During Heapsort, trace the change in value of every node in the heap.

Shiv Pratap Singh, Nazish Tabassum, Arqum Ahmad

***Index Terms*—**

I. Motivation

Heap sort is a comparison based sorting technique based on Binary Heap data structure. First find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

The purpose of this problem is used for debugging. This problem helps us in debugging because at each step we know what element is present in which node. This is very helpful if something goes wrong during sorting. It can also be used to check how many swaps are done during heap sort.

II. Algorithm

Tracing the change in value of every node during heapsort.

Input: An unsorted array A of size n which is heap.

Output: A Map S of size n in which every element's value is a vector where each node is pair stating ¡step,value>as a pair element and key is the node index.

```
1 HeapSort(A, n)
2    InitializeAnswerMap(A)
3     step = 1
4    for i = n down to 1
5        TrackChangeInNode(A, 1, i, step)
6        swap(A[1], A[i])
7        Heapify(A, i-1, 1, step)
8        step = step + 1
9 Heapify(A, size, index, step)
10    largest  =  index
11    left       =  2*index
12    right     =  left + 1
13    if left <= size and A[left] >A[largest]
14         largest = left
15     if right <= size and A[right] >A[largest]
16         largest = right
17     if largest != index
18         TrackChangeInNodeA, index, largest, step)
19         swap(A[index], A[largest])
20        Heapify(A, size, largest, step)
21 InitializeAnswerMap(A)
22    for i = 1 to n
23        S[i].push_back(0, A[i])
24 TrackChangeInNode( A, i,  j, step)
25    S[i].push_back(step, A[j])
26    S[j].push_back(step, A[i])
```

III. Description

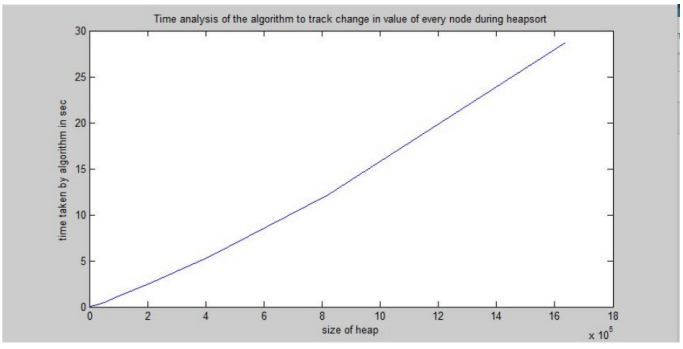
We propose a single algorithm for this problem because the problem is very easy and cann't be optimized further. The purpose of this problem is for debugging the heapsort.

The algorithm here assume that we have an unsorted heap of size n. We pass the heap to heapsort algorithm. In heapsort algorithm we first initialize the answer Map S by calling the function InitializeAnswerMap which take heap as an argument. This function simply pushes the pair (step, value) in respective node. After this we implement the general heapsort algorithm. In this algorithm we simply swap the position of last element with first element but before swapping we recorded the change of value that is going to happen in these nodes by calling the function TrackChangeInNode. This function takes four arguments (heap, first element, second element, step). After swapping the elements of first and last node we call heapify to maintain the property of heap because the first node of heap is disturbed in swapping the first and last element. Heapify function takes the four arguments (heap, size of heap, disturbed index, step). In this whenever we swap two elements to maintain the heap, we simply first call the function TrackChangeInNode for tracking the changes in node.

IV. Implementation and Results

The algorithm has been implemented in c++ language. Test case are generated manually using c++ library. For particular input size running time of input is calculated. The tool used for plotting graph is matlab.

V. Plot of time taken vs input size



VI. Conclusion

In this algorithm we assume that a heap is supplied to the algorithm of size n.

Final output is a Map S whose every element' value is a vector. The element of vector is a pair. The pair element has information of (step, value).

To achieve the results, we simply use the general heapsort algorithm and at every swapping of elements during heapsort, we track the position of change in value of nodes of heap.

The time complexity is O(n*log(n)). In calculating the time complexity that map is unordered i.e. in accessing a map element the time required is constant.

VII. Refrences

[1] http://www.geeksforgeeks.org/heap-sort/

[2] http://people.duke.edu/~hpgavin/gnuplot.html