Merge Two Heaps

Group Members

Aakash Anand (IIT2019015)

Parth Kataria (IIT2019016)

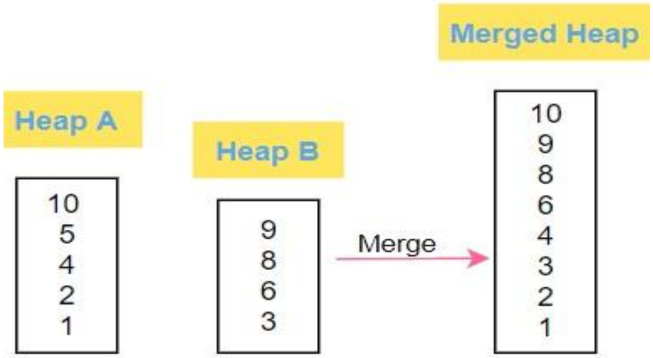
Shruti Nanda (IIT2019017)

Contents

Introduction
Algorithm design
Time complexity analysis
Conclusion
Reference

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.



Algorithm Design

We have designed two algorithm for merging two Heaps

1. **Random Merge**

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.

2. **Ordered Merge**

In this Algorithm if the given heap is min heap then we are pushing the elements of two heaps in third heap in sorted order

1. Random Merge

Algorithm:

Step 1:

We will take two heap as user input and pass them in function

Step2:

We will make a declare a new heap which will be the merge heap.

Step3:

We will iterate over first heap and push them in c[].

Step4:

We will now iterate over second Heap and push them in c[]

Finally, we will return c[] and print c[].

Complexity Analysis

Time Complexity

Best case : Ω((n+m)*(1+log(n+m))

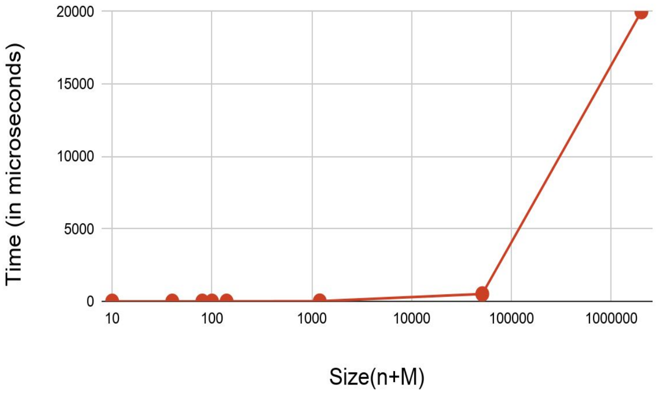
Average case : θ((n+m)*(1+log(n+m))

Worst case : O((n+m)*(1+log(n+m))

Space Complexity :

O(n+m)

Figure 4.1 Time Complexity graph for Algorithm1



Pseudo Code for 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)]=b[i]
  end for
  sort(c,c+n+m)
end function
```

```
Example:
Input
Enter the size of First Heap:
3
Enter the Elements of First heap:
3 4 5
Enter the size of Second Heap:
4
Enter the Elements of Second Heap:
5 6 7 8

OUTPUT:
3 4 5 5 6 7 8
```

2. Ordered Merge

Algorithm

Step 1:

Check the heap is Max heap or Min heap and If it is a max heap then

Step 2:

Declare two variable i=0 and j=0 and run a while till i<n and j<m;

Step 3:

If a[i] is greater than b[j] then we will push a[i] in c[] and increment the value of i.

Step 4:

If b[j] is greater than a[i] then we will push b[j] in c[] and increment the value of j.

Step 5:

We will continue the process and will push the left elements in any of heap in c[]

Step 6:

Thus c[] is the final merged heap and we will print it.

Complexity analysis

Time complexity

Best case : Ω(n+m)

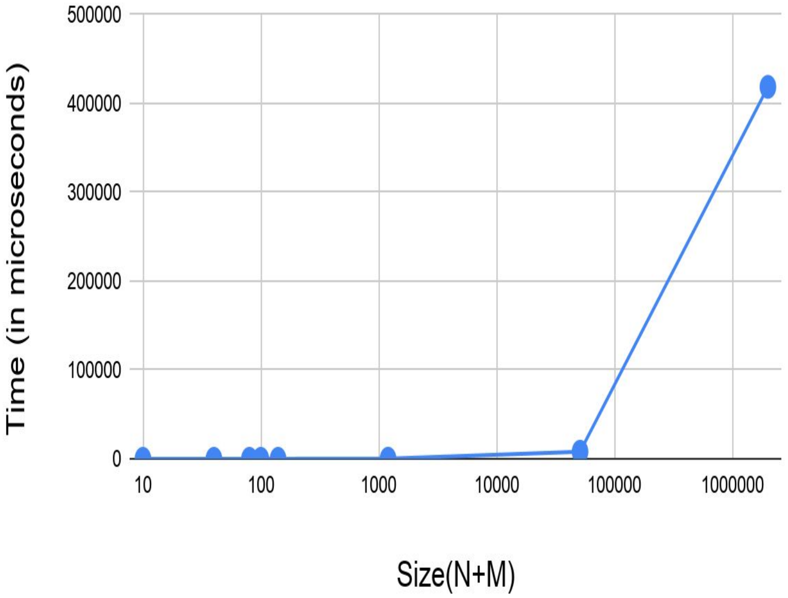
Average case : θ(n+m)

Worst case : O(n+m)

Space Complexity :

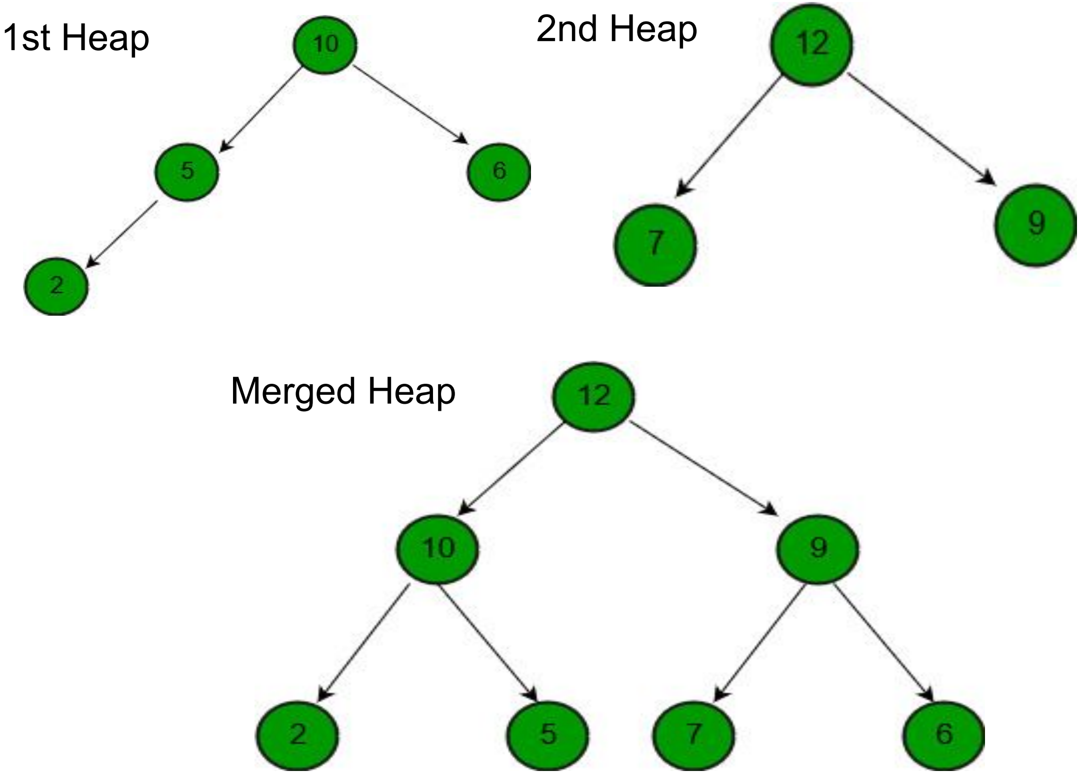
O(n+m)

Figure 4.2 Time complexity of Algorithm 2



Pseudo code

```
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
```

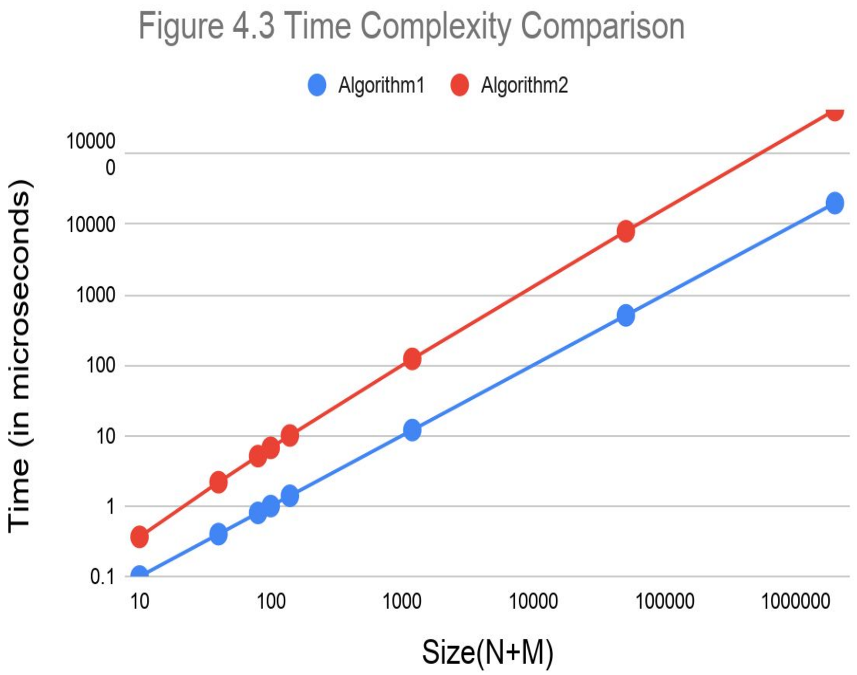


1st Heap

2nd Heap

Merged Heap

Comparison of time Complexity of two Algorithm



Here it is Time complexity Number of Iteration

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.

References:

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