To find maximum and minimum element from a list of numbers without using recursion.

Raushan Raj - IIT2018031, Rithik Seth - IIT2018032, Mrigyen Sawant - IIT2018033
Btech IT Department, Semester-4

Indian Institute Of Information Technology, Allahabad
18 March 2020

Abstract: ***In this paper, we have devised an optimized non-recursive algorithm to find the*** **maximum and minimum element from a list of numbers without using recursion.** ***Later we have discussed its space-time complexity through both apriori and aposteriori analysis.***

Keywords: Arrays, Searching, Numbers.

1. Introduction

Here we are assuming the list of numbers be list[] whose size is n where n > 0.

We start our approach for finding maximum and minimum element from a list of numbers after considering the following conditions. We are assuming the list is not sorted and in case it is sorted then the maximum and minimum element of that particular list will be the max(list[0],list[n-1]) , min(list[0],list[n-1]) respectively which is pretty simple and straightforward.

The simplest/naive approach that will strike the mind first where the list isn't sorted will be the linear search approach where we initialize the MAX and MIN to be the max(list[0],list[1]),min(list[0],list[1]) respectively, and then we compare each element starting from 3rd element with MIN and MAX and change their values accordingly.

In this paper we have devised an optimized non-recursive algorithm where we are assuming the list isn't sorted with minimum number of comparisons possible to reduce its time complexity as much as possible. Later we have discussed its space-time complexity through both apriori and aposteriori analysis.

2. Algorithm Proposed

Assumptions: Here we are assuming the list of numbers be list[] whose size is n where n > 0.

Algorithm: To find the maximum and minimum element from a list of numbers without using recursion.

Input: The first line will contain a positive number n (the size of list[]) i.e. n > 0. The second line will contain n numbers separated by space.

Output: Print maximum and minimum element of the input list.

Step 1: Start

Step 2: Read n. Print “Invalid input” if n ≤ 0 and go to step 8.

Step 3: Initialize I ← 0 , i ← -1 and list[] ← {}.

Step 4: Repeat these steps until I is equal to n.

4.1: Read list[I].

4.2: I ← I + 1

Step 5: If n is even then,

```
Initialize i ← 2
If list[0] >= list[1] then,
    Initialize MAX ← list[0]
               MIN ← list[1]
else
    Initialize MAX ← list[1]
               MIN ← list[0]
else
    Initialize MAX ← list[0]
               MIN ← list[0]
               i ← 1
```

Step 6: Repeat these steps till i < n-1

```
if list[i] > list[i+1] then,
    if list[i] > MAX then,
        MAX ← list[i]
    if list[i+1] < MIN then,
        MIN ← list[i+1]
else
    if list[i+1] > MAX then,
        MAX ← list[i+1]
    if list[i] < MIN then,
        MIN ← list[i]
i ← i + 2
```

Step 7: Print MAX and MIN respectively.

Step 8: Stop

In the above algorithm we are reducing number of comparisons by comparing in pairs and updating the MAX and MIN by comparing the list value pair i.e. (list[i],list[i+1]) with the current (MAX,MIN) pair. Hence, certain unnecessary comparisons can be avoided resulting reduced time complexity than trivial linear search.

3. Analysis and Experimental Results

3a. Apriori Analysis

Time complexity analysis:

	Time	Best F	Freq
Step 1	1	1	1
Step 2	1	1	1
Step 3	3	1	1
Step 4	3n	1	1
Step 5	5	1	1
Step 6	7(n-1)/2 + (n+1)/2	1	1
Step 7	2	1	1
Step 8	1	1	1

Worst case complexity:

$T_W \propto 1\times1 + 1\times1 + 3\times1 + 3n \times 1 + 5\times1 + (7(n-1)/2 + (n+1)/2)\times1 + 2\times1 + 1\times1$

$T_W \propto 5 + 3n + 5 + (4n - 3) + 3$

$T_W \propto 7n + 10$

$T_W \propto n$

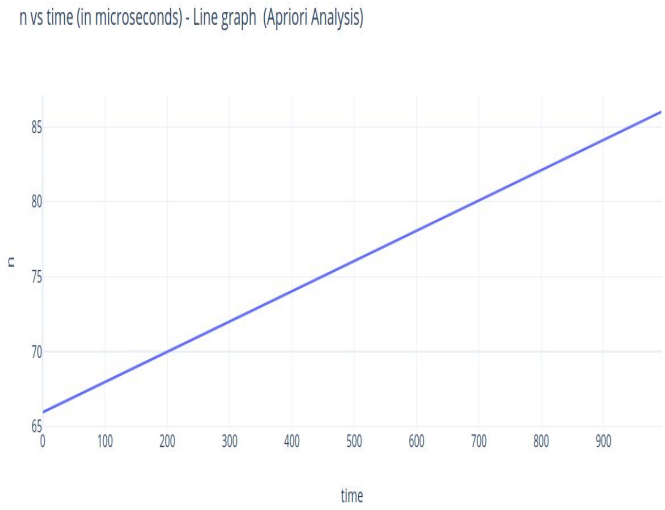
O(n)

Best and average case complexities:

The best case complexity will be same as worst since the list will be traversed till the end

irrespective of the values of the list but depend on the size of the list. Moreover, since best and worst case complexities are equal then average case complexity will also be same as that of best or worst complexity.

$O(n) \cong \Omega(n) \cong \Theta(n)$



The above line graph shows n vs time relation according to apriori analysis (Theoretical).

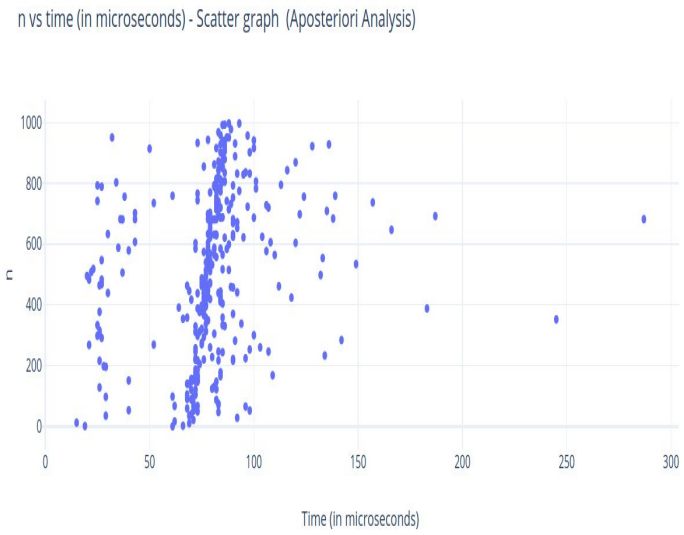
Space Time Complexity:

The above algorithm consumes 2 units of memory for min and max elements.
Therefore, the overall space-time complexity will be :

$M \propto 2$
O(1)

3b. Aposteriori Analysis

In the below scatter graph (Experimental), the x-axis represents the time taken in microseconds, y-axis represents n (number of list elements). The graph obeys linearity up to certain extent similar to that of apriori analysis.



Time complexity analysis:

n	Time (in microseconds)
253	98
692	187
67	62
379	74
282	91
558	78

4. Conclusion

From the above analysis, the optimized approach shows worst case and best case complexities of **O(n)** and **Ω(n)**. Also, the average case complexity is **$\Theta(n)$. This approach is slightly better than the naive linear search since it executes lesser comparison statements than the usual linear approach and hence, performs relatively faster and efficiently.**