|  |
| --- |
| **Experiment No. 4** |
| **Finding Maximum and Minimum** |
| Date of Performance:07/03/24 |
| Date of Submission:13/03/24 |

## 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, the following 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.

* 1. **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 of two 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 or equal to x.

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

**Example:**

|  |  |  |
| --- | --- | --- |
| **Min= 11 Max=66**  **Min= 11 Max=44** |  | **Min= 22**  **Max=66** |

**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 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.

**Time Complexity:**

The recurrence is for min-Max algorithm is:

T(n) = 0, if n = 1

T(n) = 1, if n = 2

T(n) = 2T(n/2) + 2, if n > 2

T(n) = 2T(n/2) + 2 … (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 … (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

= 23 T(n/23) + 23 + 22 + 21

……

T(n)= 2k T(n/2k) + 2k +…..+ 22 + 21

Assume n/2k=2 so n/2 = 2k

T(n) = 2k T(2) + ( 2k +…..+ 22 + 21)

T(n) = 2k T(2) + (  21 + 22 +…..+ 2k )

Using GP formula : GP = a(rk – 1)/ (r-1)

Here a = 2 and r = 2

= 2k + 2(2k-1)/(2-1)

= 2k + 2k+1-2 { Assume n/2k=2 so n/2 = 2k and n/2k+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);

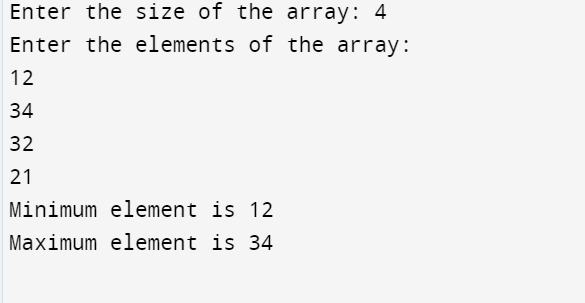
printf("Minimum element is %d\n", minmax.min);

printf("Maximum element is %d\n", minmax.max);

return 0;

}

**Output:**

****

**Conclusion:**

The minimum-maximum algorithm efficiently determines both the minimum and maximum elements in a dataset with optimal time complexity of O(n). Its simplicity lies in a single pass through the data, updating minimum and maximum values as it iterates. This approach minimizes computational overhead and memory usage, making it practical for large datasets. Versatile and adaptable to various data structures, the algorithm's straightforward logic ensures ease of implementation in diverse programming environments. In summary, the minimum-maximum algorithm provides an elegant and effective solution for extremum finding tasks in data structures.