GROUP 51 - Given a set of numbers, write an efficient algorithm, using Heap Sort, to report both the largest and the smallest elements of the Heap. Trace the movement of every element of the Heap during the execution of your algorithm.

IHM2016501[1] IHM2016005[2] BIM2016004[3] IIM2016006[4] RIT2015050[5]

I. INTRODUCTION AND LITERATURE SURVEY

A Heap is a partially sorted binary tree. Although a heap is not completely in order, it conforms to a sorting principle: every node has a value less than or greater than either of its children. Additionally, a heap is a ”complete tree” – a complete tree is one in which there are no gaps between leaves.Min-Heap Where the value of the root node is less than or equal to either of its children.Max-Heap Where the value of the root node is greater than or equal to either of its children.Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end.In max-heaps, maximum element will always be at the root and in min-heaps minimum element will always be at the root. Heap Sort uses this property of heap to sort the array. We repeat the same process for remaining element.In the question we are given an array of n elements and we have to find the minimum and maximum elements of that array using the heap sort and also trace the path of each and every element during the above mentioned process.

II. ALGORITHM DESIGN

This part of the report can be further divided into following sections.

A. Input Format

In first few lines of the code we use $rand()$ function to generate a random number and assign it to the variable n which defines the size of the array.And then we use $rand()$ function to generate n random numbers and store them in the array.So.we now have a array of size n filled with n random numbers.

- A structure $heap$ has been used which has an element of type integer to stores the key and value of type integer which stores the value present in the array.
- We have a array $a[]$ of the above mentioned structure type and also a variable temp(which is used for swapping).
- Each element of the array has a unique key value.
- This Structure has been declared globally.
- Also the variables pos[][],realn, iteration are declared globally

B. Algorithm

Algorithm 1 prints the final array

```
1: procedure PRINTARRAY($n$)
2:     for i=0 to n-1 do
3:         $arr[i].value$
```

Algorithm 2 Swaps two position of heap

```
procedure SWAP($i, j$)
    for index= 0 to realn-1 do
        $pos[index][iteration + 1] \leftarrow pos[index][iteration]$
        $temp \leftarrow arr[i]$
        $arr[i] \leftarrow arr[j]$
        $arr[j] \leftarrow temp$
        $iteration++$
        $pos[arr[i].key][iteration] \leftarrow i$
        $pos[arr[i].key][iteration] \leftarrow j$
```

Algorithm 3 HeapSort Function

```
procedure HEAPSORT(N)                                ▷
    for i=0 to n/2-2 do
        $heapify(n, i)$
    for i=0 to n-2 do
        $swap(0, i)$
        $heapify(i, 0)$
```

Algorithm 4 Prints the track of each element in each iteration

```
procedure TRACE($n$)                                 ▷
    print(———Track of each element)
    for i=0 to n-1 do
        for j=0 to n-1 do
            if arr[j].key == i then
                print(Printing index of arrr[j].value at each iteration: )
        for j=0 to n-1 do print(pos[i][j])
```

Algorithm 5 Function to build max heap

```
procedure HEAPIFY($n, i$)
    $largest \leftarrow i$
    $l \leftarrow 2 * i + 1$
    $r \leftarrow 2 * i + 2$
    if l <n & arr[l].value >arr[largest].value then $largest \leftarrow l$
    if r <n & arr[l].value >arr[largest].value then $largest \leftarrow r$
    if largest != i then
        $swap(i, largest)$
        $heapify(n, largest)$
```

Algorithm 6 Main Function

```
procedure MAIN                                       ▷ main function
    $realn \leftarrow n$
    $iteration \leftarrow 0$
    for i=0 to n-1 do
        $arr[i].value \leftarrow a[i]$
        $arr[i].key \leftarrow i$
        $pos[i][iteration] \leftarrow i$
    $heapSort(n)$
    $printArray(n)$
    $trace(n)$
    print(the max is: arr[n-1].value the min is: arr[0].value)
```

C. Output Format

As mentioned in the question we are sorting the array using heap sort and then we are tracing the path of each element and printing it.

III. ANALYSIS

A. Analysis

1) Analysis Of Heap Sort: The worst case and best case complexity for heap sort are both $O(nlogn)$. Therefore heap sort needs $O(nlogn)$ comparisons for any input array. Complexity of heap sort:

$=> O(nlogn)$

2) Best Case Analysis: Heap Sort is constructed by using Max heap .For the Max heap the best case occurs when the array is in descending order but from above the heap sort takes $O(nlogn)$ comparisons for both the best and worst case.As we need to track the movement of every element we need to run another loop having n iterations so now the complexity of over all algorithm is $O(n^2 + nlogn)$

$$=> time_{best} \propto n^2 + nlogn + K$$

$$=> O(n^2 + nlogn)$$

3) Worst Case Analysis : As discussed above in the worst case senario also the heap sort takes $O(nlogn)$ As we need to track the movement of every element we need to run another loop having n iterations so now the complexity of over all algorithm is $O(n^2 + nlogn)$

$$=> time_{worst} \propto n^2 + nlogn + K$$

$$=> O(n^2 + nlogn)$$

$$=> t_{best} = t_{worst}$$

4) Average Case Analysis : We know that

$$time_{best} <= t_{average} <= t_{worst}$$

But from above

$t_{best} = t_{worst}$

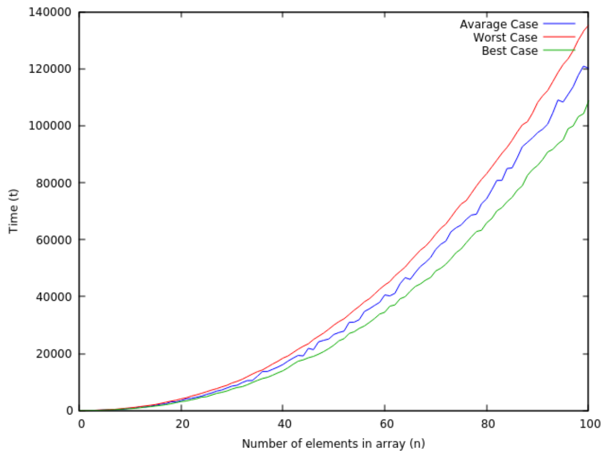
So

$=> t_{best} = t_{avg} = t_{worst}$

IV. EXPERIMENTALSTUDY

The best way to study an algorithm is by graphs and profiling.

1) Graphs-Time-Complexity: We have seen that $t_{best} = t_{avg} = t_{worst}$ So,the graphs for best case,average,worst case time complexity remain the same.We can clearly see that as though Brute force and efficient algorithm are of order n^2 efficient algorithm is a lot better than the other one.



2) Profiling:

$$NumberOfComputations$$

n	t_{avg}	t_{best}	t_{worst}
3	40	56	75
5	187	170	213
10	827	644	926
20	3448	3142	4078
50	26696	22926	29988
70	56588	49007	61943
80	74349	65902	83148
100	120230	108752	135122

- We can seen that $t_{best} = t_{avg} = t_{worst}$
- We can also see that time increases as the value of n increases and $time_{worst} \propto n^2 + nlogn$,$time_{avg} \propto n^2 + nlogn$,$time_{best} \propto n^2 + nlogn$.

V. DISCUSSIONS

A. *Understanding the Use Of File Handling and generation of random numbers*

In this context the file handling is used to generate the random test cases through which analysis and experimentation of algorithm becomes easier for handling different test cases. We have used $srand()$ and $rand()$ to generate the random values to fill the matrix with the help of $< time.h >$ header file.

B. *Tracing the content of every movement of every element*

- But tracking the movement of every element is not that easy as the array may have repetition of same digit.So,we provide unique key value to each and every element of the array and thus identify them uniquely.We then track the movement of the every element based on the value.

VI. Conclusion

- We can see that the Complexity of Heap Sort is $O(nlogn)$ in every case i.e irrespective of the input,But the complexity of over all algorithm turns out to be $O(n^2 + nlogn)$ as we need to track the movement of every element we need to run another loop having n iterations so now the complexity of over all algorithm is $O(n^2 + nlogn)$ in all the cases
- In this solution we have bounded the size of the matrix between 1 and 100 as these values are sufficient enough for us to understand and analyze the result and draw conclusions.

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Introduction to Algorithms English By Thomas H. Cormen , By Charles E. Leiserson , Ronald L. Rivest ,Clifford Stein(3rd edition)