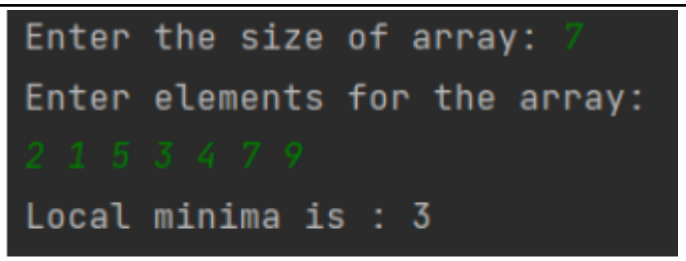
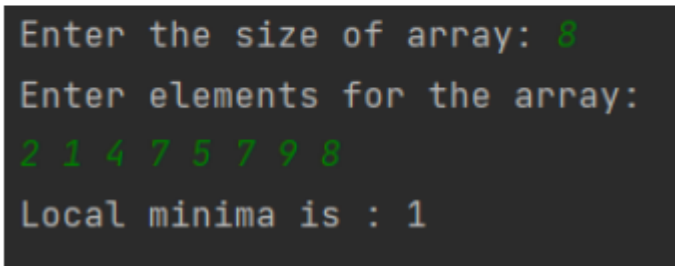
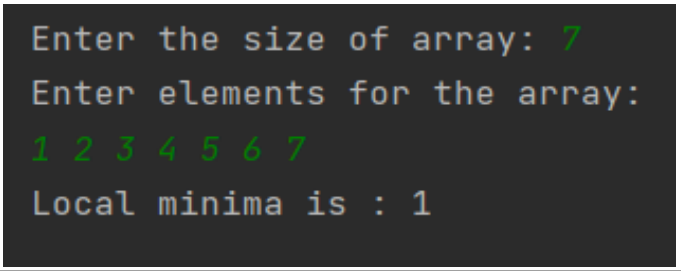
**Bharatiya Vidya Bhavan's**
**SARDAR PATEL INSTITUTE OF TECHNOLOGY**
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of AI-ML

| | |
|---|---|
| **Experiment** | 2 |
| **Aim** | Understand sorting algorithms based on Divide and Conquer approach |
| **Objective** | 1) Learn Divide and Conquer strategy<br>2)Find any one local minima in a given array |
| **Name** | Anurag Nair |
| **UCID** | 2022600035 |
| **Class** | CSE AIML |
| **Batch** | C |
| **Date of Submission** | 05/02/2024 |

| | |
|---|---|
| **Algorithm and Explanation of the technique used** | 1.Accept the size of the array from the user.<br><br>2.Accept elements for the array.<br><br>3.The localMinima function takes the array, its size, start index, and end index as parameters.<br><br>4.Compute the middle index m as (start + end) / 2.<br><br>5.Check if the middle element is a local minima:<br>If m is at the beginning (m == 0) and the element at m is less than the one following it (arr[m + 1]), return arr[m].<br>If m is at the end (m == size - 1) and the element at m is less than the one preceding it (arr[m - 1]), return arr[m].<br>If m is neither at the beginning nor at the end, and the element at m is less than both its neighbors, return arr[m].<br><br>6.If the middle element is not a local minima:<br>If the element to the left of m is smaller, recursively search the left half (start to m - 1).<br>If the element to the right of m is smaller, recursively search the right half (m + 1 to end).<br><br>7.If no local minima is found, return -1.<br><br>8.End<br><br>**Divide-and-Conquer Approach:**<br>The function localMinima takes an array, its size, and the start and end indices. It calculates the middle index and checks if the middle element is a local minima. |

| | |
|---|---|
| | If the middle element is not a local minima, it recursively searches in the left or right half of the array based on which side has a smaller neighboring element.<br>The recursion terminates when a local minima is found or when the search space reduces to a single element. |
| **Program(Code)** | |

```
import java.util.Scanner;
public class Main {
   static int localMinima(int[] arr, int size, int start,
int
            end){
        int m=(start+end)/2;
        if(m==0 && arr[m]<arr[m+1]){
            return arr[m];
        }
        else if(m==size-1 && arr[m-1]<arr[m]){
            return arr[m];
        }
        else if((m!=0 && m!=size-1) && (arr[m]<arr[m-1] &&
                arr[m]<arr[m+1] ) ){
            return arr[m];
        }
        else if (m!=0 && arr[m-1]<arr[m]) {
            return localMinima(arr, size,0,m-1);
        }
        else if (m!=size-1){
            return localMinima(arr, size,m+1,size-1);
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int size = sc.nextInt();
        int arr[]=new int[size];
        System.out.println("Enter elements for the array:");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Local minima is :
                "+localMinima(arr, size,0,size-1));
    }
}
```

| | |
|---|---|
| **Output** |  <br>  <br>  |
| **Justification of the complexity calculated** | **TIME COMPLEXITY:**<br><br>At each step of the recursion, the array size reduces by half: n/(2^k) , where k is the depth of recursion.<br><br>After k steps of halving, the size of the array becomes 1: n/(2^k)=1<br><br>Solving for k, n=2^k, that is k=log(n) to base 2.<br>Therefore, the time complexity of the algorithm is O(log n).<br><br>Best Case Time Complexity: O(1) :- Constant time when the local minima is found in the first comparison.<br><br>Worst Case Time Complexity: O(log n) :- Logarithmic time when the local minima is located at one of the ends, requiring a complete binary search.<br><br>Average Case Time Complexity: O(log n) :- Logarithmic time when the local minima is found within the array not necessarily at the ends. |
| **Conclusion** | I learned about the divide and conquer algorithm. The code implements an algorithm to find a local minima within an array using a binary search approach.The algorithm divides the array into halves at each step of the recursion, efficiently reducing the search space until it identifies a local minima. Overall, the code demonstrates an effective implementation of the local minima search algorithm. |