largest] = swap;

// Recursively heapify the affected sub-tree
heapify(arr, n, largest);
}
}
```

75)Why Heap Sort is not stable?
Heap sort algorithm is not a stable algorithm. This algorithm is not stable because the operations that are performed in a heap can change the relative ordering of the equivalent keys.

76)Is Heap Sort an example of "Divide and Conquer" algorithm?
Heap sort is **NOT** at all a Divide and Conquer algorithm. It uses a heap data structure to efficiently sort its element and not a "divide and conquer approach" to sort the elements.

77)Which sorting algorithm is better – Heap sort or Merge Sort?
The answer lies in the comparison of their time complexity and space requirement. The Merge sort is slightly faster than the Heap sort. But on the other hand merge sort takes extra memory. Depending on the requirement, one should choose which one to use.

78)Why Heap sort better than Selection sort?
Heap sort is similar to selection sort, but with a better way to get the maximum element. It takes advantage of the heap data structure to get the maximum element in constant time.

79) Which is more efficient shell or heap sort?
Ans. As per big-O notation, shell sort has O(n^{1.25}) average time complexity whereas, heap sort has O(N log N) time complexity. According to a strict mathematical interpretation of the big-O notation, heap

sort surpasses shell sort in efficiency as we approach 2000 elements to be sorted.
Note:- Big-O is a rounded approximation and analytical evaluation is not always 100% correct, it depends on the algorithms' implementation which can affect actual run time.

80) What are the two phases of Heap Sort?
The heap sort algorithm consists of two phases. In the first phase the array is converted into a max heap. And in the second phase the highest element is removed (i.e., the one at the tree root) and the remaining elements are used to create a new max heap.

81)Is Merge sort Stable?
Yes, merge sort is stable.

82)How can we make Merge sort more efficient?
Merge sort can be made more efficient by replacing recursive calls with Insertion sort for smaller array sizes, where the size of the remaining array is less or equal to 43 as the number of operations required to sort an array of max size 43 will be less in Insertion sort as compared to the number of operations required in Merge sort.

83)Analysis of Merge Sort:
A merge sort consists of several passes over the input. The first pass merges segments of size 1, the second merges segments of size 2, and the pass merges segments of size 2^{i-1}. Thus, the total number of passes is $[log_2 n]$. As merge showed, we can merge two sorted segments in linear time, which means that each pass takes $O(n)$ time. Since there are $[log_2 n]$ passes, the total computing time is ***O(nlogn)***.

84)Applications of Merge Sort:

- Merge Sort is useful for sorting linked lists in O(N log N) time. In the case of linked lists, the case is different mainly due to the difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike an array, in the linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore, the merge operation of merge sort can be implemented without extra space for linked lists.
  In arrays, we can do random access as elements are contiguous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at (x + i*4). Unlike arrays, we can not do random access in the linked list. Quick Sort requires a lot of this kind of access. In a linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have a contiguous block of memory. Therefore, the overhead increases for quicksort. Merge sort accesses data sequentially and the need of random access is low.

85) Advantages of Merge Sort:

- Merge sort has a time complexity of O(n log n), which means it is relatively efficient for sorting large datasets.
- Merge sort is a stable sort, which means that the order of elements with equal values is preserved during the sort.
- It is easy to implement thus making it a good choice for many applications.

- It is useful for external sorting. This is because merge sort can handle large datasets, it is often used for external sorting, where the data being sorted does not fit in memory.
- The merge sort algorithm can be easily parallelized, which means it can take advantage of multiple processors or cores to sort the data more quickly.
- Merge sort requires relatively few additional resources (such as memory) to perform the sort. This makes it a good choice for systems with limited resources.

86)Drawbacks of Merge Sort:

- Slower compared to the other sort algorithms for smaller tasks. Although effecient for large datasets its not the best choice for small datasets.
- The merge sort algorithm requires an additional memory space of 0(n) for the temporary array. This is to store the subarrays that are used during the sorting process.
- It goes through the whole process even if the array is sorted.
- It requires more code to implement since we are dividing the array into smaller subarrays and then merging the sorted subarrays back together.

87)Solution of the drawback for additional storage:
Use linked list.

88)Explain Quick sort-
QuickSort is a Divide and Conquer algorithm. It picks an element as a pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below)
- Pick a random element as a pivot.
- Pick median as the pivot.

The key process in **quickSort** is a partition(). The target of partitions is, given an array and an element x of an array as the pivot, put x at its correct position in a sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

89)Advantages of Quick Sort:

- It is a divide-and-conquer algorithm that makes it easier to solve problems.
- It is efficient on large data sets.
- It is a stable sort, meaning that if two elements have the same key, their relative order will be preserved in the sorted output.
- It has a low overhead, as it only requires a small amount of memory to function.

90)Disadvantages of Quick Sort:

- It has a worst-case time complexity of O(n^2), which occurs when the pivot is chosen poorly.
- It is not a good choice for small data sets.

- It can be sensitive to the choice of pivot.
- It is not cache-efficient.

91)Analysis of sorting algorithm.

Time Complexities of Searching & Sorting Algorithms:

	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$

92)Hash function and Hash tables:
Hashing is the process of generating a value from a text or a list of numbers using a mathematical function known as a hash function.
A **Hash Function** is a function that converts a given numeric or alphanumeric key to a small practical integer value.
The mapped integer value is used as an index in the hash table. In simple terms, a hash function **maps** a significant number or string to a small integer that can be used as the **index** in the hash table.
The pair is of the form **(key, value)**, where for a given key, one can find a value using some kind of a "function" that maps keys to values. The key for a given object can be calculated using a function called a hash function.
For example, given an array A, if i is the key, then we can find the value by simply looking up A[i].

93)Types of Hash functions
There are many hash functions that use numeric or alphanumeric keys. This article focuses on discussing different hash functions:

1. **Division Method.**
2. **Mid Square Method.**
3. **Folding Method.**
4. **Multiplication Method.**

methods in detail are as follows

1. Division Method:

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula:

h(K) = k mod M

Here,

k is the key value, and

M is the size of the hash table.

It is best suited that **M** is a prime number as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division.

Example:

k = 12345

M = 95

h(12345) = 12345 mod 95

= 90

k = 1276

M = 11

h(1276) = 1276 mod 11

= 0

Pros:

1. This method is quite good for any value of M.
2. The division method is very fast since it requires only a single division operation.

Cons:

1. This method leads to poor performance since consecutive keys map to consecutive hash values in the hash table.
2. Sometimes extra care should be taken to choose the value of M.

2. Mid Square Method:

The mid-square method is a very good hashing method. It involves two steps to compute the hash value-

1. Square the value of the key k i.e. k^2
2. Extract the middle **r** digits as the hash value.

Formula:

h(K) = h(k x k)

Here,

k is the key value.

The value of **r** can be decided based on the size of the table.

Example:

Suppose the hash table has 100 memory locations. So r = 2 because two digits are required to map the key to the

memory location.

k = 60

k x k = 60 x 60

= 3600

h(60) = 60

The hash value obtained is 60

Pros:

1. The performance of this method is good as most or all digits of the key value contribute to the result. This is because
2. all digits in the key contribute to generating the middle digits of the squared result.
3. The result is not dominated by the distribution of the top digit or bottom digit of the original key value.

Cons:

1. The size of the key is one of the limitations of this method, as the key is of big size then its square will double
2. the number of digits.
3. Another disadvantage is that there will be collisions but we can try to reduce collisions.

3. Digit Folding Method:

This method involves two steps:

1. Divide the key-value k into a number of parts i.e. **k1, k2, k3,….,kn**, where each part has the same number of digits except for the last part that can have lesser digits than the other parts.
2. Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula:

k = k1, k2, k3, k4, ….., kn

s = k1+ k2 + k3 + k4 +….+ kn

h(K)= s

Here,

s is obtained by adding the parts of the key k

Example:

k = 12345

k1 = 12, k2 = 34, k3 = 5

s = k1 + k2 + k3

= 12 + 34 + 5

= 51

h(K) = 51

Note:

The number of digits in each part varies depending upon the size of the hash table. Suppose for example the size of the hash table is 100, then each part must have two digits except for the last part which can have a lesser number of digits.

4. Multiplication Method

This method involves the following steps:

1. Choose a constant value A such that $0 < A < 1$.
2. Multiply the key value with A.

3. Extract the fractional part of kA.
4. Multiply the result of the above step by the size of the hash table i.e. M.
5. The resulting hash value is obtained by taking the floor of the result obtained in step 4.

Formula:

$h(K) = floor\ (M\ (kA \bmod 1))$

Here,
M is the size of the hash table.
k is the key value.
A is a constant value.

Example:
$k = 12345$
$A = 0.357840$
$M = 100$

$h(12345) = floor[\ 100\ (12345*0.357840 \bmod 1)]$
$= floor[\ 100\ (4417.5348 \bmod 1)\ ]$
$= floor[\ 100\ (0.5348)\ ]$
$= floor[\ 53.48\ ]$
$= 53$

Pros:
The advantage of the multiplication method is that it can work with any value between 0 and 1, although there are some values that tend to give better results than the rest.

Cons:
The multiplication method is generally suitable when the table size is the power of two, then the whole process of computing the index by the key using multiplication hashing is very fast.

94) Explain Hash Tables .

A hash table is a data structure that maps keys to values. It uses a hash function to calculate the index for the data key and the key is stored in the index.

An example of a hash table is as follows –

The key sequence that needs to be stored in the hash table is –

35 50 11 79 76 85

The hash function h(k) used is:

h(k) = k mod 10

Using linear probing, the values are stored in the hash table as –

0	50
1	11
2	
3	
4	
5	35
6	76
7	85
8	
9	79

Hash Table

95)Explain Characteristics of good hashing function.

The hash function should generate different hash values for the similar string.

The hash function is easy to understand and simple to compute.

The hash function should produce the keys which will get distributed, uniformly over an array.

A number of collisions should be less while placing the data in the hash table.

The hash function is a perfect hash function when it uses all the input data.

96)What is Collision?

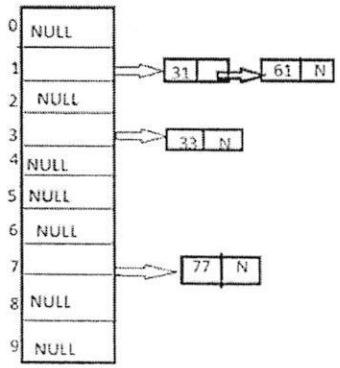
It is a situation in which the hash function returns the same hash key for more than one record, it is called as collision. Sometimes when we are going to resolve the collision it may lead to a overflow condition and this overflow and collision condition makes the poor hash function.

97)What are Collision resolution technique.

If there is a problem of collision occurs then it can be handled by apply some technique. These techniques are called as collision resolution techniques. There are generally four techniques which are described below.

1) Chaining

It is a method in which additional field with data i.e. chain is introduced. A chain is maintained at the home bucket. In this when a collision occurs then a linked list is maintained for colliding data.

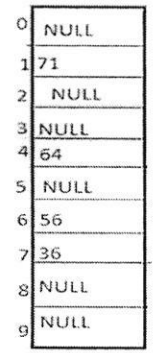


Example: Let us consider a hash table of size 10 and we apply a hash function of H(key)=key % size of table. Let us take the keys to be inserted are 31,33,77,61. In the above diagram we can see at same bucket 1 there are two records which are maintained by linked list or we can say by chaining method.

2) Linear probing

It is very easy and simple method to resolve or to handle the collision. In this collision can be solved by placing the second record linearly down, whenever the empty place is found. In this method there is a problem of clustering which means at some place block of a data is formed in a hash table.

Example: Let us consider a hash table of size 10 and hash function is defined as H(key)=key % table size. Consider that following keys are to be inserted that are 56,64,36,71.

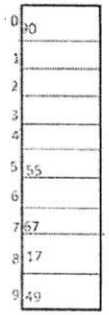


In this diagram we can see that 56 and 36 need to be placed at same bucket but by linear probing technique the records linearly placed downward if place is empty i.e. it can be seen 36 is placed at index 7.

3) Quadratic probing

This is a method in which solving of clustering problem is done. In this method the hash function is defined by the H(key)=(H(key)+x*x)%table size.

Let us consider we have to insert following elements that are:-67, 90,55,17,49.



In this we can see if we insert 67, 90, and 55 it can be inserted easily but at case of 17 hash function is used in such a manner that :-(17+0*0)%10=17 (when x=0 it provide the index value 7 only) by making the increment in value of x.

let x =1 so (17+1*1)%10=8.in this case bucket 8 is empty hence we will place 17 at index 8.

4) Double hashing

It is a technique in which two hash function are used when there is an occurrence of collision. In this method 1 hash function is simple as same as division method. But for the second hash function there are two important rules which are

-It must never evaluate to zero.

-Must sure about the buckets, that they are probed.

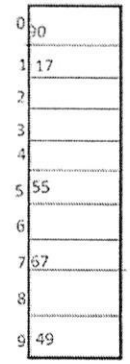
The hash functions for this technique are:

H1(key)=key % table size

H2(key)=P-(key mod P)

Where, p is a prime number which should be taken smaller than the size of a hash table.

Example: Let us consider we have to insert 67, 90,55,17,49.



In this we can see 67, 90 and 55 can be inserted in a hash table by using first hash function but in case of 17 again the bucket is full and in this case we have to use the second hash function which is H2(key)=P-(key mode P) here p is a prime number which should be taken smaller than the hash table so value of p will be the 7.

i.e. H2(17)=7-(17%7)=7-3=4 that means we have to take 4 jumps for placing the 17. Therefore 17 will be placed at index 1.

97)What is algorithm design in data structure?

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

98) Define algorithm .What are the main types of algorithms?

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output

The seven types of algorithms are the-

1. brute force-based algorithm,
2.greedy algorithm,
3.recursive algorithm,
4.backtracking algorithm
5.divide and conquer algorithm
6.dynamic programming algorithm
7.randomized algorithm.

99)Example of divide and conquer algorithm.

Merge sort,bubble sort

100) Example of greedy algorithm

Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees and the algorithm for finding optimum Huffman trees.

101) What are recursive and non recursive algorithm examples?

Merge sort and quick sort are examples of recursive sorting algorithms.

A non-recursive technique is anything that doesn't use recursion. Insertion sort is a simple example of a n on-recursive sorting algorithm.

102)what is backtracking algorithm?

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

1. Decision Problem – In this, we search for a feasible solution.
2. Optimization Problem – In this, we search for the best solution.
3. Enumeration Problem – In this, we find all feasible solutions.

103) Difference between Recursion and backtracking.

Recursion is a technique that calls the same function again and again until you reach the base case.

Backtracking is an algorithm that finds all the possible solutions and selects the desired solution from the given set of solutions.

104)What is recursion?

Recursion-The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Properties of Recursion:

- Performing the same operations multiple times with different inputs.
- In every step, we try smaller inputs to make the problem smaller.
- A base condition is needed to stop the recursion otherwise infinite loop will occur.

105)What is Backtracking?

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

Backtracking can be defined as a general algorithmic technique that considers searching every possible combination in order to solve a computational problem.

----------Set II----------

106)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

107)List out the areas where the data structure is applied?

The data structure is a vital aspect while handling data. The following are specific areas where the data structure is applied:

- Numerical analysis
- Operating systems
- A.I.(Artificial Intelligence)
- Database management
- Statistical analysis

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

The main difference between storage structure and file structure depends on the memory area that is accessed.

- **Storage structure:** It's a data structure representation in computer memory.
- **File structure:** It's a storage structure representation in the auxiliary memory.

109)Explain how does dynamic memory allocation will help you in managing data?

A dynamic memory allocation will help you effectively manage your data by allocating structured blocks to have composite structures that can be flexible, i.e. it can expand and can contract based on the need.

Also, they are capable of storing simple structured data types.

110) Explain a about the dynamic data structure?

The nature of the dynamic data structure is different compared to the standard data structures, the word dynamic data structures means that the data structure is flexible in nature. As per the need, the data structure can be expanded and contracted. Thus it helps the users to manipulate the data without worrying too much about the data structure flexibility.

111)Explain the main difference between PUSH and a POP?

The two main activities, i.e. Pushing and Popping applies the way how data is stored and retrieved in an entity.So if you check in detail, a Push is nothing but a process where data is added to the stack. On the contrary, a Pop is an activity where data is retrieved from the stack. When we discuss data retrieval it only considers the topmost available data.

112)What operations can be performed on a stack?

Mainly the following operations are performed on a stack:

- **Push operation:** To add an item to the stack. If the stack is complete, then it is in an overflow condition.
- **Pop operation:** It is used to remove an item from the stack. If it's an empty stack, then it is in underflow
  condition.
- **isEmpty operation:** If the stack is empty returns true, else false.
- **Peek or Top operation:** This returns the top element of the stack.

113)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.

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

AB+CD-*

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

Polish and Reverse Polish notations.

116)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.

117)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.

118)How would you implement a queue using a stack?

Using two stacks, you can implement a queue. The purpose is to complete the queue's en queue operation so that the initially entered element always ends up at the top of the stack.

- First, to enqueue an item into the queue, migrate all the elements from the beginning stack to the second stack,
- push the item into the stack, and send all elements back to the first stack.
- To dequeue an item from the queue, return the top item from the first stack.

119)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.

120)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.

- Every node in a B-Tree contains at most m children.
- Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
- The root nodes must have at least 2 nodes.
- All leaf nodes must be at the same level.

121)Can you tell me the minimum number of nodes that a binary tree can have?

A binary tree is allowed or can have a minimum of zero nodes. Further, a binary tree can also have 1 or 2 nodes.

122) What is the max heap in the data structure?

A max heap in a data structure is a complete binary tree where each internal node's value is greater than or equal to that node's children's values.

123)Differentiate among cycle, path, and circuit(for Graph)?

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

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

For the graph implementation, following data structures are used.

- In sequential representation, Adjacency matrix is used.
- In Linked representation, Adjacency list is used.

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

- In BFS algorithm, Queue data structure is used.
- In DFS algorithm, Stack data structure is used.

126) Explain the functionality of linked list.

A linked list consists of two parts: information and the link. In the single connected listening, the beginning of the list is marked by a unique pointer named start. This pointer does point to the first element of the list and the link part of each node consists of an arrow looking to the next node, but the last node of the list has null pointer identifying the previous node. With the help of start pointer, the linked list can be traversed easily.

127) Compare Quick and Merge sort

Comparison	Quick Sort	Merge Sort
Partition of the element in the array.	The splitting of a list of elements is not necessarily divided into half.	The Array is always divided into half(n/2)
Worst Case Complexity	$O(n^2)$	O(n log n)
Speed	Faster than another sorting algorithm for the small dataset.	Consistent speed in all type of datasets.
Additional Storage Space Requirement	Less	More
Efficiency	Inefficient for the larger array.	More efficient
Sorting Method	Internal	External

128)Define Order of precedence and Associativity, how are they used?

Order of precedence is used with the operators. When a number of operators are used in an expression, it evaluates with the priority of the operators.

Associativity is used to check whether an expression is evaluated from left-to-right or right-to-left.

Order of precedence example:

(5 > 2 + 5 && 4)

The given expression is equivalent to:

((5 > (2 + 5)) && 4)

The expression (2 + 5) will executes first and the result will be 7

Then after first part of the expression (5 > 7) executes and gives 0 (false) as an output

Finally, (0 && 4) executes and gives 0 (false).

Associativity:

1. 4 * 2 / 4

Here, operators * and / have the same precedence. Both "*" and "/" are left to right associative, i.e., the expression on the left is executed first and moves towards the right. Thus, the expression above is equivalent to:

1. ((4 * 2) / 4)
2. i.e., (4 * 2) executes first and the result will be 8 (**true**)
3. then, (8 / 4) executes and the **final** output will be 2 (**true**)

129)What is Hashing technique in the data structure?

Hashing is a faster searching technique. The process of mapping a large amount of data item to a smaller table with the help of a hashing function is called hashing. In other words, hashing is a technique to convert a range of key values into a range of indexes of an array.

In terms of java: Hashing is a way to assign a unique code for any variable or object after applying any function or algorithm on its properties.

130)List the types of trees.

- General Tree
- Binary Tree
- Forests
- Expression Tree
- Binary Search Tree
- AVL Tree
- Threaded Binary Tree

131)What is the AVL tree? What is its significance?

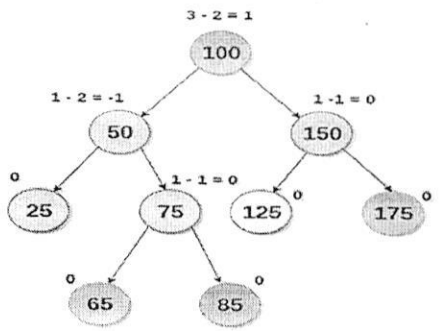
AVL tree is named after its invention by Adelson-Velsky and Landis. AVL tree is a height-balanced binary search tree of the self-balanced binary search tree in which:

- Each node of the AVL tree is associated with a balancing factor
- Balancing factor can be calculated as the difference between the heights of the left and right sub-tree cannot be more than one for each node in the tree.

  Balance Factor (k) = height (left (k)) - height (right (k))

Time complexity: Since AVL tree is balanced, its height is O(log (n)) and hence time complexity for insertion is O(log n).

The Significance of AVL tree: As binary search offer good performance while searching in balanced case, but if they are unbalanced their searching performance can be reduced hence to overcome this problem we can use AVL tree as it is a self-balanced binary search tree. Therefore, it ensures a time complexity of O(log(n)).



AVL Tree

132)Can we check whether a link list is circular or not?

Yes, we can check that a given link-list is circular or not. A link-list will be a circular link-list if it follows the two main requirements:

-If a link list is not null terminated(it points to the first node)

-If all nodes are connected in the form of cycle.

133)What type of operation can be performed using stack and queue in Data structure?

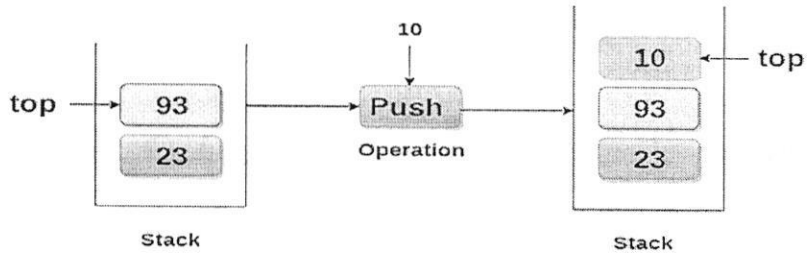
Organization of data in the computer can be done with the help of data structure so that data can be accessed easily and efficiently. There are different types of data structure used in computer, and two of them are Stack and Queue data Structure.

Stack: A stack is a type of linear-data structure, which logically represents and arrange the data in the form of a stack. As a real-life example of a stack is "plates arranged in the form of the stack." In a stack structure, any operation can be performed on data items from one end only.

A Stack structure follows a particular order for operation on data items, and that order can be LIFO (Last in First Out) or FILO (First in Last Out). A stack can be represented in the form of an array.

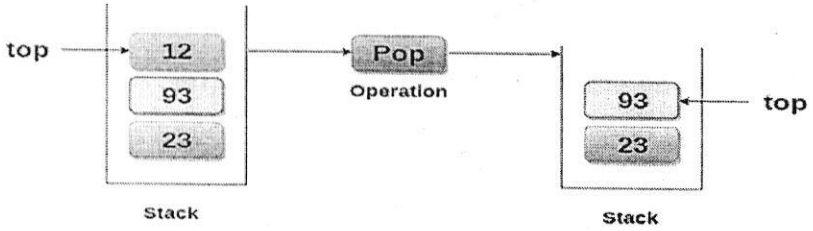
Types of Operations performed on Stack:

1.Push: Push is an operation that can be performed, to add an element in the stack.



As in the above diagram, the top element was 93 (before addition of a new element), and after performing PUSH operation, a top element is 10. Now pointer will point at the top of the stack.

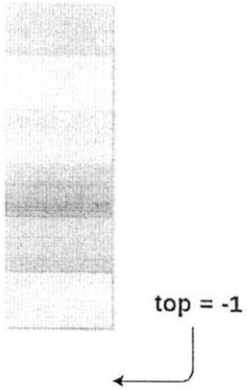
2.Pop: If we try to remove or delete an element from the Stack, then it is called as Pop operation.



As in the above diagram, if we want to delete an element from the top of the Stack, then it can be done by the pop() operation.

3.isEmpty: If we wanted to check whether the stack is empty or not, then we can perform an isEmpty operation.It will return three values: If we will perform a Pop operation on empty Stack, then it is called underflow condition.

Empty Stack



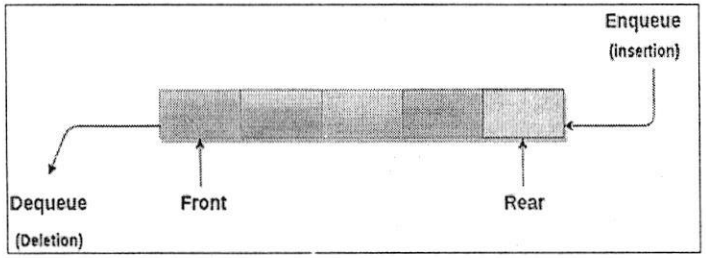
Empty	-1
Single element present	0
Stack is full	N-1
Stack overflow	N

4. **Peek or Top**: If we perform Peek operation it checks all the elements of the stack and returns the top element.

Queue: A Queue is an ordered collection of data elements same as a stack, but it enables insertion operation from one end called as REAR end and deletion operation from other end called as a FRONT end.

A Queue structure follows an order of FIFO (First in First Out) for insertion and deletion of the data element.

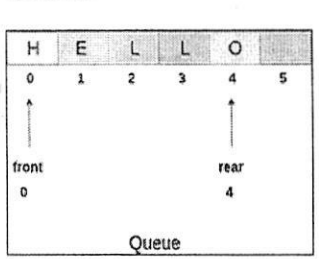
Real life example of a queue is people waiting to buy a movie ticket in a line.

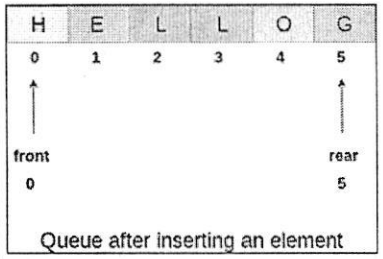


Types of Operations performed on Queue: The two main operations which can be performed On Queue are Enqueue and Dequeue.

1.Enqueue:

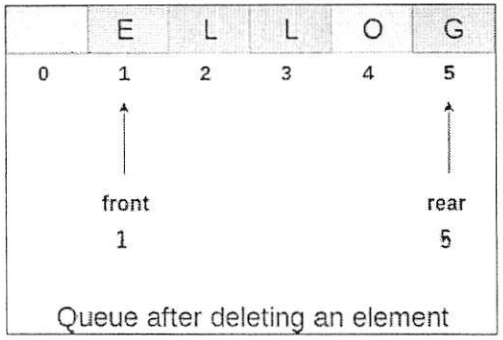
This operation is performed to add an element in the queue at the rear end. After adding an element in a queue, the count of Rear pointer increased by 1. Below is the Array representation of queue with Enqueue operation.





2.Dequeue:

This operation is performed to remove an element from the queue at the front end. After removing an element from the queue, the count of Front pointer gets decremented by 1. Below is the diagram which shows the removal of the data element from a queue.



Other operations performed on the queue are:

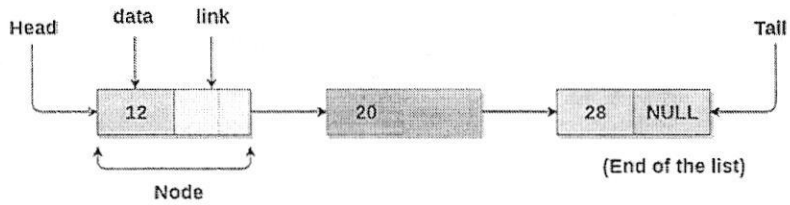
3.Peek: This operation is used to get all the data elements of queue without deletion of an element, at the front end.

4.Isfull: This operation is performed to check whether a queue is full or not.

5.Isempty: This operation is performed to check whether a queue is empty or not.

134)What is a linked list? Explain its applications?

- A Linked list is a linear data structure similar to an array, which is used to store the data in an organized way.
- In Linked list data elements are not stored in contiguous blocks.



Applications of the Linked list:

- A linked list can be used for implementations of stacks and queues.
- Implementation of graphs can be done using the linked list
- A linked list can be used for dynamic memory allocation
- A linked list can be used for implementation of graph
- It can be used for performing arithmetic operations on long integers.
- A linked list can be used with the music player for playing the songs.

135)Differentiate between graph and tree?

A tree and a graph both are the non-linear data structure, which consists of nodes and edges. The main differences between both tree and graph are given below:

Graph	Tree
In a graph, more than one path is allowed	In a tree only one path is allowed between two nodes.
In a graph, there is no any root node	It contains one root node
A graph represents a network model	A tree represents a hierarchal structure.
In a graph, there is no any pre-defined number of edges.	In a tree, the number of edges should be n-1(where n= no of nodes)

136) Which data structure is used for the dictionary?

To implement a dictionary, which type of data structure should be used depends on what we required, there are some following data structure which can be used for implementation of the dictionary.

Hash-table: If we want a simple dictionary with no option for the prefix, or nearest neighbor search then we can use Hashing or Hashtable for the dictionary.

Trie: It can be a good option if we want to add prefix and fast lookup. But, it takes more space than other data structures.

Ternary Search Tree: If we want all the qualities like trie but do not want to give the more space then we can use ternary search tree.

BK-tree: BK-tree is one of the best data structure if we want specifications like spell checker, find the similar word, etc.

137)Where is the LRU cache used in data structure?

In data structures, you use LRU (Least Recently Used) cache to organize items in order of use, enabling you to quickly find out which item hasn't been used for a long time.

138)Which Data Structure is used to implement 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 the rear end and the least recent pages will be near the front end.

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

139)How will you design a data structure for excel spreadsheets?

We can design an excel spreadsheets by using:

- Two-dimensional array (but it will take lots of space)
- Sparse matrix
- Map, etc.

140)Types of Graph

Directed or undirected

In **directed** graphs, edges point from the node at one end to the node at the other end. In **undirected** graphs, the edges simply connect the nodes at each end.

Cyclic or acyclic

A graph is **cyclic** if it has a cycle—an unbroken series of nodes with no repeating nodes or edges that connects back to itself. Graphs without cycles are **acyclic**.

Weighted or unweighted

If a graph is **weighted**, each edge has a "weight." The weight could, for example, represent the distance between two locations, or the cost or time it takes to travel between the locations.

141)What are Advanced graph algorithms

If you have lots of time before your interview, these advanced graph algorithms pop up occasionally:

- **Dijkstra's Algorithm:** Finds the shortest path from one node to all other nodes in a *weighted* graph.
- **Topological Sort:** Arranges the nodes in a *directed, acyclic* graph in a special order based on incoming edges.
- **Minimum Spanning Tree:** Finds the cheapest set of edges needed to reach all nodes in a *weighted* graph.

142)What is the advantage of using Bellman-Ford over Dijkstra?

The Bellman-Ford algorithm finds single source shortest paths by repeatedly relaxing distances until there are no more distances to relax. Relaxing distances is done by checking if an intermediate point provides a better path than the currently chosen path.

143)How do you detect a loop in a linked list?

Approach 1)

The first approach we can **use Hash Table** in hash table basically we record each node's reference (or memory address) in a hash table. if the current node is null we reached the end of the Linked List and its not a cycle otherwise if the node's reference present in the Hash table, then there is a loop.

Approach 2)

1. Traverse Linked List using two-pointer.
2. Move slow pointer by one position
3. Move fast pointer by two positions.
4. If these two pointers meet at the same node then there is a Loop. If they don't meet in the same position (pointer) then Linked List doesn't have a loop.

144) Can you store a duplicate key in Hash map?

No, duplicate keys are not allowed in HashMap. The key in a entry pair is a unique reference to the Value, also to the location of that pair in the HashMap. When there is an attempt to put an entry with an existing key, the existing entry will be replaced with the new one. In such a case, the `put()` method of HashMap returns the existing entry that has been deleted.

HashMap does not allow for duplicate keys however, it does allow duplicate values. This means there can be multiple different keys with the same values.

HashMap also allows a null key, however as this is a key there can only be one null key per HashMap.

You can have multiple null values.

145)What is infix, prefix, and postfix in data structure?

The way to write arithmetic expressions is known as notation. There are three types of notations used in an arithmetic expression, i.e., without changing the essence or output of expression. These notations are:

1.Prefix (Polish) Notation - In this, the operator is prefixed to operands, i.e. ahead of operands.

2.Infix Notation - In this, operators are used in between operands.

3.Postfix (Reverse-Polish) Notation - In this, the operator is postfixed to the operands, i.e., after the operands.

146)What is the difference between Hashing and Hash Tables?

Hashing: is simply the act of turning a data chunk of arbitrary length into a fixed-width value (hereinafter called a hash value) that can be used to represent that chunk in situations where dealing with the original data chunk would be inconvenient.

A hash table It is an abstract data type that maps keys to values. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found.

Hash table is a data structure that stores some information, and the information has basically two main components, i.e., key and value.

147)Why do we use Big O instead of Big Theta(θ)

Because, when analyzing performance, we are usually only interested in the worst-case scenario. Knowing the upper bound is therefore sufficient. When it performs better than expected for given input. Some Algorithms don't have tight bound at all. Moreover, tight bounds are often more difficult to compute.

148) What do you mean by Bipartite graph :

A bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V such that every edge connects a vertex in U to one in V.

149)What do you mean by Eulerian directed path?

An Eulerian trail (or Eulerian path) is a path in a graph that visits every edge exactly once.

150)What are Properties of Minimum Spanning Tree (MST)?

For a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have multiple spanning trees.

A **Minimum Spanning Tree(MST)** or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree having a weight less than or equal to the weight of every other possible spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

Necessary conditions for Minimum Spanning Tree:

1.It must not form a cycle i.e, no edge is traversed twice.

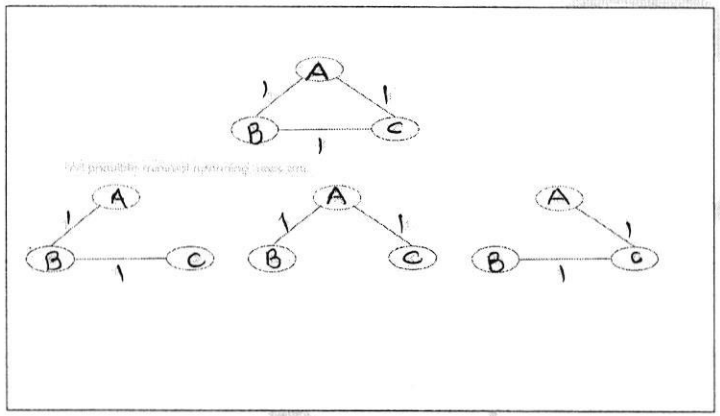
2.There must be no other spanning tree with lesser weight.

properties of MST:

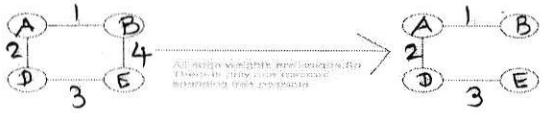
Possible Multiplicity:

If **G(V, E)** is a graph then every spanning tree of graph **G** consists of **(V – 1)** edges, where **V** is the number of vertices in the graph and **E** is the number of edges in the graph. So, **(E – V + 1)** edges are not a part of the spanning tree. There may be several minimum spanning trees of the same weight. If all the edge weights of a graph are the same, then every spanning tree of that graph is minimum.

Consider a complete graph of three vertices and all the edge weights are the same then there will be three spanning trees(which are also minimal) of the same path length are possible. Below is the image to illustrate the same:



Each of the spanning trees has the same weight equal to **2**.



Minimum Cost Subgraph

For all the possible spanning trees, the minimum spanning tree must have the minimum weight possible. However,there may exist some more spanning with the same weight that of minimum spanning tree, and those all may also be considered as Minimum Spanning tree.

- **Minimum Cost Edge:** If the minimum cost edge of a graph is unique, then this edge is included in any MST. For example, in the above figure, the edge **AB** (of the least weight) is always included in MST.

- If a new edge is added to the spanning tree then it will become cyclic because every spanning tree is minimally acyclic. In the above figure, if edge **AD** or **BC** is added to the resultant MST, then it will form a cycle.
- The spanning tree is minimally connected, i.e., if any edge is removed from the spanning tree it will disconnect the graph. In the above figure, if any edge is removed from the resultant MST, then it will disconnect the graph.

151) Algorithms for finding Minimum Spanning Tree(MST):-

Prim's Algorithm
Kruskal's Algorithm

152) Difference between Prim's and Kruskal's algorithm for MST (minimum spanning tree)

Kruskal's algorithm for MST

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together.
A single graph can have many different spanning trees.
A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree.
The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

Below are the steps for finding MST using Kruskal's algorithm

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning-tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Prim's algorithm for MST

Like Kruskal's algorithm, Prim's algorithm is also a Greedy algorithm.
It starts with an empty spanning tree.
The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included.
At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Below are the steps for finding MST using Prim's algorithm

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices
   - Pick a vertex u which is not there in mstSet and has minimum key value.
   - Include u to mstSet.
   - Update the key value of all adjacent vertices of u. To update the key values, iterate through all
   - adjacent vertices. For every adjacent vertex v, if the weight of edge u-v is less than the previous key
   - value of v, update the key value as the weight of u-v

Both **Prim's and Kruskal's algorithm** finds the Minimum Spanning Tree and follow the Greedy approach of problem-solving, but there are few **major differences between them**.

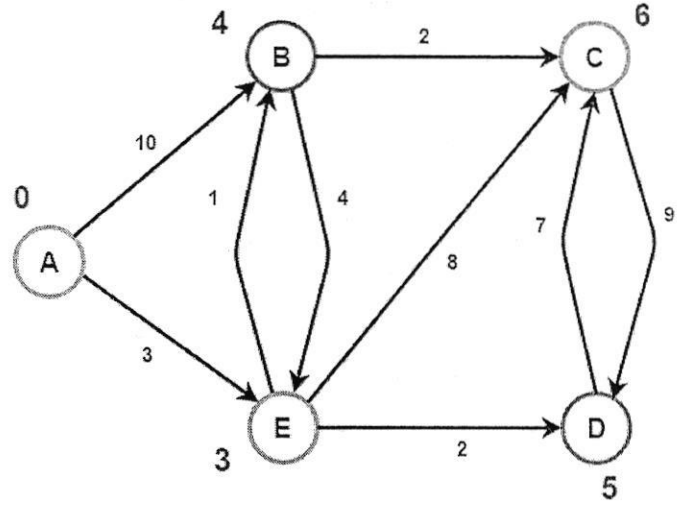
Prim's Algorithm	Kruskal's Algorithm
It starts to build the Minimum Spanning Tree from any vertex in the graph.	It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph.
It traverses one node more than one time to get the minimum distance.	It traverses one node only once.
Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to O(E log V) using Fibonacci heaps.	Kruskal's algorithm's time complexity is O(E log V), V being the number of vertices.
Prim's algorithm gives connected component as well as it works only on connected graph.	Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components
Prim's algorithm runs faster in dense graphs.	Kruskal's algorithm runs faster in sparse graphs.
It generates the minimum spanning tree starting from the root vertex.	It generates the minimum spanning tree starting from the least weighted edge.
Applications of prim's algorithm are Travelling Salesman Problem, Network for roads and Rail tracks connecting all the cities etc.	Applications of Kruskal algorithm are LAN connection, TV Network etc.
Prim's algorithm prefer list data structures.	Kruskal's algorithm prefer heap data structures

153)Explain Single-Source Shortest Paths(SSSP algorithm) – Dijkstra's Algorithm

Given a source vertex s from a set of vertices V in a weighted digraph where all its edge weights w(u, v) are non-negative, find the shortest path weights d(s, v) from source s for all vertices v present in the graph.

For example,

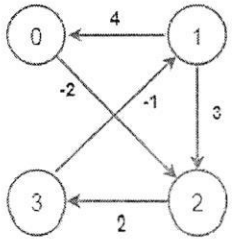
Vertex	Minimum Cost	Route
A —> B	4	A —> E —> B
A —> C	6	A —> E —> B —> C
A —> D	5	A —> E —> D
A —> E	3	A —> E



Dijkstra's algorithm don't work for negative edges.

154) Explain All-Pairs Shortest Paths(APSP Algorithm) – Floyd Warshall Algorithm

Given a set of vertices V in a weighted graph where its edge weights w(u, v) can be negative, find the shortest path weights d(s, v) from every source s for all vertices v present in the graph. If the graph contains a negative-weight **cycle**, report it.
For example, consider the following graph:



The adjacency matrix containing the shortest distance is:

```
0 -1 -2  0
4  0  2  4
5  1  0  2
3 -1  1  0
```

The shortest path from:

- vertex 0 to vertex 1 is [0 —> 2 —> 3 —> 1]
- vertex 0 to vertex 2 is [0 —> 2]
- vertex 0 to vertex 3 is [0 —> 2 —> 3]
- vertex 1 to vertex 0 is [1 —> 0]
- vertex 1 to vertex 2 is [1 —> 0 —> 2]
- vertex 1 to vertex 3 is [1 —> 0 —> 2 —> 3]
- vertex 2 to vertex 0 is [2 —> 3 —> 1 —> 0]
- vertex 2 to vertex 1 is [2 —> 3 —> 1]
- vertex 2 to vertex 3 is [2 —> 3]
- vertex 3 to vertex 0 is [3 —> 1 —> 0]
- vertex 3 to vertex 1 is [3 —> 1]
- vertex 3 to vertex 2 is [3 —> 1 —> 0 —> 2]

We have already covered **single-source the shortest paths** in separate posts. We have seen that:

- For graphs having non-negative edge weights, Dijkstra's Algorithm runs in $O(E + V \times \log(V))$
- For graphs containing negative edge weights, Bellman–Ford runs in $O(V \times E)$.

This introduce **All-Pairs Shortest Paths** that return the shortest paths between every pair of vertices in the graph containing negative edge weights.

155)Compare Dijkstra's and Floyd–Warshall algorithms to find shortest path in graph

Main Purposes:

- Dijkstra's Algorithm is one example of a single-source shortest or SSSP algorithm, i.e., given a source vertex it finds shortest path from source to all other vertices.
- Floyd Warshall Algorithm is an example of all-pairs shortest path algorithm, meaning it computes the shortest path between all pair of nodes.

Time Complexities :

- Time Complexity of Dijkstra's Algorithm: $O(E \log V)$
- Time Complexity of Floyd Warshall: $O(V^3)$

Other Points:

- We can use Dijskstra's shortest path algorithm for finding all pair shortest paths by running it for every vertex. But time complexity of this would be O(VE Log V) which can go (V^3 Log V) in worst case.
- Another important differentiating factor between the algorithms is their working towards distributed systems. Unlike Dijkstra's algorithm, Floyd Warshall can be implemented in a distributed system, making it suitable for data structures such as Graph of Graphs (Used in Maps).
- Lastly Floyd Warshall works for negative edge but no negative cycle, whereas Dijkstra's algorithm don't work for negative edges.

156) Difference between BFS and DFS

		BFS	DFS
1.	Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	Data Structure	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.
4.	Technique	BFS can be used to find a single source shortest path in an unweighted graph because, in BFS, we reach a vertex with a minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
5.	Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
6.	Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).

7.	Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.
8.	Suitable for Decision Trees their winning	BFS considers all neighbors first and therefore not suitable for decision-making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, and then explore all paths through this decision. And if this decision leads to win situation, we stop.
9.	Time Complexity	The Time complexity of BFS is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.
10.	Visiting of Siblings/ Children	Here, siblings are visited before the children.	Here, children are visited before the siblings.
11.	Removal of Traversed Nodes	Nodes that are traversed several times are deleted from the queue.	The visited nodes are added to the stack and then removed when there are no more nodes to visit.
12.	Backtracking	In BFS there is no concept of backtracking.	DFS algorithm is a recursive algorithm that uses the idea of backtracking
13.	Applications	BFS is used in various applications such as bipartite graphs, shortest paths, etc.	DFS is used in various applications such as acyclic graphs and topological order etc.
14.	Memory	BFS requires more	DFS requires less

		memory.	memory.
15.	Optimality	BFS is optimal for finding the shortest path.	DFS is not optimal for finding the shortest path.
16.	Space complexity	In BFS, the space complexity is more critical as compared to time complexity.	DFS has lesser space complexity because at a time it needs to store only a single path from the root to the leaf node.
17.	Speed	BFS is slow as compared to DFS.	DFS is fast as compared to BFS.
18.	When to use?	When the target is close to the source, BFS performs better.	When the target is far from the source, DFS is preferable.

157)What is complexity analysis in algorithm?

Algorithmic complexity is **a measure of how long an algorithm would take to complete given an input of size n**.

If an algorithm has to scale, it should compute the result within a finite and practical time bound even for large values of n. For this reason, complexity is calculated asymptotically as n approaches infinity.

What is complexity in data structure with example?

Time complexity is **a type of computational complexity that describes the time required to execute an algorithm**. The time complexity of an algorithm is the amount of time it takes for each statement to complete. As a result, it is highly dependent on the size of the processed data.

What are the two types of complexity analysis?

The complexity of an algorithm computes the amount of time and spaces required by an algorithm for an input of size (n). The complexity of an algorithm can be divided into two types. **The time complexity and the space complexity**.

158)Explain asymptotic notations that are used to represent the time complexity of an algorithm

There are three asymptotic notations

- **Θ Notation (theta)** - The Θ Notation is used to find the average bound of an algorithm i.e. it defines an upper bound and a lower bound, and your algorithm will lie in between these levels.
- **Big O Notation-** The Big O notation defines the upper bound of any algorithm i.e. you algorithm can't take more time than this time. In other words, we can say that the big O notation denotes the maximum time taken by an algorithm or the worst-case time complexity of an algorithm. So, big O notation is the most used notation for the time complexity of an algorithm.

- **Ω Notation** - The Ω notation denotes the lower bound of an algorithm i.e. the time taken by the algorithm can't be lower than this. In other words, this is the fastest time in which the algorithm will return a result. Its the time taken by the algorithm when provided with its best-case input.

159)Can doubly-linked be implemented using a single pointer variable in every node?

A doubly linked list can be implemented using a single pointer.

160)What is the postfix form of: (X + Y) * (Z - C)

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

----------Set III----------

161)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)

```
/* Java Program to implement a stack using  two queue */
 import java.util.*;

 class GG {

   static class Stack {
      // Two inbuilt queues
      static Queue<Integer> q1
         = new LinkedList<Integer>();
      static Queue<Integer> q2
         = new LinkedList<Integer>();

      // To maintain current number of
      // elements
      static int curr_size;

      static void push(int x)
      {
         // Push x first in empty q2
         q2.add(x);

         // Push all the remaining
         // elements in q1 to q2.
         while (!q1.isEmpty()) {
            q2.add(q1.peek());
```

```
        q1.remove();
    }

    // swap the names of two queues
    Queue<Integer> q = q1;
    q1 = q2;
    q2 = q;
}

static void pop()
{

    // if no elements are there in q1
    if (q1.isEmpty())
        return;
    q1.remove();
}

static int top()
{
    if (q1.isEmpty())
        return -1;
    return q1.peek();
}

static int size() { return q1.size(); }
}

// driver code
public static void main(String[] args)
{
    Stack s = new Stack();
    s.push(1);
    s.push(2);
    s.push(3);

    System.out.println("current size: " + s.size());
    System.out.println(s.top());
    s.pop();
    System.out.println(s.top());
    s.pop();
    System.out.println(s.top());

    System.out.println("current size: " + s.size());
}
```



```
}
```

162)How to implement a queue using a stack?

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

```
// Java program to implement Queue using
// two stacks with costly enQueue()

import java.util.*;

class GG
{
static class Queue
{
    static Stack<Integer> s1 = new Stack<Integer>();
    static Stack<Integer> s2 = new Stack<Integer>();

    static void enQueue(int x)
    {
        // Move all elements from s1 to s2
        while (!s1.isEmpty())
        {
            s2.push(s1.pop());
            //s1.pop();
        }

        // Push item into s1
        s1.push(x);

        // Push everything back to s1
        while (!s2.isEmpty())
        {
            s1.push(s2.pop());
            //s2.pop();
        }
    }

    // Dequeue an item from the queue
    static int deQueue()
    {
        // if first stack is empty
        if (s1.isEmpty())
        {
            System.out.println("Q is Empty");
```

```
            System.exit(0);
        }

        // Return top of s1
        int x = s1.peek();
        s1.pop();
        return x;
    }
};
// Driver code
public static void main(String[] args)
{
    Queue q = new Queue();
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);

    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
}
}
```

163)Write a program to reverse a linked list

```
// Java program for reversing the linked list

class LinkedList {

    static Node head;

    static class Node {

        int data;
        Node next;

        Node(int d)
        {
            data = d;
            next = null;
        } }
    /* Function to reverse the linked list */
    Node reverse(Node node)
    {
        Node prev = null;
        Node current = node;
```



```
        Node next = null;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }

        node = prev;
        return node;
    }


    // prints content of double linked list
    void printList(Node node)
    {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        LinkedList list = new LinkedList();
        list.head = new Node(85);
        list.head.next = new Node(15);
        list.head.next.next = new Node(4);
        list.head.next.next.next = new Node(20);

        System.out.println("Given linked list");
        list.printList(head);
        head = list.reverse(head);
        System.out.println("");
        System.out.println("Reversed linked list ");
        list.printList(head);
    }
}
```

164) where are Tree Data Structures used?

Tree data structures are used in a variety of applications. Following are some of them:

- Arithmetic expression handling
- Symbol table creation
- Lexical analysis

- Hierarchical data modeling

165)What are the time complexities of linear search and binary search?

Binary search is more effective as it takes lesser comparisons to search for an element in an array. The time complexity for linear search is O(n), while it is O(log n) for binary search.

166)What is the use of void pointers?

Void pointers are used because of their capability to store any pointer, which is pointing to a wide variety of data.

It is used to implement heterogeneous linked lists in many programming languages.

167)What are the minimum nodes binary trees can have?

Binary trees can have zero nodes or a minimum of 1 or 2 as well. It can be zero in a case where all of the nodes have a NULL value.

168)What Data Structures make use of pointers?

Pointers are used in a variety of data structures. They are majorly used in the following data structures: Binary trees,Linked lists,Queues,Stacks

169)What is the use of dynamic Data Structures?

Dynamic data structures provide users with a lot of flexibility in terms of the provision of data storage and manipulation techniques, which can change during the operation of the algorithm or the execution of the program.

170)Pointers allocate memory for data storage. True or False?

False, pointer operations such as declaration will not allocate any memory for the storage of data. But, memory is allocated for the variable that the pointer is pointing to. Memory processing begins only when the program begins its execution.

171)What is the difference between a File Structure and a Data Structure?

file structure-

-Data stored on disk
-Standard file storage policies
-Low compatibility with external apps

Data structure-

-Data stored on both RAM and disk
-Customized storage policies
-High compatibility with external apps

172)What are multi-dimensional arrays?

Multi-dimensional arrays are arrays that span across more than one dimension. This means that they will have more than one index variable for every point of storage. This is primarily used in cases where data cannot be represented or stored using only one dimension.

173)How is a variable stored in memory when using Data Structures?

A variable is stored based on the amount of memory that is needed. First, the required quantity of memory is assigned, and later, it is stored based on the data structure being used. Using concepts such as dynamic allocation ensures high efficiency and that the storage units can be supplied based on the requirements in real time.

174)Why should heap be used over a stack?

The heap data structure is more efficient to work with when compared to stack in a way that memory allocation in a head is dynamic and can be allocated and removed as per the requirement. The memory structure and access time of a stack are comparatively slow.

175)What is the meaning of Data Abstraction?

Data abstraction is one of the widely used tools in data structures. The goal is to break down complex entities into smaller problems and solve these by using the concepts of data structures. This provides users with the advantage of being focused on the operations and not worried about how the data is stored or represented in the memory.

176) What is the meaning of the stack overflow condition?

Stack overflow is the term given when the stack is full and an element cannot be inserted into the stack anymore.

Stack overflow happens when top = Maxsize – 1

177)What are the disadvantages of implementing queues using arrays?

There are two main downsides when implementing queues using arrays.

Array sizing: The queue has to be constantly extended to make way for more elements that get implemented.

Always extending the size of the array will not be feasible as there will be a discrepancy in the creation of the correct array size.

Memory dumps: The memory that is used to store the queue elements cannot be reused to actually store the queue. This is because of the working of queues where insertion happens at the head node only.

178)How can elements be inserted in the circular queue?

There are two cases in which items can be placed in a circular queue. They are as follows:

--When front != 0 and rear = max -1. This makes it possible as the queue will not be full, and new elements can be inserted here.

--When rear != max -1. This ensures that the rear is incremented to the maximum allocation size, and values can be inserted easily to the rear end of the queue.

179)Generics in Java-

Generics means parameterized types. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

180)Why Generics?

The Object is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature.
Generics in Java are similar to templates in C++.
For example, classes like HashSet, ArrayList, HashMap, etc., use generics

181)Types of Java Generics

Generic Method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

Generic Classes: A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma.

The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.

Generic Class

Like C++, we use <> to specify parameter types in generic class creation. To create objects of a generic class,
we use the following syntax.

```
// To create an instance of generic class
BaseType <Type> obj = new BaseType <Type>()
```

Note: In Parameter type we can not use primitives like 'int','char' or 'double'.

182)what is Generic Function?

We can also write generic functions that can be called with different types of arguments based on the type of arguments passed to the generic method. The compiler handles each method.

183)Advantages of Generics:

Programs that use Generics has got many benefits over non-generic code.

1. Code Reuse: We can write a method/class/interface once and use it for any type we want.

2. Type Safety: Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time). Suppose you want to create an ArrayList that store name of students, and if by mistake the programmer adds an integer object instead of a string, the compiler allows it. But, when we retrieve this data from ArrayList, it causes problemsat runtime.

```
// Java program to demonstrate that NOT using
// generics can cause run time exceptions
```



```
import java.util.*;

class Test
{
   public static void main(String[] args)
   {
      // Creating  an ArrayList without any type specified
      ArrayList al = new ArrayList();

      al.add("Sachin");
      al.add("Rahul");
      al.add(10); // Compiler allows this

      String s1 = (String)al.get(0);
      String s2 = (String)al.get(1);

      // Causes Runtime Exception
      String s3 = (String)al.get(2);
   }}
```

Output :

```
Exception in thread "main" java.lang.ClassCastException:
  java.lang.Integer cannot be cast to java.lang.String
   at Test.main(Test.java:19)
```

How do Generics Solve this Problem?
When defining ArrayList, we can specify that this list can take only String objects.

```
// Using Java Generics converts run time exceptions into
// compile time exception.
import java.util.*;

class Test
{
   public static void main(String[] args)
   {
      // Creating a an ArrayList with String specified
      ArrayList <String> al = new ArrayList<String> ();

      al.add("Sachin");
      al.add("Rahul");

      // Now Compiler doesn't allow this
      al.add(10);
```

```
        String s1 = (String)al.get(0);
        String s2 = (String)al.get(1);
        String s3 = (String)al.get(2);
    }
}
```

Output:

```
    15: error: no suitable method found for add(int)
    al.add(10);
     ^
```

3. Individual Type Casting is not needed: If we do not use generics, then, in the above example, every time we
4. retrieve data from ArrayList, we have to typecast it. Typecasting at every retrieval operation is a big headache
5. If we already know that our list only holds string data, we need not typecast it every time.

```
// We don't need to typecast individual members of ArrayList

import java.util.*;

class Test {
   public static void main(String[] args)
   {
      // Creating a an ArrayList with String specified
      ArrayList<String> al = new ArrayList<String>();

      al.add("Sachin");
      al.add("Rahul");

  // Typecasting is not needed
      String s1 = al.get(0);
      String s2 = al.get(1);
   }
}
```

6. Generics Promotes Code Reusability: With the help of generics in Java, we can write code that will work with different types of data. For example,

   public <T> void genericsMethod (T data) {...}
   Here, we have created a generics method. This same method can be used to perform operations on integer data,string data, and so on.

7. Implementing Generic Algorithms: By using generics, we can implement algorithms that work on

different types of objects, and at the same, they are type-safe too.

184)What is the Collection framework in Java?

Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

185)Explain various interfaces used in Collection framework?

Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework.

Collection interface:

1.Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.

Syntax:

public interface Collection<E>extends Iterable

Where <E> represents that this interface is of Generic type

2.List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

Syntax:

public interface List<E> extends Collection<E>

3.Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface

Syntax:

public interface Set<E> extends Collection<E>

4.Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).

Syntax:

public interface Queue<E> extends Collection<E>

5.Dequeue interface: it is a double-ended-queue. It allows the insertion and removal of elements from both ends.

. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.

Syntax:

public interface Dequeue<E> extends Queue<E>

6.Map interface: A Map (java.util.Map) represents a key, value pair storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

186) What is the difference between Set and Map?

- Set contains values only whereas Map contains key and values both.
- Set contains unique values whereas Map can contain unique Keys with duplicate values.
- Set holds a single number of null value whereas Map can include a single null key with n number of null values.

187)What is the difference between HashSet and HashMap?

- HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
- HashSet implements Set interface whereas HashMap implements the Map interface
- HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
- HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

188)What is the difference between Collection and Collections?

The Collection is an interface whereas Collections is a class.
The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

189)What is the advantage of the generic collection?

There are three main advantages of using the generic collection.

1. If we use the generic class, we don't need typecasting.
2. It is type-safe and checked at compile time.
3. Generic confirms the stability of the code by making it bug detectable at compile time.

190)What is hash-collision in Hashtable and how it is handled in Java?

Two different keys with the same hash value are known as hash-collision. Two separate entries will be kept in a single hash bucket to avoid the collision. There are two ways to avoid hash-collision.
-Separate Chaining
-Open Addressing

191) What is the default size of load factor in hashing based collection?

The default size of load factor is 0.75. The default capacity is computed as initial capacity * load factor. For example, 16 * 0.75 = 12. So, 12 is the default capacity of Map.

192) What is the difference between poll() and remove() method of Queue interface?

Though both poll() and remove() method from Queue is used to remove the object and returns the head of the queue, there is a subtle difference between them.
--If Queue is empty() then a call to remove() method will throw Exception,
--while a call to poll() method returns null. By the way, exactly which element is removed from the queue depends upon the queue's ordering policy and varies between different implementations,
for example, Priority Queue keeps the lowest element as per Comparator or Comparable at head position.

193) Difference between list and set.

-List is an ordered collection of elements and hence it maintains insertion order of elements while Set doesn't maintain any ordering of the elements.
-List allows duplicate elements while Set doesn't allow any elements. List is index-based, i.e.we can access elements in List based on Index, whereas we can't access Set using index.
-Concrete implementation of List Interface are ArrayList, LinkedList, etc.
Concrete implementation of Set implementations are HashSet, LinkedHashSet, TreeSet etc.
-we can insert any number of null values in List. But we can have only a single null value at most in Set.
-ListIterator can be used to traverse a List in both directions (forward and backward) however it cannot be used to traverse a Set.
-Using Iterator, we can traverse the elements in Set and List.

194)Difference between Iterator and ListIterator

-We can traverse the elements in a List using Iterator in a forward direction.

Using ListIterator, we can traverse the elements in the forward and backward direction both.

- Iterator can be used in these collection types: List, Set, and Queue.

ListIterator can be used in List collection only.

Iterator has the following functionalities:

- boolean hasNext()
- E next()
- void remove()

ListIterator has the following functionalities:

- void add (E e)
- boolean hasNext()
- boolean hasPrevious()
- void remove()

-Iterator can only perform remove operation while traversing the elements in a collection. If we try to add elements,it will throw ConcurrentModificationException.
ListIterator can perform add, remove operation while traversing the elements in a collection. We won't get any exception while adding element using ListIterator.

195)What is the difference between Comparable and Comparator?

The difference between Comparable and Comparator is:

Comparable	Comparator
Comparable provides compareTo() method to sort elements in Java.	Comparator provides compare() method to sort elements in Java.
Comparable interface is present in java.lang package.	Comparator interface is present in java.util package.
The logic of sorting must be in the same class whose object you are going to sort.	The logic of sorting should be in a separate class to write different sorting based on different attributes of objects.
The class whose objects you want to sort must implement the comparable interface.	Class, whose objects you want to sort, do not need to implement a comparator interface.
It provides single sorting sequences.	It provides multiple sorting sequences.
This method can sort the data according to the natural sorting order.	This method sorts the data according to the customized sorting order.
It affects the original class. i.e., the actual class is altered.	It doesn't affect the original class, i.e., the actual class is not altered.
Implemented frequently in the API by Calendar, Wrapper classes, Date, and String.	It is implemented to sort instances of third-party classes.
All wrapper classes and String class implement the comparable interface.	The only implemented classes of Comparator are Collator and RuleBasedColator.

196)Explain linked list supported by Java

Two types of linked list supported by Java are:

- **Singly Linked list:** Singly Linked list is a type of data structure. In a singly linked list, each node in the list stores the contents of the node and a reference or pointer to the next node in the list. It does not store any reference or pointer to the previous node.
- **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.

197)What is the difference between failfast and failsafe?

Failfast	Failsafe
It does not allow collection modification while iterating.	It allows collection modification while iterating.
It can throw ConcurrentModificationException	It can't throw any exception.
It uses the original collection to traverse the elements.	It uses an original collection copy to traverse the elements.
There is no requirement of extra memory.	There is a requirement of extra memory

198) Explain the methods provided by the Queue interface?

Methods of Java Queue interface are:

Method	Description
boolean add(object)	Inserts specified element into the Queue. It returns true in case it a success.
boolean offer(object)	This method is used to insert the element into the Queue.

Object remove()	It retrieves and removes the queue head.
Object poll()	(): It retrieves and removes queue head or return null in case if it is empty.
Object poll()	It retrieves and removes queue head or return null in case if it is empty.
Object element()	Retrieves the data from the Queue, but does not remove its head.
Object peek()	Retrieves the data from the Queue but does not remove its head, or in case, if the Queue is the Queue is empty, it will retrieve null.

199)What are the two ways to remove duplicates from ArrayList?

Two ways to remove duplicates from ArrayList are:

- **HashSet:** Developer can use HashSet to remove the duplicate element from the ArrayList. The drawback is it cannot preserve the insertion order.
- **LinkedHashSet:** Developers can also maintain the order of insertion by using LinkedHashSet instead of HashSet.

200)Name the collection classes that gives random element access to its elements

Collection classes that give random element access to its elements are:

- ArrayList
- HashMap,
- TreeMap
- Hashtable.

----------Set IV----------

201) Why Create Data Structures?

Data structures serve a number of important functions in a program. They ensure that each line of code performs its function correctly and efficiently, they help the programmer identify and fix problems with his/her code, and they help to create a clear and organized code base.

202) 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.

203) Elaborate on different types of array data structure

There are several different types of arrays:

One-dimensional array: A one-dimensional array stores its elements in contiguous memory locations, accessing them using a single index value. It is a linear data structure holding all the elements in a sequence.

- **Two-dimensional array:** A two-dimensional array is a tabular array that includes rows and columns and stores data. An M × N two-dimensional array is created by grouping M rows and N columns into N columns and rows.
- **Three-dimensional array:** A three-dimensional array is a grid that has rows, columns, and depth as a third dimension. It comprises a cube with rows, columns, and depth as a third dimension. The three-dimensional array has three subscripts for a position in a particular row, column, and depth. Depth (dimension or layer) is the first index, row index is the second index, and column index is the third index.

204)How Can You Store the Elements of a 2D Array in the Memory?

There are two techniques used to store 2D arrays:

- Row-major ordering

All of the rows of the array are stored contiguously in this technique. You start with the first row, then move on to the second, and so forth, until the entire array has been stored in the memory.

- Column-major ordering

Under this technique, the columns of the array are stored contiguously instead of the rows. You start with the first column and move on to the subsequent ones sequentially.

205)Why Is Algorithm Analysis Important?
Algorithm analysis is the process of assessing the computational capabilities of a particular algorithm and determining whether it can serve a particular use case. This is important to do before any programming project because it prevents unforeseen challenges and gives programmers an idea of what they can achieve with a particular algorithm.

206)What Is Topological Sorting in a Graph?
Topological sorting is a way of ordering the vertices in a graph. The vertices are ordered in such a way that if there is an edge from vertex v1 to vertex v2, then v1 is placed before v2 in the topological sorting order.

207)Can Dynamic Memory Allocation Help in Managing Data?
Yes, dynamic memory allocation can help in managing data. It is easier to work with data when it can be assigned to memory locations dynamically.

208)What Is a Jagged Array?
A jagged array is an array of arrays, which means that it is an array that contains other arrays. The arrays within a jagged array can be of varying lengths.

209)What are some examples of divide and conquer algorithms?

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

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

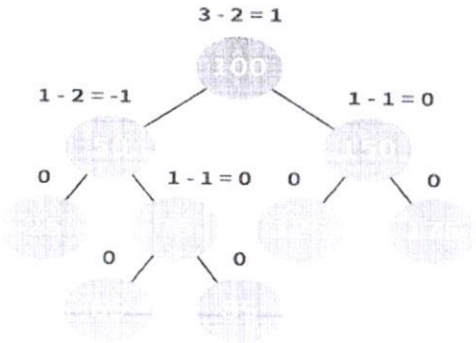
210) When is a binary search best applied?

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.

211) How can you reference all elements in a one-dimensional array?

You can access all of a one-dimensional array's elements via an indexed loop. A counter or timer counts down to 0 from the highest array size (n-1). You can use the loop counter as an array subscript and reference all the items in succession.

212) What is AVL tree data structure, its operations, and its rotations? What are the applications for AVL trees?
AVL trees are height balancing binary search trees named after their inventors Adelson, Velski, and Landis. The AVL tree compares the heights of the left and right subtrees and ensures that the difference is less than one. This distinction is known as the Balance Factor.
BalanceFactor = height(left-subtree) − height(right-subtree)



We can perform the following two operations on AVL tree:

- **Insertion:** Insertion in an AVL tree is done in the same way that it is done in a binary search tree. However, it may cause a violation in the AVL tree property, requiring the tree to be balanced. Rotations can be used to balance the tree.
- **Deletion:** Deletion can also be performed in the same manner as in a binary search tree. Because deletion can disrupt the tree's balance, various types of rotations are used to rebalance it.

An AVL tree can balance itself by performing the four rotations listed below:

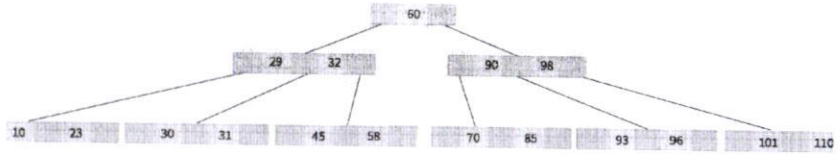
- **Left rotation:** When a node is inserted into the right subtree of the right subtree and the tree becomes unbalanced, we perform a single left rotation.
- **Right rotation:** If a node is inserted in the left subtree of the left subtree, the AVL tree may become unbalanced. The tree then requires right rotation.
- **Left-Right rotation:** The RR rotation is performed first on the subtree, followed by the LL rotation on the entire tree.
- **Right-Left rotation:** The LL rotation is performed first on the subtree, followed by the RR rotation on the entire tree.

Following are some real-time applications for AVL tree data structure:

- AVL trees are typically used for in-memory sets and dictionaries.
- AVL trees are also widely used in database applications where there are fewer insertions and deletions but frequent data lookups are required.
- Apart from database applications, it is used in applications that require improved searching.

213) what is a B-tree data structure? What are the applications for B-trees?

The B Tree is a type of m-way tree that is commonly used for disc access. A B-Tree with order m can only have m-1 keys and m children. One of the primary reasons for using a B tree is its ability to store a large number of keys in a single node as well as large key values while keeping the tree's height relatively small. A B-tree of order 4 is shown below in the image:



Following are the key properties of a B-tree data structure:

- All of the leaves are at the same height.
- The term minimum degree't' describes a B-Tree. The value of t is determined by the size of the disc block.
- Except for root, every node must have at least t-1 keys. The root must contain at least one key.
- All nodes (including root) can have no more than 2*t - 1 keys.
- The number of children of a node is equal to its key count plus one.
- A node's keys are sorted in ascending order. The child of two keys k1 and k2 contains all keys between k1 and k2.
- In contrast to Binary Search Tree, B-Tree grows and shrinks from the root.

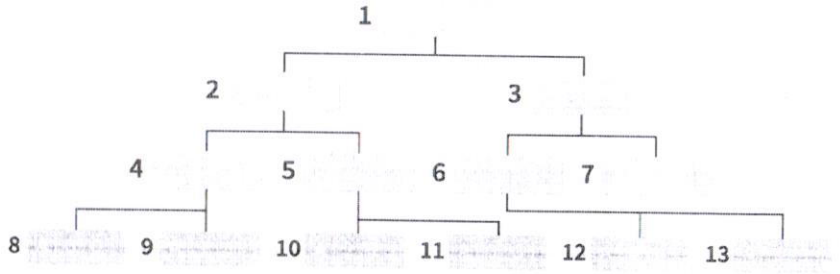
Following are real-time applications of a B-Tree data structure:

- It is used to access data stored on discs in large databases.
- Using a B tree, you can search for data in a data set in significantly less time.
- The indexing feature allows for multilevel indexing.
- The B-tree approach is also used by the majority of servers.

214) Define Segment Tree data structure and its applications.

A segment Tree is a binary tree that is used to store intervals or segments. The Segment Tree is made up of nodes that represent intervals. Segment Tree is used when there are multiple range queries on an array and changes to array elements.

The segment tree of array A[7] will look like this:



Segment Tree

Following are key operations performed on the Segment tree data structure:

- Building Tree: In this step, we create the structure and initialize the segment tree variable.
- Updating the Tree: In this step, we change the tree by updating the array value at a point or over an interval.
- Querying Tree: This operation can be used to run a range query on the array.

Following are real-time applications for Segment Tree:

- Used to efficiently list all pairs of intersecting rectangles from a list of rectangles in the plane.
- The segment tree has become popular for use in pattern recognition and image processing.
- Finding range sum/product, range max/min, prefix sum/product, etc
- Computational geometry
- Geographic information systems
- Static and Dynamic RMQ (Range Minimum Query)
- Storing segments in an arbitrary manner

215) Define Trie data structure and its applications

The word "Trie" is an abbreviation for "retrieval." Trie is a data structure that stores a set of strings as a sorted tree. Each node has the same number of pointers as the number of alphabet characters. It can look up a word in the dictionary by using its prefix. Assuming that all strings are formed from the letters 'a' to 'z' in the English alphabet, each trie node can have a maximum of 26 points.

Trie is also referred to as the digital tree or the prefix tree. The key to which a node is connected is determined by its position in the Trie. Trie allows us to insert and find strings in O(L) time, where L is the length of a single word. This is clearly faster than BST. Because of how it is implemented, this is also faster than Hashing. There is no need to compute a hash function. There is no need to handle collisions (like we do in open addressing and separate chaining)

Another benefit of Trie is that we can easily print all words in alphabetical order, which is not easy with hashing. Trie can also perform prefix search (or auto-complete) efficiently.

The main disadvantage of tries is that they require a large amount of memory to store the strings. We have an excessive number of node pointers for each node

Following are some real-time applications for Trie data structure:

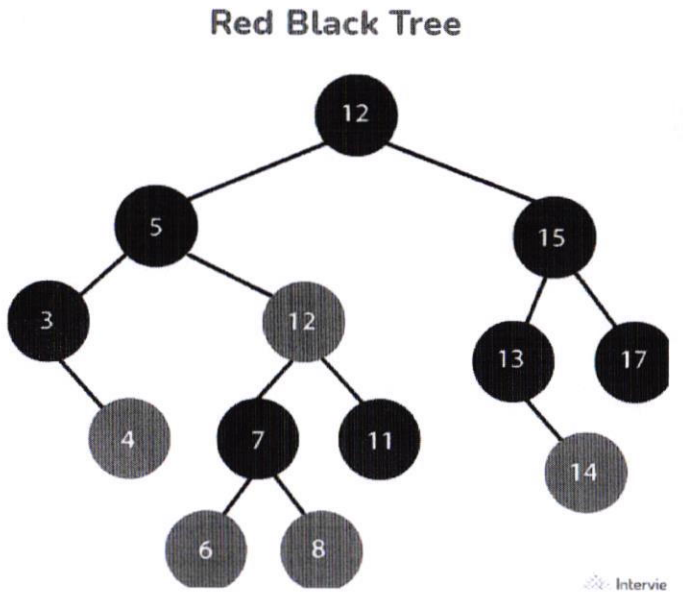
- Auto-Complete and Search for Search Engines
- Genome Analysis
- Data Analytics
- Browser History
- Spell Checker

216) Define Red-Black Tree and its applications

Red Black Trees are a type of self-balancing binary search tree. Rudolf Bayer invented it in 1972 and dubbed it "symmetric binary B-trees."

A red-black tree is a Binary tree in which each node has a colour attribute, either red or black. By comparing the node colours on any simple path from the root to a leaf, red-black trees ensure that no path is more than twice as long as any other, ensuring that the tree is generally balanced.

Red-black trees are similar to binary trees in that they both store their data in two's complementary binary formats. However, red-black trees have one important advantage over binary trees: they are faster to access. Because red-black trees are so fast to access, they are often used to store large amounts of data. Red-black trees can be used to store any type of data that can be represented as a set of values.



Every Red-Black Tree Obeys the Following Rules:

- Every node is either red or black.
- The tree's root is always black.
- There are no two red nodes that are adjacent.
- There is the same number of black nodes on every path from a node to any of its descendant's NULL nodes.
- All of the leaf nodes are black.

Following are some real-time applications for the Red-Black Tree data structure:

- The majority of self-balancing BST library functions in C++ or Java use Red-Black Trees.
- It is used to implement Linux CPU Scheduling.
- It is also used to reduce time complexity in the K-mean clustering algorithm in machine learning.
- MySQL also employs the Red-Black tree for table indexes in order to reduce searching and insertion time.

217) 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.

218) 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 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 the 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.

219)Sorting Techniques

Sorting : Arranging elements in a given order.

-Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical .

- Following are some of the examples of sorting in real-life scenarios:

Telephone Directory – The telephone directory stores the telephone numbers of people sorted by their names, so that the names can be searched easily.

Dictionary – The dictionary stores words in an alphabetical order so that searching of any word becomes easy.

In sorting Orders can be -

1. Ascending order : nth element <= (n+1)th element
2. Increasing order : nth element < (n+1)th element
3. Descending order : nth element >= (n+1)th element
4. Decreasing order : nth element > (n+1)th element

Sorting algorithms :

a) Comparison based : elements are compared to sort them

b) Non-comparison based : elements need not require comparison --- counting sort, shell sort.

Some of the popular sorting algorithms are :

1. Bubble sort
2. Selection sort
3. Insertion sort
4. Merge sort
5. Quick sort
6. Heap sort
7. Topological sort
8. Bucket sort
9. Radix sort
10. Shell sort
11. Counting sort

Best case for sorting algorithm : the given list of elements is already in required sorting order.
Worst case for sorting algorithm : the given list of elements is in reverse order of required order.
Average case for sorting algorithm : the given list of elements are in random order.

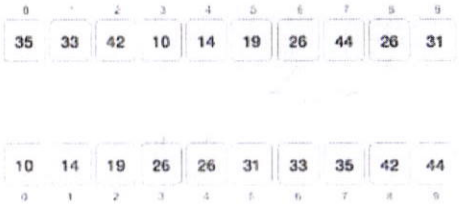
Types of Sorting-

In-place Sorting and Not-in-place(out-place) Sorting-

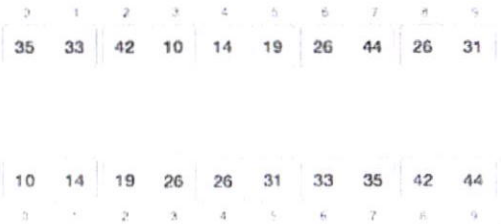
-Sorting algorithms may require some extra space for comparison and temporary storage of few data elements.

-Algorithms do not require any extra space and sorting is said to happen in-place, or for example,

within the array itself called as in-**place sorting.**
E.g. Bubble sort
-In some sorting algorithms, the program requires space which is more than or equal to the elements being sorted. Sorting which uses equal or more space is called **not in-place sorting.**
E.g. Merge-sort
Stable and Not Stable Sorting-
In a sorting algorithm, after sorting the contents, does not change the sequence of similar content in which they appear, it is called **stable sorting.**



If a sorting algorithm, after sorting the contents, changes the sequence of similar content in which they appear, it is called **unstable sorting**.



Stability of an algorithm matters when we wish to maintain the sequence of original elements, like in a tuple for example.

Adaptive and Non-Adaptive Sorting Algorithm
- sorting algorithm is said to be **adaptive,** if it takes advantage of already 'sorted' elements in the list that is to be sorted. That is, while sorting if the source list has some element already sorted, adaptive algorithms will take this into account and will try not to re-order them.
-A **non-adaptive algorithm** is one which does not take into account the elements which are already sorted. They try to force every single element to be re-ordered to confirm their sortedness.

220)Heap sort is a comparison-based sorting technique based on Binary Heap data structure.

Heap Sort Algorithm
To solve the problem follow the below idea:
First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
  - Swap the root element of the heap (which is the largest element) with the last element of the heap.
  - Remove the last element of the heap (which is now in the correct position).
  - Heapify the remaining elements of the heap.
- The sorted array is obtained by reversing the order of the elements in the input array.

Complexity Analysis of Heap Sort
Time Complexity: O(N log N)
Auxiliary Space: O(1)
Important points about Heap Sort:

- Heap sort is an in-place algorithm.
- Its typical implementation is not stable but can be made stable
- Typically 2-3 times slower than well-implemented QuickSort. The reason for slowness is a lack of locality of reference.