Department of Computer Science and Engineering (Data Science)

Experiment 2

(Divide and Conquer)

Aim: Implementations min-max algorithm using divide and conquer.

Theory:

Max-Min problem is to find a maximum and minimum element from the given array. We can effectively solve it using divide and conquer approach.

In the traditional approach, the maximum and minimum element can be found by comparing each element and updating Max and Min values as and when required. This approach is simple but it does (n – 1) comparisons for finding max and the same number of comparisons for finding the min. It results in a total of 2(n – 1) comparisons. Using a divide and conquer approach, we can reduce the number of comparisons.

Divide and conquer approach for Max. Min problem works in three stages.

- If a1 is the only element in the array, a1 is the maximum and minimum.
- If the array contains only two elements a1 and a2, then the single comparison between two elements can decide the minimum and maximum of them.
- If there are more than two elements, the algorithm divides the array from the middle and creates two subproblems. Both subproblems are treated as an independent problem and the same recursive process is applied to them. This division continues until subproblem size becomes one or two.

After solving two subproblems, their minimum and maximum numbers are compared to build the solution of the large problem. This process continues in a bottom-up fashion to build the solution of a parent problem.

Algorithm:

```
Algorithm DC_MAXMIN (A, low, high)
// Description : Find minimum and maximum element from array using divide and conquer approach
// Input : Array A of length n, and indices low = 0 and high = n - 1
// Output : (min, max) variables holding minimum and maximum element of array

if n == 1 then
  return (A[1], A[1])
else if n == 2 then
  if A[1] < A[2] then
    return (A[1], A[2])
  else
```

```
    return (A[2], A[1])
else
  mid ← (low + high) / 2
  [LMin, LMax] = DC_MAXMIN (A, low, mid)
  [RMin, RMax] = DC_MAXMIN (A, mid + 1, high)
  if LMax > RMax then   // Combine solution
    max ← LMax
  else
    max ← RMax
  end
  if LMin < RMin then    // Combine solution
    min ← LMin
  else
    min ← RMin
  end
  return (min, max)
end
```

Complexity:

Time complexity: O(n)

Example: