**Answers**

**1.ans** In simple words, data structure means if you have collection of data how you should organize it so that you can perform basic data manipulation, search, sort, etc. operations as quick as possible in others words effectively.

**2.ans** There are few implementations of data structure they are:

1. Arrays : Here data is allocated in continuous manner, so you can reach an element or insert an element at O(1). Size is fixed.

2. LinkedList: Here data is allocated randomly but size is unlimited until it uses all the memory of system.

3. Stack: Here data is stored in the form of LIFO or FILO. If you application data needs to be processed as above said then we can use Stack.

4. Queue: Here data is stored in the form of FIFO or LILO. If you application data needs to be processed as above said then we can use Stack.

5. Tree: Here data stored in hierarchal form to achieve quick traversing by storing the data in an organized way, there are many kind of trees they are like BST, AVL, MHeap.

6. Graph: Here data is stored in the form of nodes and vertices.

**3.ans**

**LinkedList > Array**

1. Size is not fixed
2. Quick Insertion and deletion
3. Less memory wastage as LinkedList is dynamic memory allocation.

**4.ans**

Struct Node

{

Int n;

Struct Node \*next;

};

Struct Node \*first = (struct Node\*)malloc(sizeof(struct Node));

5.ans Doubly LinkedList is useful when we want to traverse back and forth in linked list. Mainly at the time of deletion it is more useful.

6.ans

1. In stack data should be inserted only from one direction and deleted from the same direction
2. Push and pop are operations of stack
3. Need a top pointer and you can go only to top directly.
4. In Arrays data can be inserted from any where and deleted from any where
5. Insertion, deletion are operations of arrays
6. No top pointer, we can go to any index directly

7.ans If you use predefined queues from STL or utility package then we need two queues one for storing data and other for its priority. If we program queue we can add another variable in the node like priority with int type.

8.ans

There are mainly three traversal techniques they are

Pre-order, in-order, post-order.

Here the data is looks as shown below:

Pre-order : root left right;

In-order : left root right;

Post-order : left right root;

Another traversal is Level-order : here the data is printed level wise;

9.ans

The main difference between binary tree and binary search tree is binary tree is not organized and bst is organized, this means in bst searching, sorting, etc. operations are quick as compared to binary tree.

10.ans

Graph is a great ds to you when you have a data like locations, people, nodes, etc. because everything looks like a graph where people, locations are nodes connected to each other. Google maps, facebook, etc uses graphs to store the connections of locations and people.

11.ans

Yes it is possible but not efficient, because of non continuous memory and no indexing.

12.ans

A memory leak is a type of resource leak that occurs when a [computer program](https://en.wikipedia.org/wiki/Computer_program) incorrectly manages [memory allocations](https://en.wikipedia.org/wiki/Memory_allocation)in a way that memory which is no longer needed is not released. A memory leak may also happen when an [object](https://en.wikipedia.org/wiki/Object_(computer_science)) is stored in memory but cannot be accessed by the running code.

13.ans

We can check binary tree is a bst by the structure of the tree. Like if left node is less than its parent node and right node is greater than its parent node, and if this rule satisfies for every node in the tree then we can say that it is bst else binary tree.

14.ans

Stack is useful for recursion. In recursion the functions form a stack while they are in run time so that the last function can return its value to the second last function.

15.ans

Stacks can be used for expression evaluation.

* 1. Stacks can be used to check parenthesis matching in an expression.
  2. Stacks can be used for Conversion from one form of expression to another. Stacks can be used for Memory Management.
  3. Stack data structures are used in backtracking problems.

16.ans

Question is incomplete.

17.ans

Sorting stack using temp stack

1. Create Stack s and temp, int min;
2. Add elements in stack s;
3. Now for i=0 ; i< Stack s length ; i++
4. For j=i; j< Stack s length ; j++
5. Element = pop Stack s
6. Check the element with min and find min
7. Push the element into temp
8. Repeat step 4
9. Now push the min into Stack s
10. Push all the elements of temp into Stack s except min
11. Repeat step 3
12. Stack is sorted, end

18.ans

Program to reverse a queue

Queue q;

Stack s;

Int len=q1.length(), temp;

For(int i=0;i<q1.size();i++)

{

s.push(q.poll());

}

For(int i=0;i<s1.size();i++)

{

q.add(s.poll());

}

19.ans

Queue q, q1;

Stack s;

Int k;

For(int i=0;i<k;i++)

{

s.push(q.poll());

}

For(int i=0;i<s1.size();i++)

{

q1.add(s.poll());

}

For(int i=0;i<s1.size()-k;i++)

{

q1.add(q.poll());

}

q=q1;

20.ans

LinkedList l;

Int len,value;

Iterator I = l.iterator();

While(I.hasNext())

{

Len++;

I.next();

}

I = l.iterator();

For(int i=0;i< len-n ;i++)

{

Value = I.next();

}

Return value;

21.ans

    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;

    }

22.ans

class Demo{

      static void changeArr(int[] input)

    {

        int newArray[] = Arrays.copyOfRange(input,  0,   input.length);

        Arrays.sort(newArray);

        int i;

        Map<Integer, Integer> ranks

            = new HashMap<>();

        int rank = 1;

        for (int index = 0;index < newArray.length; index++)

{

            int element = newArray[index];

              if (ranks.get(element) == null)

{

                ranks.put(element, rank);

                rank++;

            }

       }

        for (int index = 0; index < input.length; index++)

{

            int element = input[index];

            input[index]  = ranks.get(input[index]);

        }

    }

23.ans

int V;

LinkedList<Integer> adj[];

    Boolean isCyclic(int v, Boolean visited[], int parent)

    {

        visited[v] = true;

        Integer i;

        Iterator<Integer> it = adj[v].iterator();

        while (it.hasNext())

        {

            i = it.next();

            if (!visited[i])

            {

                if (isCyclicUtil(i, visited, v))

                    return true;

            }

            else if (i != parent)

               return true;

        }

        return false;

    }

Boolean isTree()

    {

        Boolean visited[] = new Boolean[V];

        for (int i = 0; i < V; i++)

            visited[i] = false;

        if (isCyclicUtil(0, visited, -1))

            return false;

        for (int u = 0; u < V; u++)

            if (!visited[u])

                return false;

        return true;

    }

24.ans

   public static int kthSmallest(Integer[] arr, int k)

    {

        Arrays.sort(arr);

        return arr[k - 1];

    }

25.ans

V=vertices and E=edges;

One solution is to solve in O(VE) time using [Bellman–Ford](https://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/). If there are no negative weight cycles, then we can solve in O(E + VLogV) time using [Dijkstra’s algorithm](https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/).