Design Analysis of Algorithm Assignment-3

Kaushal - IIT2019030,
Sarvesh - IIT2019031,
Aarushi - IIT2019032

Abstract

In this paper we have to do a max/min search so here we will use a randomly generated array and find the minimum and maximum element from the array. Here, we have discussed the complexities of the different algorithm used for the problem, along with experimental analysis.

1 Keywords

Minimum, Maximum, Array, Sorting, Heap.

2 Introduction

According to the given problem statement, we had to find the maximum and minimum element of the array. Here we have to apply search for the maximum and the minimum of all the element in the array.This problem can be solved in the two different methods i.e Recursive and Iterative, but as stated in the problem we have only discussed the iterative version of the algorithm. So, basically for this problem we can have many algorithm but only the most popular and the most acceptable algorithm are discussed in the paper. Also we have discussed the time and space complexity of all the different algorithms used and accordingly came up with the best suitable algorithm for the problem.

3 Application

lot of problems in different practical fields of Computer Science, Database Management System, Networks, Data Mining and Artificial intelligence. Searching is common fundamental operation and solve to searching problem in a different formats of these field[**4**]. Search algorithms aim to find solutions or objects with specified properties and constraints in a large solution search space or among a collection of objects. A solution can be a set of value assignments to variables that will satisfy the constraints or a sub-structure of a given discrete structure. In addition, there are search algorithms, mostly probabilistic, that are designed for the prospect[**3**].There are a number of real life examples also where searching plays a very important role, for example, in a library system , we search for a particular category of books or in our phone book we search for any particular user on the basis of their names or self determined properties. Therefore, an efficient searching algorithm is preferred.

4 Algorithm

For the given problem statement, we have prepared three algorithm designs. All of these three algorithms are independently and briefly discussed in the below subsections:-

4.1 Algorithm 1

The first algorithm is the brute force one. In this algorithm we have to just iterate through the whole array and keep track of the minimum and maximum element in the array and in the last when we have done the complete iteration of the array, we will output the maximum and minimum element of the randomly generated array.

4.2 Algorithm 2

The second one is based on the sorting algorithm [**1**]. So we have used the insertion sort to sort the contents of the array. After the sorting, we will output the first element of the array as minimum and last element of the array as the maximum element.

4.3 Algorithm 3

The third algorithm is basically a application of the heap. To find the maximum element of the array, we will insert the all the element of the array in the min heap[**2**]. Since the min heap readjust itself to maintain the maximum element on the top and we will output it. For minimum element, we will use the max heap and will do as stated in the process while finding the maximum element.Also heap is implemented with the help of the binary tree.

5 Pseudo codes

5.1 Brute Force:

```
function MAIN()
Get n
a[n+1]
Get random inputs
mx ← 0
mn ← Infinity
for i ← 0to(n − 1) do

if a[i] > mx then
   mx ← a[i]

end if

if a[i] < mn then
   mn ← a[i]

end if
i ← i + 1

end for
print("MAXIMUM ELEMENT")
print(mx)
print("MINIMUM ELEMENT")
print(mn)
return 0
```

5.2 By Sorting:

```
function insertionSort(arr[],n)
i,key,j
for i ← 1to(n − 1) do
key ← arr[i]
j ←i - 1
while j ← 0&arr[j] ¿ key do
arr[j + 1] ← arr[j]
j ← j − 1
end while
arr[j + 1] ← keyi + 1
ENDfor
return

function answer(A[],size)
print("MAXIMUM ELEMENT")
print(A[size-1])
print("MINIMUM ELEMENT")
print(A[0])
return

function Main()

Get n
a[n+1]
Get random inputs
insertionSort(a,n)
answer(a,n)
return 0
```

5.3 By Heap:

```
function Main()
Get n
a[n+1]
Get random inputs
mx ← Max_priority_rqueue
mn ← Min_priority_queue
for i← 0to(n − 1) do
mn.push(k)
mx.push(k)
i+1
END for

print("MAXIMUM ELEMENT")
print(mx.top())
print("MINIMUM ELEMENT")
print(mn.top())
return 0
```

6 Complexity analysis

6.1 Time Complexity:

Calculating time complexities of individual algorithms.

6.1.1 Algorithm 1: Using for Loop

Here, we go through the complete array and checks for the smallest or the largest number as compared to the previously initialised variables to store the maximum and the minimum elements. So, as we go through the array once completely the time complexity of this algorithm is O(N).

6.1.2 Algorithm 2: Using insertion sort

In this method we have used insertion sort method to sort the array in ascending order, so that the largest and the smallest elements can be accessed in O(1). The best case complexity occurs if the array is already in sorted order, insertion sort compares O(n) elements and performs no swaps. So, the best case time complexity is $\Omega(N)$. The worst case time complexity for insertion sort is $N(N-1) \approx O(N^2)$

6.1.3 Algorithm 3: Using heap

Here we have created a min heap for the generated array. Therefore we can get the the maximum and minimum elements separated easily. The time complexity for generating heap is O(Nlog(N)), so the time complexity of this algorithm is also O(Nlog(N)).

6.2 Space Complexity:

6.2.1 Algorithm 1: Using for Loop

As for this algorithm we are not using any memory so the space complexity for this is O(1).

6.2.2 Algorithm 2: Using Insertion Sort

Insertion sort is a stable sort with a space complexity of O(1).

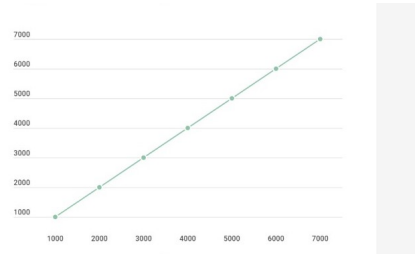
6.2.3 Algorithm 3: Using Heap

As for this algorithm we are using two heaps which stores all the elements of the array so the space complexity for this is O(2N) which is generally written as O(N).

7 Experimental Study

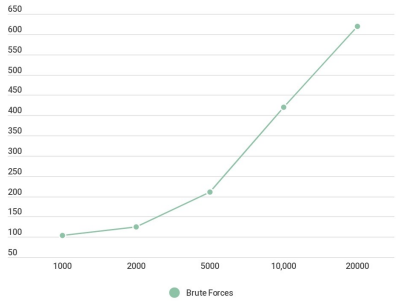
7.1 Apriori Analysis:

Apriori analysis means,analysis is performed prior to running it on a specific system.This analysis is a stage where a function is defined using some theoretical model.Hence,we determine the time and space complexity of an algorithm by just looking at the algorithm rather than running it on a particular system with a different memory,processor, and compiler.So,as we discussed under the heading complexity analysis we arrived at the conclusion that the best time complexity is O(n).



7.2 Aposteriori Analysis:

Aposteriori analysis of an algorithm means we perform analysis of an algorithm only after running it on a system.It directly depends on the system and changes from system to system.So for the a aposteriori analysis of the algorithm,we have run our code on the compiler and get values of the time.



Value of n	Time(in ms)
1000	104
2000	125
5000	210
10,000	420
20,000	620

8 Conclusion

Through this problem, we learnt about finding the minimum and maximum element of the array using the different algorithm. Also we have the learnt the application of the insertion Sort, Heap algorithms. Also learnt about the deducing the time and space complexity of the divide and conquer algorithms.

9 References

1.Insertion Sort
2.Heap
3.Searching Algorithms
4.Application of searching