Merge Two Heaps

Akash Anand, Parth Kataria, Shruti Nanda
iit2019015@iiita.ac.in, iit20190016@iiita.ac.in, iit2019017@iiita.ac.in
IV semester,Department of Information Technology
Indian Institute of Information Technology, Allahabad

Abstract-This paper introduces algorithms to merge two heaps .The idea is to design,explain and analyse algorithms and conclude with the most efficient algorithm in terms of time and space complexities.

1. INTRODUCTION

A heap is a tree-based data structure in which all the nodes of the tree are in a specific order.On basis of order there are two heaps min heap and max heap.We are given two heaps and we have returned a merged heap of the given heaps.
This report further contains:

1. Algorithm Design and Analysis
2. Experimental study and complexity.
3. Conclusion

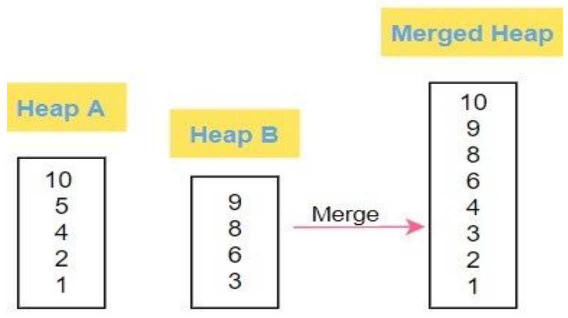


Fig 1.1:Merging two max heaps

2. ALGORITHM DESIGN

We are considering the cases when both heaps are max heap or both are min heap.
1. Take input of the two heaps into arrays.
Name these arrays as array a and array b and their respective sizes be n and m.
2. Perform any one of the following defined algorithms and store result in array c.

Approach 1:Random Merge

```
function  merge(int a[],int b[],int c[])
   for  i ← 0 to n do
        c[i]=a[i]
   end for
   for  i ← 0 to m do
        c[i+sizeof(a)]=a[i]
   end for
   sort(c,c+n+m)
end  function
```

In this approach sorting is performed according to the property of the heap.
In descending order if a and b are max heap and in ascending order if a and b are min heap.

Approach 2:Ordered Merge

```
function  merge(int a[],int b[],int c[])
  If max heap then
    int i=0,j=0;
    while  i<n && j<m
       If a[i]>=b[j] then
         push a[i] in c[] and i++
       If a[i]<b[j] then
         push b[j] in c[] and j++
    end while
   while i<n
       push a[i] to c[i] and i++
    end while
    while(j<m)
        Push b[i] to c[i] and j++
     end while
end  function
```

Same algorithm can be performed for min heap.

3. EXPERIMENTAL STUDY AND ANALYSIS

A priori analysis:
For Approach1:
1)Time Complexity:
Best case : Ω((n+m)*(1+log(n+m))
Average case : θ((n+m)*(1+log(n+m))
Worst case : O((n+m)*(1+log(n+m))
2)Space Complexity-O(n+m)

For Approach2:
1)Time Complexity:
Best case : Ω(n+m)
Average case : θ(n+m)
Worst case : O(n+m)
2)Space Complexity-O(n+m)

S no.	N	M	Algorithm1	Algorithm2
1	5	5	10	33.21928095
2	20	20	40	212.8771238
3	40	40	80	505.7542476
4	50	50	100	664.385619
5	70	70	140	998.0996224
6	200	1000	1200	12274.58243
7	5000	100	5100	62813.03581
8	1000000	1000000	2000000	41863137.14

4. TIME COMPLEXITY

The algorithms were tested against random heaps of variable sizes.The observation thus obtained from this experiment is represented in the form of individual and campirison graphs given below:

Figure 4.1 Time Complexity graph for Algorithm1

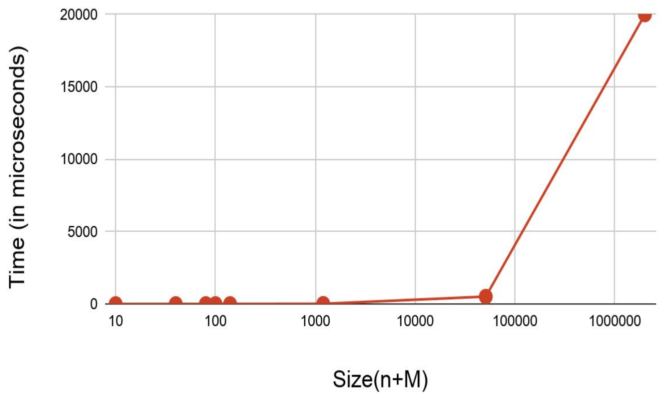


Figure 4.2 Time complexity of Algorithm 2

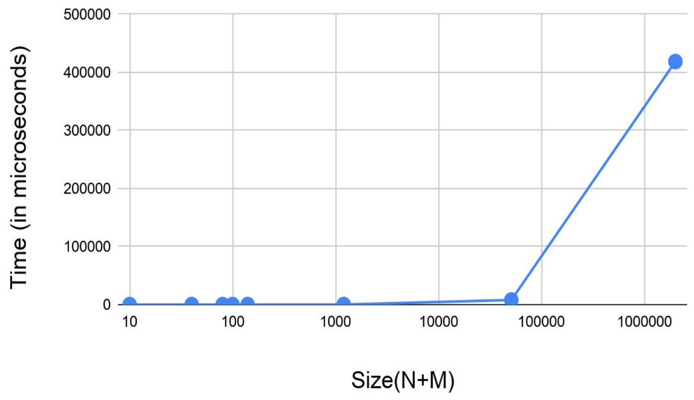
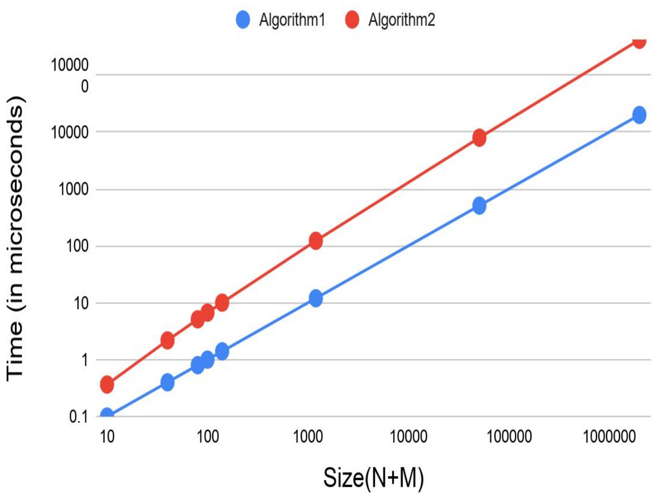


Figure 4.3 Time Complexity Comparison



5. CONCLUSION

After observing and analysing the above algorithms we can conclude that the algorithm based on the second approach is much more effective than the first based on time complexity and both are similar in terms of space complexity.

6. REFERENCES

1. https://www.geeksforgeeks.org/merge-two-binary-max-heaps/
2. https://www.geeksforgeeks.org/time-complexity-of-building-a-heap/