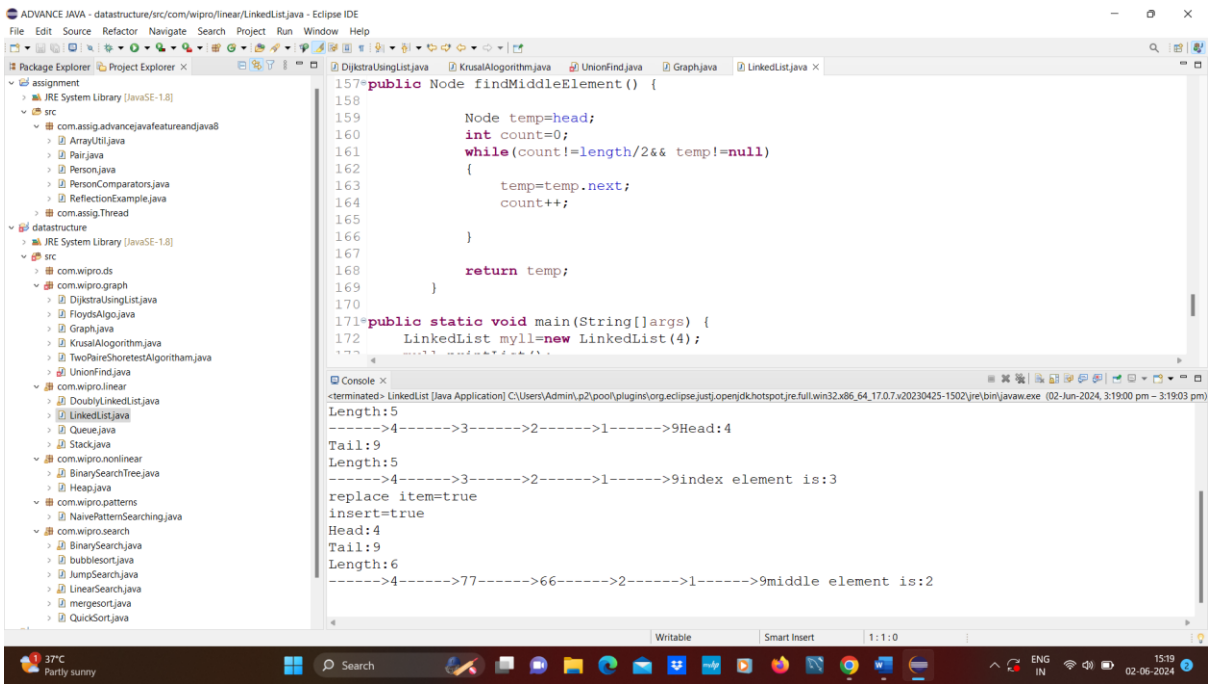
NAME:VISHAKHA AVINASH KALE

ASSIGNMENT:Linear Data Structure

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.

```
public Node findMiddle() {
 Node temp=head;
 int count=0;
 while(count!=length/2&& temp!=null)
 {
 temp=temp.next;
 count++;
 }
return temp;
 }
```

OUTPUT:



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.

Initialization:

Let's denote the queue as Q and the stack as S.

The goal is to transfer elements between Q and S to sort the elements in Q.

Sorting Steps:

While Q is not empty, perform the following operations:

a. Find the Minimum Element:

Initialize a variable min with a value larger than any element in Q (e.g., Integer.MAX_VALUE).

Dequeue all elements from Q one by one.

For each element, compare it with min. If it is smaller, update min with this element.

Push each dequeued element onto S.

Once all elements are transferred to S, min will hold the smallest element from Q.

b. Transfer Elements Back to Q:

Initialize a variable countMin to keep track of how many times min appears.

While S is not empty, perform the following operations:

Pop an element from S.

If the element is equal to min, increment countMin.

If the element is not equal to min, enqueue it back to Q.

c. Place the Minimum Element(s) in Sorted Position:

Enqueue the min element(s) back to Q based on the value of countMin.

Repeat Steps:Repeat the above steps until Q is sorted. In each iteration, the smallest remaining elements are placed in their correct

positions in Q.

Consider the queue Q with elements: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5].

First pass:

Find min = 1, and enqueue two 1s to Q.

Q becomes [1, 1, 3, 4, 5, 9, 2, 6, 5, 3, 5].

Second pass:

Find min = 2, and enqueue 2 to Q.

Q becomes [1, 1, 2, 3, 4, 5, 9, 6, 5, 3, 5].

Third pass:

Find min = 3, and enqueue three 3s to Q.

Q becomes [1, 1, 2, 3, 3, 3, 4, 5, 9, 6, 5, 5].

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.

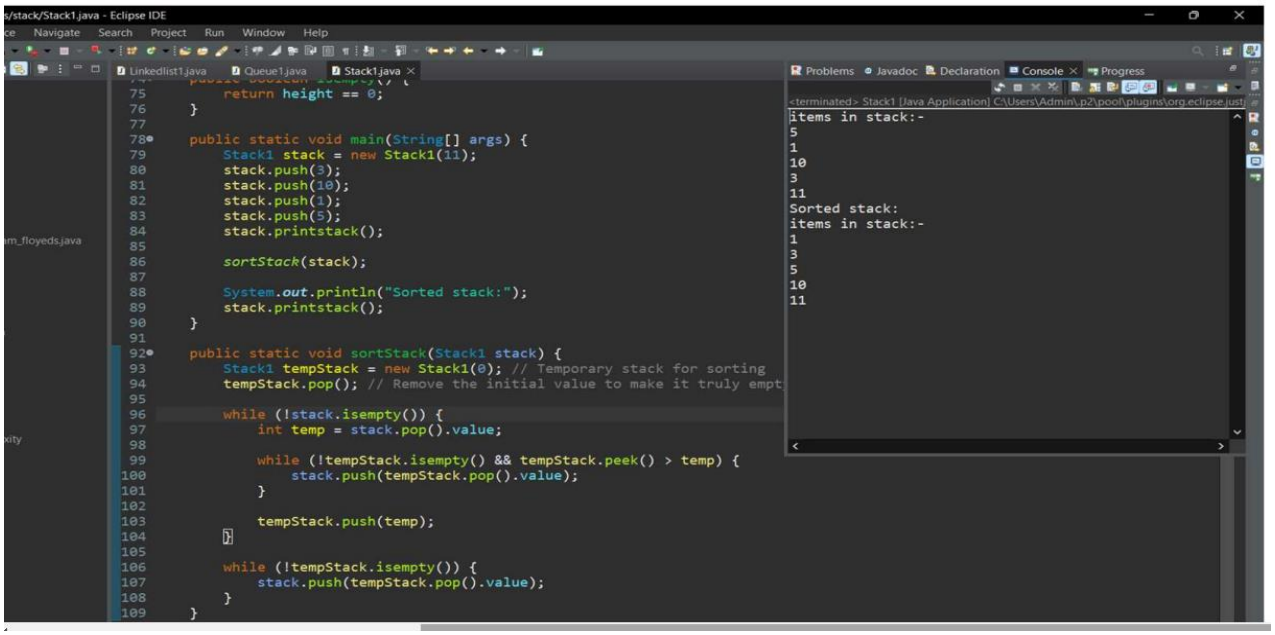
```
public static void sortStack(Stack1 stack) {
 Stack1 tempS = new Stack1(0);
 tempS.pop();


while (!stack.isempty()) {
 int temp = stack.pop().value;


while (!tempS.isempty() && tempS.peek() > temp){ stack.push(tempS.pop().value);
 }
tempS.push(temp);
 }
while (!tempS.isempty()) {
```

stack.push(tempS.pop().value);

}

}

OUTPUT:



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.

private void removeDuplicates() {

Node current = head;

while (current != null && current.next != null) { if (current.value == current.next.value) {

current.next = current.next.next;

length--;

} else {

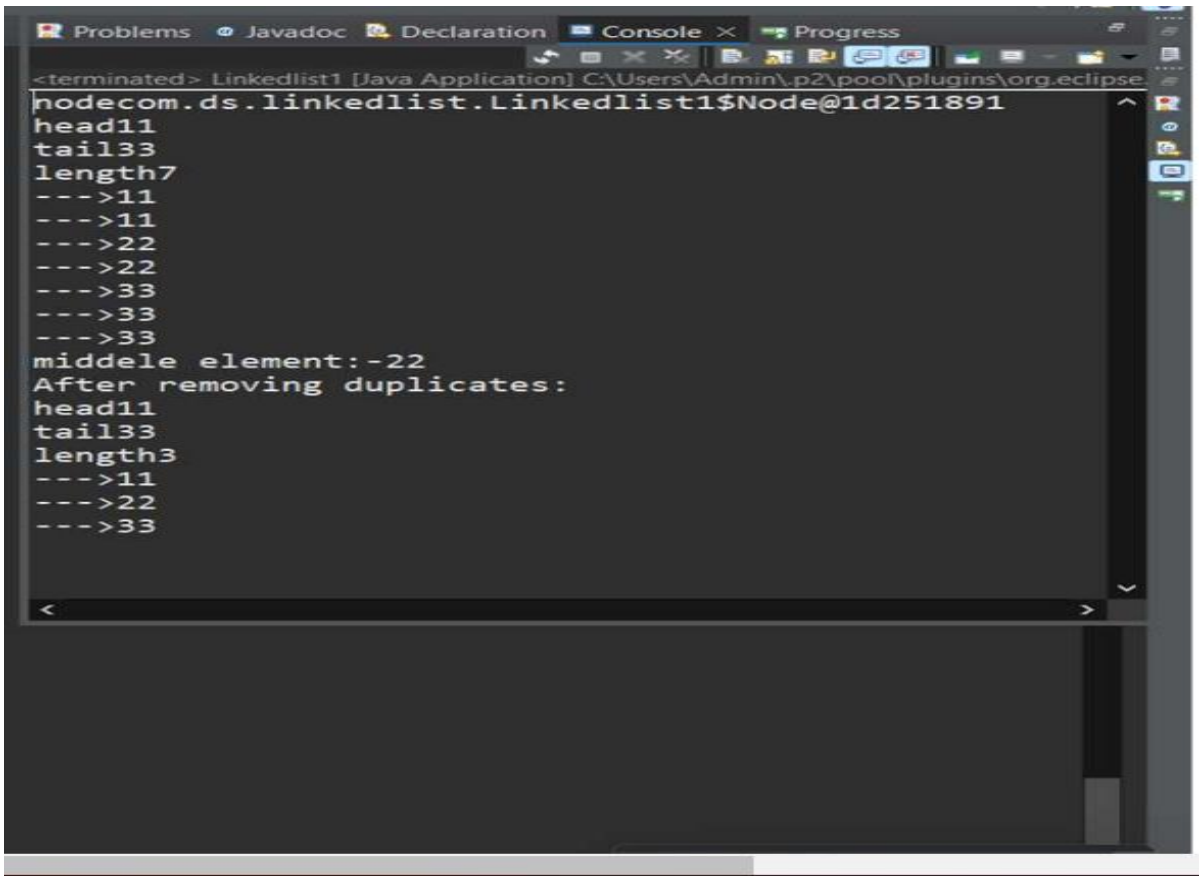
current = current.next;

}

}

}

OUTPUT:



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

public static boolean containsSequence(Stack1 stack, int[]sequence){

Stack1 reversedStack = new Stack1(0);

reversedStack.pop();

while (!stack.isempty()) {

reversedStack.push(stack.pop().value);

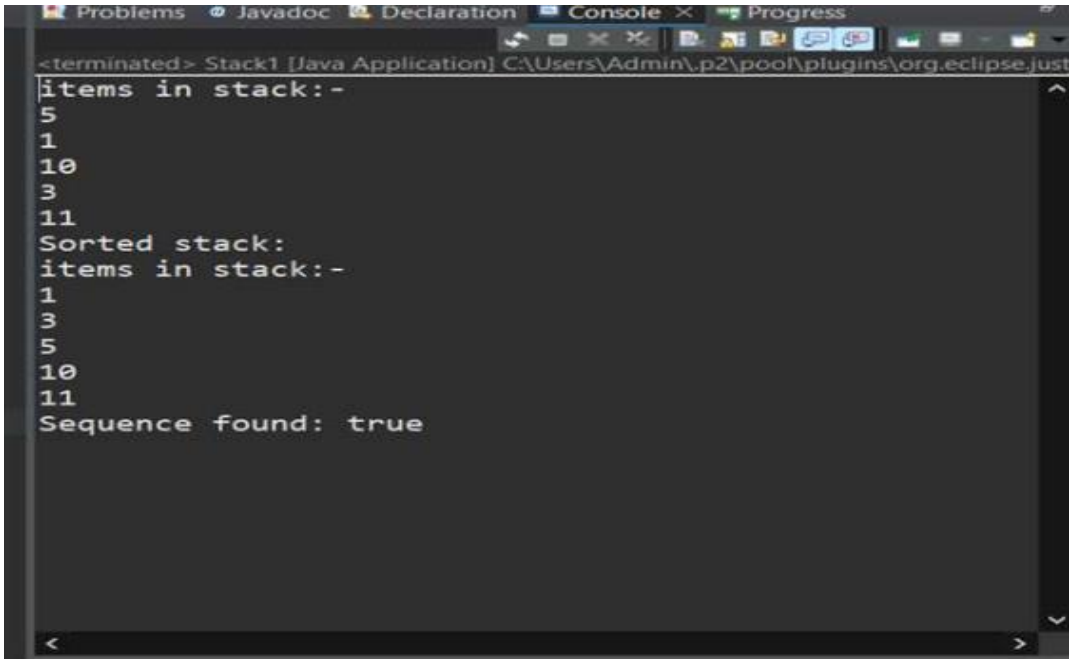
}

for (int i = sequence.length - 1; i >= 0; i--) { if (reversedStack.isempty() || reversedStack.pop().value!= sequence[i]) {

while (!reversedStack.isempty()) {

```
 stack.push(reversedStack.pop().value); }
return false;
 }
 }
while (!reversedStack.isempty()) {
 stack.push(reversedStack.pop().value);
 }
return true;
 }
```

OUTPUT:



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).

```
public static Linkedkist1 mergeLists(Linkedkist1 l1, Linkedkist1 l2)
{
 if (l1.head == null) return l2;
 if (l2.head == null) return l1;
 Node dummy = new Node(0); // dummy node to simplify merge
```

logic

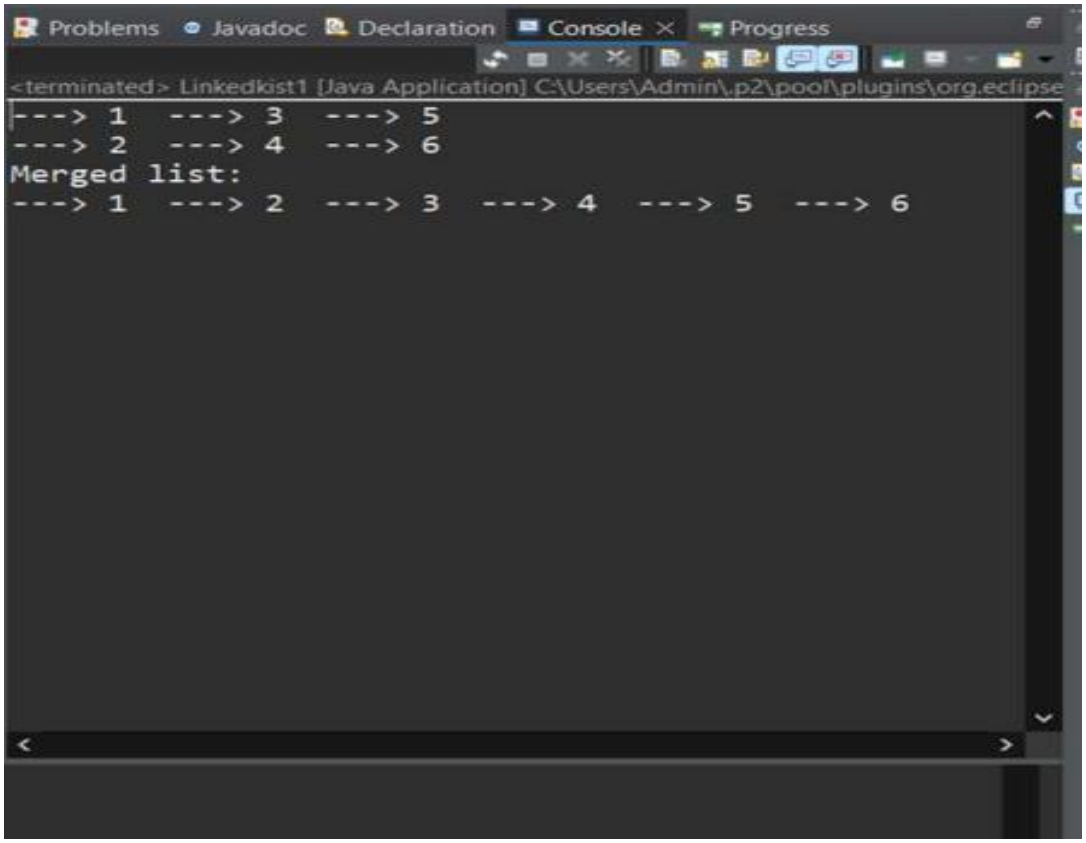
```
Node tail = dummy;
Node h1 = l1.head;
Node h2 = l2.head;
while (h1 != null && h2 != null) {
if (h1.value < h2.value) {
tail.next = h1;
h1 = h1.next;
} else {
tail.next = h2;
h2 = h2.next;
}
tail = tail.next;
}
if (h1 != null) {
tail.next = h1;
} else {
tail.next = h2;
}
l1.head = dummy.next;
l1.tail = l1.head;
l1.length = 0;
Node temp = l1.head;
while (temp != null) {
l1.length++;
if (temp.next == null) {
l1.tail = temp;
}
temp = temp.next;
```

}

Return l1;

}

OUTPUT:



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.

```
package com.assig.linear;
public class CircularQueueBinarySearch {


public static int search(int[] array, int target) {
 int low = 0;
 int high = array.length - 1;
```

```
while (low <= high) {
 int mid = low + (high - low) / 2;

if (array[mid] == target) {
return mid;
 }

 if (array[low] <= array[mid]) {
 if (array[low] <= target && target < array[mid]) {
 high = mid - 1;
 } else {
 low = mid + 1;
 }
 } else {
 if (array[mid] < target && target <= array[high]) {
 low = mid + 1;
 } else {
 high = mid - 1;
 }
 }
 }

 return -1;
 }

 public static void main(String[] args) {
 int[] array = { 6, 7, 8, 9, 1, 2, 3, 4, 5 };
 int target = 3;
 int index = search(array, target);
```

```
 if (index != -1) {
 System.out.println("Element " + target + " found at
index: " + index);
 } else {
 System.out.println("Element " + target + " not found.");
 }
 }
}
```

OUTPUT: