Government College of Engineering, Jalgaon
(An Autonomous Institute of Government of Maharashtra)

Name :	**Semester :** VI	**PRN :**
Class : T. Y. B.Tech Computer	**Academic Year :** 2024-25	
Course Teacher : Mr. Vinit Kakde	**Subject :** CO356U DAA Lab	
Date of Performance :	**Date of Completion :**	

Practical no. 4

Aim : Implement algorithms using divide and conquer approach (finding min-max problem) .

Min-Max :-

The **Min-Max Divide and Conquer** algorithm is a technique used to find both the **minimum** and **maximum** values in an array by breaking down the problem into smaller subproblems. The basic idea behind the divide and conquer approach is to divide the array into smaller chunks, solve the problem for each chunk, and then combine the results.

Key Concepts:

1. **Divide**:
   - Split the array into two halves (or smaller subarrays).
2. **Conquer**:
   - Solve the problem (find the minimum and maximum) for each subarray.
3. **Combine**:
   - Combine the results from the two subarrays to get the global minimum and maximum.

Complexity Analysis of Min-Max:-

The function findMinMax() follows the Divide and Conquer approach, where the array is divided into two halves recursively.

- RecurrenceRelation:

Each recursive call splits the array into two halves:
T(n)=2T(n/2)+O(1)

- Solving this recurrence using Master's Theorem,
  - o a=2 (number of subproblems)
  - o b=2 (array is divided into half each time)
  - o f(n)=O(1) (constant work done at each level)

Since f(n)=O(1), we apply the case where f(n) = O(n^c) and c = 0, so the complexity is:O(n)
Thus, the time complexity is:O(n)
Space Complexity Analysis:

- The algorithm uses recursive calls, which require stack space.
- Since the recursion depth is $\log_2(n)$ (because we divide the array into halves), the space complexity due to function calls is: O(logn)
- Apart from recursion, we only use a few constant variables, which take O(1) space.

Thus, the overall space complexity is:O(logn)

Best Case Time Complexity	O(n)
Worst Case Time Complexity	O(n)
Average Case Time Complexity	O(n)
Space Complexity	O(log n)

Min-Max Algorithm :-

```
1. Function FindMinMax(arr, low, high):

2. If low == high:

     a. Return (arr[low], arr[low]

3. If high == low + 1:a. If arr[low] < arr[high]:
          - Return (arr[low], arr[high])
```

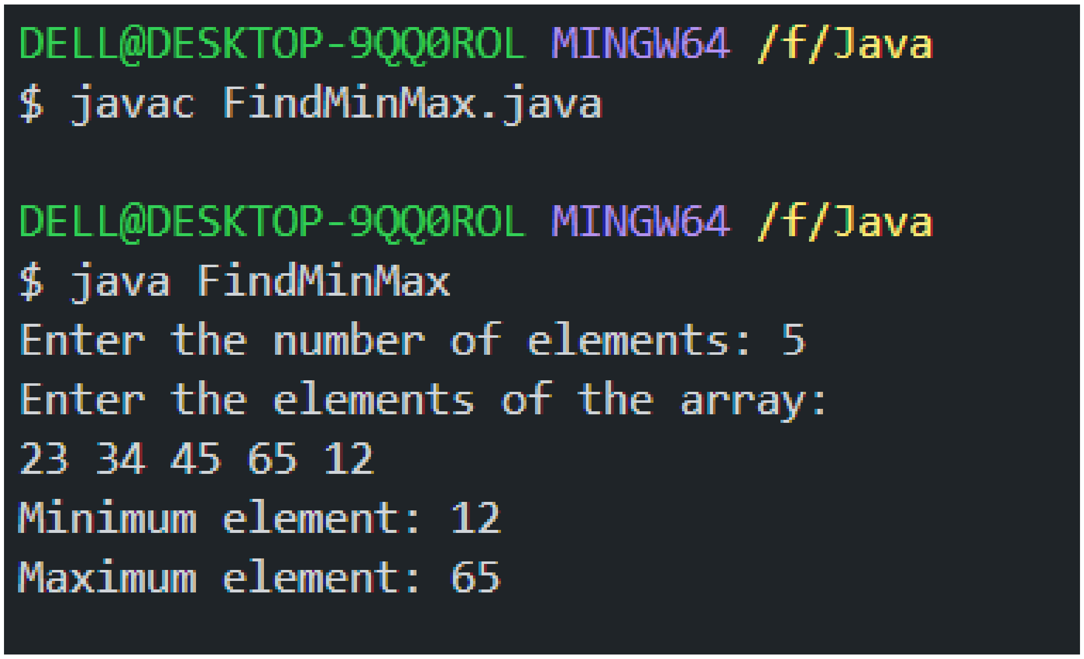
```
        b. Else:
           - Return (arr[high], arr[low])
 4. Else:
        a. Find mid = (low + high) / 2
        b. Recursively call FindMinMax(arr,
        low, mid) → Get min1, max1
        c. Recursively call FindMinMax(arr, mid+1, high) → Get min2, max2
        d. Set overall min = min(min1, min2)
        e. Set overall max = max(max1, max2)
        f. Return (overall min, overall max)
```

Program:

```
Import java.util.Scannr;

class MinMax {
    int min, max;

    MinMax(int min, int max) {

        this.min = min;

        this.max = max;

    }

}


public class FindMinMax {

    public static MinMax findMinMax(int[] arr, int low, int high) {

        if (low == high) {

            return new MinMax(arr[low], arr[low]);

        }


        if (high == low + 1) {

            if (arr[low] < arr[high]) {

                return new MinMax(arr[low], arr[high]);

            } else {

                return new MinMax(arr[high], arr[low]);
```

```java
            }
        }

        int mid = (low + high) / 2;
        MinMax leftResult = findMinMax(arr, low, mid);
        MinMax rightResult = findMinMax(arr, mid + 1, high);

        int min = Math.min(leftResult.min, rightResult.min);
        int max = Math.max(leftResult.max, rightResult.max);

        return new MinMax(min, max);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        MinMax result = findMinMax(arr, 0, n - 1);

        System.out.println("Minimum element: " + result.min);
        System.out.println("Maximum element: " + result.max);

        sc.close();
    }
}
```

Output :-



```
DELL@DESKTOP-9QQ0ROL MINGW64 /f/Java
$ javac FindMinMax.java

DELL@DESKTOP-9QQ0ROL MINGW64 /f/Java
$ java FindMinMax
Enter the number of elements: 5
Enter the elements of the array:
23 34 45 65 12
Minimum element: 12
Maximum element: 65
```

Question :-

1. What is the time complexity of the Min-Max Divide and Conquer algorithm?

Answer: The time complexity is O(n), as each element is compared a constant number of times.

2. What is the space complexity of the Min-Max Divide and Conquer algorithm?

Answer: The space complexity is O(log n) due to the recursion stack.

3. How many comparisons does the Min-Max Divide and Conquer algorithm make in the worst case?

Answer: It makes 2n - 2 comparisons in the worst case

4. Can the Min-Max Divide and Conquer algorithm be modified to find only the minimum or maximum?

Answer: Yes, by focusing on either the minimum or maximum comparisons during recursion.

5. How does the algorithm handle arrays of size 1 or 2?

Answer: For size 1, the element is both min and max; for size 2, it directly compares the two elements.

Name and sign of teacher

Mr. Vinit Kakde