Heap Notes 2

Friday, 24 January 2020 4:10 PM

→ Add
→ delete min
→ delete any ele
→ Build → Insertion
Build → Down heapify

Heap

Max Element → Root node

$i \xrightarrow{L} 2i+1$

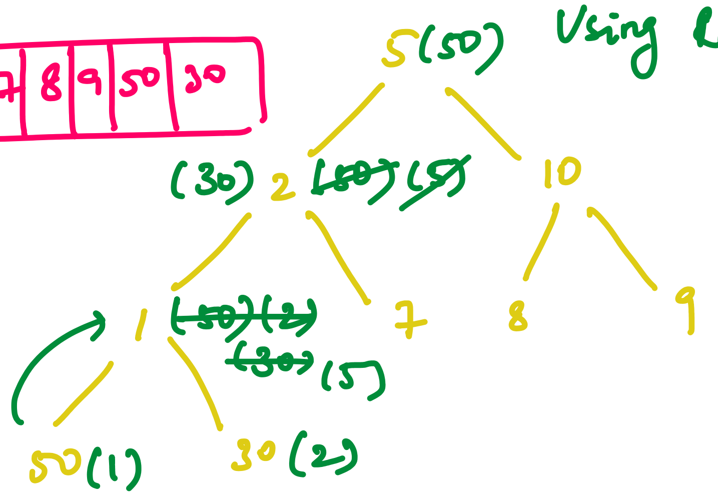
$i \xrightarrow{R} 2i+2$

* leaf nodes are already max heap

BUILD

5	2	10	17	8	9	50	30

Using Recursion



Max Heap

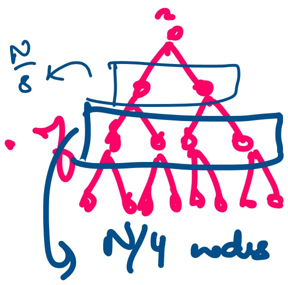
→ If you know that both subtrees are heap, then you can go top - down.
(otherwise $\boxed{\text{bottom up}}$)

→ Here we are going in bottom - up fashion.

Time Complexity of building heap →

$\boxed{O(n)}$

For a complete BST, total no. of leaf nodes $\frac{n}{2}$.



$S = 2^{h-1} \times 0 + 2^{h-2} \times 1 + 2^{h-3} \times 2$ - - - -

$$① \quad S = 2^{h-1}\left(\overset{h\times 1}{\frac{1}{2}} + \frac{2}{2^2} + \frac{3}{2^3} \cdots\cdots \frac{h}{2^{h-1}}\right) \to \text{AGP}$$

$$② \quad 2S = 2^{h-1}\left(1 + \frac{2}{2} + \frac{3}{2^2} + \cdots\cdots \frac{h}{2^{h-2}}\right)$$

$$2S - S = 2^{h-1}\left(1 + \left(\frac{2}{2} - \frac{1}{2}\right) + \left(\frac{3}{2^2} - \frac{2}{2^2}\right)\cdots\cdots \cdots \frac{h}{2^{h-1}}\right)$$

$$S = 2^{h-1}\left(\underbrace{1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \frac{1}{2^{h-2}}}_{\text{GP}} + \frac{h}{2^{h-1}}\right)$$

$$S = 2^{h-1}\left(\frac{1\times(1-(1/2)^{h-1})}{1-1/2} + \frac{h}{2^{h-1}}\right)$$

$$S = 2^{h-1}\left(\frac{(2^{h-1}-1)/2^{h-1}}{1-1/2} + \frac{h}{2^{h-1}}\right)$$

$$S = 2^{h-1}\left(\frac{2^{h-1}-1}{2^{h-2}} + \frac{h}{2^{h-1}}\right)$$

$$S = \cancel{2^{h-1}} \times \frac{1}{\cancel{2^{h-1}}}\left[\left(2^{h-1}-1\right)\times 2 + h\right]$$

$$S = 2^h - 2 + h \qquad h = \log_2 n$$

$$S = 2^{\log_2 n} - 2 + \log_2 n$$

$$S = n - 2 + \log_2 n$$

$$\boxed{O(n)}$$

$$* \quad \boxed{TC = O(n)} \to \text{Down heapify at each level}$$

Q. Now can the rope can be solved easily ??

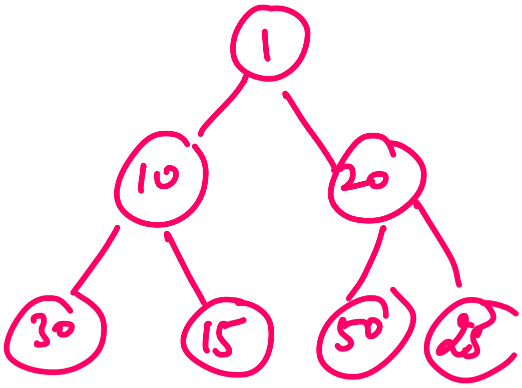
→ Use Min heap

DELETE

Q. How to extract a value from Min heap?

→

1	10	20	30	15	50	25



→ We are storing the elements in array.

→ We need first element.

→ TC → Remove first element and shift all the other elements O(n)

→ But the 2i+1 & 2i+2 property will be distorted.

→ which is the element which can be removed from the array in O(1) time ??

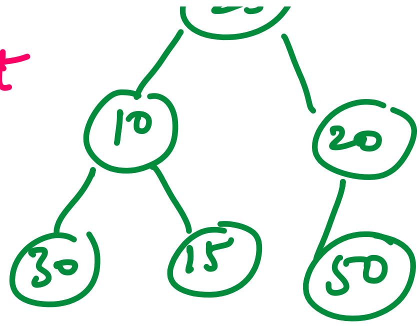
→ Using this fact to remove the first element

→ swap the first & last element. (1 & 25)

25	10	20	30	15	50	1

→ Now shift the last pointer and remove 1

→ At this point, left subtree is a heap because we haven't touched any of its elements but the right subtree is not.



Now do heapify again.

→ TC → O(log n)

We always do level order traversal.

// call the minheapify func^n with the assumption that left subtree and right subtree are already min heap.

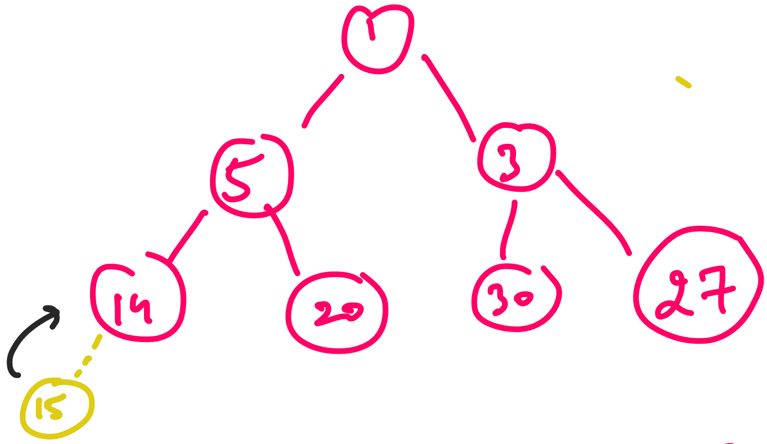
```
Minheapify (i) {
    small;
    l = 2i+1;
    r = 2i+2;

    if (r < size && h[r] < h[i])
            small = r;
    if ( l < size && h[i] < h[small])
            small = l;
    if (small != i) {

        swap (h[i], h[small]);
        Minheapify (small);

}
```

Functions for Heap

1. Heapify
2. getmin
3. Insert an element for heap.

ADD

Insert 15



→ How to insert in $O(1)$?
Insert at last.
Call heapify again

* To find the last element of heap, maintain a size variable

TC → $O(\log n)$ for inserting a element

For $\boxed{i \rightarrow \lfloor i/2 \rfloor}$ (parent element)

Ques Given 1, 23, 12, 9, 30, 2, 50

Find k^{th} largest, Here $k = 3$
ans = 23

Brute Force → Sort the array

Brute force → sort the array

→ $O(n \log n)$

optimized → Build Max heap

→ Delete $k-1$ elements
→ Print k^{th} element

For Building Max heap → $O(n)$
for extracting k elements → $O(k \log n)$

$$\boxed{TC \rightarrow O(n) + O(k \log n)}$$

If $k < n$

This approach is kind of sorting only

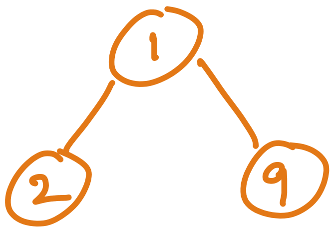
Lets consider $\boxed{23, 30, 50}$ k numbers

Out of these 23 is the smallest no.

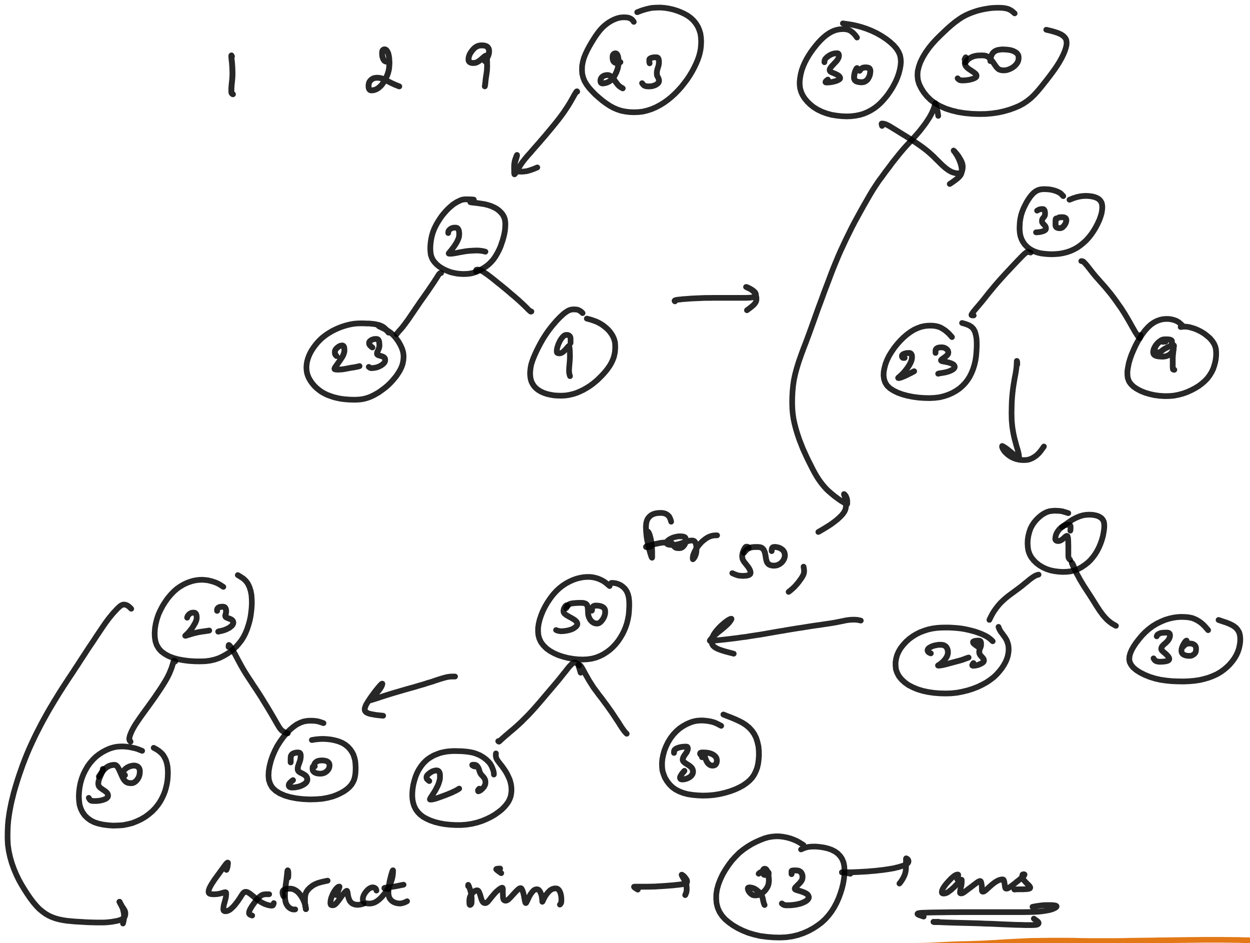
→ Build a Min heap of size k (3).

Since 23 is the minimum, we will eventually find it at the top of the heap.

Given →



I want 23 to be the minimum element of the heap. So, for any element 'less than' 23 should be popped out.



$$\boxed{\text{Time Complexity} \rightarrow O(k) \; [\text{For Building heap}] + O(n-k) \log k}$$

Q2 Merge K sorted Array

$k = 3$

$n = 4$

arr = { {1, 3, 5, 7},
{2, 4, 6, 8},
{0, 9, 10, 11}, }

Output → 0 1 2 3 4 5 6 7 8

1 9 10 11

Brute force → ① Create output array of size $n * k$

② Sort the array

TC → $O(\underline{n}k \log \underline{n}k)$

Another efficient solⁿ : $(nk\log k)$

Google Q3 $n*k\ (n^2)$

A : 1, 7, 11 ⎤ n sorted
B : 2, 4, 6 ⎦ n

Let $k=3$

Return minimum k sum pair (a, b)

Here ans → (1, 2) (1, 4) (1, 6)

$(a, \underline{b}) = (b, \underline{a})$

I Using 2 pointers → wont work

Observation: If $k = 1$

$(1, 2)$ $\quad (0, 0) \to$ A, B elements

1st pair

$a_1 \quad a_2 \quad a_3 \quad a_4$

$b_1 \quad b_2 \quad b_3 \quad b_4$

Possibilities for second pair $\to (b_1, a_2)$, (a_1, b_2)

let say (b_2, a_1) was second pair

then

$\boxed{(0, 0)} \to$ 1st min

$\boxed{(b_1, a_2) \quad (b_2, a_1)} \to$ Mins these (2nd min)

$(b_2, a_2) \quad (b_3, a_1)$

3 contenders for third minimum

Dry run:

$(1, 2) \to$ Put in heap

Then

$(2, 2) \to$ extract

$(1, 5)$

$(1, 2)$

$(2, 2)$

$(1, 5)$

$\to (1, 5) \to$ Extract

(1, 6) (2, 5)

. . . .

For every pair (i, j) push

(i, j+1) & (i+1, j)

→ Dont push when any index exceeds size of the array.

→ Also keep track of what all pairs are already in the heap.

(can use a hash map to keep track of what all pairs are already present in heap)

* After selecting the first choice, your concern is only for some elements and not all of the pairs.

Time Complexity :

Depends on maximum size of heap at any point of time. ie, k.

For building a heap → o(k)

Since, we are doing k operations → o(k log k)

O(k) + O(k log k)

Space Complexity : O(k)

* Be comfortable with libraries.
Priority queues →