Extraction of maximum and minimum element during each iteration of Heapsort Algorithm

Maharshi Roy, Rajat Kumar Sahu, Amrit Daimary

ism2014006@iiita.ac.in
ihm2014501@iiita.ac.in
irm2014001@iiita.ac.in

Abstract—*This paper provides a procedure to extract both the maximum and minimum element during each iteration of heapsort algorithm. Thus during each iteration, the size of the heap reduces by 2 instead of 1.*

Keywords — *Heapsort, Maxheap, Heapify, Max element, Min element*

MOTIVATION

Heapsort is an important comparison based sorting algorithm. Heapsort has a major advantage over other sorting algorithms due to the fact that it is optimal both in case of worst-case time complexity and space complexity. Heapsort is an in-place sorting algorithm, hence requires no extra space.i.e. *O(1)*. In this respect it seems similar to quicksort. However, quicksort is disadvantageous in terms of worst-case complexity. Heapsort has a worst-case time complexity of $O(nlog_2n)$, unlike quicksort which has $O(n^2)$. However, in the algorithm proposed we need to perform extraction of both the minimum and maximum element at each iteration hence we have taken an output array that would account for an extra space of *O(n)*.

METHODOLOGY AND APPROACH

The heapsort algorithm which we will modify to attain the objective is a max-heap. Thus, we can extract the maximum element quickly as it is the 1st element in the array (topmost element). However, the minimum element can belong to any of the leaves. Thus, we need to iterate the leaves and find out the minimum during each total iteration of the heapsort. After finding out the minimum element, we can put it in the output solution, and replace it with the last element. We then call float-up operation at that location to maintain the heap property.

Algorithm:-

.

```
floatup (A[], n, index) {
    parent=index/2;
    while(parent>=1 and A[parent]<arr[index]){
       swap(A[index],A[parent])
       index=parent
       parent=index/2;
    }

maxheapify (A[], n, index) {
       largest=index
       left=2*index
       right=left+1

       if (left<=n && A[left] > A[largest]) {
              largest=left
       }

       if (right<=n && A[right] > A[largest]) {
              largest=right
       }

       if (index!=largest) {
              swap(A[index],A[largest])
              maxheapify(A,n,largest);
       }
}

heapsort (A[], O[], n) {
       k=1
       for (i from n/2 to 1) {
              maxheapify(A,n,i)
       }
       for (i from n to 1) {
              O[k]=A[1]
              A[1]=A[i]
              i=i-1
              maxheapify(A,i,1)
              if (i<1)
                  break
              small=i/2+1
              for (j from i/2 + 2 to i) {
                     if(arr[small] > arr[j]) {
                            small=j;
                     }
              }
              O[n-k+1]=A[small]
              k=k+1
              A[small]=A[i]
              i=i-1
              floatup(A,small);
       }
}
```

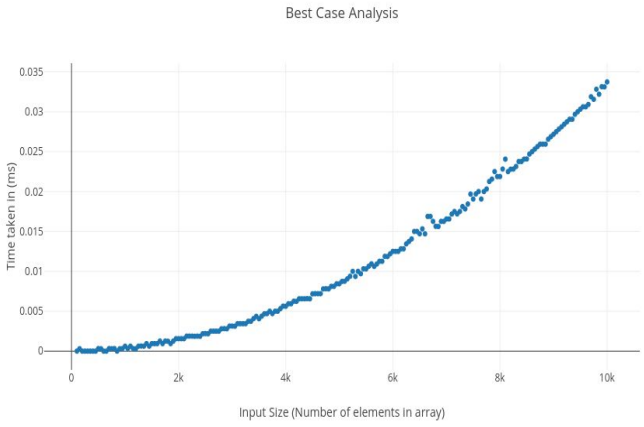
where A is the unsorted array, O is the output array and n is the initial number of elements present in the array.

IMPLEMENTATION & RESULTS

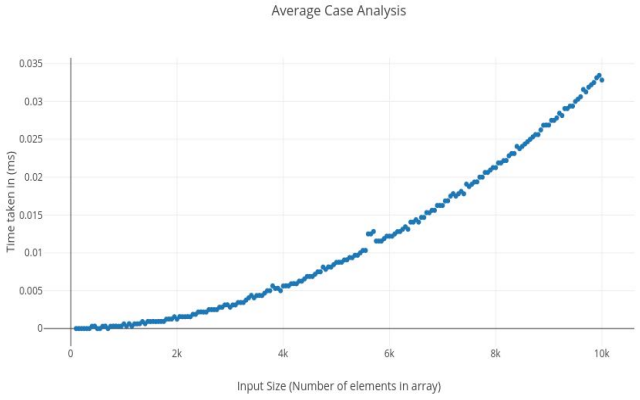
The algorithm was implemented in C++ and Time complexity analysis were performed on all the three methods, and line-graphs were obtained for varying inputs. All these operations were done in GNUPlot. All the tasks was performed in a 64-bit machine (x86-64 architecture). The line-graphs for the various methods are given as follows:-

Time Complexity Plot:-

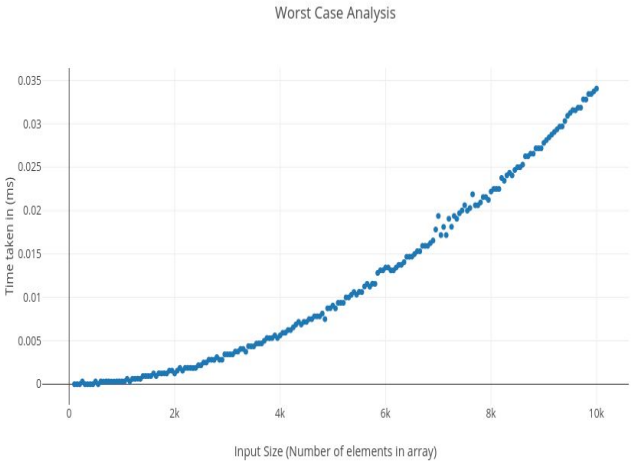
Best-Case Analysis



Average-Case Analysis



Worst-Case Analysis



REFERENCES

[1] Heapsort Article from Wikipedia