Design and Analysis of Algorithm for Sorting an Array by Finding the Max and Min elements using Divide and Conquer Method and Placing them at their Correct Position at every Recursive Call

Nairit Banerjee
IIT2016505

Kamal Nayan Chaubey
IIT2016510

Ashutosh Vishwakarma
IIT2016017

Mandeep Chakma
IIT2016012

March 30, 2018

Abstract

This document is a final report of our study on sorting of a given array by continuously tracking the max and min elements at every recursive step and placing the max and min elements at their correct position. The max and min elements should be found using divide and conquer approach.

Keywords: *Divide and Conquer, Recursion, Maximum, Minimum, Mid*

1 Introduction And Literature Survey

In mathematics and computer science,the divide and conquer approach plays an important role in solving some big and tough problems.
Divide and Conquer works in two parts:
(1)Divide : At first a large problem is divided into many small(sub) problems of the same type.
(2)Conquer : Then, recursively solve these sub problems and combine them to find an overall solution of the initial problem.
Our task is to sort an array by finding the max and min element at each step recursively and placing them at their correct position.Solution goes as follows:
(1)Divide : we will divide the array till each partition has exactly two elements remaining in them.
(2)Conquer : Then with these two elements in each partiton we get min and max element,and recursively we combine the results of all the partition and we get the min and max element of the array.
Thus we put them in their correct position and reduce the size of array by two and implement the same startegy in the remaining array till the whole array gets exhausted. In this way we get the sorted array.
For example : we have array := 4,12,16,20,2,9. Divide part will break it into following sub parts : 4,12 , 16,16 , 20,2 , 9,9. Conquering part will give us the result as min = 2 , max = 20. So we place 2 at 0 index and 20 at 5th index.Now the same will be repeated for the array of size reduced by 2,one from each end. Similar procedure will place 4 at 1 and 16 at 4 and at last conquer 12 will be placed at 3 and 9 at 2. So our final sorted array will look like 2,4,9,12,16,20.

2 Algorithm Design

Firstly we took the complete array and finded the maximum and minimum elements and their index in the array. Now we swap the minimum element from the first position in the array and the maximum element is swapped from the last position in the array. Now the same procedure is repeated for the array with starting index incremented by 1 and the ending index of the array decremented by 1. So basically the size of the array is reduced by 2. Basically for finding the maximum and minimum element and their index we use divide and conquer algorithm and the same is shown in the function max_range and min_range respectively. This divide and conquer uses recursion.

3 Psuedo code for every function used in the algorithm

Algorithm 1 MIN_RANGE

```
procedure MIN_RANGE($(l, r)$)
    if $l == r$ then
        if $a[l] < min$ then
            $min \leftarrow a[l]$
            $min\_i \leftarrow l$
        end if
        return a[l]
    else if $l == r - 1$ then
        if $a[l] < a[r]$ then
            if $a[l] < min$ then
                $min \leftarrow a[r]$
                $min\_i \leftarrow r$
            end if
        return a[l]
        end if
        if $a[r] < min$ then
            $min \leftarrow a[r]$
            $min\_i \leftarrow r$
        end if
        return a[r]
    end if
    $mid \leftarrow (l + r)/2$
    $x \leftarrow$ MIN_RANGE(l,mid)
    $y \leftarrow$ MIN_RANGE(mid+1,r)
    if $x \leq y - 1$ then
        return x
    end if
    return y
end procedure
```

Algorithm 2 MAX_RANGE

```
procedure MAX_RANGE($(l, r)$)
    if $l == r$ then
        if $a[l] > max$ then
            $max \leftarrow a[l]$
            $max\_i \leftarrow l$
        end if
        return a[l]
    else if $l == r - 1$ then
        if $a[l] > a[r]$ then
            if $a[l] > max$ then
                $max \leftarrow a[l]$
                $max\_i \leftarrow l$
            end if
        return a[l]
        end if
        if $a[r] > max$ then
            $max \leftarrow a[r]$
            $max\_i \leftarrow r$
        end if
        return a[r]
    end if
    $mid \leftarrow (l + r)/2$
    $x \leftarrow$ MAX_RANGE(l,mid)
    $y \leftarrow$ MAX_RANGE(mid+1,r)
    if $x \geq y + 1$ then
        return x
    end if
    return y
end procedure
```

Algorithm 3 Main_Program

```
procedure MAIN
    $l \leftarrow 0$
    $r \leftarrow n - 1$
    while $l < r$ do
        MIN_RANGE$(l, r)$
        MAX_RANGE$(l, r)$
        $min \leftarrow 1000000000$
        $max \leftarrow -1000000000$
        if $min\_i \neq l$ then
            $x \leftarrow a[min\_i]$
            $a[min\_i] \leftarrow a[l]$
            $a[l] \leftarrow x$
            if $l == max\_i$ then
                $max\_i \leftarrow min\_i$
            end if
        end if
        if $min\_i \neq r$ then
            $x \leftarrow a[max\_i]$
            $a[max\_i] \leftarrow a[r]$
            $a[r] \leftarrow x$
        end if
        $l \leftarrow l + 1$
        $r \leftarrow r - 1$
    end while
    print the final sorted array
end procedure
```

4 Analysis and Discussions

4.1 Time Complexity

The time complexity for the above stated algorithm can be computed by adding units of time taken by each statement multiplied by the number of times the statement is executed. Hence,

$T = \Sigma(cost(i) * no.of.times(i))$,for all statements

The running time will be dependent on the number of swaps required to sort the given array. The maximum and minimum element are comuted using Divide and Conquer irresptive of the type of input.

$$T_{best} \propto (n/2 + 1)(28 + 4n + 4)$$

$$T_{avg} \propto (n/2+1)(28+4n+7c_1+4) \text{ where } c_1 < n$$

$$T_{worst} \propto (n/2 + 1)(28 + 4n + 7n + 4)$$

Hence the order of growth for best case, average case and worst case are

$$\Omega(n^2), \Theta(n^2), O(n^2) \text{ respectively.}$$

We observe that in case of worst case, maximum swaps ae required. In average case, not all elements are swapped. Whereas for best case, no swaps are required.

By varying the m,

Table 1: Data for plotting running time graph

n	T_{best}	T_{avg}	T_{worst}
5	75	79	95
7	129	149	165
11	298	323	362
15	508	529	613

We will get the following running time graph for n ranging from 5 to 15. The x-axis represents the number of elements in the array and y-axis represents the units of time taken.

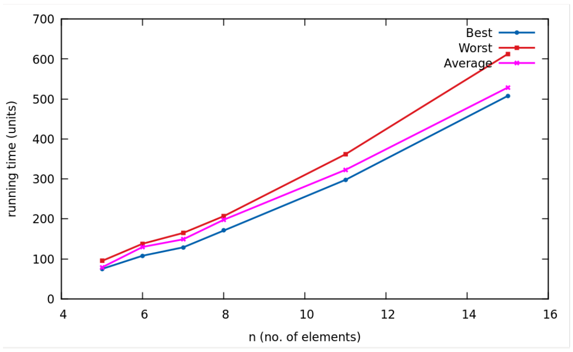

Figure 1: Running time graph.

4.2 Space Complexity

Apart from a few variables, we are not using any extra space. All swapping are done on the input array itself. Hence,

$$Space(n) \propto 1$$

5 Experimental Study

We may assume the following examples to study the various running times possible,

Best Case: If the input array is already sorted, no swapping is required, hence it results to best case. For example, 1, 2, 3, 4, 5

Average Case: When some but not all elements of an array are misplaced(not in sorted order), we need some swappings to be done. Hence it results to average case. For example 1, 4, 2, 5, 3 required 3 swaps in order to get sorted.

Worst Case: When the array is in complete un-sorted order(descending in our case), every element is required to be swapped to its correct position. Hence it results into worst case. For example 5, 4, 3, 2, 1 requires 4 swaps to get sorted.

Thus we see that even if order of growth of the algorithm is same for all three cases, running time differs because of the number of swaps required for each.

6 Conclusion

In this paper we proposed an algorithm to sort a given array by finding the max and min element at each step recursively where max and min element is found using divide and conquer approach.

Divide and conquer approach is used in following algorithms : Binary Search, Merge Sort, Quick Sort, Closest Pair of Point, Strassen's Algorithm, Cooley–Tukey Fast Fourier Transform (FFT) algorithm, Karatsuba algorithm for fast multiplication

References

[1] Introduction To Algorithms (Second Edition) by Thomas H.Cormen, Charles E.Leiserson , R.L.Rivest and Clifford Stein.

[2] How to Solve it by Computer by R.G.Dromey.

[3] The C Programming Language by Brian Kernighan and Dennis Ritchie.

[4] Divide and Conquer - GeeksforGeeks: https://www.geeksforgeeks.org/divide-and-conquer-introduction/

[5] Finding max and min using divide and conquer approach: https://stackoverflow.com/questions/35748297/finding-max-and-min-using-divide-and-conquer-approach