ANALYSIS OF DIFFERENT HEAPS

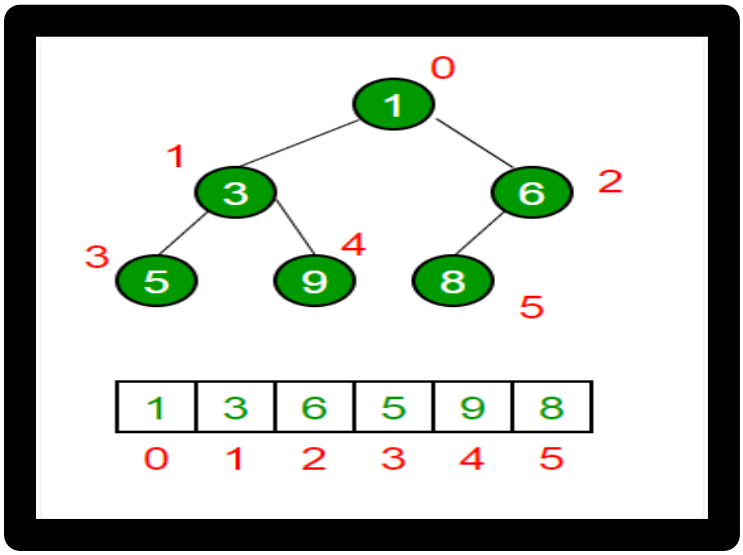
Aarushi

2019CSB1059

Contents

BINARY HEAPS

A binary heap can be viewed as an array where each node is stored at a position i and has a parent(if i !=0) at (i-1)//2 and children(if i is not a leaf node) at indices 2*i+1 and 2*i+2.



Array representation of binary heap (Source)

Insert() operation: It involves adding a node at the leaf level and then percolating it upwards till it's parent is less than (for min heap)/ greater than(for max heap) it. Time complexity: **O(logn)**.

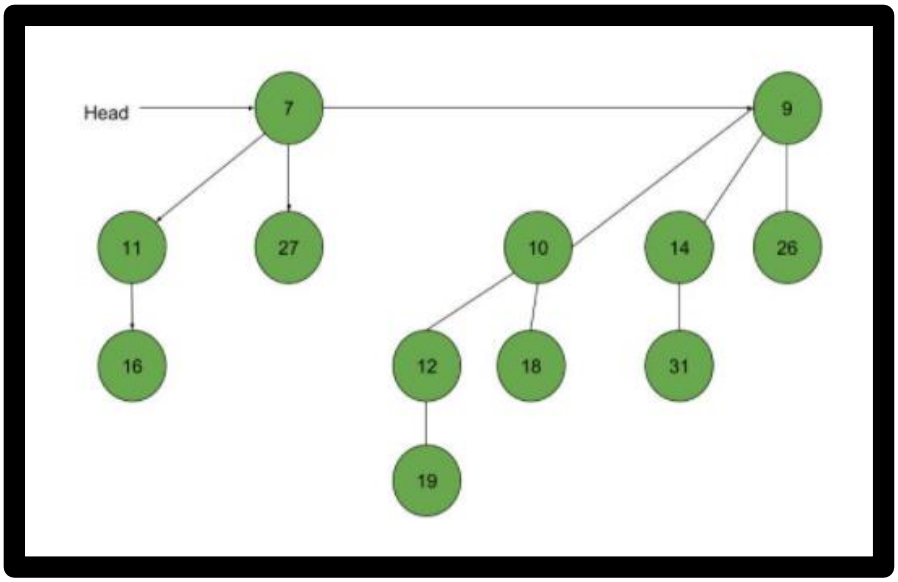
Getmin() (for min heap)/ Getmax(for max heap) operation: It simply reads the first element of an array as it is the min/max of all elements. Time complexity: **O(1)**

Decrease key(): Decrease current value of a node and percolate up as in insert() operation. Time complexity: **O(logn).**

Extract min(): Replace min with a leaf node and percolate down till the possible point. Time complexity: **O(logn).**

BINOMIAL HEAPS

A binomial heap is a collection of one or more binary trees. A binary tree of order k can be made by suitable merging of 2 binomial trees of order k-1.



A binomial heap having trees of rank 2 and rank 3 (Source)

Insert(): Create a new heap of rank 1 and merge it with the previous heap using Union() operation. Time complexity: **O(logn)**

Decrease key(): This operation is the same as in binary heap. Time complexity: **O(logn)**

Extract min(): We remove the minimum and send all its children to the main heap and use Union() operation on the heap. Time complexity: **O(logn)**

Union(): Firstly take the 2 heaps and make a new heap which stores trees of these two in increasing order of their ranks. Now we have to merge nodes of the same ranks in the heap. nodes can have the same rank if they come from diff heaps or if the merge operation on two previous nodes made a node of that rank. hence we can conclude that at most 3 nodes have the same rank at a time. Hence the following 3 cases can arise for 2 consecutive nodes:

CASE 1: The nodes have different ranks-> Simply move ahead.

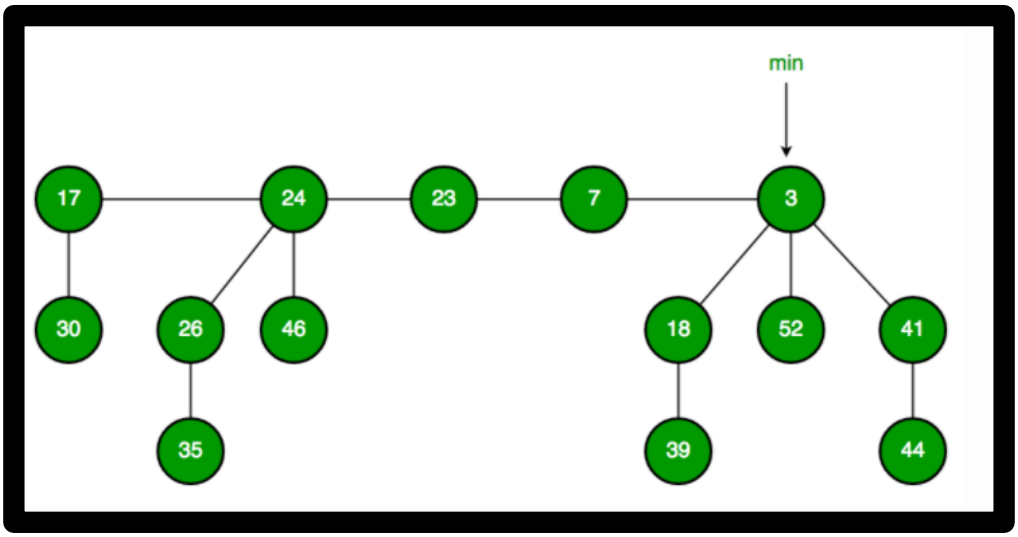
CASE 2: The nodes have the same ranks and a node after them also has the same rank-> Simply move ahead.

CASE 3: The nodes have the same ranks but a node after them also has different rank-> Make the bigger node as a child of the smaller node. Time complexity: **O(logn)**

Find min(): We aways store a pointer to the minimum node. Just get the value of that pointer. Time complexity: **O(logn)**

FIBONACCI HEAPS

It is a collection of trees with min heap property where all nodes are connected through a doubly linked list.



A fibonacci heap (Source)

NOTE: the time complexities are amortised.

Insert(): Insert the new node into the main linked list and update min if required. Time complexity: **O(1)**

Extract min(): Delete the min node and put all it's children in the main linked list. Now start merging the children according to their ranks similar to the merge in binomial heaps. Update the min pointer. Time complexity: **O(logn)**

Decrease key(): Decrease the value of node to the required value. If the min heap property is not violated don't do anything. If the property is violated 2 cases arise:

CASE 1: Parent of that node is unmarked-> Cut off the node and put in the main list mark the parent.

CASE 2: Parent of that node is marked-> Cut off the node and put it in the main list. Repeat the same procedure with the parent.

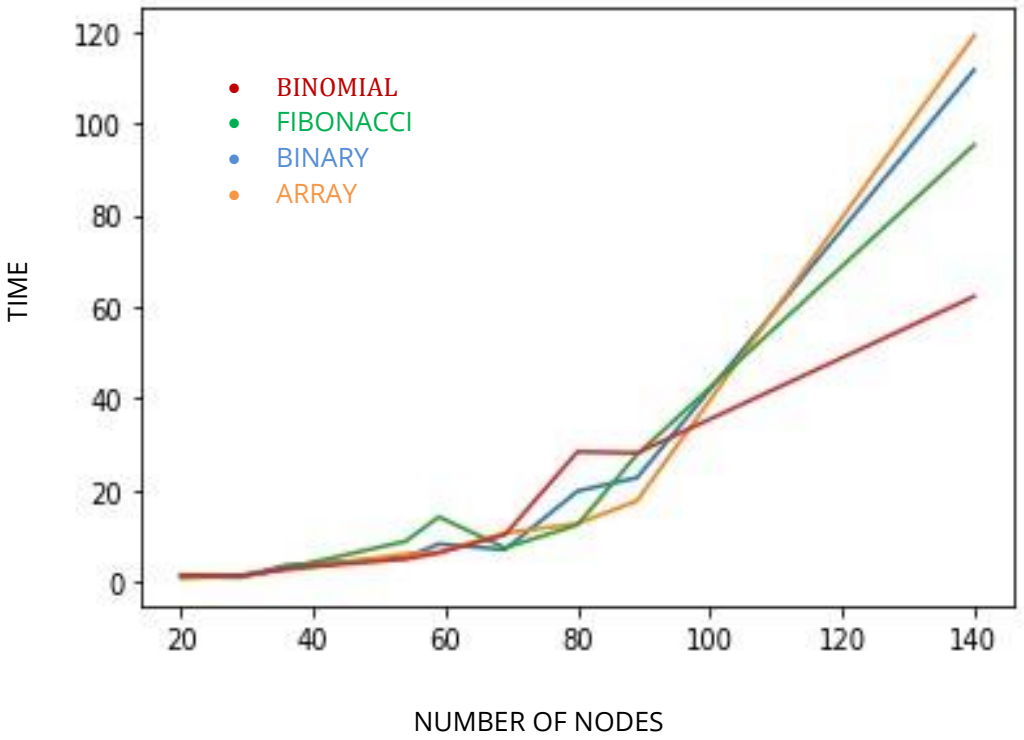
Update the min pointer if necessary. Time complexity: **O(1)**

Get min(): We aways store a pointer to the minimum node. Just get the value of that pointer. Time complexity: **O(1)**

TIME COMPLEXITY COMPARISON

Operation	Linked list	Binary heap	Binomial heap	Fibonacci heap
Insert node	**O(1)**	**O(logn)**	**O(logn)**	**O(1)**
Extract min	**O(n)**	**O(logn)**	**O(logn)**	**O(logn)**
Decrease key	**O(1)**	**O(logn)**	**O(logn)**	**O(1)**
Meld	**O(1)**	**O(n)**	**O(logn)**	**O(1)**
Delete node	**O(1)**	**O(logn)**	**O(logn)**	**O(logn)**
Find min	**O(n)**	**O(1)**	**O(logn)**	**O(1)**

PLOT OF TIME TAKEN



EXPECTATIONS AND OBSERVATIONS

I expected Fibonacci heaps to work the best at the right extreme and arrays at the left extreme. Though the arrays seem to do the same, according to my observation binomial is coming out to be best at the right extreme. However, it is possible that going further would result in parallelism between expectations and observations.