DAA432C
ASSIGNMENT 4

GIVEN A HEAP, WRITE AN EFFICIENT ALGORITHM TO LOCATE THE 1ST SMALLEST, 2ND SMALLEST, ..., K'TH SMALLEST ELEMENT IN THE HEAP, FOR SOME GIVEN INPUT VALUE OF K. DO THE NECESSARY EXPERIMENTATION AND ANALYSIS WITH YOUR ALGORITHM

PREPARED BY

SHELDON TAURO
ASWIN VB
AVINASH YADAV
NISTALA VENKATA SHARMA

Indian Institute Of Information Technology
Allahabad

2018

Given a Heap, write an efficient algorithm to locate the 1st smallest, 2nd smallest, ..., k'th smallest element in the Heap, for some given input value of k. Do the necessary experimentation and analysis with your algorithm

Sheldon Tauro*, Aswin VB†, Avinash Yadav‡ and N.V.K. Sharma§
Indian Institute of Information Technology,Allahabad
Allahabad-211012
Email: *iit2016137@iiita.ac.in, †iit2016106@iiita.ac.in, ‡itm2016004@iiita.ac.in, §ism2016005@iiita.ac.in

I. Introduction and literature survey

The heap is one maximally efficient implementation of an abstract data type called a priority queue, and in fact priority queues are often referred to as "heaps", regardless of how they may be implemented. A common implementation of a heap is the binary heap, in which the tree is a binary tree. The heap data structure, specifically the binary heap, was introduced by J. W. J. Williams in 1964, as a data structure for the heapsort sorting algorithm. Heaps are also crucial in several efficient graph algorithms such as Dijkstra's algorithm.

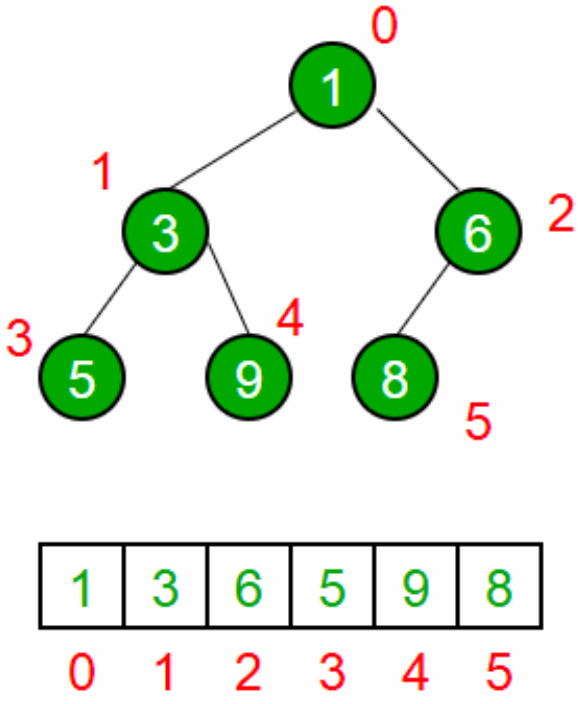

Fig. 1. array representation (level order) of a binary heap

In a heap, the highest (or lowest) priority element is always stored at the root. A heap is not a sorted structure and can be regarded as partially ordered. As visible from the heap-diagram, there is no particular relationship among nodes on any given level, even among the siblings. When a heap is a complete binary tree, it has a smallest possible heighta heap with N nodes and for each node a branches always has log(n) height. A heap is a useful data structure when you need to remove the object with the highest (or lowest) priority

Heap sort is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element.

Let us first define a Complete Binary Tree. A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

A Binary Heap is either Min Heap or Max Heap. In a Min Binary Heap, the key at root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree. Max Binary Heap is similar to Min Heap.

Binary heaps may be represented in a very space-efficient way (as an implicit data structure) using an array alone. The first (or last) element will contain the root. The next two elements of the array contain its children. The next four contain the four children of the two child nodes, etc. Thus the children of the node at position n would be at positions 2n and 2n + 1 in a one-based array, or 2n + 1 and 2n + 2 in a zero-based array. This allows moving up or down the tree by doing simple index computations. Balancing a heap is done by sift-up or sift-down operations (swapping elements which are out of order). As we can build a heap from an array without requiring extra memory (for the nodes, for example), heapsort can be used to sort an array in-place.

II. Algorithm Design

a is an array represented as level order of given heap. n is the number of elements in a heap. k is the number of times we have to print the smallest value in a min-heap. $index$ is the place in heap from that place we have to heapify the heap. $size$ is the current size of array by removing minimum simultaneously.

Algorithm 1 heapify as a min heap

function $heapify(int\ *a, int\ index, int\ size)$
$temp \leftarrow index$
$min1 \leftarrow a[index]$
if index*2 <= size **and** min1>a[index*2] **then**
  $min1 \leftarrow a[index*2]$
  $temp \leftarrow index*2$
end if
if index*2+1 <= size **and** min1>a[index*2+1] **then**
  $min1 \leftarrow a[index*2+1]$
  $temp \leftarrow index*2+1$
end if
if temp != index **then**
  $swap(a[temp], a[index])$
  $heapify(a, temp, size)$
end if

Algorithm 2 Main function

function
$sort(a, a+n)$
for $i \leftarrow n$ **to** n-k **do**
  $a[1] \leftarrow a[i]$
  $heapify(a, 1, i-1)$
end for

Explanation

We have an array of random generated number represented as level order of a binary heap. By sorting we have created a min-heap.Now we have generated k which is less than n.Till k we are removing the root of the min-heap.As far as we are removing the root of min-heap we are heapifying the heap by replacing the last child of the heap.

III. Analysis

N is the number of nodes in the heap. we have a function heapify(). In which we are heapifying the heap by giving index and size of the heap.

A. Time Analysis

$$time_{HeapifyWorst} = 12 * log(n)$$

$$time_{HeapifyBest} = 7$$

$$\Omega(g(n)) = \Omega(k * (6 + time_{HeapifyBest}))$$

$$\Omega(g(n)) = \Omega(13 * k)$$

$$O(f(n)) = (6 * k + 12 * k * log(n))$$

$$BestCase = O(k)$$

$$WorstCase = \Omega(k * log(n))$$

$$AverageCase = \Theta(k * log(n))$$

- From the above analysis it is clear the the best case is of the order of k when all the nodes are equal which results in lesser number of recursions during heapify . Worst case is of the order of $k * log(n)$ when all nodes are different and the recursions are maximum. The average time complexity will be in between the best and worst case.

IV. Experimental study

TimeComplexity

Then Time Complexity is $O(klogn)$, where n is the number of nodes and k is the number of smallest elements to be found.

- When we plot for $klogn$ vs t It can be noted that, the graph is linear since the time is proportional to klogn for average and worst case and proportional to k for best case as in fig(2).The green line is the graph for best case,violet for worst case and the blue line is for average case.

n	k	Best	Experimental	Worst
668	353	4589	36599	41876.5
367	178	2314	16842	19274.3
385	170	2210	16220	18548.6
950	659	8567	71448	82190.2
159	153	1989	10895	14361.1

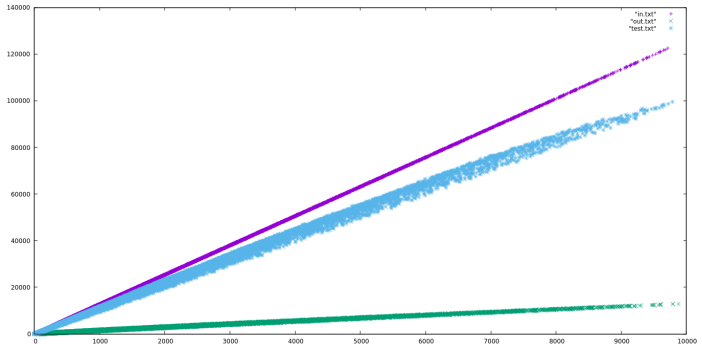

Fig. 2. Complexity Graph between klogn vs time.

V. Discussions

The heap data structure has many applications.

- Heapsort: One of the best sorting methods being in-place and with no quadratic worst-case scenarios.
- Selection algorithms: A heap allows access to the min or max element in constant time, and other selections

(such as median or kth-element) can be done in sub-linear time on data that is in a heap.

- Graph algorithms: By using heaps as internal traversal data structures, run time will be reduced by polynomial order. Examples of such problems are Prim's minimal-spanning-tree algorithm and Dijkstra's shortest-path algorithm.
- Priority Queue: A priority queue is an abstract concept like "a list" or "a map"; just as a list can be implemented with a linked list or an array, a priority queue can be implemented with a heap or a variety of other methods.
- Table below is having time complexity of operations in heap.

Time complexity for operations	
Operation	Time Complexity
find-min	O(1)
delete-min	O($logn$)
insert	O($logn$)

VI. Conclusion

The problem was to find 1st,2nd,...,kth smallest element in a heap. As given in the problem we will be given a heap if the heap is not min heap we make it min heap by sorting the heap. Then we delete the root node which will be the first smallest element in the heap and then apply heapify algorithm on it to make it a min heap again.This process is continued till k times to obtain kth smallest element. So in each removal what we remove is the i_{th} smallest element in the heap. The time complexity of this algorithm is of the order of klogn when we plot klogn vs t the graph obtained is linear

References

[1] IDAA CLass Lectures.

[2] https://www.geeksforgeeks.org/binary-heap/

[3] https://en.wikipedia.org/wiki/Heap$_(data_structure)$