Q.1 Merge Two Sorted Linked Lists

Given heads of two sorted linked lists namely ‘head1’ & ‘head2’, merge them into one single sorted linked list & return the head of the new sorted list.

Stubbed Code -

```
class MergeTwoLists {
   public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
         // WRITE YOUR CODE HERE
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      int n1 = sc.nextInt();
      ListNode head1 = createList(sc, n1);
      int n2 = sc.nextInt();
      ListNode head2 = createList(sc, n2);
      ListNode mergedHead = mergeTwoLists(head1, head2);
      printList(mergedHead);
   }
}
```

Sample Input -

```
3
1 3 5
3
2 4 6
```

Sample Output -

```
1 2 3 4 5 6
```

Test Cases -

Input -

```
5
1 2 3 4 5
3
100 200 300
```

Output -

```
1 2 3 4 5 100 200 300
```

Input -

```
1
10
1
```

1
Output -
1 10

Input -
1
1
1
1
Output -
1 1

Q.2 Check for balanced parentheses
Given a string ‘str’, check if the string has balanced parentheses or not.

Stubbed Code -
```
import java.util.Scanner;
public class BalancedParanthesis {
   public static boolean isBalanced(String str) {
         // WRITE YOUR CODE HERE
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      String str = sc.nextLine();
      if(isBalanced(str)) System.out.println("Balanced");
      else System.out.print("Unbalanced");
   }
}
```

Sample Output -
[({})]
Sample Output -
Balanced

Test Cases -
Input -
(()())
Output -
Balanced

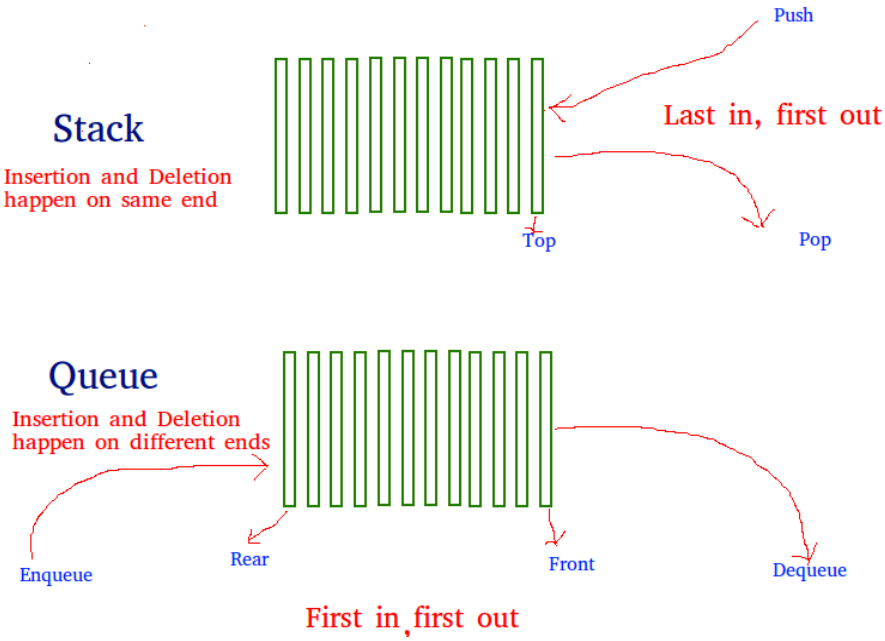
Input -
(()
Output -
Unbalanced

Input -
)(
Output -
Unbalanced

Q.3 Queue & it's operations

We are given a Queue data structure that supports standard operations like enqueue() and dequeue(). We need to implement a Stack data structure using only instances of Queue and queue operations allowed on the instances.

Example -



Stubbed Code -

```
import java.util.*;
class Main {
   static class Stack {
      static Queue<Integer> q1 = new LinkedList<>();
      static Queue<Integer> q2 = new LinkedList<>();
      static int curr_size;
      Stack() {
         curr_size = 0;
      }
      static int size() {
         return curr_size;
      }
      // WRITE YOUR CODE HERE
```

```
    }
    public static void main(String[] args) {
        Stack s = new Stack();
        Scanner sc = new Scanner(System.in);
        int n=sc.nextInt();
        for(int i=0; i<n; i++){
            int value = sc.nextInt();
            s.push(value);
        }
        System.out.println("Size - " + s.size());
        System.out.println("Top - " + s.top());
    }
}
```

Sample Input -
5
1 2 3 4 5
Sample Output -
Size - 5
Top - 5

Test Cases -
Input -
5
1 2 3 4 10
Output -
Size - 5
Top - 10

Input -
10
1 2 3 4 5 6 7 8 90 100
Output -
Size - 10
Top - 100

Input -
1
10
Output-
Size - 1
Top - 10

Q.4 Left Shift a List

Given a list of numbers, you have to delete a specific user requested & left shift the remaining elements.
Example, given a list, you have to delete the key ‘10’ from the list. The node from the list will be removed and the remaining list is to be printed.

Stubbed Code -

```
import java.util.Scanner;
public class LinkedList {
    Node head;
    static class Node {
        int data;
        Node next;
        Node(int data) {
            this.data = data;
            this.next = null;
        }
    }
    LinkedList() { this.head = null; }
    public void push(int data) {
            // WRITE YOUR CODE HERE
    }
    public void delete(int data) {
            // WRITE YOUR CODE HERE
    }
    public void display() {
        if(head == null) return;
        else {
            Node temp = head;
            while(temp != null) {
                System.out.print(temp.data + " ");
                temp = temp.next;
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        LinkedList ll = new LinkedList();
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        for(int i=0; i < n; i++) {
            int element = sc.nextInt();
            ll.push(element);
        }
        int k = sc.nextInt();
```

```
        ll.delete(k);
        ll.display();
    }
}
```

Sample Input -
5
1 2 3 4 5
3
Sample Output -
1 2 4 5

Test Cases -
Input -
10
74 74 273 271 38 47 33 2 1 184
9
Output -
74 74 273 271 38 47 33 2 1 184

Input -
1
10
10
Output -
LinkedList is Empty!

Input -
2
1 1
1
Output -
1

Q.5 'k' Reverse a LinkedList
Given a linked list of size N. The task is to reverse every k node (where k is an input to the function) in the linked list. If the number of nodes is not a multiple of k then left-out nodes, in the end, should be considered as a group and must be reversed

Stubbed Code -

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
```

```
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class Main {
    static class Pair {
        Node first, second;
        Pair(Node first, Node second) {
            this.first = first;
            this.second = second;
        }
    }
    static Node reverse(Node head, int k) {
            // WRITE YOUR CODE HERE
    }
    static void print(Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
    }
    static Node insertAtEnd(Node head, int x) {
        Node current = new Node(x);
        if (head == null) return current;
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = current;
        return head;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Node head = null;
        int n = sc.nextInt(), k = sc.nextInt();
        for (int i = 0; i < n; i++) {
            int x = sc.nextInt();
            head = insertAtEnd(head, x);
        }
        head = reverse(head, k);
        print(head);
    }
```

}

Sample Input -
8 4
1 2 3 4 5 6 7 8
Sample Output -
4 3 2 1 8 7 6 5

Test Cases -
Input -
8 3
1 2 3 4 5 6 7 8
Output -
3 2 1 6 5 4 8 7

Input -
8 5
1 2 3 4 5 6 7 8
Output -
5 4 3 2 1 8 7 6

Input -
10 2
63 354 364 21 -74 475 34 40 100 10
Output -
354 63 21 364 475 -74 40 34 2 100 10

Q.6 Order of a LinkedList - I
Given a singly linked list: A0→A1→…→An-1→An, reorder it to:
A0→An→A1→An-1→A2→An-2→… For example: Given 1->2->3->4->5 its reorder is 1->5->2->4->3.
Note: It is recommended to do this in-place.

Stubbed Code -
```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    Node(int x) {
        this.data = x;
        this.next = null;
    }
}
public class Main {
```

```
    static Node insertAtEnd(Node head, int x) {
        Node current = new Node(x);
        if (head == null) return current;
        Node tmp = head;
        while (tmp.next != null) tmp = tmp.next;
        tmp.next = current;
        return head;
    }
    static void reorderList(Node head) {
            // WRITE YOUR CODE HERE
    }
    static void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Node head = null;
        int n = sc.nextInt();
        while (n-- > 0) {
            int x = sc.nextInt();
            head = insertAtEnd(head, x);
        }
        reorderList(head);
        printList(head);
    }
}
```

Sample Input -
5
1 2 3 4 5
Sample Output -
1 5 2 4 3

Test Cases -
Input -
8
12 43 223 565 7876 1232 4567 10
Output -
12 10 43 4567 223 1232 565 7876

Input -

10
2 63 354 364 21 -74 475 34 40 100
Output -
2 100 63 40 354 34 364 475 21 -74

Input -
7
1 2 3 4 5 6 7
Output -
1 7 2 6 3 5 4

Q.7 Ordered of a LinkedList - II
Given a sorted doubly linked list and an element X, you need to insert the element X into the correct position in the sorted DLL.

Stubbed Code -

```
import java.util.Scanner;
class Node {
    int data;
    Node prev, next;
    Node(int data) {
        this.data = data;
        this.prev = this.next = null;
    }
}
public class Main {
    static void printList(Node head) {
        while (head != null) {
            System.out.print(head.data + " ");
            head = head.next;
        }
    }
    static boolean isChecked(Node head) {
        int lengthF = 0, lengthB = 0;
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
            lengthF++;
        }

        while (temp.prev != null) {
            temp = temp.prev;
            lengthB++;
        }
```

```
        return lengthF == lengthB;
    }
    static Node sortedInsert(Node head, int x) {
            // WRITE YOUR CODE HERE
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        Node head = null, tail = null;
        int x = sc.nextInt();
        head = new Node(x);
        tail = head;
        for (int i = 0; i < n - 1; i++) {
            x = sc.nextInt();
            Node temp = new Node(x);
            tail.next = temp;
            temp.prev = tail;
            tail = temp;
        }
        x = sc.nextInt();
        head = sortedInsert(head, x);
        if (isChecked(head)) {
            printList(head);
        } else {
            System.out.println("The node has not been inserted correctly. Please fix the prev and
next pointers");
        }
    }
}
```

Sample Input -

```
5
1 9 16 25 78
2
```

Sample Output -

```
1 2 9 16 25 78
```

Test Cases -

Input -

```
10
2 63 354 364 21 -74 475 34 40 100
-10
```

Output -

-10 2 63 354 364 21 -74 475 34 40 100

Input -
8
12 43 223 565 7876 1232 4567 10
200
Output -
12 43 200 223 565 7876 1232 4567 10

Input -
5
1 9 16 25 78
2
Output -
1 2 9 16 25 78

Q.8 Sort a LinkedList
Given a singly linked list, sort the list (in ascending order) using insertion sort algorithm.
Constraints -
1 <= N, M <= 105
1 <= x, y <= 106

Stubbed Code -

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    Node(int x) {
        data = x;
        next = null;
    }
}
public class LinkedListSorted {
    public static Node insertAtEnd(Node head, int x) {
        Node current = new Node(x);
        if (head == null) return current;
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = current;
        return head;
    }
    public static Node sortedInsert(Node head, Node newNode) {
```

```
        // WRITE YOUR CODE HERE
    }
    public static Node insertionSort(Node head) {
        Node sorted = null;
        Node current = head;
        while (current.next != null) {
            Node next = current.next;
            sorted = sortedInsert(sorted, current);
            current = next;
        }
        return sorted;
    }
    public static void printList(Node node) {
        while (node != null) {
            System.out.print(node.data + " ");
            node = node.next;
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Node head = null;
        int n = sc.nextInt();
        while (n-- > 0) {
            int x = sc.nextInt();
            head = insertAtEnd(head, x);
        }
        head = insertionSort(head);
        printList(head);
        sc.close();
    }
}
```

Sample Input -
5
10 20 5 1 4
Sample Output -
1 4 5 10 20

Test Cases -
Input -
5
2 1 3 4 5
Output -
1 2 3 4 5

Input -
10
2 63 354 364 21 -74 475 34 40 100
Output -
-74 2 21 34 40 63 100 354 364 475

Input -
8
12 43 223 565 7876 1232 4567 10
Output -
10 12 43 223 565 1232 4567 7876

Q.9 Palindrome List
Given a singly linked list of size N of integers. The task is to check if the given linked list is palindrome or not.

Stubbed Code -

```
import java.util.Scanner;
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
public class LinkedListPalindrome {
    public static Node insertAtEnd(Node head, int x) {
        Node current = new Node(x);
        if (head == null) return current;
        Node temp = head;
        while (temp != null) {
            temp = temp.next;
        }
        temp.next = current;
        return head;
    }
    public static boolean isPalindrome(Node head) {
            // WRITE YOUR CODE HERE
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
```

```
        Node head = null;
        for (int i = 0; i < n; i++) {
            int x = scanner.nextInt();
            head = insertAtEnd(head, x);
        }
        System.out.println(isPalindrome(head) ? 1 : 0);
        scanner.close();
    }
}
```

Sample Input -
8
1 2 3 4 4 3 2 1
Sample Output -
1

Test Cases -
Input -
7
1 2 3 4 3 2 2
Output -
0

Input -
8
10 20 30 40 40 30 20 10
Output -
1

Input -
7
1 2 3 4 3 2 1
Output -
1

Q.10 Sort a stack
Write a logical piece of code which sorts an unsorted stack into a sorted stack.

Stubbed Code -

```
import java.util.Iterator;
import java.util.Scanner;
import java.util.Stack;
public class SortStack {
    public static Stack<Integer> sortStack(Stack<Integer> input) {
```

```
        // WRITE YOUR CODE HERE
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Stack<Integer> stack = new Stack<>();
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) stack.push(sc.nextInt());
        Stack<Integer> sortedStack = sortStack(stack);
        Iterator<Integer> itr = sortedStack.iterator();
        while(itr.hasNext()) {
            int elt = itr.next();
            System.out.print(elt + " ");
        }
    }
}
```

Sample Input -
5
20 10 30 50 40
Sample Output -
10 20 30 40 50

Test Cases -
Input -
5
8000 8565 8050 1234 111
Output -
111 1234 8000 8050 8565

Input -
5
99999 89565 8050 1234999 111
Output -
111 8050 89565 99999 1234999

Input -
5
20 10 30 50 40
Output -
10 20 30 40 50

Q.11 Change the queue
Write a piece of code which reverse first k elements of a queue.

Stubbed Code -

```
import java.util.*;
import java.util.LinkedList;
public class ReverseFirstKElements {
    public static Queue<Integer> reverseFirstKElements(Queue<Integer> queue, int k) {
            // WRITE YOUR CODE HERE
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Queue<Integer> queue = new LinkedList<>();
        int n = sc.nextInt();
        int k = sc.nextInt();
        for (int i = 0; i < n; i++) {
            queue.offer(sc.nextInt());
        }
        queue = reverseFirstKElements(queue, k);
        while (!queue.isEmpty()) {
            System.out.print(queue.poll() + " ");
        }
    }
}
```

Sample Input -

```
6 4
1 2 3 4 5 6
```

Sample Output -

```
4 3 2 1 5 6
```

Test Cases -

Input -

```
5 0
1 2 3 4 5
```

Output -

```
1 2 3 4 5
```

Input -

```
4 4
10 20 30 40
```

Output -

```
40 30 20 10
```

Input -

```
3 5
1 2 3
```

Output -

1 2 3

Q.12 Reverse the LinkedList

Tag - LinkedList

The code below is supposed to reverse a singly linked list, but it contains a logical error.

Java Stubbed Code -

```
import java.util.Scanner;
class Node {
   int data;
   Node next;
   Node(int data) {
      this.data = data;
      this.next = null;
   }
}
public class ReverseLinkedList {
   Node head;
   ReverseLinkedList() { this.head = null; }
   public void insert(int data) {
      Node newNode = new Node(data);
      if(head == null) head = newNode;
      else {
         Node temp = head;
         while(temp.next != null) temp = temp.next;
         temp.next = newNode;
      }
   }
   public static Node reverse(Node head) {
         // WRITE YOUR CODE HERE
   }
   public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
         ReverseLinkedList list = new ReverseLinkedList();
         int n = sc.nextInt()
         for(int i=0; i < n; i++) {
                  int element = sc.nextInt();
                  list.insert(element);
         }
         Node reversed = reverse(head);
         while (reversed != null) {
                  System.out.print(reversed.data + " ");
                  reversed = reversed.next;
         }
```

```
    }
}
```

Sample Input -
4
1 2 3 4
Sample Output -
4 3 2 1

Test Cases -
Input -
1
10
Output -
10

Input -
2
100 200
Output -
200 100

Input -
5
1 3 5 6 2
Output -
2 6 5 3 1

Q.13 Maximum Nesting Depth of the Parenthesis
Given a valid parentheses string s, return the nesting depth of s. The nesting depth is the maximum number of nested parentheses.

Java Stubbed Code -

```
package com.company;
import java.util.Scanner;
class Solution {
    public static int maxDepth(String s) {
            // WRITE YOUR CODE HERE
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        System.out.println( maxDepth(s) );
    }
```

}
Sample Input -
(1+(2*3)+((8)/4))+1
Sample Output -
3

Test Cases -
Input -
(1)+((2))+(((3)))
Output -
3

Input -
()(())((()()))
Output -
3

Input -
()
Output -
1