**MMU\_KEKA Technical Mock Test:**

**ANSWERS:**

1. Height
2. H=o(logn)
3. Option d
4. Inorder sequence gives decreasing order of elements.
5. Preorder traversal
6. Option d
7. O(m+n)
8. Underflow
9. -18
10. Stack
11. 2
12. FIFO(first in first out)principle
13. Dequeue
14. Find and return the position of the given element in the list
15. Return the element at the tail of the list and remove it from the list
16. Print success if a particular element is equal to 1.
17. O(n+m)time,o(1)space
18. Queue
19. Collections.sort(listobj);
20. Test -10

**CODING:**

**class LinkedList**

**{**

**Node head;**

**class Node**

**{**

**int data;**

**Node next;**

**Node(int d) {data = d; next = null; }**

**}**

**Node reverse(Node head, int k)**

**{**

**Node current = head;**

**Node next = null;**

**Node prev = null;**

**int count = 0;**

**while (count < k && current != null)**

**{**

**next = current.next;**

**current.next = prev;**

**prev = current;**

**current = next;**

**count++;**

**}**

**if (next != null)**

**head.next = reverse(next, k);**

**return prev;**

**}**

**public void push(int new\_data)**

**Node new\_node = new Node(new\_data);**

**new\_node.next = head;**

**head = new\_node;**

**}**

**void printList()**

**{**

**Node temp = head;**

**while (temp != null)**

**{**

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

**temp = temp.next;**

**}**

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

**}**

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

**{**

**LinkedList llist = new LinkedList();**

**llist.push(9);**

**llist.push(8);**

**llist.push(7);**

**llist.push(6);**

**llist.push(5);**

**llist.push(4);**

**llist.push(3);**

**llist.push(2);**

**llist.push(1);**

**System.out.println("Given Linked List");**

**llist.printList();**

**llist.head = llist.reverse(llist.head, 3);**

**System.out.println("Reversed list");**

**llist.printList();**

**}**

**}**

**3) class Node {**

**int data;**

**Node left, right;**

**public Node(int item)**

**{**

**data = item;**

**left = right = null;**

**}**

**}**

**class BinaryTree {**

**Node root;**

**static int max\_level = 0;**

**void leftViewUtil(Node node, int level)**

**{**

**if (node == null)**

**return;**

**if (max\_level < level) {**

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

**max\_level = level;**

**}**

**leftViewUtil(node.left, level + 1);**

**leftViewUtil(node.right, level + 1);**

**}**

**void leftView()**

**{**

**leftViewUtil(root, 1);**

**}**

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

**{**

**BinaryTree tree = new BinaryTree();**

**tree.root = new Node(12);**

**tree.root.left = new Node(10);**

**tree.root.right = new Node(30);**

**tree.root.right.left = new Node(25);**

**tree.root.right.right = new Node(40);**

**tree.leftView();**

**}**

**4)** **import java.util.\*;**

**class linkeList {**

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

**{**

**Node one = new Node(1);**

**Node two = new Node(2);**

**Node three = new Node(3);**

**Node four = new Node(4);**

**Node five = new Node(3);**

**Node six = new Node(2);**

**Node seven = new Node(1);**

**one.ptr = two;**

**two.ptr = three;**

**three.ptr = four;**

**four.ptr = five;**

**five.ptr = six;**

**six.ptr = seven;**

**boolean condition = isPalindrome(one);**

**System.out.println("isPalidrome :" + condition);**

**}**

**static boolean isPalindrome(Node head)**

**{**

**Node slow = head;**

**boolean ispalin = true;**

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

**while (slow != null) {**

**stack.push(slow.data);**

**slow = slow.ptr;**

**}**

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

**int i = stack.pop();**

**if (head.data == i) {**

**ispalin = true;**

**}**

**else {**

**ispalin = false;**

**break;**

**}**

**head = head.ptr;**

**}**

**return ispalin;**

**}**

**}**

**class Node {**

**int data;**

**Node ptr;**

**Node(int d)**

**{**

**ptr = null;**

**data = d;**

**}**

**}**