Answers

Q1->Data Structure is collection of data values the relationships among them, and the functions or operations that can be applied to the data.. It is a data organization, management, and storage format that enables efficient access and modification .

Q2->Some of the data structures and its applications are as follows:

1. Arrays : Arrays are the simplest data structures that stores items of the same data type in a continuous memory location.

Its application are as follows:

i)Arrangement of leader-board of a game can be done simply through arrays to store the score and arrange them in descending order to clearly make out the rank of each player in the game.

ii)A simple question Paper is an array of numbered questions with each of them assigned to some marks.

2. Linked List : A linked list is a sequence data structure, which connects elements, called nodes, through links.

Its applications are as follows :

i)Web pages can be accessed using the previous and the next URL links which are linked using linked list.

ii)The music players also use the same technique to switch between music.

3. Stacks : A stack is a data structure which uses LIFO order.

Its applications are as follows :

i)Converting infix to postfix expressions.

ii)Undo operation is also carried out through stacks.

4. Queue : A queue is a data structure which uses FIFO order.

Its applications are as follows :

i)Operating System uses queue for job scheduling.

ii)To handle congestion in networking queue can be used.

Q3->Advantages of linked lists over arrays are as follows:

1)Less Memory wastage due to dynamic memory allocation: The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the

allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, upper limit is rarely reached.

2)Insertion and deletion is easy in linked list : Insertion/deletion of an element at beginning in a linked list is O(1) operation while in array it is O(n).

Q4->struct node{

int data;

struct node *next;

};

Q5->Each node of a doubly linked lists have two pointers which contains the address of both previous and next node. Due to this it has its own usages over singly linked list. Its usages are as follows:

1) It is can be used to implement stacks as well as heaps and binary trees whereas Singly linked list are generally used for implementation of stacks.

2) It allows two way traversal.

Q6->Differences between arrays and stack are as follows:

1) In stack the element can be entered and removed only from top because it follows LIFO policy, whereas in arrays elements can be inserted or removed from any point in the array.

2) In a stack only the topmost element's value can be viewed at a time whereas in arrays there is no such bound.

Q7->Minimum 2 queues are needed to implement a priority queue. One is used for storing data while the other is used for storing priorities.

Q8->Different types of tree traversal are as follows :

1)Inorder Traversal : Inorder Traversal is the one the most used variant of DFS(Depth First Search) Traversal of the tree.In this method the left subtree of a node is visited first, then the node is visited and then the right subtree is visited.These steps are repeated recursively until all the nodes are visited.

2)Preorder Traversal : Preorder Traversal is another variant of DFS. In this method the node is visited first, then the left subtree of the node is visited and then the right subtree of the node is visited.These steps are repeated recursively until all the nodes are visited.

3) Postorder Traversal: It is opposite of preorder. In this method the left subtree of node is visited first, then the right subtree of the node is visited and then node is visited.These steps are repeated recursively until all the nodes are visited.

4) Level Order Traversal : Level order traversal follows BFS(Breadth-First Search) to visit/modify every node of the tree. In this method we will visit all the nodes present at the same level one-by-one from left to right and then move to the next level to visit all the nodes of that level.

Q9->

The applications of graph data structures are as follows:

1) Google maps uses graphs for building transportation systems, where intersection of two(or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.

2) In Facebook, users are considered to be the vertices and if they are friends then there is an edge running between them. Facebook's Friend suggestion algorithm uses graph theory. Facebook is an example of undirected graph.

3) In Computer science graphs are used to represent the flow of computation

Q10->Yes, Binary search is possible on the linked list if the list is sorted and you know the count of elements in list.

Q1111->

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

12->

A memory leak is a type of resource leak that occurs when a computer program incorrectly manages memory allocations in a way that memory which is no longer needed is not released. A memory leak may also happen when an object is stored in memory but cannot be accessed by the running code.

13->

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->

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->

Stacks can be used for expression evaluation.

a. Stacks can be used to check parenthesis matching in an expression.

b. Stacks can be used for Conversion from one form of expression to another. Stacks can be used for Memory Management.

c. Stack data structures are used in backtracking problems.

16->

Question is incomplete.

17->

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->

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->

```
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->

```
        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->

```
        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->
   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->

```
  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->

```
  public static int kthSmallest(Integer[] arr, int k)
  {
      Arrays.sort(arr);

      return arr[k - 1];
  }
```

25->

V=vertices and E=edges;

One solution is to solve in O(VE) time using Bellman–Ford. If there are no negative weight cycles, then we can solve in O(E + VLogV) time using Dijkstra's algorithm.