**21)Reverse a linked list**

**import** java.util.Scanner;

**public** **class** DoubleNode {

Node<Integer> head;

Node<Integer> tail;

}

**class** Node<T>{

T data;

Node<T> next;

Node(T data)

{

**this**.data=data;

next=**null**;

}

}

**public** **class** ReverseLLbetter{

**static** Scanner *sc*=**new** Scanner(System.***in***);

**public** **static** Node<Integer> reverseLLR(Node<Integer> head)

{

**if**(head==**null** || head.next==**null**)

{

**return** head;

}

Node<Integer> reverseTail=head.next;

Node<Integer> smallHead=*reverseLLR*(head.next);

reverseTail.next=head;

head.next=**null**;

**return** smallHead;

}

**public** **static** DoubleNode reverseLLbetter(Node<Integer> head)

{

**if**(head==**null** || head.next==**null**)

{

DoubleNode ans = **new** DoubleNode();

ans.head=head;

ans.tail=head;

**return** ans;

}

DoubleNode smallAns=*reverseLLbetter*(head.next);

smallAns.tail.next=head;

head.next=**null**;

DoubleNode ans=**new** DoubleNode();

ans.head=smallAns.head;

ans.tail=head;

**return** ans;

}

**public** **static** **void** printList(Node<Integer> head)

{

**while**(head !=**null**)

{

System.***out***.print(head.data+" ");

head=head.next;

}

System.***out***.println();

}

**public** **static** Node<Integer> takeInt()

{

Node<Integer> head=**null**,tail=**null**;

**int** data=*sc*.nextInt();

**while**(data!=-1)

{

Node<Integer> newNode=**new** Node<>(data);

**if**(head==**null**)

{

head=newNode;

tail=newNode;

}

**else**

{

tail.next=newNode;

tail=tail.next;

}

data=*sc*.nextInt();

}

**return** head;

}

**public** **static** **void** main(String[] args){

Node<Integer> node=*takeInt*();

*printList*(*reverseLLR*(node));

}

}

**7) What are the minimum number of Queues needed to implement the priority queue?**

TWO(2)

**13) How will you check if a given Binary Tree is a Binary Search Tree or not?**

**class** BinaryTreeNode<T> {

**public** T data;

**public** BinaryTreeNode<T> left;

**public** BinaryTreeNode<T> right;

BinaryTreeNode(T data)

{

**this**.data=data;

left=**null**;

right=**null**;

}

}

**public** **static** **boolean** isBST2(BinaryTreeNode<Integer> root,**int** min,**int** max)

{

**if**(root==**null**)

**return** **true**;

**if**(root.data < min || root.data > max)

{

**return** **false**;

}

**boolean** isLeftok=*isBST2*(root.left,min,root.data-1);

**boolean** isRightok=*isBST2*(root.right,root.data,max);

**return** isLeftok&&isRightok;

}

**2) What are some of the applications of DS?**

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

**4) Write the syntax in C to create a node in the singly linked list**

Struct Node

{

**int** data;

Node \*next;

};

**6) Find out the Kth smallest element in an unsorted array?**

**import** java.util.Arrays;

**public** **class** Question6

{

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

{

Arrays.*sort*(arr);

**return** arr[k - 1];

}

**public** **static** **void** main(String[] args)

{

**int** arr[] ={9,8,3,4,1,2,8,7,5,6,3};

**int** k = 4;

System.***out***.println((arr, k));

}

}

**3)What are the advantages of a Linked list over an array?**

Arrays have Fixed Size, before implementing we have to give the memory and insertion and deletion is difficult. but if talk about LL they have dynamic size ,no need to assign size before implementing moreover insertion and deletion is more easy as compared to Arrays

**5)What are the different types of traversal techniques in a tree?**

Inorder Traversal, Postorder Traversal, Preorder Traversal, Level Order Traversal.

**6) Sorting a stack using a temporary stack?**

**package** HourOfCode;

**import** java.util.\*;

**class** Day7

{

**public** **static** Stack<Integer> reverse(Stack<Integer> Original)

{

Stack<Integer> reverse = **new** Stack<Integer>();

**while**(!Original.isEmpty())

{

**int** temp = Original.pop();

**while**(!reverse.isEmpty() && reverse.peek() > temp)

{

Original.push(reverse.pop());

}

reverse.push(temp);

}

**return** reverse;

}

**public** **static** **void** main(String args[])

{

Stack<Integer> Original = **new** Stack<Integer>();

Original.add(1);

Original.add(2);

Original.add(3);

Original.add(4);

Original.add(5);

Original.add(6);

Stack<Integer> reverse = *reverse*(Original);

System.***out***.println("Sorted numbers are:");

**while** (!reverse.empty())

{

System.***out***.print(reverse.pop()+" ");

}

}

}

**18)Program to reverse a queue ?**

**package** HourOfCode;

**import** java.util.\*;

**class** Day8

{

**public** **static** Stack<Integer> reverse(Queue<Integer> Original)

{

Stack<Integer> reverse = **new** Stack<Integer>();

**while**(!Original.isEmpty())

{

**int** temp = Original.remove();

**while**(!reverse.isEmpty() && reverse.peek() > temp)

{

Original.add(reverse.pop());

}

reverse.push(temp);

}

**return** reverse;

}

**public** **static** **void** main(String args[])

{

Queue<Integer> Original = **new** LinkedList<>();

Original.add(1);

Original.add(2);

Original.add(3);

Original.add(4);

Original.add(5);

Original.add(6);

Original.add(7);

Stack<Integer> reverse = *reverse*(Original);

System.***out***.println("Sorted numbers are:");

**while** (!reverse.empty())

{

System.***out***.print(reverse.pop()+" ");

}

}

}

**4)What is the use of a doubly-linked list when compared to that of a singly**

**linked list?**

The complexity of insertion and deletion is O(n) but in doubly LL is O(1).

In singly LL it can traverse only through one direction but in doubly we can traverse in two direction.

 singly LL store pointer of only one node so consumes lesser memory, on other hand Doubly LL uses more memory per node(two pointers).

**9)Why it is said that searching a node in a binary search tree is efficient than that of**

**a simple binary tree?**

Binary Search Tree allows for fast retrieval of elements stored in the tree as each node key is thoroughly compared with the [root](http://www.differencebetween.net/science/nature/difference-between-root-and-stem/) node, which discards half of the tree. It is already sorted as all element in left tree is smaller than root element and all element in right tree is greater than root element.  
  
**10) What are the applications of Graph DS?**

Social networks(Facebook), Google map, WWW(World Wide Web, biological networks, product recommendation graphs, neural networks, road networks, blockchains, and bitcoin transaction graphs.

**11) Can we apply Binary search algorithm to a sorted Linked list?**

Yes, we can implement in in java language with the use of Array List.

**12) When can you tell that a Memory Leak will occur?**

Memory leak occurs when programmers create a memory in heap and forget to delete it.