

Task 02:

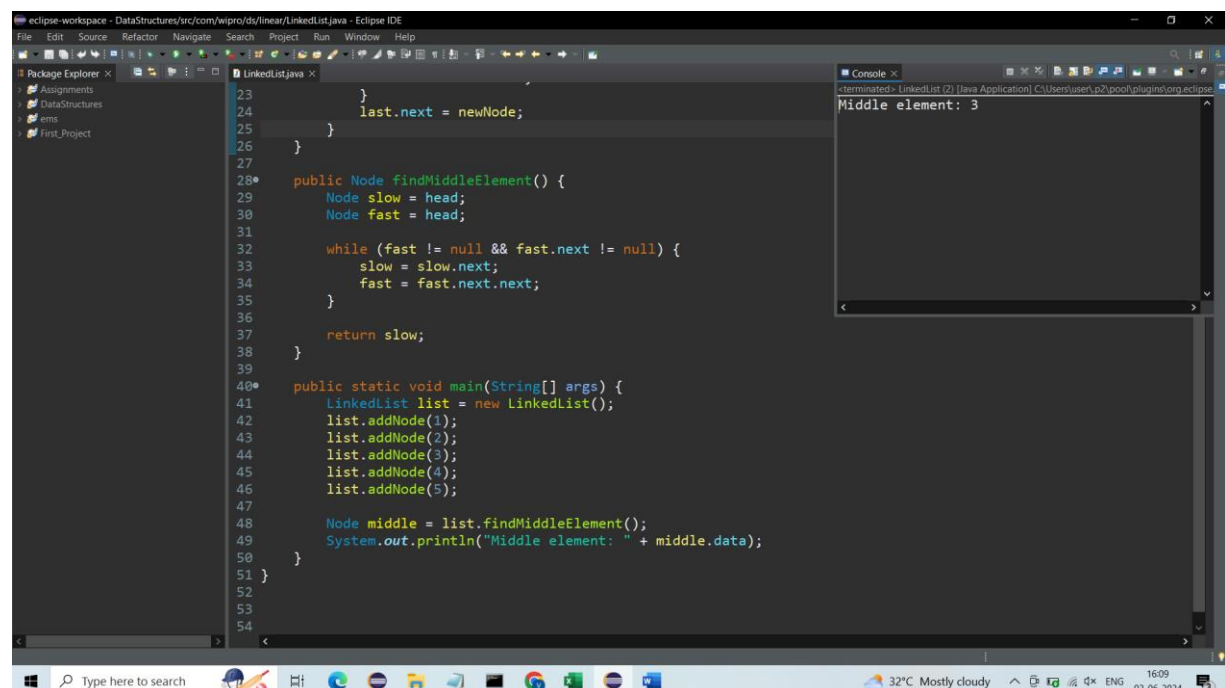
Linked List Middle Element Search

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.

Function: -

```
public Node findMiddleElement() {  
    Node slow = head;  
    Node fast = head;  
  
    while (fast != null && fast.next != null) {  
        slow = slow.next;  
        fast = fast.next.next;  
    }  
  
    return slow;  
}
```

Output: -



The screenshot shows the Eclipse IDE with a Java project. The main editor displays the implementation of the `findMiddleElement` function and a `main` method. The `main` method creates a linked list with nodes containing values 1 through 5. It then calls `findMiddleElement` and prints the middle element's data. The console output shows "Middle element: 3".

```
23  
24     last.next = newNode;  
25 }  
26  
27  
28* public Node findMiddleElement() {  
29     Node slow = head;  
30     Node fast = head;  
31  
32     while (fast != null && fast.next != null) {  
33         slow = slow.next;  
34         fast = fast.next.next;  
35     }  
36  
37     return slow;  
38 }  
39  
40* public static void main(String[] args) {  
41     LinkedList list = new LinkedList();  
42     list.addNode(1);  
43     list.addNode(2);  
44     list.addNode(3);  
45     list.addNode(4);  
46     list.addNode(5);  
47  
48     Node middle = list.findMiddleElement();  
49     System.out.println("Middle element: " + middle.data);  
50 }  
51 }  
52  
53  
54
```

Console Output:
Middle element: 3

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 for additional space, you can follow these steps:

1. Enqueue and Dequeue Operations with Stack:

- Use the stack as a temporary storage to help with sorting.
- You will perform operations such as push and pop on the stack to manipulate the elements.

2. Sorting Process:

- While the queue is not empty, dequeue elements from the queue one by one and push them onto the stack.
- At this point, the stack contains elements in reverse order compared to their original order in the queue.

3. Extracting Minimum Element:

- Pop elements from the stack and enqueue them back into the queue until the stack is empty.
- During this process, keep track of the minimum element encountered.

4. Finding the Minimum Element in the Queue:

- Iterate through the queue to find the minimum element.
- You can dequeue and enqueue elements from the queue, comparing each element with the minimum element found so far.
- Once the minimum element is found, dequeue and enqueue it to the end of the queue.

5. Repeat Until Queue is Sorted:

- Repeat steps 3 and 4 until the queue is empty.
- After each iteration, the minimum element will be moved to the end of the queue, gradually sorting the queue in ascending order.

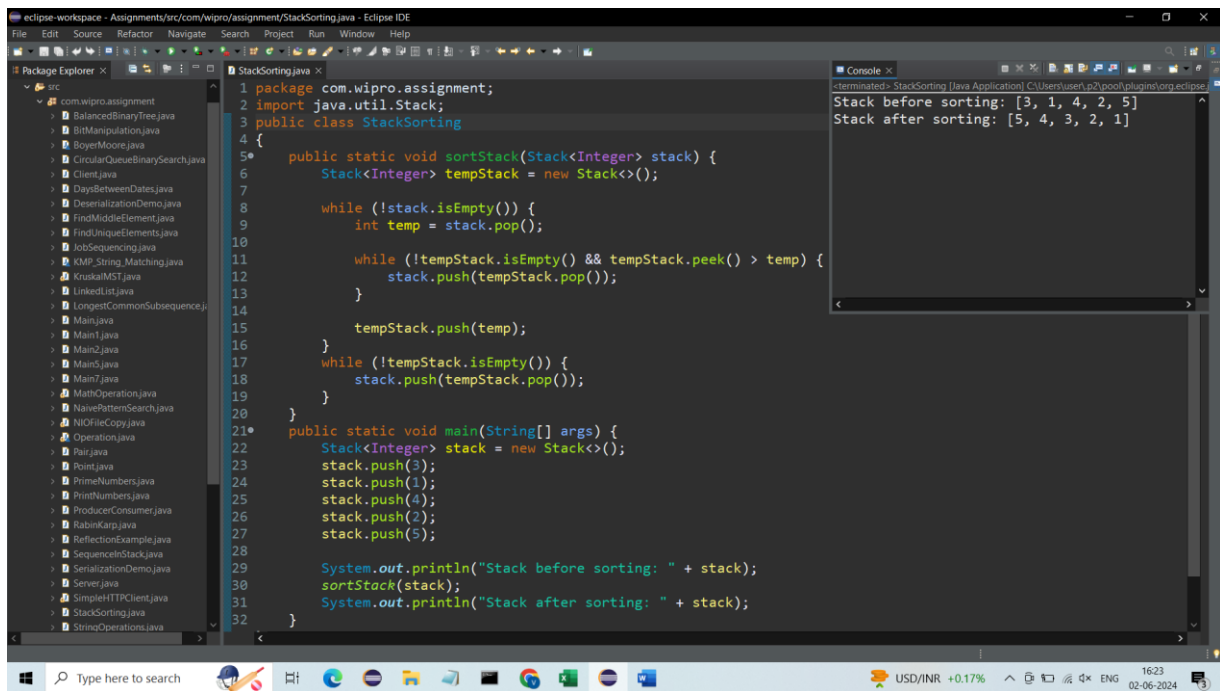
Task 4: Stack Sorting In-Place

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.

Function: -

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

Output: -



```
1 package com.wipro.assignment;
2 import java.util.Stack;
3 public class StackSorting
4 {
5     public static void sortStack(Stack<Integer> stack) {
6         Stack<Integer> tempStack = new Stack<>();
7
8         while (!stack.isEmpty()) {
9             int temp = stack.pop();
10
11             while (!tempStack.isEmpty() && tempStack.peek() > temp) {
12                 stack.push(tempStack.pop());
13             }
14
15             tempStack.push(temp);
16         }
17         while (!tempStack.isEmpty()) {
18             stack.push(tempStack.pop());
19         }
20     }
21     public static void main(String[] args) {
22         Stack<Integer> stack = new Stack<>();
23         stack.push(5);
24         stack.push(1);
25         stack.push(4);
26         stack.push(2);
27         stack.push(5);
28
29         System.out.println("Stack before sorting: " + stack);
30         sortStack(stack);
31         System.out.println("Stack after sorting: " + stack);
32     }
33 }
```

Console

```
<terminated> StackSorting [Java Application] C:\Users\user\p2\pool\plugins\org.eclipse
Stack before sorting: [3, 1, 4, 2, 5]
Stack after sorting: [5, 4, 3, 2, 1]
```

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.

Algorithm: -

1. Initialization:

Start with the head of the sorted linked list.

2.Traversal:

Traverse the linked list one node at a time.

3.Duplicate Check:

Compare the data value of the current node with the data value of the next node.

If they are equal, it means a duplicate is found.

4.Removing Duplicates:

To remove duplicates efficiently, simply bypass the next node by updating the next reference of the current node to skip the duplicate node.

Ensure proper memory management by deleting the duplicate node (if dynamic memory allocation is involved).

5.Continue Traversal:

Continue traversing the linked list until the end is reached.

6.Return:

Once traversal is complete, the linked list will have all duplicates removed, leaving only unique elements in sorted order.

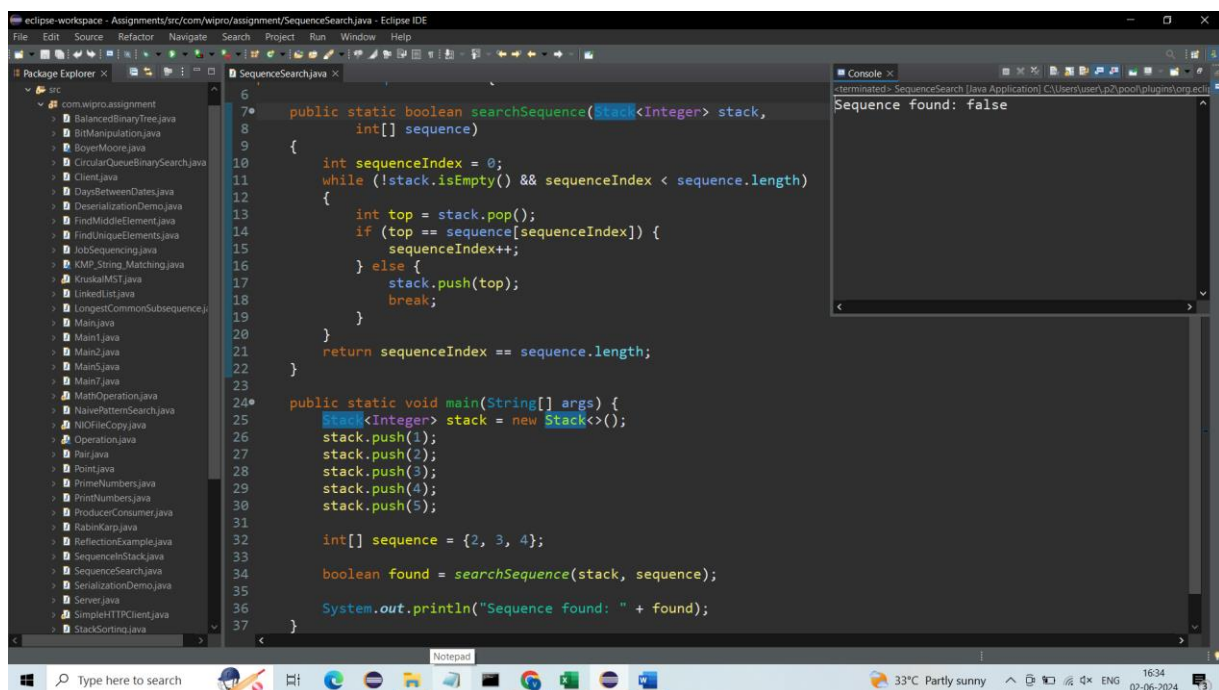
Task 6: Searching for a Sequence in a Stack

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.

Function: -

```
public static boolean searchSequence(Stack<Integer> stack, int[] sequence) {
    int sequenceIndex = 0;
    while (!stack.isEmpty() && sequenceIndex < sequence.length) {
        int top = stack.pop();
        if (top == sequence[sequenceIndex]) {
            sequenceIndex++;
        } else {
            stack.push(top);
            break;
        }
    }
    return sequenceIndex == sequence.length;
}
```

Output: -



The screenshot shows the Eclipse IDE with the following components:

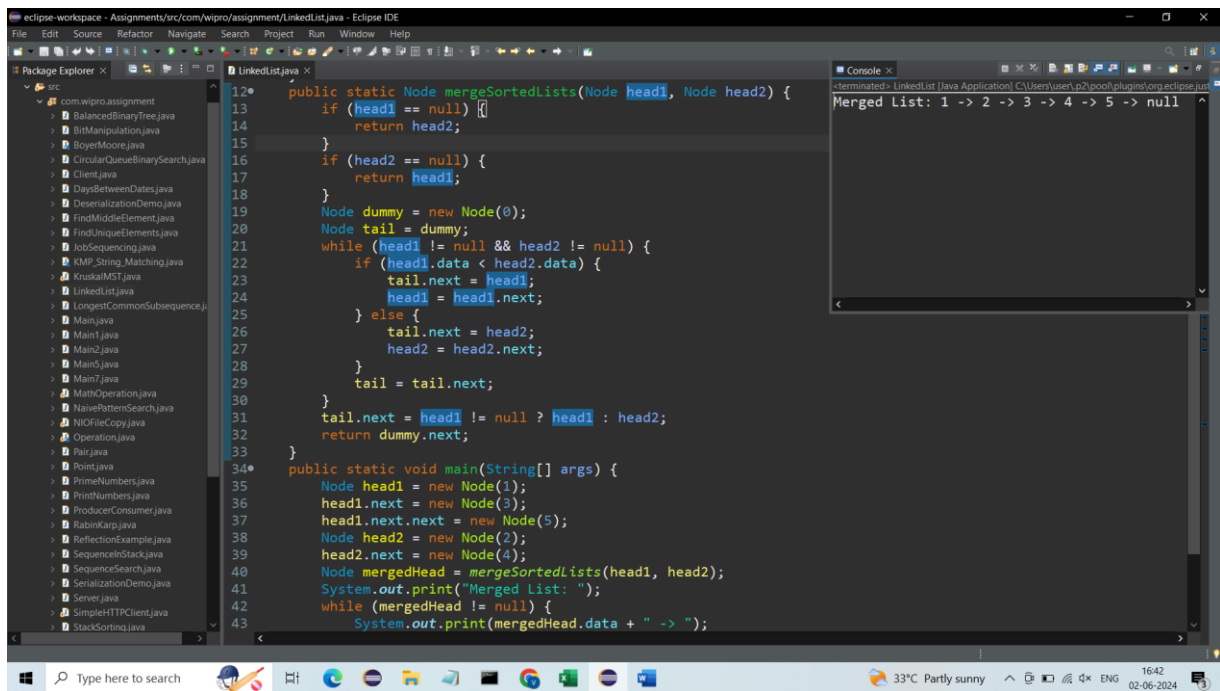
- Package Explorer:** A list of Java files in the 'com.wipro.assignment' package, including 'BalancedBinaryTree.java', 'BitManipulation.java', 'BoyerMoore.java', 'CircularQueueBinarySearch.java', 'Client.java', 'DaysBetweenDates.java', 'DeserializeDemo.java', 'FindMiddleElement.java', 'FindUniqueElements.java', 'JobSequencing.java', 'KMP_String_Matching.java', 'KruskalMST.java', 'LinkedList.java', 'LongestCommonSubsequence.java', 'Main.java', 'Main2.java', 'Main3.java', 'Main4.java', 'MathOperation.java', 'NaivePatternSearch.java', 'NIOFileCopy.java', 'Operation.java', 'Pair.java', 'Point.java', 'PrimeNumbers.java', 'PrintNumbers.java', 'ProducerConsumer.java', 'RabinKarp.java', 'ReflectionExample.java', 'SequenceSearch.java', 'SerializeDemo.java', 'Server.java', 'SimpleHTTPClient.java', and 'StackSorting.java'.
- Editor:** The 'SequenceSearch.java' file is open, showing the 'searchSequence' function and a 'main' method. The 'main' method creates a stack, pushes elements 1 through 5, and calls 'searchSequence' with the sequence {2, 3, 4}. The output is 'Sequence found: false'.
- Console:** The console shows the output 'Sequence found: false'.
- Taskbar:** The Windows taskbar is visible at the bottom, showing the system clock as 16:34 on 02-06-2024.

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 Node mergeSortedLists(Node head1, Node head2) {  
    if (head1 == null) {  
        return head2;  
    }  
    if (head2 == null) {  
        return head1;  
    }  
    Node dummy = new Node(0);  
    Node tail = dummy;  
    while (head1 != null && head2 != null) {  
        if (head1.data < head2.data) {  
            tail.next = head1;  
            head1 = head1.next;  
        } else {  
            tail.next = head2;  
            head2 = head2.next;  
        }  
        tail = tail.next;  
    }  
    tail.next = head1 != null ? head1 : head2;  
    return dummy.next;  
}
```

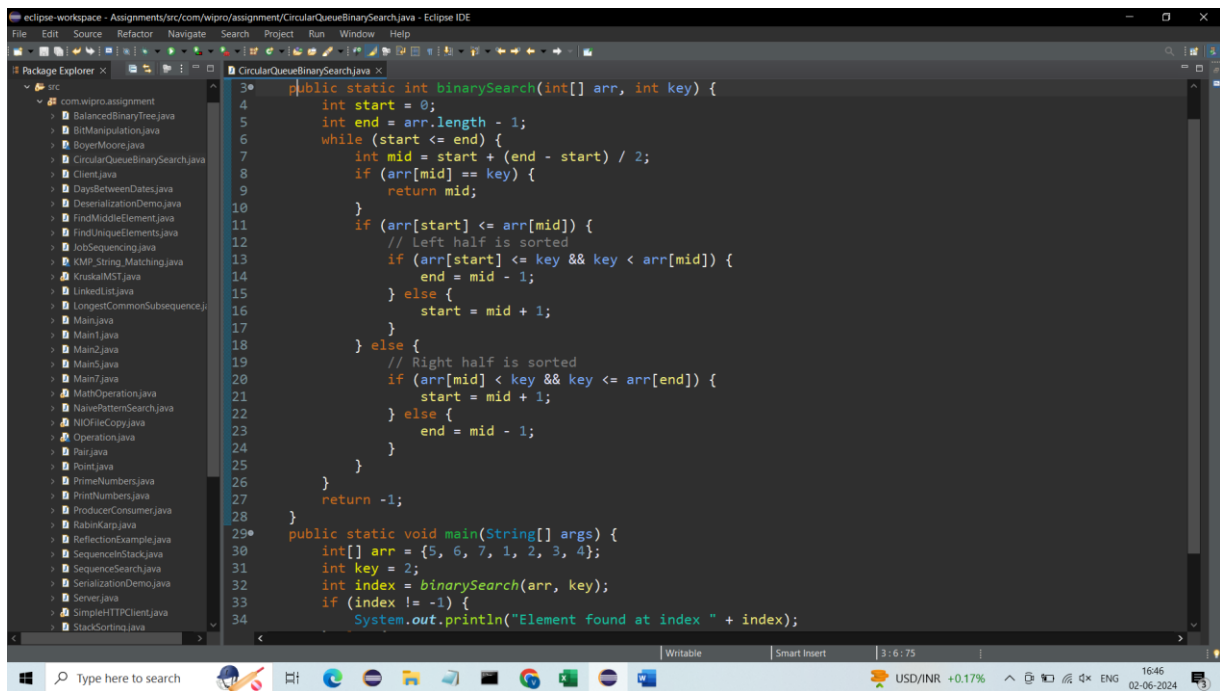
Output: -



Task 8: Circular Queue Binary Search

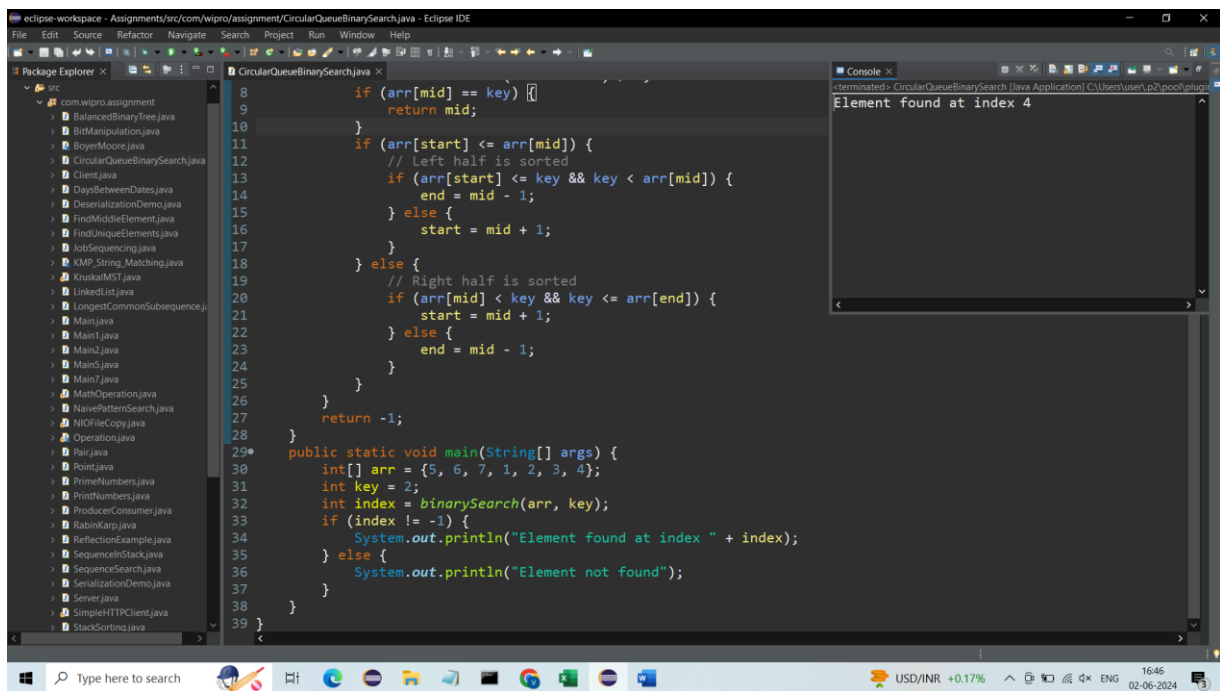
Consider a circular queue (implemented using a fixedsize array) where the elements are sorted but have been rotated at an unknown index. Describe an approach to perform a binary search for a given element within this circular queue.

Approach: -



```
1 public static int binarySearch(int[] arr, int key) {
2     int start = 0;
3     int end = arr.length - 1;
4     while (start <= end) {
5         int mid = start + (end - start) / 2;
6         if (arr[mid] == key) {
7             return mid;
8         }
9         if (arr[start] <= arr[mid]) {
10            // Left half is sorted
11            if (arr[start] <= key && key < arr[mid]) {
12                end = mid - 1;
13            } else {
14                start = mid + 1;
15            }
16        } else {
17            // Right half is sorted
18            if (arr[mid] < key && key <= arr[end]) {
19                start = mid + 1;
20            } else {
21                end = mid - 1;
22            }
23        }
24    }
25    return -1;
26 }
27
28 public static void main(String[] args) {
29     int[] arr = {5, 6, 7, 1, 2, 3, 4};
30     int key = 2;
31     int index = binarySearch(arr, key);
32     if (index != -1) {
33         System.out.println("Element found at index " + index);
34     }
```

Output: -



```
1 public static int binarySearch(int[] arr, int key) {
2     int start = 0;
3     int end = arr.length - 1;
4     while (start <= end) {
5         int mid = start + (end - start) / 2;
6         if (arr[mid] == key) {
7             return mid;
8         }
9         if (arr[start] <= arr[mid]) {
10            // Left half is sorted
11            if (arr[start] <= key && key < arr[mid]) {
12                end = mid - 1;
13            } else {
14                start = mid + 1;
15            }
16        } else {
17            // Right half is sorted
18            if (arr[mid] < key && key <= arr[end]) {
19                start = mid + 1;
20            } else {
21                end = mid - 1;
22            }
23        }
24    }
25    return -1;
26 }
27
28 public static void main(String[] args) {
29     int[] arr = {5, 6, 7, 1, 2, 3, 4};
30     int key = 2;
31     int index = binarySearch(arr, key);
32     if (index != -1) {
33         System.out.println("Element found at index " + index);
34     } else {
35         System.out.println("Element not found");
36     }
37 }
```

Console Output: Element found at index 4