NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

# ASSIGNMENT DAY - (1-6)

**Task 1: Linked List Middle Element Search**

**You are given a singly linked list. Write a function to find the middle element without using any extra space and only one traversal through the linked list.**

**Ans)**

**Code:-**

```
package com.wipro.linear; public
class LinkedListMiddle {

  private Node head;
  private int length;

  class Node {

     int value; Node next;
     public Node(int value)
     {

        this.value = value;

     }
  }
  public void addNode(int value) {

     Node newNode = new Node(value);
     if (head == null) {
        head = newNode;

     } else {
        Node current = head;

        while (current.next != null) {
           current = current.next;
        }
        current.next = newNode;
     }
     length++;
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
  }
  public Node findMiddle()
     { if (head == null) {
     return null;
     }
     Node slowPtr = head;
     Node fastPtr = head;
     while (fastPtr != null && fastPtr.next != null) {
        fastPtr = fastPtr.next.next;
        slowPtr = slowPtr.next;
     }
     return slowPtr;
  }
public static void main(String[] args) {
     LinkedListMiddle linkedList = new LinkedListMiddle();
     linkedList.addNode(14);
     linkedList.addNode(26); linkedList.addNode(33);
     linkedList.addNode(45);
     linkedList.addNode(58);


     Node middle = linkedList.findMiddle(); if
     (middle != null) {
        System.out.println("Middle element value: " +
middle.value);
     } else {
        System.out.println("The list is empty.");
     }
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
  }
}
```

**OUTPUT:**

```
Middle element value: 33
```

**Task 2: Queue Sorting with Limited Space**
**You have a queue of integers that you need to sort. You can only use additional space equivalent to one stack. Describe the steps you would take to sort the elements in the queue. Ans)**

**Code:**

```
package com.wipro.non.linear;

import java.util.LinkedList;

import java.util.Queue;

import java.util.Stack;
public class QueueSorting {

  public static void sortQueue(Queue<Integer> queue) {
    Stack<Integer> stack = new Stack<>();


    while (!queue.isEmpty()) {

      int temp = queue.poll(); while (!stack.isEmpty()
      && stack.peek() > temp) {


        queue.offer(stack.pop());
      }


      stack.push(temp);

    }
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
    while (!stack.isEmpty()) {

        queue.offer(stack.pop());
    }
    reverseQueue(queue);
```

```
  }

  private static void reverseQueue(Queue<Integer> queue)
{
    Stack<Integer> tempStack = new Stack<>();
    while (!queue.isEmpty()) {
        tempStack.push(queue.poll());


    }

    while (!tempStack.isEmpty()) {

        queue.offer(tempStack.pop());


    }


  }
  public static void main(String[] args) {
    Queue<Integer> queue = new LinkedList<>();
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        queue.offer(3);
        queue.offer(1);
        queue.offer(4);
        queue.offer(5);




        queue.offer(9);
        queue.offer(2);




        queue.offer(6); queue.offer(5);
        System.out.println("Queue before sorting: " + queue);

        sortQueue(queue);

        System.out.println("Queue after sorting: " + queue);
    }

}
```

**OUTPUT:-**

```
Queue before sorting: [3, 1, 4, 5, 9, 2, 6, 5]
Queue after sorting: [1, 2, 3, 4, 5, 5, 6, 9]
```

**Task 3: Stack Sorting In-Place**

**You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.**

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

**CODE:-**

```
package com.wipro.non.liniear;

import java.util.Stack; public class StackAss { public
static void sortStack(Stack<Integer> stack) {
Stack<Integer> tempStack = new Stack<>();


    while (!stack.isEmpty()) { int

        current = stack.pop();


        while (!tempStack.isEmpty() && tempStack.peek() >

current) { stack.push(tempStack.pop());


        }

        tempStack.push(current);


    }

    while (!tempStack.isEmpty()) {
        stack.push(tempStack.pop());
    }

}
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        stack.push(34);

        stack.push(3); stack.push(31);



        stack.push(98); stack.push(92);



        stack.push(23);


        System.out.println("Original stack: " + stack);
        sortStack(stack);

        System.out.println("Sorted stack from bottom to top: "
+ stack);
    }

}
```

**OUTPUT:**

```
Original stack: [34, 3, 31, 98, 92, 23]

Sorted stack from bottom to top: [98, 92, 34, 31, 23, 3]
```

**Task 4: Removing Duplicates from a Sorted Linked List A sorted linked list has been constructed with repeated elements. Describe an algorithm to remove all duplicates from the linked list efficiently. Ans)**

**CODE:-**

```
package com.wipro.non.liniear;
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
public class LinkedListDup
 { Node head; class Node {
 int data;
    Node next;



    Node(int d) {

       data = d;
       next = null;



    }

 }
```

```
 void removeDuplicates() { Node
    current = head; Node previous =
    head; while (current!= null) { if
    (current.data == previous.data) {









          previous.next = current.next;
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        } else { previous =
            current;


        }

        current = current.next;

    }

}

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) {
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        LinkedListDup list = new LinkedListDup(); list.head
        = list.new Node(1);


        list.head.next = list.new Node(2);


        list.head.next.next = list.new Node(2);


        list.head.next.next.next = list.new Node(3);


        list.head.next.next.next.next = list.new Node(3);
        list.head.next.next.next.next.next = list.new Node(3);
        list.head.next.next.next.next.next.next = list.new


Node(4); list.head.next.next.next.next.next.next.next =
        list.new


Node(5);   list.head.next.next.next.next.next.next.next.next
        =


list.new Node(5);
        System.out.println("Original Linked List:");
        list.printList();


        list.removeDuplicates();
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        System.out.println("Linked List after removing

duplicates:"); list.printList();


    }
}
```

**OUTPUT:-**

```
Original Linked List:
1 2 2 3 3 3 4 5 5

Linked List after removing duplicates:

1 2 3 4 5
```

**Task 5: Searching for a Sequence in a Stack**

**Given a stack and a smaller array representing a sequence, write a function that determines if the sequence is present in the stack. Consider the sequence present if, upon popping the elements, all elements of the array appear consecutively in the stack. ANS) CODE:-**

```
package com.wipro.linear;
import java.util.Stack; public
class SequenceInStack {
    public static boolean isSequenceInStack(Stack<Integer>

stack, int[] sequence) { if (sequence == null ||
        sequence.length == 0) {


            return true;


        }

        if (stack == null || stack.size() < sequence.length) {
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        return false;

    }

    Stack<Integer> tempStack = new Stack<>();

    boolean found = false;

    while (!stack.isEmpty()) {

            int element = stack.pop();

       tempStack.push(element); if
            (element == sequence[0]) {
          boolean match = true;

                for (int i = 1; i < sequence.length; i++) {
```

```
            if (stack.isEmpty() || stack.peek() != sequence[i])


{
                match = false;

                break;
            } else {
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
                tempStack.push(stack.pop());

            }
```

```
        }

        if (match) {

            found = true;
            break;


        } else {while (tempStack.size() > sequence.length)
{
                stack.push(tempStack.pop());

            }

        }

    }
    }
     while (!tempStack.isEmpty()) {

    stack.push(tempStack.pop());
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        }

        return found;

    }

    public static void main(String[] args) {

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


        stack.push(1); stack.push(2);



        stack.push(3);

        stack.push(4);

        stack.push(5); int[]
        sequence = {5, 4, 3};
         boolean result = isSequenceInStack(stack, sequence);

        System.out.println("Is sequence in stack: " + result);


    }

}
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

**OUTPUT:-**

```
Is sequence in stack: true
```

**Task 6: Merging Two Sorted Linked Lists You are provided with the heads of two sorted linked lists. The lists are sorted in ascending order. Create a merged linked list in ascending order from the two input lists without using any extra space (i.e., do not create any new nodes). Ans)**

```
package com.wipro.linear; public
class MergeSortedLinkedLists {


  static class Node {int
     value; Node next;
     public Node(int value)
     {this.value = value;
     }


  }
  public static Node mergeTwoLists(Node l1, Node l2) {
     if (l1 == null) return l2; if (l2 == null) return l1;


     Node dummy = new Node(0);
     Node tail = dummy;

     while (l1 != null && l2 != null)
        { if (l1.value <= l2.value) {
        tail.next = l1; l1 = l1.next;
        } else {


            tail.next = l2;
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
            l2 = l2.next;
        }
        tail = tail.next;
    }

    if (l1 != null) {
        tail.next = l1;
    } else {
        tail.next = l2;
    }
    return dummy.next;
}

public static void printList(Node head) {
    Node current = head;

    while (current != null) {
        System.out.print(current.value + " ");
        current = current.next;
    }
    System.out.println();

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

    Node l1 = new Node(1); l1.next
    = new Node(3);
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        l1.next.next = new Node(5);


        Node l2 = new Node(2);

        l2.next = new Node(4); l2.next.next
        = new Node(6);
        System.out.println("List 1: "); printList(l1);


        System.out.println("List 2: ");
        printList(l2);

                mergedList                      Node=
                                                mergeTwoLists(l1,
                                                l2);

        Systen   .out.println                    ("Merged List: ");
                printList(mergedList                    );

    }
}
```

**OUTPUT:-**

```
List 1:

1 3 5
List 2:
2 4 6

Merged List:

1 2 3 4 5 6
```

**Task 7: Circular Queue Binary Search**

**Consider a circular queue (implemented using a fixed-size array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.**

**Ans)**

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
package Ass; public class CircularQueueBinarySearch { public static int searchInCircularQueue(int[] arr, int

target) { int
    left = 0;

    int right = arr.length - 1;

    while (left <= right) { int mid =
      left + (right - left) / 2;



      if (arr[mid] == target) {
```

```
        return mid;
      }


      if (arr[left] <= arr[mid]) {

        if (arr[left] <= target && target < arr[mid]) {


          right = mid - 1;

        } else {
          left = mid + 1;
        }
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        } else { if (arr[mid] < target && target <=
            arr[right]) {



                left = mid + 1;

            } else { right =
                mid - 1;


            }
        }

    }

    return -1;


}
public static void main(String[] args) {
    int[] circularQueue = {12, 14, 18, 21, 3, 6, 8, 9};
    int target = 8;

    int result = searchInCircularQueue(circularQueue,

target);
```

NAME-AMIT KUMAR KUSHWAHA
EAMIL-amitsingh35873@gmail.com

```
        System.out.println("Index of target " + target + ": " +
result);
    }
}
```

**OUTPUT:-**

```
Index of target 8: 6
```

**Explanation**

1. **Initialization**: Start with the whole array (left = 0, right = arr.length - 1).
2. **Middle Calculation**: Compute the middle index.
3. **Middle Element Check**: If the middle element is the target, return its index.
4. **Identify Sorted Half**: Determine if the left half or the right half of the array is sorted.
5. **Adjust Search Range**:
    - If the left half is sorted and the target lies within this range, adjust right.
    - If the right half is sorted and the target lies within this range, adjust left.
6. **Continue Search**: Repeat the process until the target is found or the search range is exhausted.
7. **Return Result**: If the target is found, return the index. Otherwise, return -1.

This approach ensures an efficient binary search operation on a circular queue with a time complexity of O(log n)O(\log n)O(logn).