1. **What do you mean by a Data structure?**

Data structure is a way of organizing the data in such a way that we can access the data very efficiently in terms of time, memory. There are different data structures like array, linkedlist, stack, queue, tree, graph and all these data structures have different uses but there main purpose is to reduce complexity and increase efficiency.

1. **What are some of the applications of DS?**

Arrays,Linked List,Queue,Stack,Tree,Graph.

1. **What are the advantages of a Linked list over an array?**

Both these data structures can store data in contiguous manner. Linked List is dynamic in nature whereas Arrays are not dynamic in nature. While creating an array we should know about the size of array in advance and only that much sized array will be created not more than that size but in case of linked list we can create a list of any sized . Insertion and deletion in arrays are costly operation whereas in case of linkedlist we can easily insert or delete an item

1. **Write the syntax in C to create a node in the singly linked list.**

#include<stdio.h>

#include<stdlib.h>

Struct node

{

Int data;

Struct node \*next;

}

1. **What is the use of a doubly-linked list when compared to that of a singly linked list?**

In singly linked we can traverse only in forward direction but in case of doubly linked list we cn traverse both in forward as well as backward direction.

1. **What is the difference between an Array and Stack?**

Both are linear data structure. In case of stack it is First in first out where the most recently added item should be deleted first i.e top of the stack and when we want to insert at that time also we have to insert at the top of the stack. We can’t insert item at the specified position wheareas in case of array we can insert , delete any item from any index.

1. **What are the minimum number of Queues needed to implement the priority queue?**

2 queues

1. **What are the different types of traversal techniques in a tree?**
2. Preorder traversal-> Root –left- right
3. Inorder traversal ->Left-root-right
4. Postorder traversal-> Left-right-root
5. Level order traversal-> Prints level wise node

Preorder,inorder and postorder are used BFS(Breadth first search) and level order traversal used DFS(Depth first search)

1. **Why it is said that searching a node in a binary search tree is efficient than that of a simple binary tree**?]

In case of binary tree we have to traverse every node in order to search for a particular node and its time complexity will become O(n) but in case of binary search tree all nodes are placed in a tree in such a way that nodes on the left subtree are less than root node and nodes on right subtree are greater than root node. So when we search any node we compare the node with root node and if data < root.data then we only search in left subtree and now overall complexity becomes O(logn).

1. **Can we apply Binary search algorithm to a sorted Linked list?**

Yes, Binary search is possible on the linked list if the list is ordered and you know the count of elements in list. But While sorting the list, you can access a single element at a time through a pointer to that node i.e. either a previous node or next node.

1. **When can you tell that a Memory Leak will occur?**

In C/C++, programmer is responsible for both creation and destruction of objects. Usually programmer neglects destruction of useless objects. Due to this negligence, at certain point, for creation of new objects, sufficient memory may not be available and entire program will terminate abnormally causing **OutOfMemoryErrors**.

1. **How will you check if a given Binary Tree is a Binary Search Tree or not?**

If all left subtree is less than root node and all right subtree is greater than root node then we can say a tree is binary search tree

1. **Which data structure is ideal to perform recursion operation and why?**

We use stack data structure to perform recursion. In recursion we break a big problem into small pieces and then first solve those small pieces and after that we use the result of that small problem in order to find a solution for big problem and for that we have to maintain a stack of all the sub problems so that we can use them later in the same sequence in which they are break.

1. **What are some of the most important applications of a Stack?**

**In recursion we internally use stack,used to convert infix to postfix expressions, checking for balanced paranthsis , used in backtracking problems**

1. **Sorting a stack using a temporary stack**

public static void sort(Stack<Integer> st)  
{  
 Stack<Integer> temprarySt=new Stack<>();  
 while (!st.isEmpty())  
 {  
 int temp=st.pop();  
 while (!temprarySt.isEmpty()&&temprarySt.peek()>temp)  
 {  
 st.push(temprarySt.pop());  
 }  
 temprarySt.push(temp);  
 }  
  
 while (!temprarySt.isEmpty())  
 {  
 System.*out*.println(temprarySt.pop());  
 }  
}

1. **Program to reverse a queue**

public static void reverseQueue(Queue<Integer> qu)  
{  
 Stack<Integer> st=new Stack<>();  
 while (!qu.isEmpty())  
 {  
 st.push(qu.poll());  
 }  
 while (!st.isEmpty())  
 {  
 qu.add(st.pop());  
 }  
}

1. **Reverse first k elements of the queue**

public static void revereFirstKElements(Queue<Integer> qu,int k)  
{  
 Stack<Integer> st=new Stack<>();  
 if(k>qu.size())  
 {  
 return;  
 }  
 for (int i=0;i<k;i++)  
 {  
 st.push(qu.poll());  
 }  
 while (!st.isEmpty())  
 {  
 qu.add(st.pop());  
 }  
 for (int i=0;i<qu.size()-k;i++)  
 {  
 int item=qu.peek();  
 qu.poll();  
 qu.add(item);  
 }  
   
}

1. **Program to return the nth node from the end in a linked list**

public int getLength(Node n,int k){  
 if(n == null){  
 return k;  
 }  
 return getLength(n.next, k+1);  
}  
public int nthNode(Node n, int nth){  
 int getLengthFromStart = getLength(n,0) – nth;  
 int ret = 0;  
 Node current = n;  
 while(getLengthFromStart-- > 0){  
 ret = Node.data;  
 current = current.next;  
 }  
 return ret;  
}

1. **Reverse a linked list**

class Node{  
 Node next;  
 int i;  
 public Node(){  
 next = null;  
 }  
 public Node(int i){  
 this.i = i;  
 this.next = null;  
 }  
}  
public Node reverseNode(Node n){  
 Node prev = null;  
 Node current = n;  
 while(n != null){  
 Node revNode = new Node();  
 revNode.i = n.i;  
 revNode.next = prev;  
 prev = revNode;  
 n = n.next;  
 }  
 return prev;}

1. **Replace each element of the array by its rank in the array**

public class rankArray  
{  
 public static void main(String[] args)  
 {  
 int[] ar={100,5,70,2};  
 *rank*(ar);  
 }  
 public static void rank(int[] ar)  
 {  
 ArrayList<Integer> list=new ArrayList<>();  
 for (int i=0;i<ar.length;i++)  
 {  
 list.add(ar[i]);  
 }  
 Collections.*sort*(list);  
 for(int i=0;i<ar.length;i++)  
 {  
 System.*out*.println(list.indexOf(ar[i])+1);  
 }  
 }  
}

1. **Check if a given graph is a tree or not**

To check if a given graph is a tree or not, we have to check if the graph contains a cycle. A graph is a tree if it doesn’t contain a cycle. Since it is not mentioned if it is a directed or un-directed graph, a particular cycle detection algorithm can’t be prescribed but techniques used to detect cycle are BFS or DFS cycle detection algorithm and Topological sorting

1. **Find out the Kth smallest element in an unsorted array**

public int kthSmallestElement(int[] ar , int k)  
{  
 Arrays.*sort*(ar);  
 return ar[k-1];  
}

1. **How to find the shortest path between two vertices**

Shortest path between two vertices is determined by using a shortest path algorithm called Dijkstra’s Algorithm. Dijkstra’s Algorithm is also called single source shortest path algorithm