

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.

Program:

```
package Assignments.Day6;

import java.util.Arrays;

public class Task1 {

    private static int[] trades = new int[100];
    private static int numTrades = 0;

    public static void addTrade(int trade) {
        if (numTrades == trades.length) {
            trades = Arrays.copyOf(trades, trades.length * 2);
        }
        trades[numTrades++] = trade;
        heapSort(trades, numTrades);
    }

    public static void heapSort(int[] arr, int size) {
        for (int i = size / 2 - 1; i >= 0; i--)
            heapify(arr, size, i);
        for (int i = size - 1; i > 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }

    public static void heapify(int[] arr, int n, int i) {
        int largest = i; // Initialize largest as root
        int left = 2 * i + 1; // left = 2*i + 1
        int right = 2 * i + 2; // right = 2*i + 2
```

```

        if (left < n && arr[left] > arr[largest])
            largest = left;

        if (right < n && arr[right] > arr[largest])
            largest = right;

        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }

    public static void main(String[] args) {
        // Example of adding trades
        addTrade(12);
        addTrade(11);
        addTrade(13);

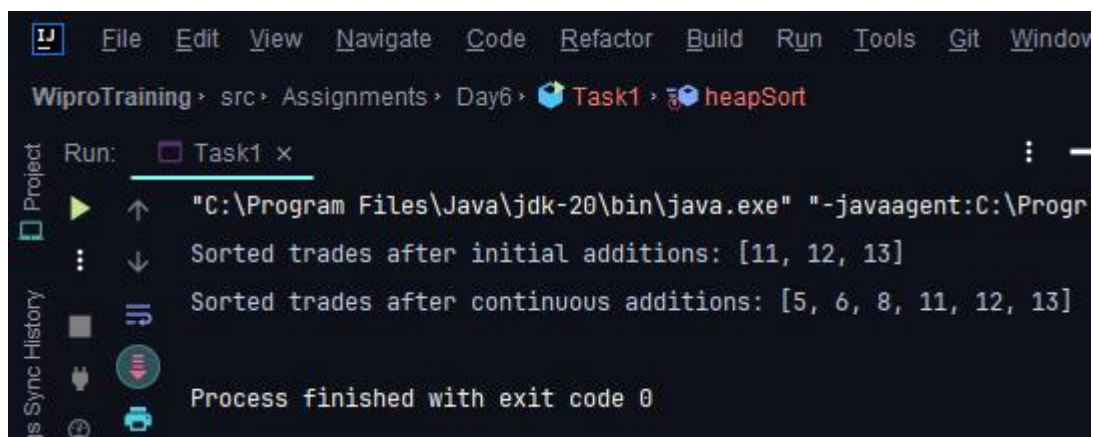
        System.out.println("Sorted trades after initial additions: " +
            Arrays.toString(Arrays.copyOf(trades, numTrades)));

        addTrade(5);
        addTrade(8);
        addTrade(6);

        System.out.println("Sorted trades after continuous additions: " +
            Arrays.toString(Arrays.copyOf(trades, numTrades)));
    }
}

```

Output:



The screenshot shows an IDE window with the following content:

- File Explorer:** WiproTraining > src > Assignments > Day6 > Task1 > heapSort
- Run Configuration:** Run: Task1 x
- Output Window:**

```

"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Progr
Sorted trades after initial additions: [11, 12, 13]
Sorted trades after continuous additions: [5, 6, 8, 11, 12, 13]
Process finished with exit code 0

```

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.

Program:

```
package Assignments.Day6;

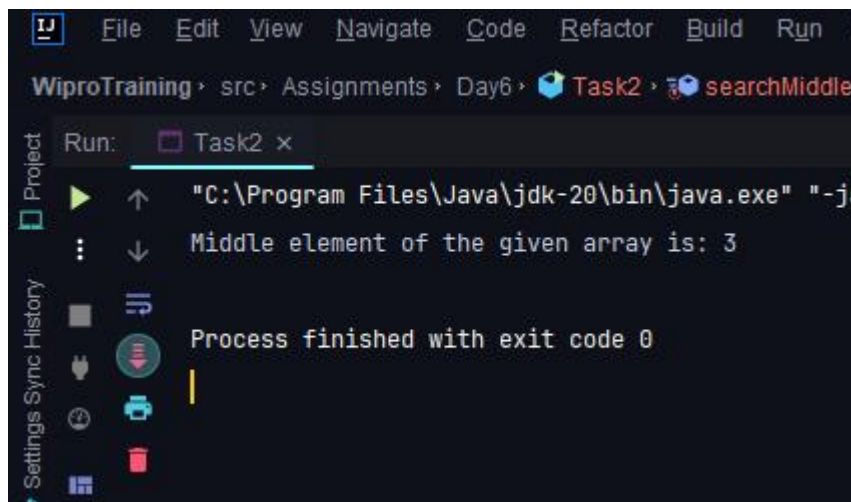
public class Task2 {
    Node head;
    static class Node {
        int data;
        Node next;
        Node(int data){
            this.data = data;
            next = null;
        }
    }
    private static void insert(Task2 list, int data){
        Node new_node = new Node(data);
        if(list.head == null){
            list.head = new_node;
        }else {
            Node last = list.head;
            while (last.next != null){
                last = last.next;
            }
            last.next = new_node;
        }
    }
    private static void searchMiddle(Task2 list){
        Node slow = list.head;
        Node fast = list.head;
        while( fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        if(slow != null) {
            System.out.println("Middle element of the given array is: "+ slow.data);
        }else{
            System.out.println("No Elements");
        }
    }
    public static void main(String[] args) {
        Task2 list = new Task2();
    }
}
```

```

        insert(list, 1);
        insert(list, 2);
        insert(list, 3);
        insert(list, 4);
        insert(list, 5);
        searchMiddle(list);
    }
}

```

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.

Program:

```

package Assignments.Day6;
import java.util.LinkedList;
import java.util.Queue;

public class Task3 {

    private static void sortQueue(Queue<Integer> q){
        for (int i = 1; i <= q.size() ; i++) {
            int minIndex = minIndexFind(q,q.size() - i);
            insertMinToRear(q, minIndex);
        }
    }

    private static void insertMinToRear(Queue<Integer> q, int minIndex) {

```

```

int minValue = 0;
int size = q.size();
for (int i = 0; i < size; i++) {
    if(!q.isEmpty()) {
        int cur = q.peek();
        q.poll();
        if (i != minIndex) {
            q.add(cur);
        } else {
            minValue = cur;
        }
    } else {
        System.out.println("No elements");
    }
}
q.add(minValue);
}

```

```

private static int minIndexFind(Queue<Integer> q, int index) {
    int minIndex = -1;
    int minValue = Integer.MAX_VALUE;
    int size = q.size();
    for (int i = 0; i < size; i++) {
        if (!q.isEmpty()) {
            int cur = q.peek();
            q.poll();
            if (cur <= minValue && i <= index) {
                minIndex = i;
                minValue = cur;
            }
            q.add(cur);
        }
    }
    return minIndex;
}

```

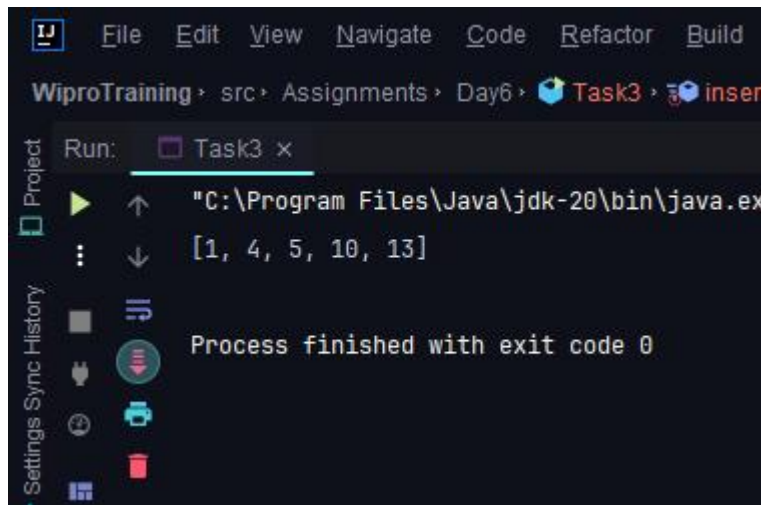
```

public static void main(String[] args) {
    Queue<Integer> q= new LinkedList<>();
    q.add(5);
    q.add(4);
    q.add(10);
    q.add(13);
    q.add(1);
    sortQueue(q);
    System.out.println(q);
}

```

```
}
```

Output:



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.

Program:

```
package Assignments.Day6;

import java.util.Stack;

public class Task4 {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        stack.add(5);
        stack.add(13);
        stack.add(1);
        stack.add(4);
        sortStack(stack);
    }

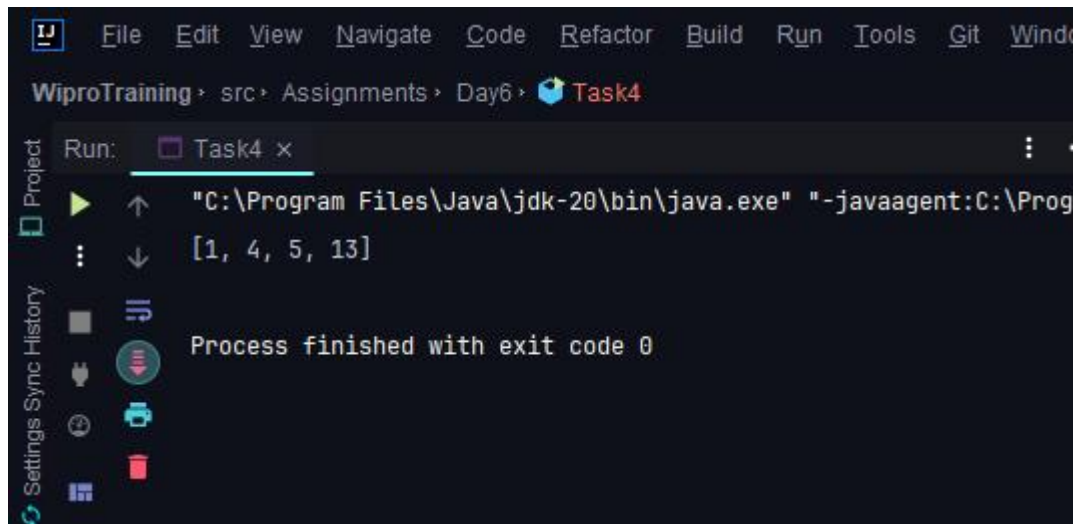
    private static void sortStack(Stack<Integer> stack) {
        Stack<Integer> tempStack = new Stack<>();
        while (!stack.isEmpty()){
            int cur = stack.pop();
            while (!tempStack.isEmpty() && tempStack.peek() > cur){
                stack.push(tempStack.pop());
            }
            tempStack.push(cur);
        }
    }
}
```

```

    }
    tempStack.push(cur);
  }
  System.out.println(tempStack);
}
}

```

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.

Program:

```

package Assignments.Day6;

class ListNode {
    int val;
    ListNode next;
    ListNode(int val) {
        this.val = val;
    }
}

public class Task5 {

    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) {

```

```

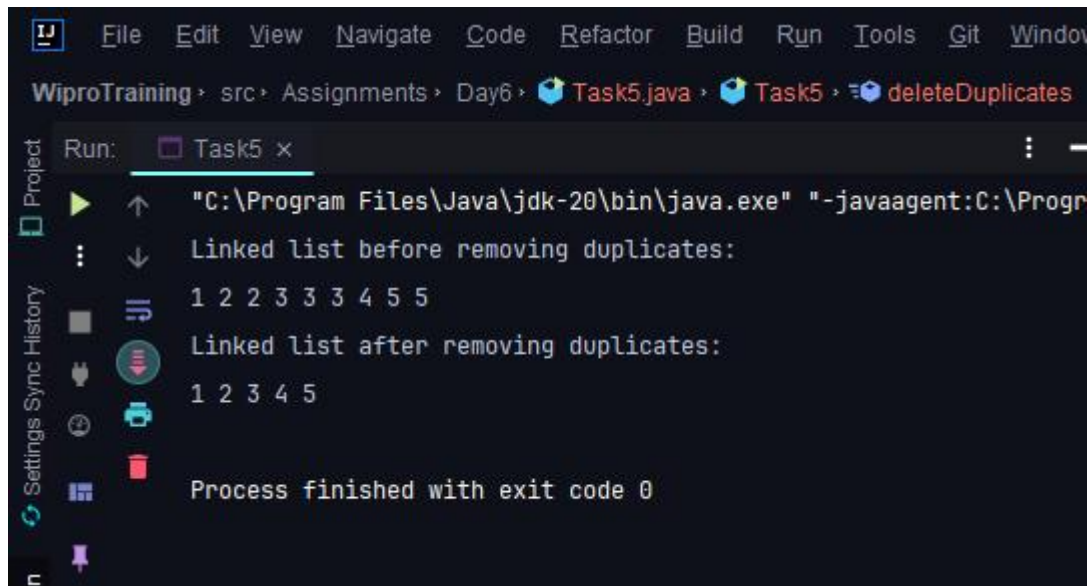
        return head;
    }
    ListNode current = head;
    while (current != null && current.next != null) {
        if (current.val == current.next.val) {
            current.next = current.next.next;
        } else {
            current = current.next;
        }
    }
    return head;
}

public static void printList(ListNode head) {
    ListNode current = head;
    while (current != null) {
        System.out.print(current.val + " ");
        current = current.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    // Sorted linked list: 1 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 5 -> 5
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(2);
    head.next.next.next = new ListNode(3);
    head.next.next.next.next = new ListNode(3);
    head.next.next.next.next.next = new ListNode(3);
    head.next.next.next.next.next.next = new ListNode(4);
    head.next.next.next.next.next.next.next = new ListNode(5);
    head.next.next.next.next.next.next.next.next = new ListNode(5);
    System.out.println("Linked list before removing duplicates:");
    printList(head);
    Task5 solution = new Task5();
    ListNode newHead = solution.deleteDuplicates(head);
    System.out.println("Linked list after removing duplicates:");
    printList(newHead);
}
}

```


Output:



```
WiproTraining > src > Assignments > Day6 > Task5.java > Task5 > deleteDuplicates

Run: Task5 x

"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Progr
Linked list before removing duplicates:
1 2 2 3 3 3 4 5 5
Linked list after removing duplicates:
1 2 3 4 5
Process finished with exit code 0
```

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.

Program:

```
package Assignments.Day6;
```

```
import java.util.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 class Task6 {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        int [] arr = {1,2};
        stack.push(1);
        stack.push(2);
        stack.push(3);
```

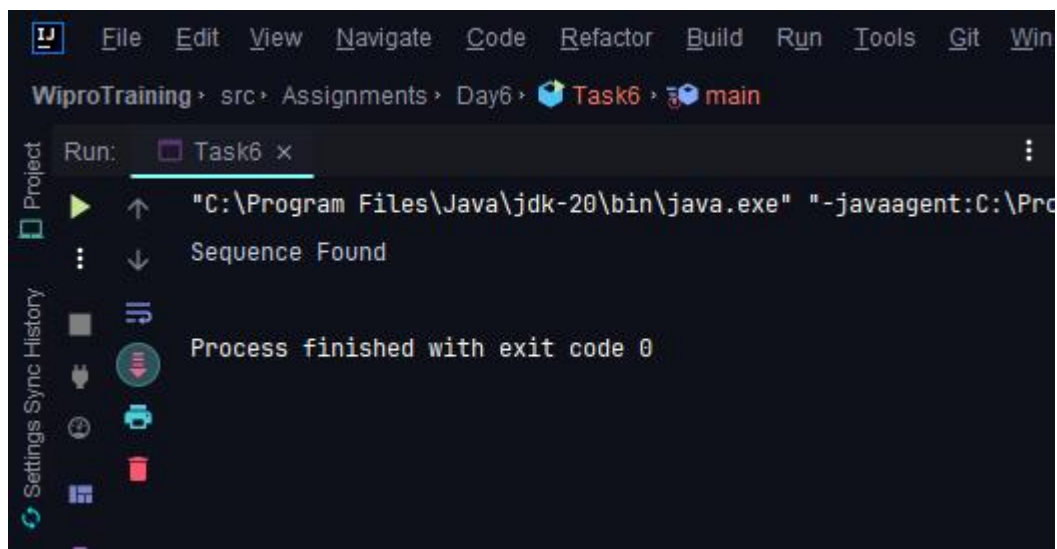
```

        stack.push(4);
        if(findSequence(stack, arr)){
            System.out.println("Sequence Found");
        }else{
            System.out.println("Sequence Not Found");
        }
    }
}

private static boolean findSequence(Stack<Integer> stack, int[] arr) {
    boolean ans = false;
    while(!stack.isEmpty()) {
        if (arr[arr.length - 1] == stack.peek()) {
            for (int i = arr.length - 1; i >= 0; i--) {
                if (arr[i] == stack.peek()) {
                    ans = true;
                    stack.pop();
                } else {
                    ans = false;
                }
            }
        } else {
            stack.pop();
        }
    }
    return ans;
}
}

```

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).

Program:

```
package Assignments.Day6;
```

```
public class Task7 {  
    public static class ListNode {  
        int val;  
        ListNode next;  
        ListNode() {}  
        ListNode(int val) { this.val = val; }  
        ListNode(int val, ListNode next) {  
            this.val = val; this.next = next;  
        }  
    }  
  
    public static ListNode mergeTwoLists(ListNode list1, ListNode list2) {  
        ListNode dummy = new ListNode(0);  
        ListNode current = dummy;  
        while(list1 != null && list2 != null){  
            if(list1.val <= list2.val){  
                current.next = list1;  
                list1 = list1.next;  
            }else{  
                current.next = list2;  
                list2 = list2.next;  
            }  
            current = current.next;  
        }  
        if(list1 != null){  
            current.next = list1;  
            list1 = list1.next;  
        }  
        if(list2 != null){  
            current.next = list2;  
            list2 = list2.next;  
        }  
        return dummy.next.next.next;  
    }  
  
    public static void main(String[] args) {  
        ListNode list1 = new ListNode();  
    }  
}
```

```

        ListNode list2 = new ListNode();
        insert(list1, 1);
        insert(list2,2);
        insert(list1,3);
        insert(list2,4);
        insert(list1,10);
        insert(list1,15);
        insert(list2, 12);

        displayMergedList(list1,list2);

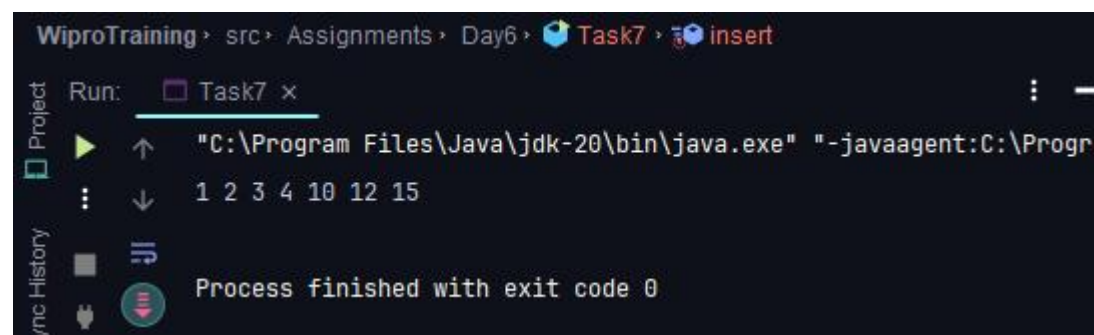
    }

    private static void insert(ListNode list, int data){
        ListNode new_node = new ListNode(data);
        if(list.next == null){
            list.next = new_node;
        }else{
            ListNode last = list.next;
            while (last.next != null){
                last = last.next;
            }
            last.next = new_node;
        }
    }

    private static void displayMergedList(ListNode list1, ListNode list2) {
        ListNode cur = mergeTwoLists(list1,list2);
        while (cur != null){
            System.out.print(cur.val+ " ");
            cur = cur.next;
        }
        System.out.println();
    }
}

```

Output:



```

WiproTraining > src > Assignments > Day6 > Task7 > insert
Run: Task7 x
"C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Progr
1 2 3 4 10 12 15
Process finished with exit code 0

```

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.

Program:

```
package Assignments.Day6;

public class Task8 {
    public int search(int[] nums, int target) {
        int n = nums.length;
        if (n == 0) return -1;
        int pivot = findPivot(nums);
        if (pivot == -1) {
            return binarySearch(nums, 0, n - 1, target);
        }
        if (nums[pivot] == target) {
            return pivot;
        }
        if (target >= nums[0] && target <= nums[pivot - 1]) {
            return binarySearch(nums, 0, pivot - 1, target);
        } else {
            return binarySearch(nums, pivot + 1, n - 1, target);
        }
    }

    private int findPivot(int[] nums) {
        int low = 0;
        int high = nums.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;

            if (mid > 0 && nums[mid] < nums[mid - 1]) {
                return mid;
            }
            if (nums[mid] >= nums[low]) {
                // Left part is sorted, pivot must be in the right
                low = mid + 1;
            } else {
                // Right part is sorted, pivot must be in the left part
                high = mid - 1;
            }
        }
        // If no pivot found, the array is not rotated
        return -1;
    }
}
```

```

    }
    private int binarySearch(int[] nums, int low, int high, int target) {
        while (low <= high) {
            int mid = (low + high) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        Task8 solution = new Task8();
        int[] nums1 = {4, 5, 6, 7, 0, 1, 2};
        int target1 = 0;
        System.out.println("Index of " + target1 + " in nums1: " +
solution.search(nums1,
                target1));
        int[] nums2 = {4, 5, 6, 7, 0, 1, 2};
        int target2 = 3;
        System.out.println("Index of " + target2 + " in nums2: " +
solution.search(nums2,
                target2));
    }
}

```

Output:



The screenshot shows an IDE window with the following content:

- Run:** Task8 x
- Process:** "C:\Program Files\Java\jdk-20\bin\java.exe" "-javaagent:C:\Pr
- Output:**

```

Index of 0 in nums1: -1
Index of 3 in nums2: -1
Process finished with exit code 0

```

