CS2023 - Data Structures and Algorithms

In Class Lab Exercise

Week 08

Index Number: 200105F

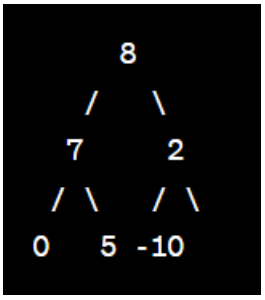
GitHub Link : https://github.com/UlinduP/CS2023/tree/main/In%20Class%20Labs/Lab%208

Screenshots of terminal output

```
s Labs\Lab 8\" ; if ($?) { g++ heap.cpp -o heap } ; if ($?) { .\heap }
Input array
4 17 3 12 9 6
Sorted array
3 4 6 9 12 17
PS D:\_FoE\_Fourth Semester\Data Structures and Algo\Labs\CS2023\In Class Labs\Lab 8>
```

```
                                                    > cd "d:\_FoE\_Fourth Semester\Data Structures and Algo\Labs
8\" ; if ($?) { g++ heap.cpp -o heap } ; if ($?) { .\heap }
Input array
0 5 2 -10 7 8
Sorted array
-10 0 2 5 7 8
```

Max heap drawn for the custom input



The time complexity of heap sort can be analyzed as follows:

1. Building the heap: The time complexity of building a heap from an array of n elements is O(n), as each element needs to be compared and possibly swapped with its parent node until the heap property is satisfied.
2. Extracting the maximum element: The time complexity of extracting the maximum element from a heap of size n is O (log n), as the maximum element is always at the root of the heap, and removing it requires rearranging the remaining elements to maintain the heap property.
3. Sorting the array: After extracting the maximum element and placing it at the end of the array, the remaining heap needs to be reconstructed. This step takes O (log n) time, and it needs to be repeated n times, as there are n elements in the array. Therefore, the time complexity of the sorting step is O (n log n).

Overall, the time complexity of heap sort is O (n log n), which is the same as the time complexity of other popular comparison-based sorting algorithms such as merge sort and quicksort. However, heap sort has the advantage of being an in-place sorting algorithm, which means that it requires only a constant amount of extra memory beyond the input array.