Experiment No. 4
Finding Maximum and Minimum
Date of Performance:07/03/2024
Date of Submission:14/03/2024

Experiment No. 4

Title: Finding Maximum and Minimum

Aim: To study, implement, analyze Finding Maximum and Minimum Algorithm using Greedy method

Objective: To introduce Greedy based algorithms

Theory:

Maximum and Minimum can be found using a simple naïve method.

1. **Naïve Method:**

Naïve method is a basic method to solve any problem. In this method, the maximum and minimum number can be found separately. To find the maximum and minimum numbers, thefollowing straightforward algorithm can be used.

The number of comparisons in Naive method is 2n - 2.

The number of comparisons can be reduced using the divide and conquer approach. Following is the technique.

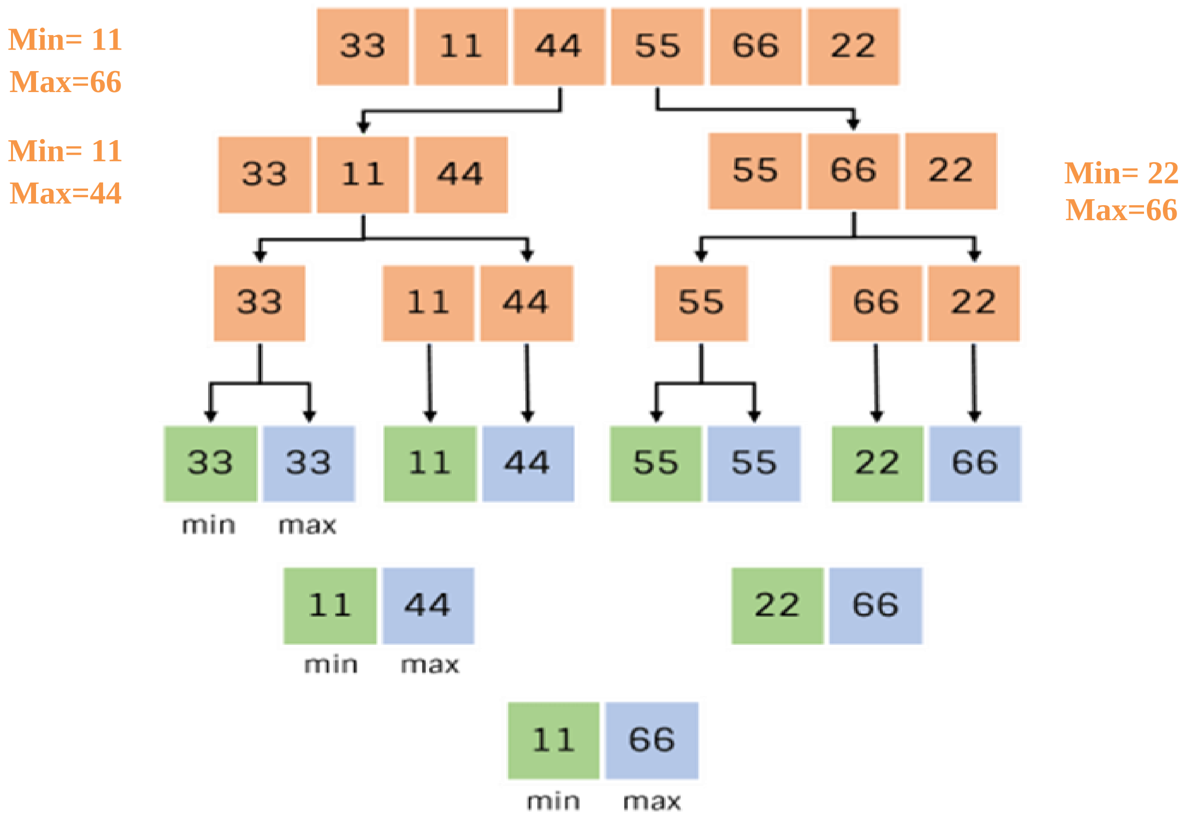
2. **Divide and Conquer Approach:**

In this approach, the array is divided into two halves. Then using recursive approach maximum and minimum numbers in each halves are found. Later, return the maximum oftwo maxima of each half and the minimum of two minima of each half.

In this given problem, the number of elements in an array is y−x+1, where y is greater than orequal to x.

DC_MAXMIN (A, low, high) will return the maximum and minimum values of an array numbers[x...y].

Example:



Logic used:

- The given list has more than two elements, so 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.
- If a_1 is the only element in the array, a_1 is the maximum and minimum.
- If the array contains only two elements a_1 and a_2, then the single comparison between two elements can decide the minimum and maximum of them.

Time Complexity:

The recurrence is for min-Max algorithm is:

$T(n) = 0, \quad \text{if } n = 1$

$T(n) = 1, \quad \text{if } n = 2$

$T(n) = 2T(n/2) + 2, \quad \text{if } n > 2$

$T(n) = 2T(n/2) + 2 \ldots (1)$

Substituting n by (n / 2) in Equation (1)

$T(n/2) = 2T(n/4) + 2$

$= T(n) = 2(2T(n/4) + 2) + 2$

$= 4T(n/4) + 4 + 2 \ldots (2)$

By substituting n by n/4 in Equation (1),

$T(n/4) = 2T(n/8) + 2$

Substitute it in Equation (1),

$T(n) = 4[2T(n/8) + 2] + 4 + 2$

$= 8T(n/8) + 8 + 4 + 2$

$= 2^3\, T(n/2^3) + 2^3 + 2^2 + 2^1$

……

$T(n)= 2^k\, T(n/2^k) + 2^k + \ldots.. + 2^2 + 2^1$

Assume $n/2^k=2$ so $n/2 = 2^k$

$T(n) = 2^k\, T(2) + (2^k + \ldots.. + 2^2 + 2^1)$

$T(n) = 2^k\, T(2) + (2^1 + 2^2 + \ldots.. + 2^k)$

Using GP formula : $GP = a(r^k - 1)/ (r-1)$

Here a = 2 and r = 2

$= 2^k + 2(2^k-1)/(2-1)$

$= 2^k + 2^{k+1}-2$ $\quad$ { Assume $n/2^k=2$ so $n/2 = 2^k$ and $n/2^{k+1} =2(n/2)=n$ }

$= n/2 + n - 2$

$= 1.5\, n - 2$

Time Complexity = O(n)

Algorithm:

DC_MAXMIN (A, low, high)

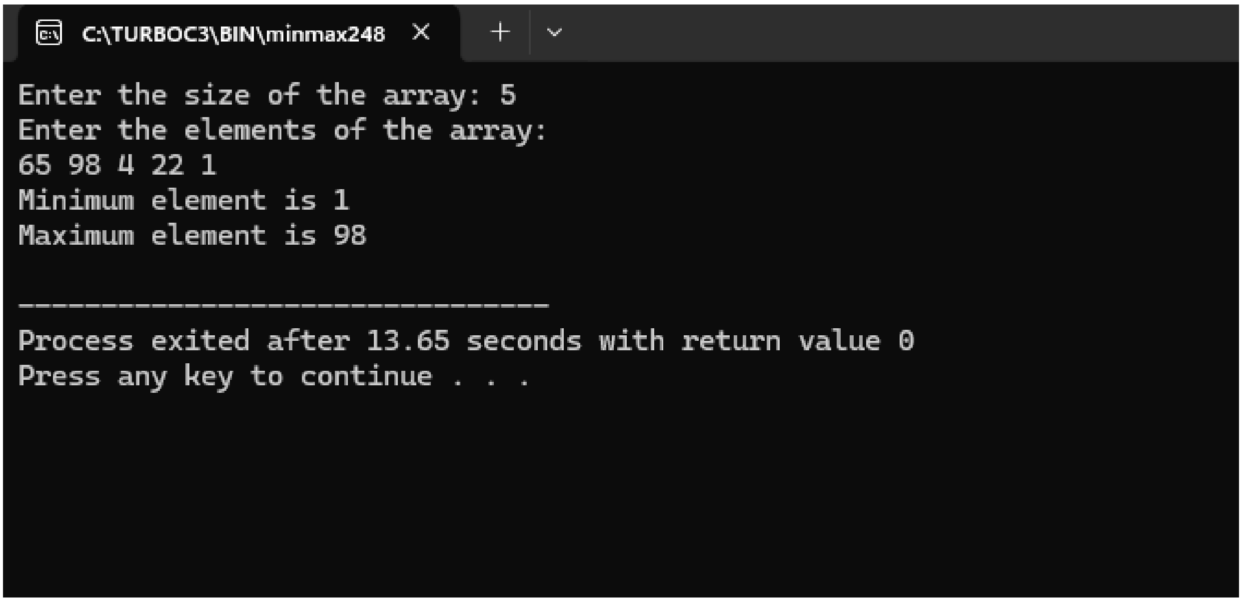
```
// Input: Array A of length n, and indices low = 0 and high = n-1
// Output: (min, max) variables holds minimum and maximum
if low = = high, Then          // low = = high
   return (A[low], A[low])
else if low = = high - 1 then       //low = = high - 1
         if A[low] < A[high] then
             return (A[low], A[high])
         else
             return (A[high], A[low])
     else
        mid ← (low + high) / 2
        [LMin, LMax] = DC_MAXMIN (A, low, mid)
        [RMin, RMax] = DC_MAXMIN (A, mid + 1, high)
        // Combine solution
        if LMax > RMax, Then
            max ← LMax
        else
            max ← RMax
  end
      if LMin < RMin, Then    // Combine solution.
          min ← LMin
      else
          min ← RMin
      end
   return (min, max)
end
```

Code:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
struct Pair {
   int min;
   int max;
};

struct Pair getMinMax(int arr[], int n) {
   struct Pair minmax;
   int i;

   if (n == 1) {
      minmax.max = arr[0];
      minmax.min = arr[0];
      return minmax;
   }

   if (arr[0] > arr[1]) {
      minmax.max = arr[0];
      minmax.min = arr[1];
   }
   else {
      minmax.max = arr[1];
      minmax.min = arr[0];
   }

   for (i = 2; i < n; i++) {
      if (arr[i] > minmax.max)
         minmax.max = arr[i];
      else if (arr[i] < minmax.min)
         minmax.min = arr[i];
   }

   return minmax;
}

int main() {
   int arr_size;
```

```
    printf("Enter the size of the array: ");
    scanf("%d", &arr_size);

    int arr[arr_size];

    printf("Enter the elements of the array:\n");
    for (int i = 0; i < arr_size; i++) {
        scanf("%d", &arr[i]);
    }

    struct Pair minmax = getMinMax(arr, arr_size);
    printf("Minimum element is %d\n", minmax.min);
    printf("Maximum element is %d\n", minmax.max);

    return 0;
}
```

Output:



```
C:\TURBOC3\BIN\minmax248

Enter the size of the array: 5
Enter the elements of the array:
65 98 4 22 1
Minimum element is 1
Maximum element is 98

--------------------------------
Process exited after 13.65 seconds with return value 0
Press any key to continue . . .
```

Conclusion:

In conclusion, the MIN MAX algorithm efficiently determines the minimum and maximum elements in an array by employing a divide-and-conquer approach. By recursively splitting the array into smaller subproblems and comparing pairs of elements, it achieves a time complexity of O(n). This algorithm is straightforward to implement and offers a practical solution for finding extreme values in arrays of various sizes, making it a valuable tool in algorithmic problem-solving