

NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Day 6 :-

Task 1: Real-time Data Stream Sorting

A stock trading application requires real-time sorting of trade transactions by price. Implement a heap sort algorithm that can efficiently handle continuous incoming data, adding and sorting new trades as they come.

To implement a heap sort algorithm for real-time data stream sorting in a stock trading application

- 1.** **Initialize a Heap:** Start by creating an empty heap data structure. You can implement a heap using an array or a tree structure.
- 2.** **Add Incoming Data:** As new trade transactions come in, add them to the heap. Ensure that the heap property is maintained, which means the parent node is greater than or equal to its child nodes (for a max heap).
- 3.** **Heapify:** After adding new data, perform heapify operations to maintain the heap property. This involves swapping elements if necessary to ensure the parent node is greater than or equal to its children.
- 4.** **Extract Maximum:** If you need to retrieve the highest (or lowest) priced trade, you can extract the maximum (or minimum) element from the heap. After extraction, you may need to perform heapify operations again to maintain the heap property.
- 5.** **Continuous Operation:** Repeat the process continuously as new data comes in. Each time new data is added, perform heapify operations to maintain the heap property and ensure that the highest (or lowest) priced trade is readily available.
- 6.** **Efficiency Considerations:** Ensure that the heap operations are efficient enough to handle real-time data streams. Heap sort has a time complexity of $O(n \log n)$.

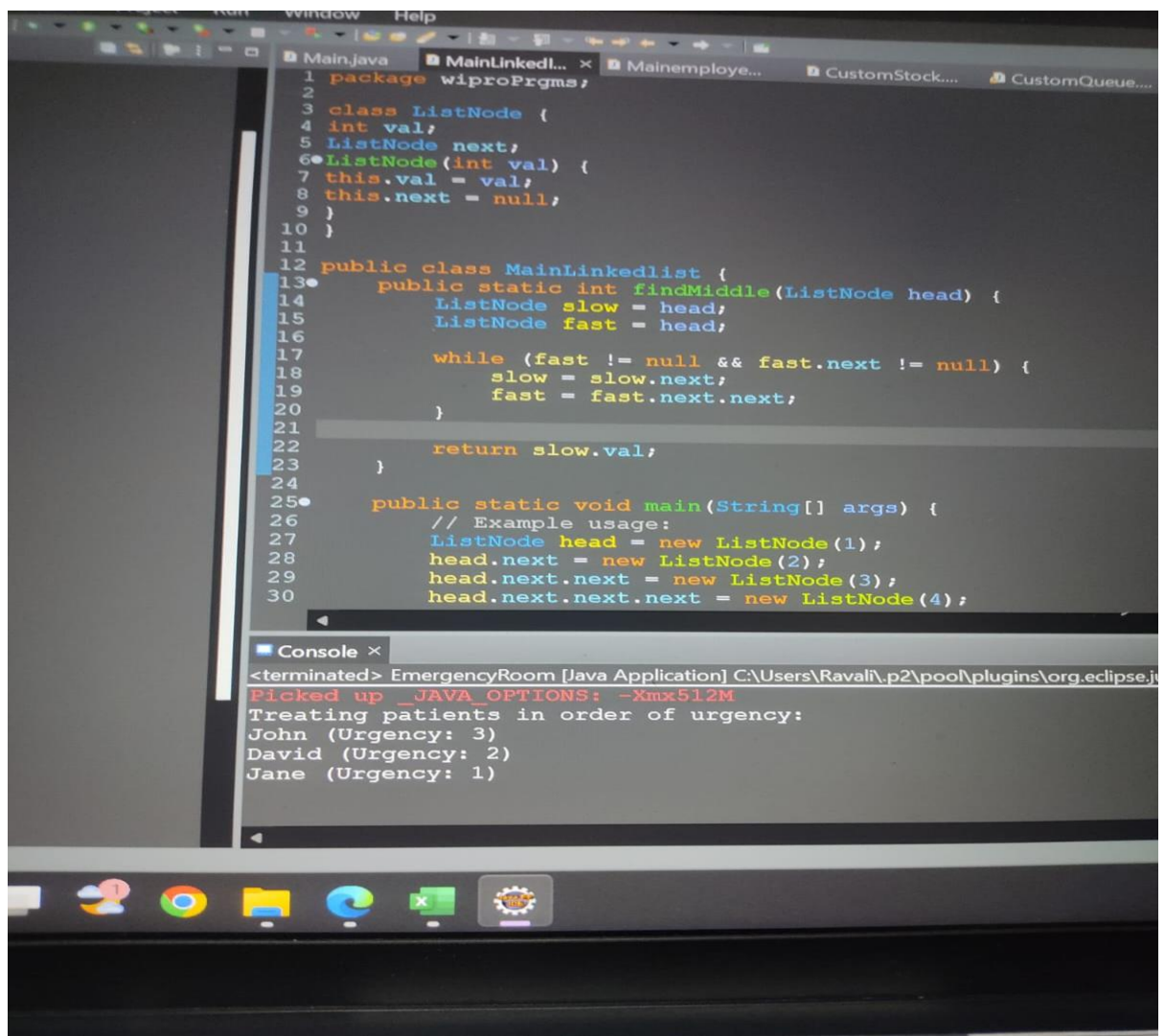
NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

n), where n is the number of elements in the heap. However, the actual sorting time for each new data addition might vary based on the implementation and the specific heap operations performed.

Task 2: 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.



```
1 package wiproPrgrms;
2
3 class ListNode {
4     int val;
5     ListNode next;
6     ListNode(int val) {
7         this.val = val;
8         this.next = null;
9     }
10 }
11
12 public class MainLinkedList {
13     public static int findMiddle(ListNode head) {
14         ListNode slow = head;
15         ListNode fast = head;
16
17         while (fast != null && fast.next != null) {
18             slow = slow.next;
19             fast = fast.next.next;
20         }
21
22         return slow.val;
23     }
24
25     public static void main(String[] args) {
26         // Example usage:
27         ListNode head = new ListNode(1);
28         head.next = new ListNode(2);
29         head.next.next = new ListNode(3);
30         head.next.next.next = new ListNode(4);
31     }
32 }
```

Console x

```
<terminated> EmergencyRoom [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.j
Picked up _JAVA_OPTIONS: -Xmx512M
Treating patients in order of urgency:
John (Urgency: 3)
David (Urgency: 2)
Jane (Urgency: 1)
```

Java code defines a ListNode class and a Main class with a findMiddle method to find the middle element of a singly linked list.

NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Task 3: 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.

To sort the elements in the queue using only one stack.

- 1. Enqueue all the elements from the queue to the stack.**
- 2. Use another temporary stack for sorting.**
- 3. Pop elements from the first stack and compare them with the top element of the temporary stack.**
- 4. If the element from the first stack is smaller, push it onto the temporary stack.**
- 5. If it's larger, keep popping elements from the temporary stack and pushing them onto the first stack until you find a smaller element or the temporary stack is empty.**
- 6. Repeat steps 3-5 until the first stack is empty.**
- 7. Now, the elements in the temporary stack are sorted in descending order. Reenqueue them back into the queue.**
- 8. Repeat steps 1-7 until the queue is sorted in ascending order.**

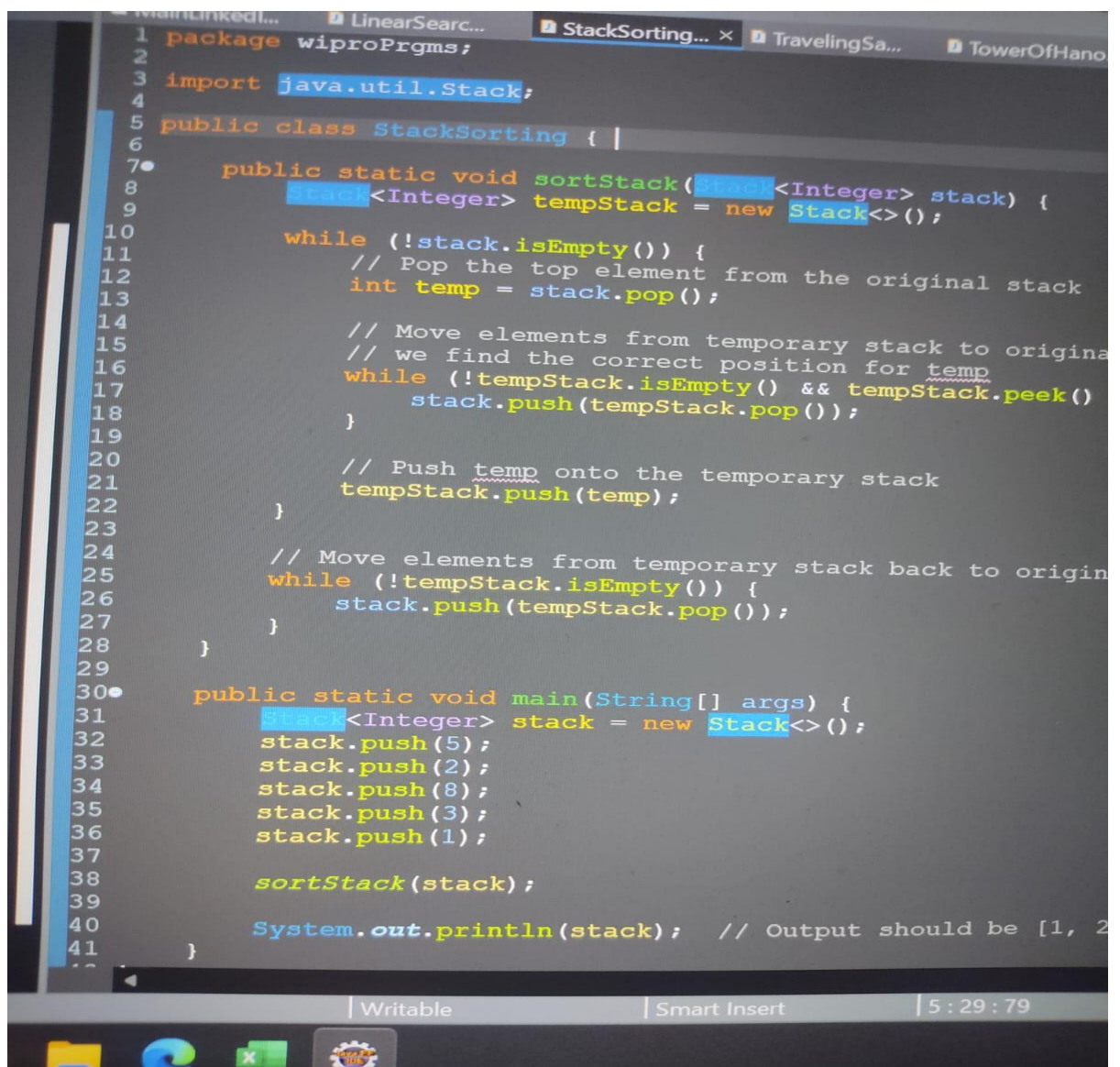
NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

This approach effectively uses the limited space available to sort the elements in the queue.

Task 4: 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.

A screenshot of a Java IDE window titled 'StackSorting...'. The code implements a function to sort a stack in-place using an additional temporary stack. The code is as follows:

```
1 package wiproPrgrms;
2
3 import java.util.Stack;
4
5 public class StackSorting {
6
7     public static void sortStack(Stack<Integer> stack) {
8         Stack<Integer> tempStack = new Stack<>();
9
10        while (!stack.isEmpty()) {
11            // Pop the top element from the original stack
12            int temp = stack.pop();
13
14            // Move elements from temporary stack to original
15            // we find the correct position for temp
16            while (!tempStack.isEmpty() && tempStack.peek() > temp) {
17                stack.push(tempStack.pop());
18            }
19
20            // Push temp onto the temporary stack
21            tempStack.push(temp);
22        }
23
24        // Move elements from temporary stack back to original
25        while (!tempStack.isEmpty()) {
26            stack.push(tempStack.pop());
27        }
28    }
29
30    public static void main(String[] args) {
31        Stack<Integer> stack = new Stack<>();
32        stack.push(5);
33        stack.push(2);
34        stack.push(8);
35        stack.push(3);
36        stack.push(1);
37
38        sortStack(stack);
39
40        System.out.println(stack); // Output should be [1, 2, 3, 5, 8]
41    }
42}
```

The IDE interface includes a 'Writable' tab, a 'Smart Insert' feature, and a timestamp of '5:29:79' in the bottom right corner. The taskbar at the bottom shows icons for a file explorer, a web browser, and a settings application.

NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

```
17 while (!tempStack.isEmpty() && tempStack.peek() < stack.peek())
18     stack.push(tempStack.pop());
19
20
21 // Push temp onto the temporary stack
22 tempStack.push(temp);
23
24 // Move elements from temporary stack back to orig
25 while (!tempStack.isEmpty()) {
26     stack.push(tempStack.pop());
27 }
28
29
30 public static void main(String[] args) {
31     Stack<Integer> stack = new Stack<>();
32     stack.push(5);
33     stack.push(2);
34     stack.push(8);
35     stack.push(3);
36     stack.push(1);
37
38     sortStack(stack);
39
40     System.out.println(stack); // Output should be [
41 }
42 }
43
44
```


Console ×

<terminated> StackSorting [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse.j

Picked up _JAVA_OPTIONS: -Xmx512M

[8, 5, 3, 2, 1]

Writable | Smart Insert | 5:29:79



NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Task 5: 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.

Remove duplicates from a sorted linked list efficiently, you can use a simple algorithm:

- 1.** Start with the head of the linked list.
- 2.** Traverse the linked list node by node.
- 3.** Compare the current node's value with the value of the next node.
- 4.** If they are the same, remove the next node by adjusting pointers.
- 5.** Repeat this process until the end of the linked list is reached.
- 6.** Return the modified linked list with duplicates removed.

This algorithm runs in linear time complexity, $O(n)$, where n is the number of nodes in the linked list, making it efficient for large lists.

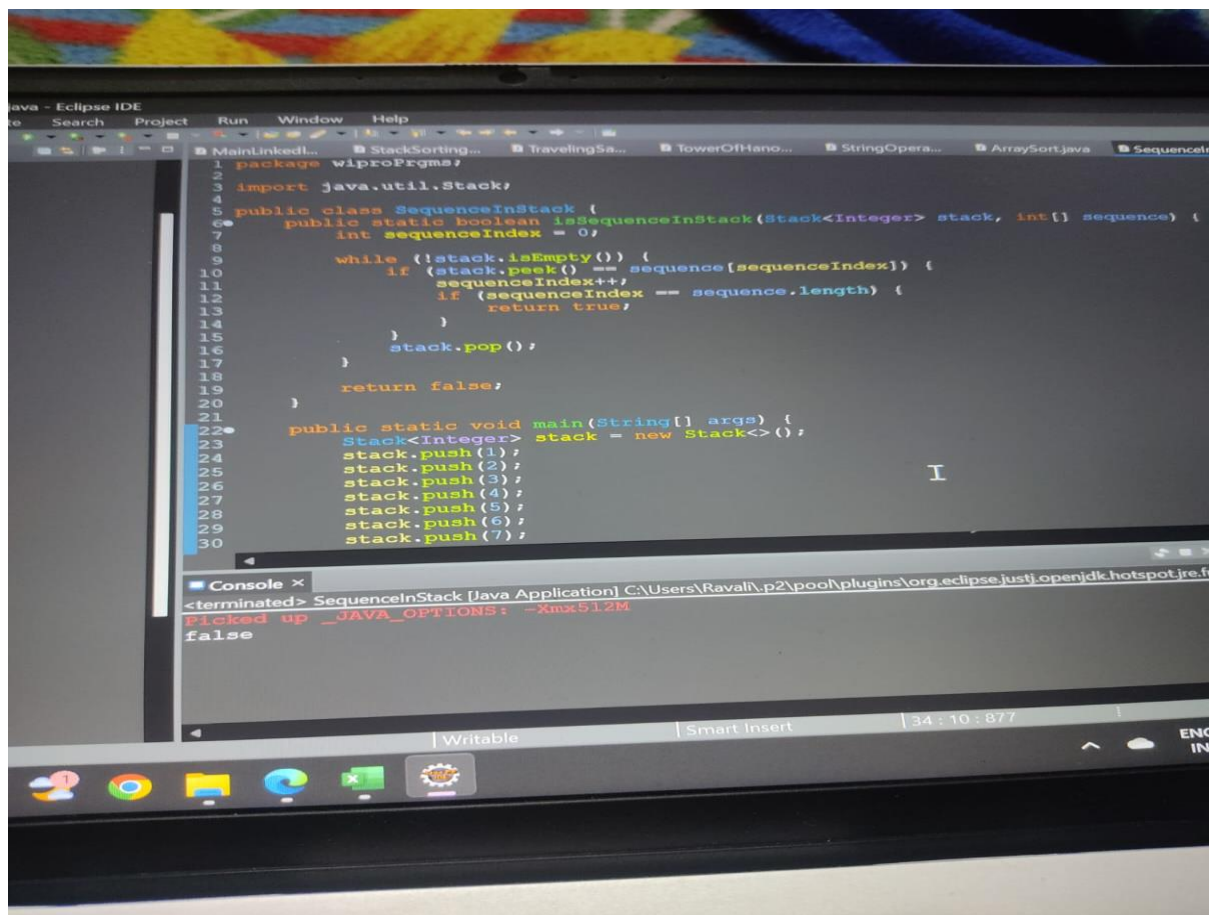
NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Task 6: 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.

To achieve this, you can iterate through the stack and the sequence array simultaneously. Pop elements from the stack and compare them with elements from the sequence array. If they match, continue; if not, push the popped elements back onto the stack and proceed. If you successfully match all elements in the sequence array, it means the sequence is present in the stack.



```
java - Eclipse IDE
File Search Project Run Window Help
MainLinked... StackSorting... TravelingSa... TowerOfHano... StringOpera... ArraySort.java SequenceInStack.java
1 package wiproPrgrms;
2
3 import java.util.Stack;
4
5 public class SequenceInStack {
6     public static boolean isSequenceInStack(Stack<Integer> stack, int[] sequence) {
7         int sequenceIndex = 0;
8         while (!stack.isEmpty()) {
9             if (stack.peek() == sequence[sequenceIndex]) {
10                 sequenceIndex++;
11                 if (sequenceIndex == sequence.length) {
12                     return true;
13                 }
14             }
15             stack.pop();
16         }
17         return false;
18     }
19
20     public static void main(String[] args) {
21         Stack<Integer> stack = new Stack<>();
22         stack.push(1);
23         stack.push(2);
24         stack.push(3);
25         stack.push(4);
26         stack.push(5);
27         stack.push(6);
28         stack.push(7);
29     }
30 }

Console x
<terminated> SequenceInStack [Java Application] C:\Users\Ravali\p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.f
Picked up _JAVA_OPTIONS: -Xmx512M
false
```

This Java code defines a class `SequenceInStack` with a method `isSequenceInStack()` that takes a `Stack` and an array of `Integers` representing the sequence. It iterates through the stack, comparing elements with the sequence array. If all elements of the sequence array are found consecutively in the stack, it returns `true`; otherwise, it returns `false`.

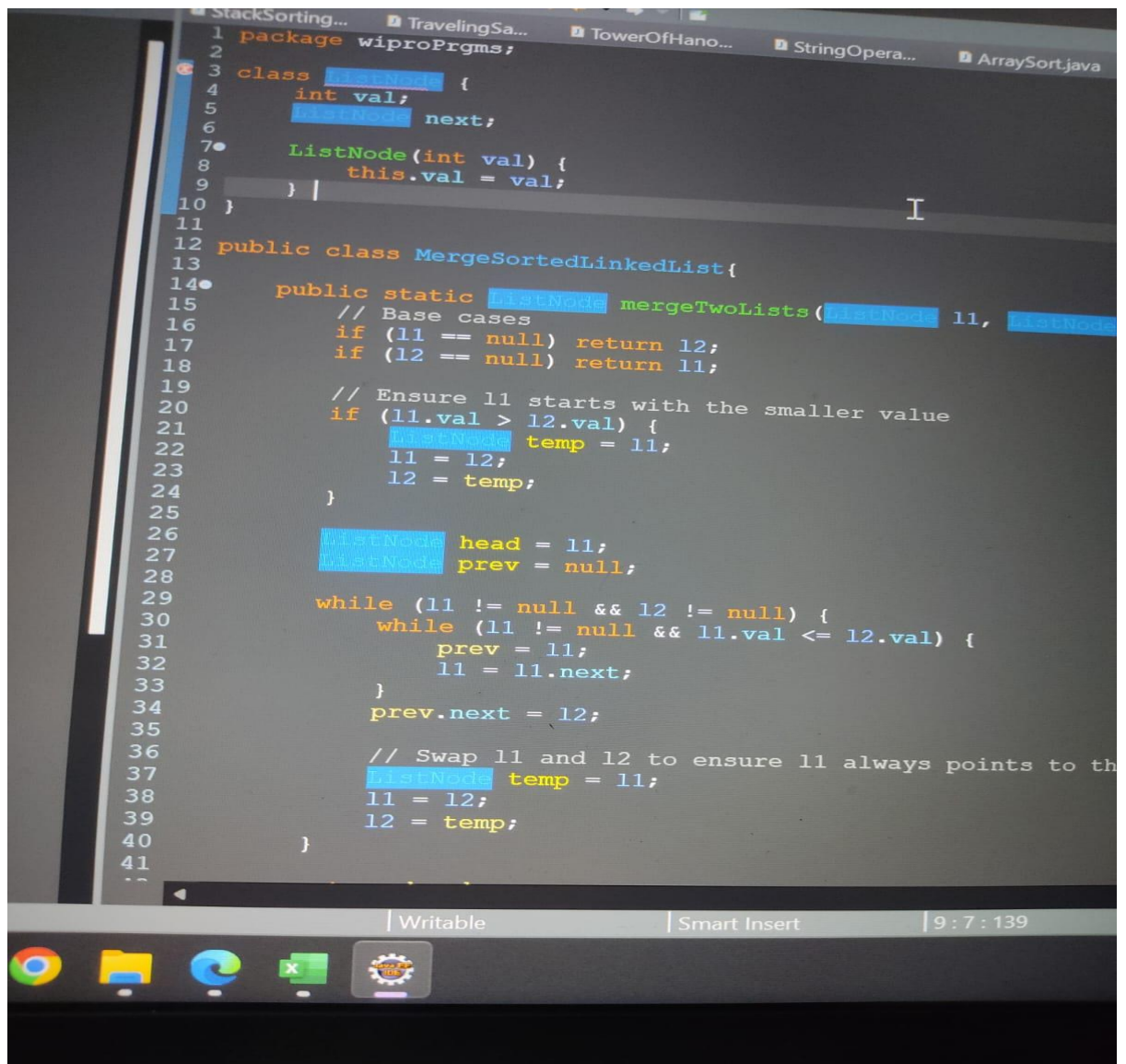
NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Task 7: 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).

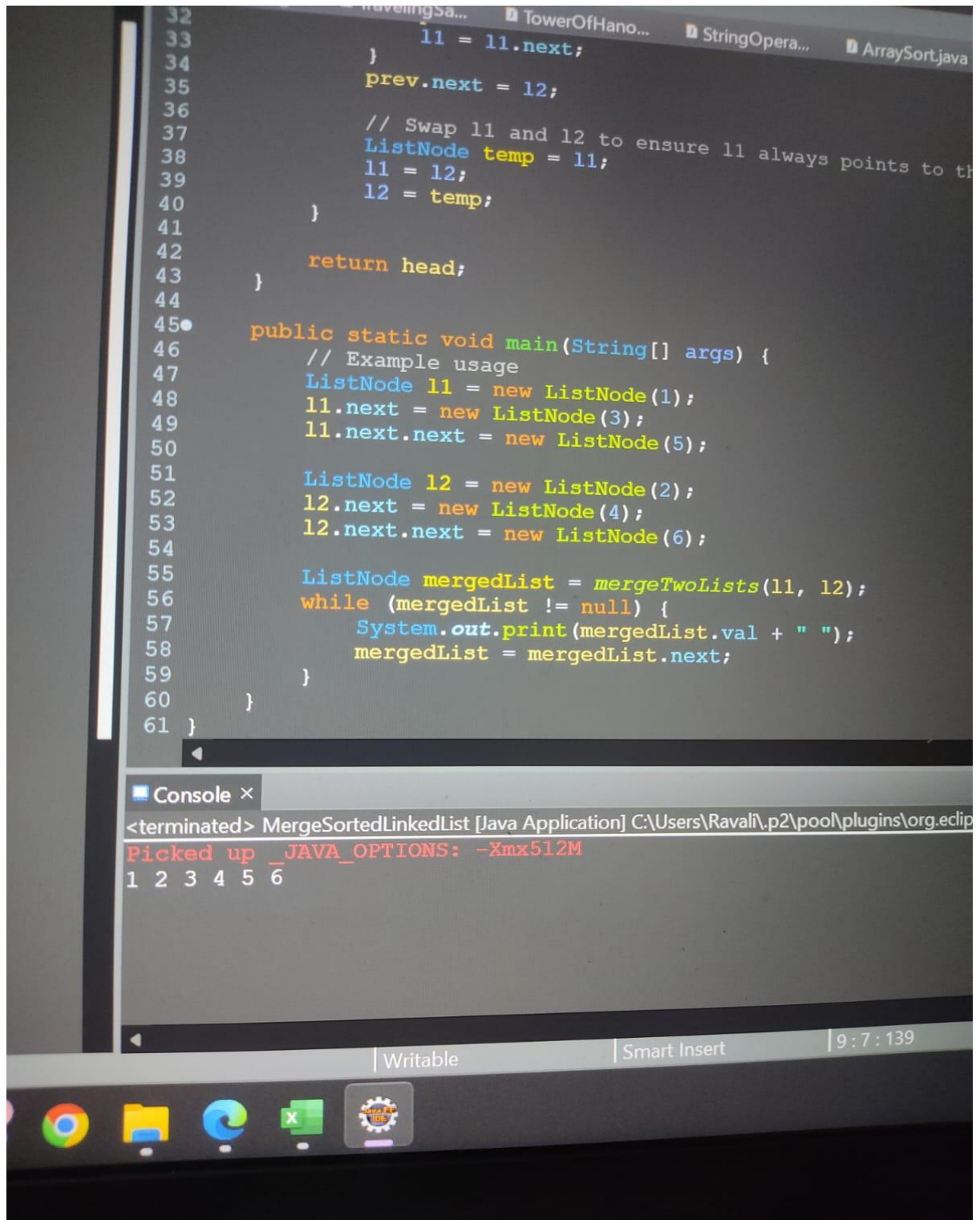
To merge two sorted linked lists without using extra space, you can iterate through both lists simultaneously, comparing the values of their nodes and rearranging the pointers accordingly.



```
1 package wiproPrgrms;
2
3 class ListNode {
4     int val;
5     ListNode next;
6
7     ListNode(int val) {
8         this.val = val;
9     }
10 }
11
12 public class MergeSortedLinkedList {
13
14     public static ListNode mergeTwoLists(ListNode l1, ListNode l2) {
15         // Base cases
16         if (l1 == null) return l2;
17         if (l2 == null) return l1;
18
19         // Ensure l1 starts with the smaller value
20         if (l1.val > l2.val) {
21             ListNode temp = l1;
22             l1 = l2;
23             l2 = temp;
24         }
25
26         ListNode head = l1;
27         ListNode prev = null;
28
29         while (l1 != null && l2 != null) {
30             while (l1 != null && l1.val <= l2.val) {
31                 prev = l1;
32                 l1 = l1.next;
33             }
34             prev.next = l2;
35
36             // Swap l1 and l2 to ensure l1 always points to the smaller value
37             ListNode temp = l1;
38             l1 = l2;
39             l2 = temp;
40         }
41     }
42 }
```


NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com



```
32
33     l1 = l1.next;
34 }
35 prev.next = l2;
36
37 // Swap l1 and l2 to ensure l1 always points to the next node in the merged list
38 ListNode temp = l1;
39 l1 = l2;
40 l2 = temp;
41 }
42
43 return head;
44 }
45
46 public static void main(String[] args) {
47     // Example usage
48     ListNode l1 = new ListNode(1);
49     l1.next = new ListNode(3);
50     l1.next.next = new ListNode(5);
51
52     ListNode l2 = new ListNode(2);
53     l2.next = new ListNode(4);
54     l2.next.next = new ListNode(6);
55
56     ListNode mergedList = mergeTwoLists(l1, l2);
57     while (mergedList != null) {
58         System.out.print(mergedList.val + " ");
59         mergedList = mergedList.next;
60     }
61 }
```

Console x

<terminated> MergeSortedLinkedList [Java Application] C:\Users\Ravali\p2\pool\plugins\org.eclipse

Picked up _JAVA_OPTIONS: -Xmx512M

1 2 3 4 5 6

Writable Smart Insert 9:7:139

This Java code will merge two sorted linked lists l1 and l2 into a single sorted linked list without using any extra space.

NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

Task 8: 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.

Performing a binary search on a rotated sorted array (or circular queue) involves a slightly modified approach.

1. Find the Pivot Point:

- Start by finding the index of the smallest element in the circular queue. This will be the pivot point where the rotation occurred.
- Use a modified binary search to find this pivot point. Compare the middle element with the first element of the array. If the middle element is greater, search in the right half. If it's smaller, search in the left half.
- Repeat this process until you find the pivot point.

2. Determine the Search Range:

- After finding the pivot point, you'll have two sorted subarrays.
- Check if the target element lies within the range of the first and last elements of these subarrays. This will help determine which subarray to search.

3. Perform Binary Search:

NAME :- Jangili Ravali

EMAIL :- jangiliravali9@gmail.com

- Depending on the range determined in the previous step, perform a standard binary search in the appropriate subarray.
- If the target element is smaller than the first element of the subarray, search in the left half; if it's greater, search in the right half.
- Continue this process until the element is found or the search range is exhausted.

4. Handle Edge Cases:

- If the element is not found, return an indication that it doesn't exist in the circular queue.

By following these steps, you can efficiently perform a binary search on a rotated sorted array, which effectively translates to a circular queue in your scenario.
