Today's Content

Connecting the Robes

Heap Introduction

Insertion

Heapify

Extract Min

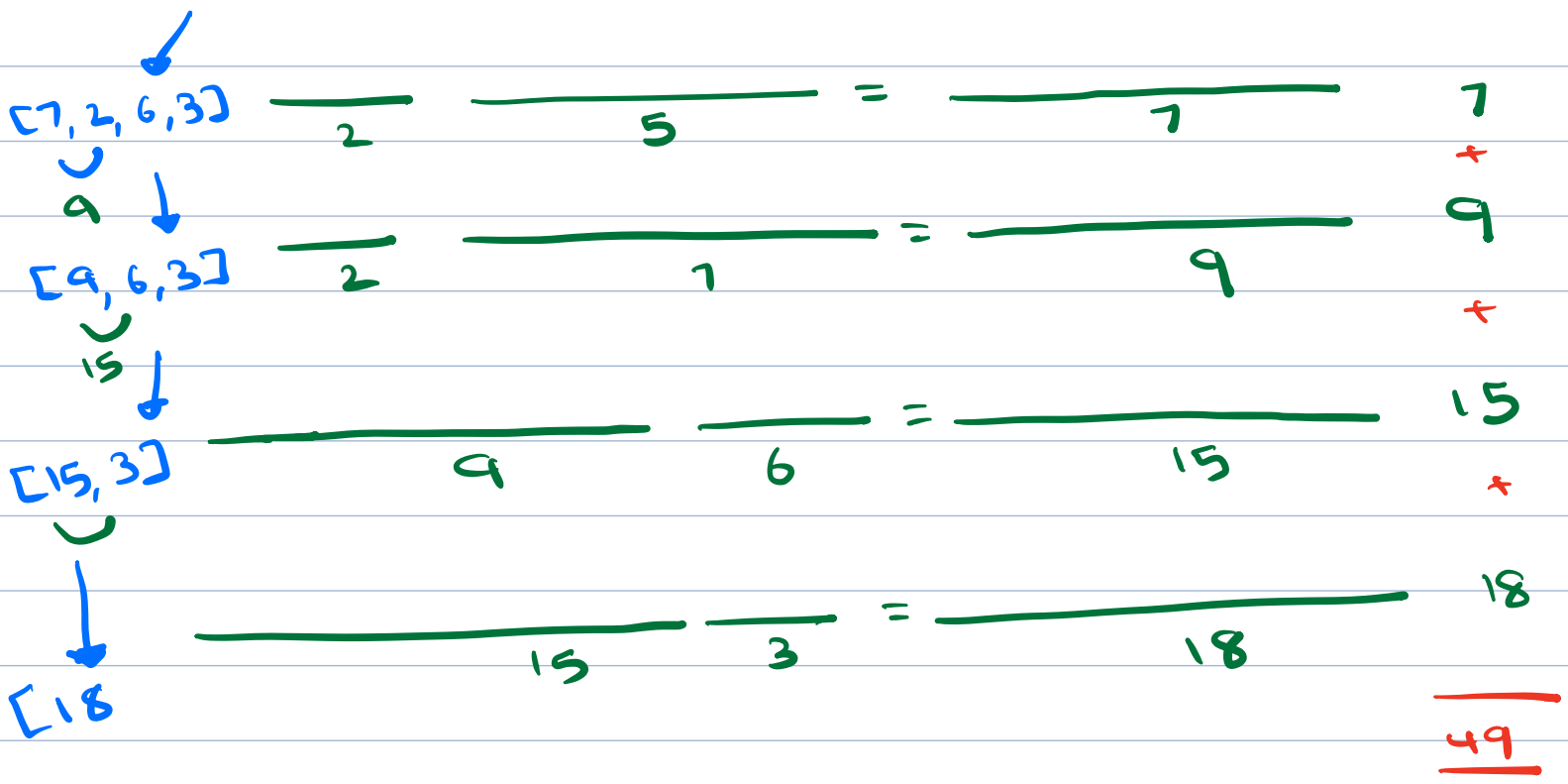
Build Heap

Q. Connecting the Ropes

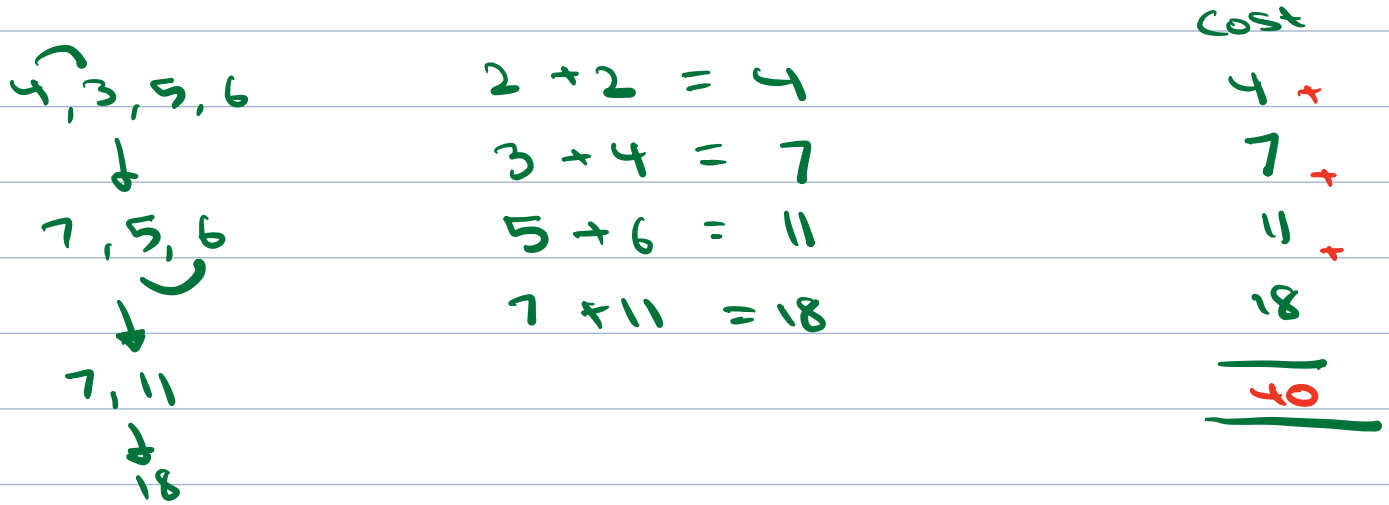
You can connect any two ropes together, there's a cost associated to connect them = sum of length of ropes that you're connecting.

Find min. cost required to connect all ropes.

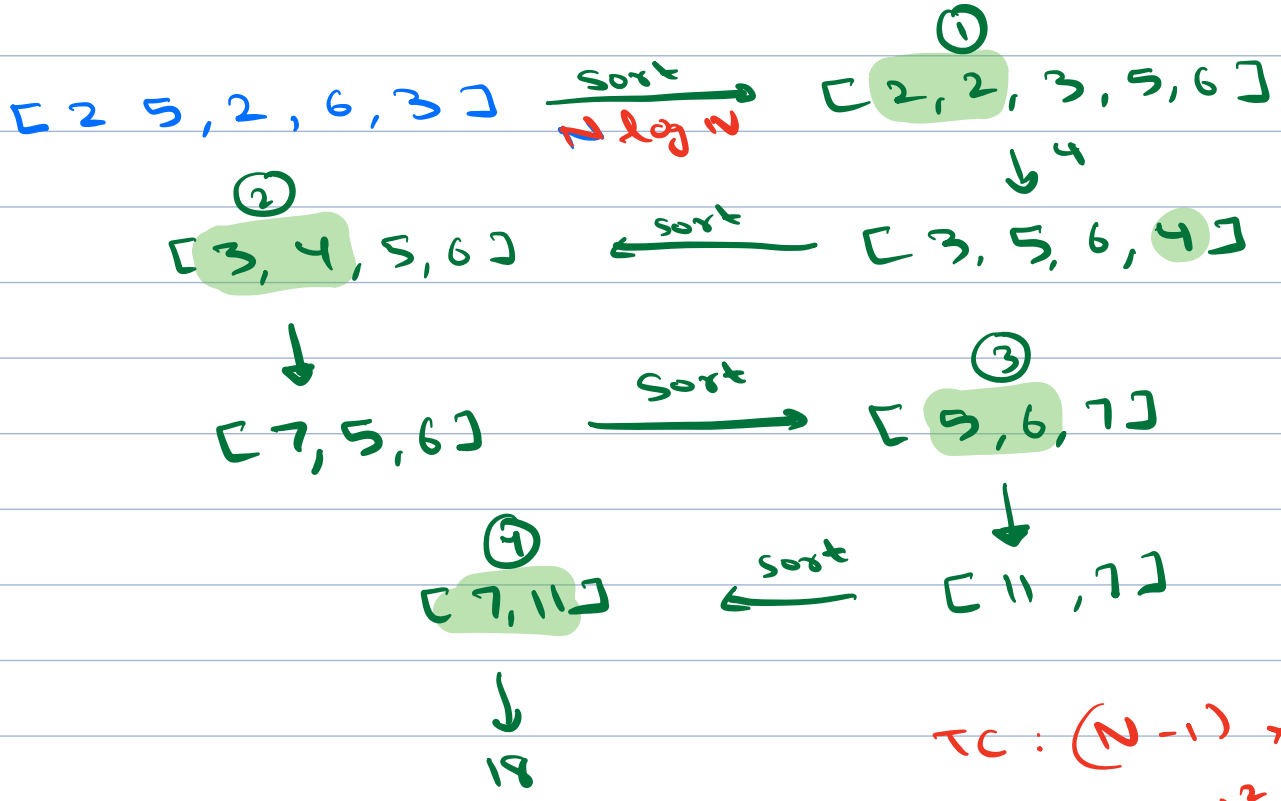
[2, 5, 2, 6, 3]



Sort [2 5, 2, 6, 3] → [2, 2, 3, 5, 6]



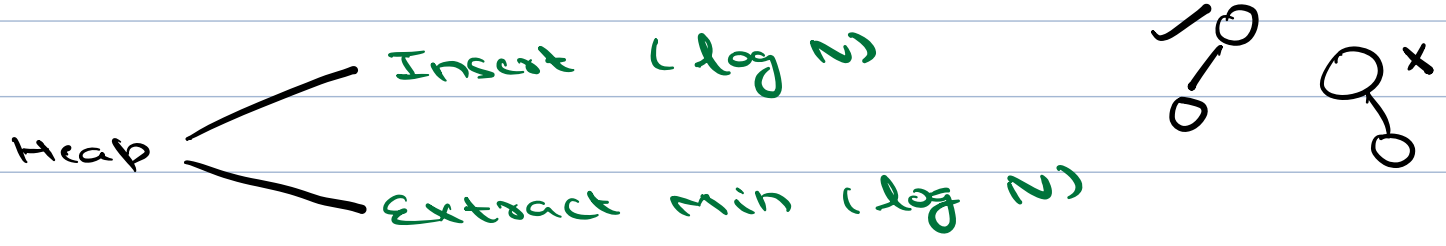
Idea: Always pick 2 smallest ropes and combine them.



TC: $(N-1) \times N \log N$

$O(N^2 \log N)$

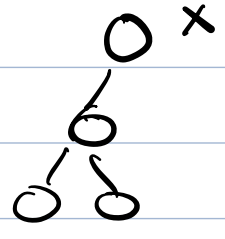
We need a DS which is optimized for finding min element



Heap DS

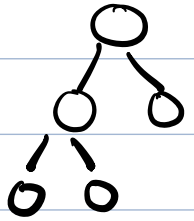
↓

Binary Tree



① Complete BT (CBT)

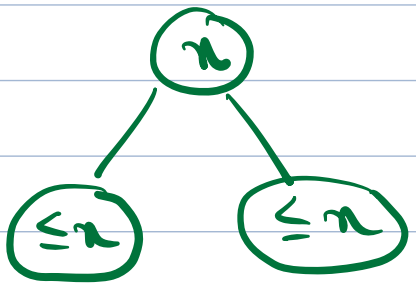
All levels are filled completely except last level, data can be filled from left to right



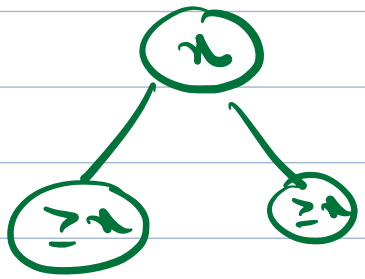
② Order of elements

↓

Heap Order Property [HOP]

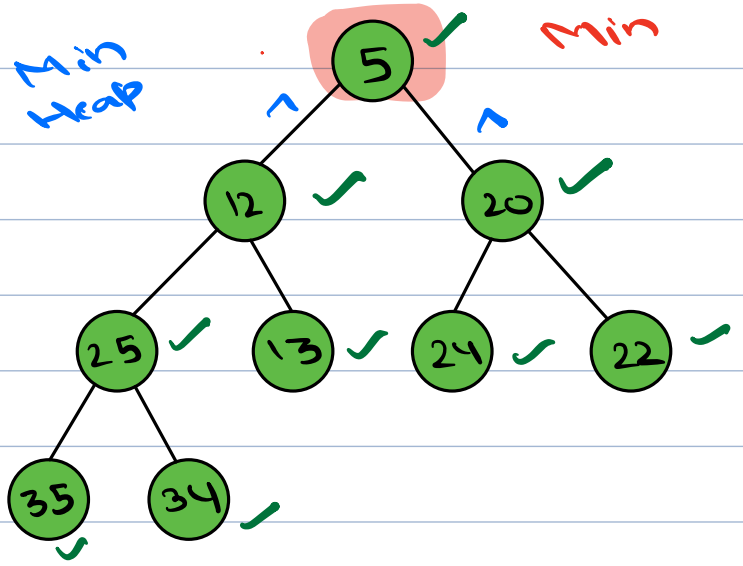


Max Heap Node ≥ Children



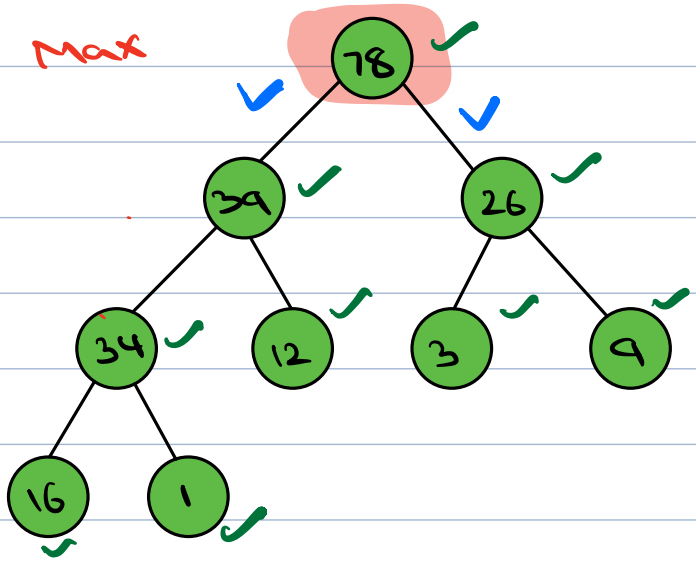
Min Heap Node ≤ Children

Min Heap



1. Complete BT
2. HOP

Max Heap

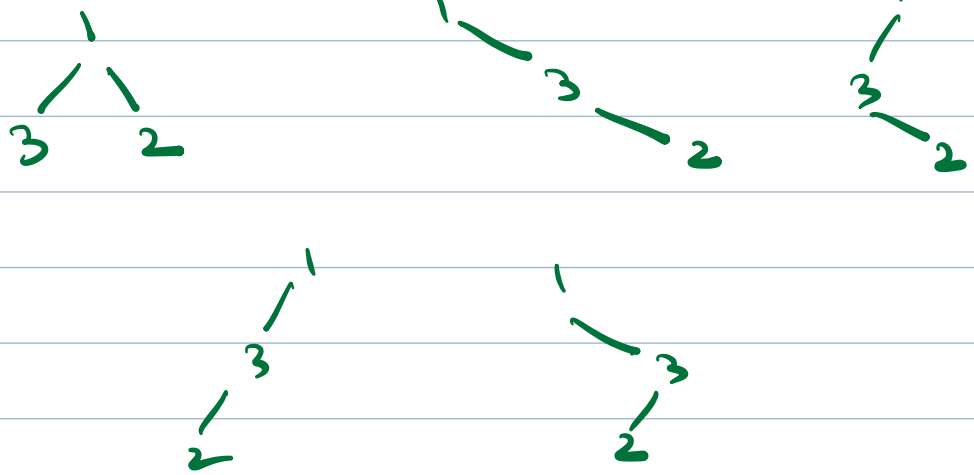


1. Complete BT
2. HOP

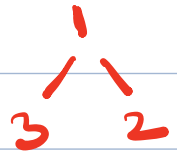
Min Heap → min el is at root O(1)

Max Heap → Max el is at root O(1)

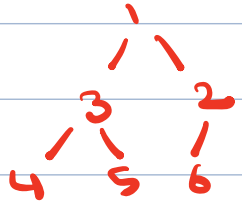
Level order Traversal [1,3,2]



[1,3,2]



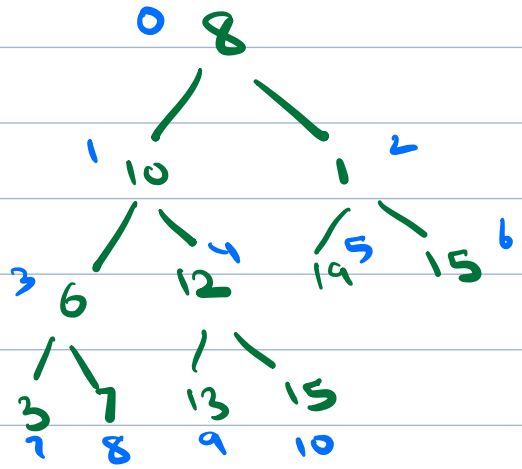
[1,3,2,4,5,6]



Visualize array as Tree (CBT)

8	10	1	6	12	19	15	3	7	13	15
0	1	2	3	4	5	6	7	8	9	10

Heap → CBT → Arr (Level order)

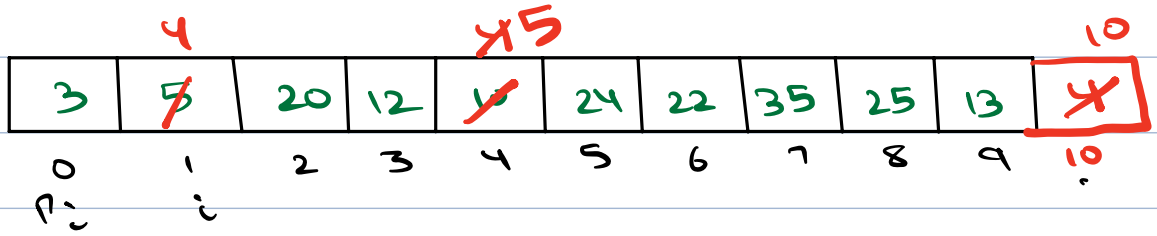


Parent	Child
0 →	1, 2
	[2*0+1, 2*0+2]
3 →	7, 8
	[2*3+1, 2*3+2]
4 →	9, 10
	[2*4+1, 2*4+2]
i →	[2i+1, 2i+2]

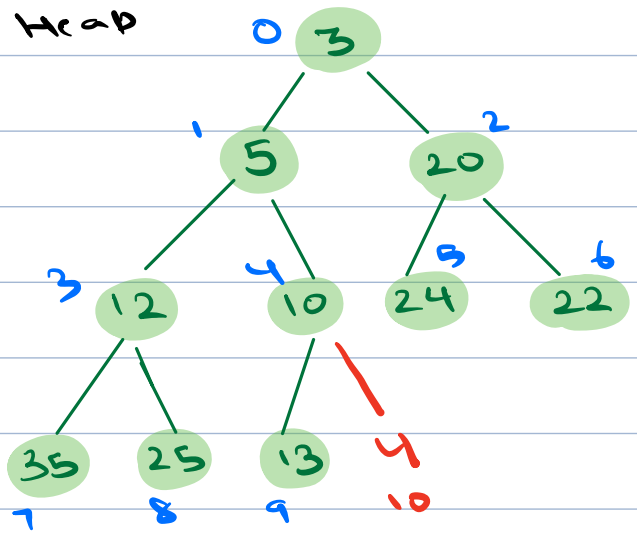
$c_i \xrightarrow{\text{par}} \frac{c_i - 1}{2}$

Insertion in minHeap

Insert 4



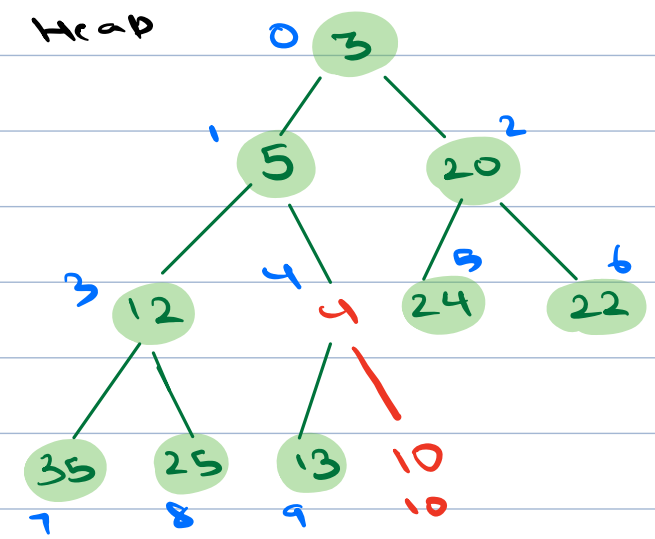
Min Heap



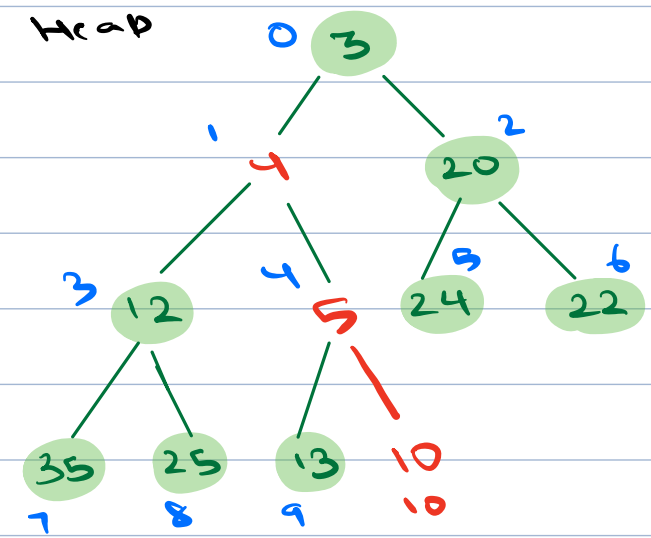
$\frac{i-1}{2}$

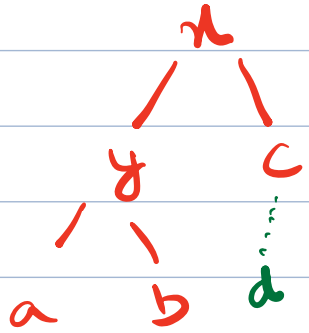
i	Pi	ar[i] > ar[Pi]
10	4	4 > 10 No
		Swap
4	1	4 > 5 No
		Swap
1	0	4 > 3 Yes
		Break

Min Heap



Min Heap

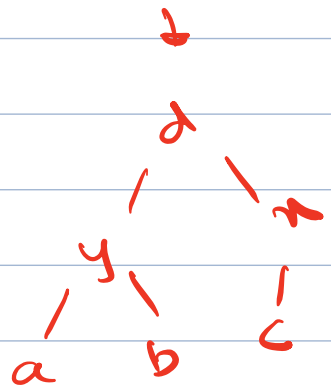




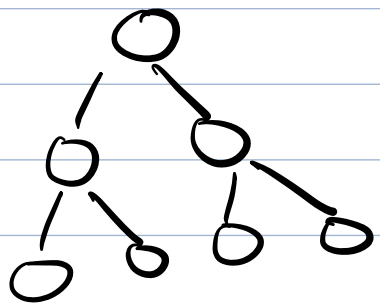
$x < y < a$
$x < b$
$x < c$

swap $d < c$

swap $d < x$

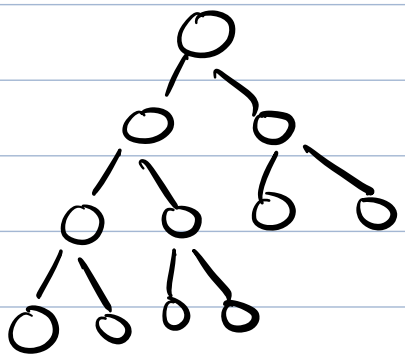


$d < x$
$< c$



$N \rightarrow \log_2 N$

7 $\log_2 7 = 2.\_$



11 $\log_2 11 = 3.\_$

Dynamic array

```
void insert(int x, int[] heap) {
    heap.addLast(x)
    i = heap.size() - 1

    while (i > 0) {
        int Pi = (i-1)/2
        if (heap[i] < heap[Pi]) {
            swap(heap[i], heap[Pi])
            i = Pi
        }
        else {
            break
        }
    }
}
```

TC: O(Height of tree)

↓

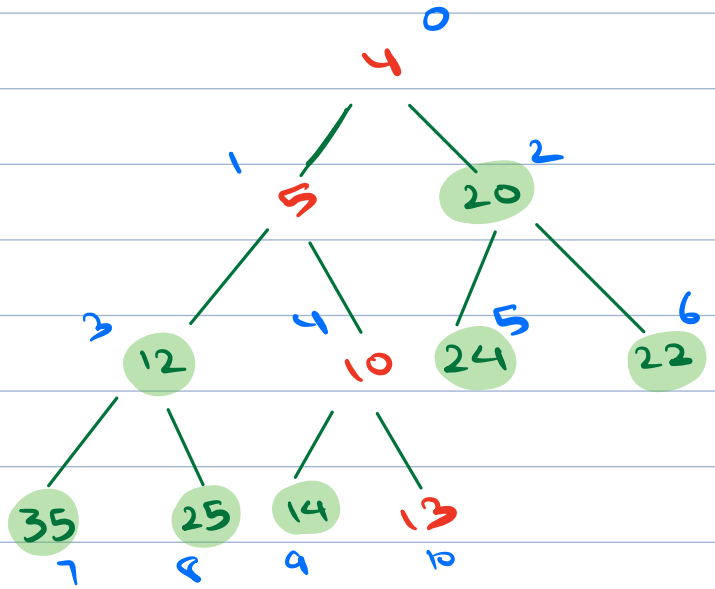
TC: $O(\log_2 N)$

SC: O(1)

Heapify

Given a min heap, all elements are following heap property except for first ele, fix the heap

0	1	2	3	4	5	6	7	8	9	10
~~13~~ 4	~~4~~ 5	20	12	~~5~~ 10	24	22	35	25	14	~~10~~ 13



N = 11 (Heap size)

idx → 0 - 10

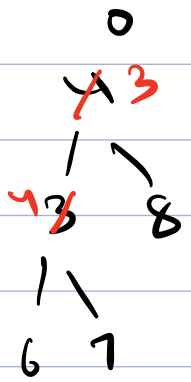
Invalid idx ≥ N

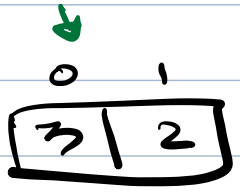
i	2i+1	2i+2
5	11 X	12 X
6	13 X	14 X

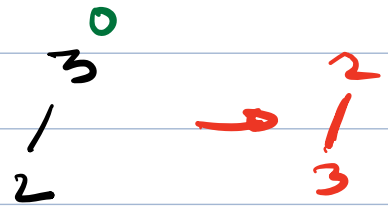
i	child idx	min(a[i], a[2i+1], a[2i+2])
0	1, 2	min(13, 4, 20) = 4 → idx 1 swap (0, 1)
1	3, 4	min(13, 12, 5) = 5 (idx 4) swap(1, 4)
4	9, 10	min(13, 14, 10) = 10 (idx 10) swap(4, 10)
10	21, 22	Break

```
void heapify (heap[], N, i){
    while(2i+1 < N){
        int x = min(heap[i], heap[2i+1], heap[2i+2])
        if (x == heap[i])
            break
        else if (x == heap[2i+1]){
            swap(heap[i], heap[2i+1])
            i = 2i+1
        }
        else if (x == heap[2i+2]){
            swap(heap[i], heap[2i+2])
            i = 2i+2
        }
    }
}
```

NOTE *





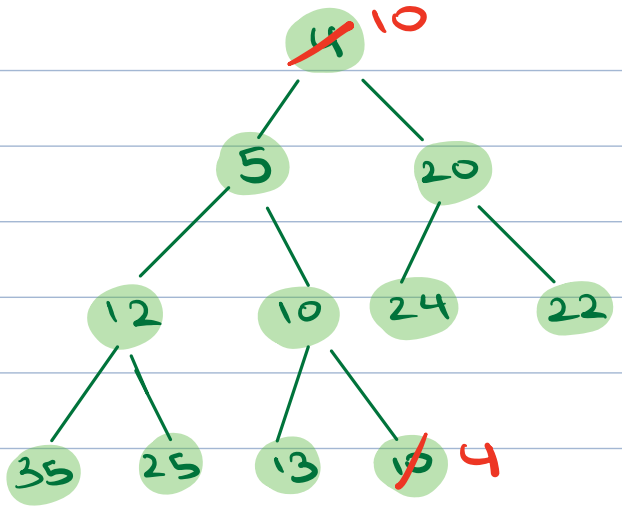


TC: O(log N)
SC: O(1)

10:55

Extract / Remove min

10										4
~~4~~	5	20	12	10	24	22	35	25	13	~~10~~
0	1	2	3	4	5	6	7	8	9	10



idx

N = 11 → 0 - 10

N-- → 10 → 0 - 9

① swap heap[0] with heap[last]

② remove last el

③ heapify with idx 0

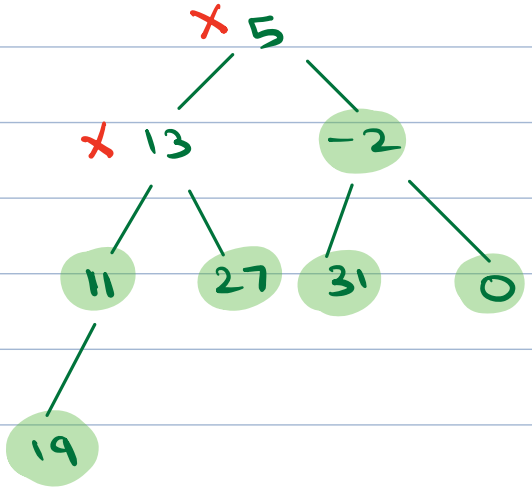
TC : O(log N)

SC : O(1)

Heap [ ] → Min el → heap[0]

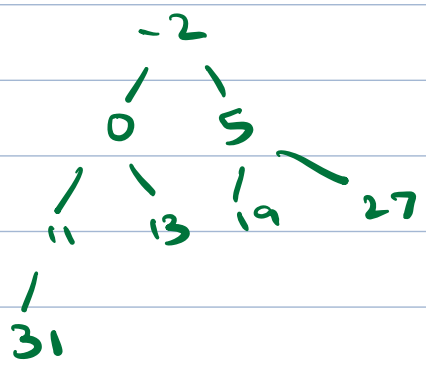
Build Heap [Min Heap]

[5 13 -2 11 27 31 0 19]



Idea 1 : Sort the arr

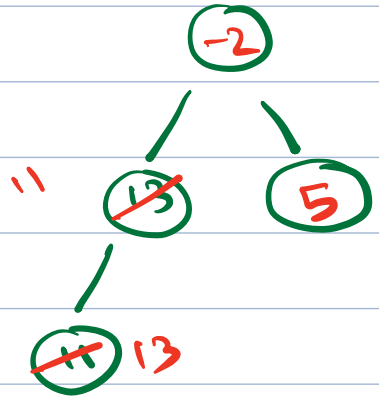
[-2 0 5 11 13 19 27 31]



TC: O(NlogN)

[5 13 -2 11 27 31 0 19]

Idea 2: Insert elements one by one in heap



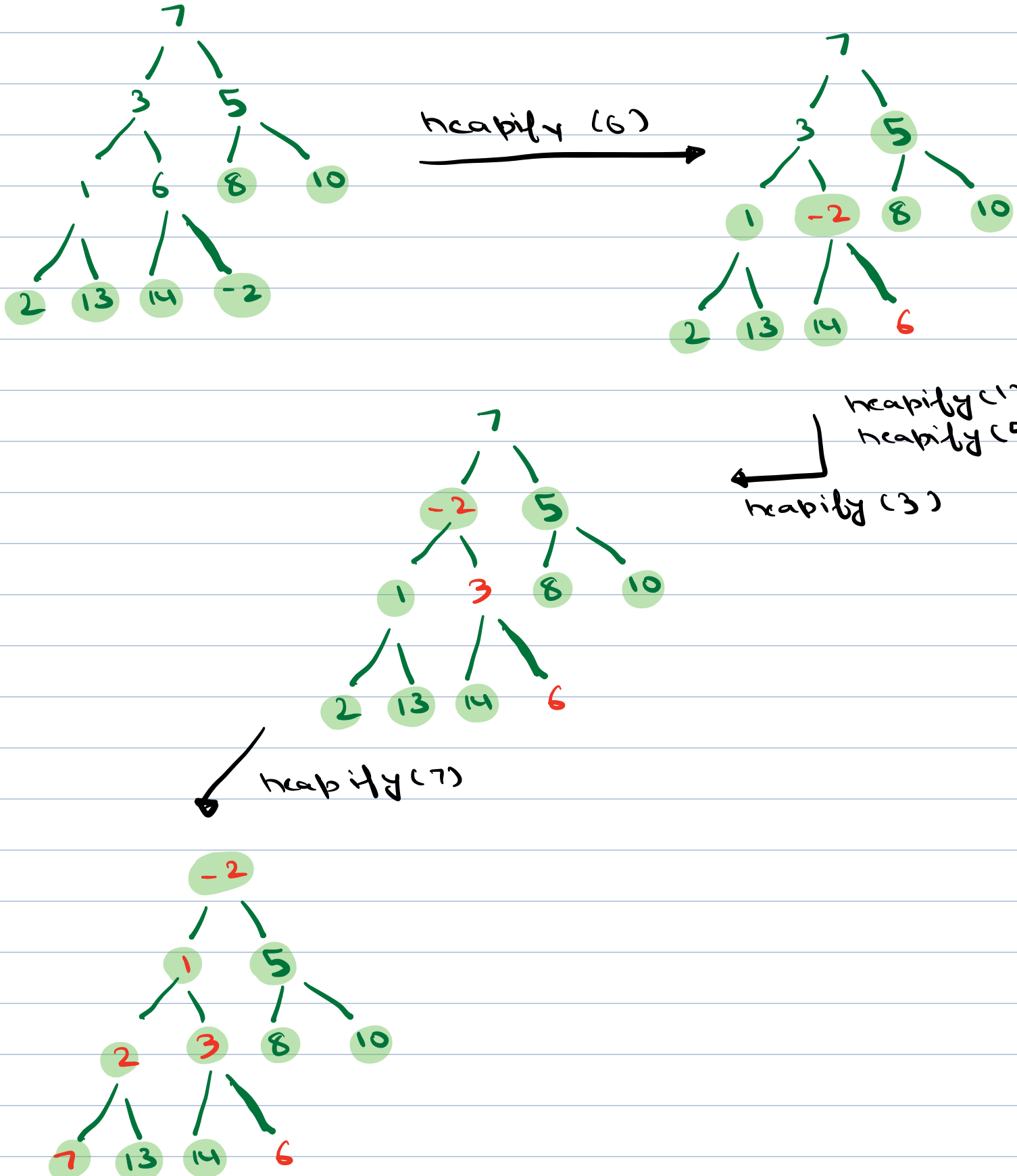
heap

[5 13] -2

-2 5

TC: O(N log$_2$N)

Idea 3: [7, 3, 5, 1, 6, 8, 10, 2, 13, 14, -2]



Last Non-Leaf Node

↓

Parent of Last leaf

Heap

↓

N

Last leaf (ll) = N-1

Parent = $\frac{i-1}{2}$ = $\frac{N-1-1}{2}$ = $\frac{N-2}{2}$ = $\frac{N}{2}-1$

Last Non-Leaf Node = $\frac{11-2}{2} = \frac{9}{2} = 4$

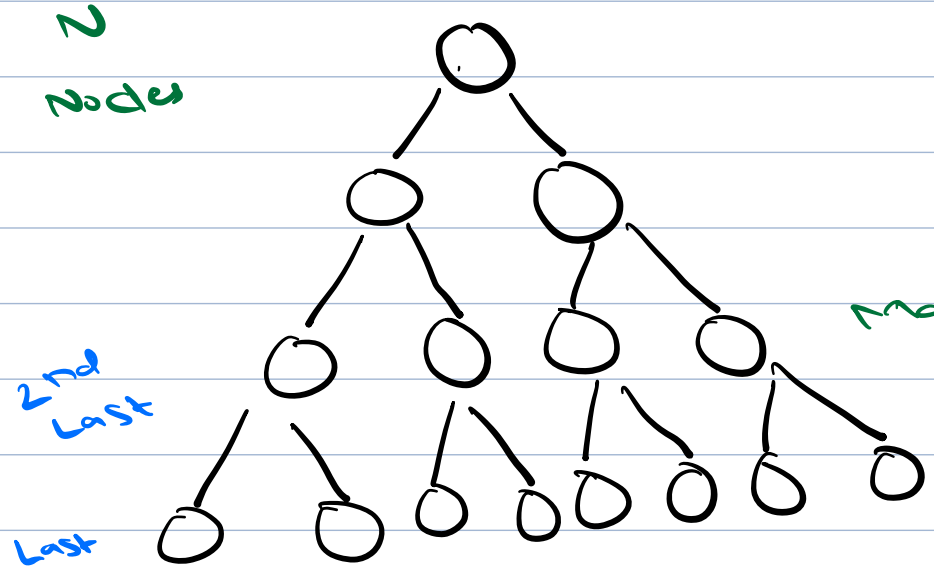
for (i = $\frac{N-2}{2}$; i ≥ 0 ; i--) {

  heapify (heap, i)

}

TC: O(N)

$\times \frac{N}{2} \log_2 N$



Max No. of nodes in Last level = N/2

Max no. of nodes in 2nd last level = N/4

3rd Last → N/8

Total swaps

$$= \frac{N}{2} * 0 + \frac{N}{4} * 1 + \frac{N}{8} * 2 + \frac{N}{16} * 3 + \dots$$

($\frac{N}{2} * 0 = 0$)

$$= \frac{N}{2}\left(\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \dots\right)$$

AGP

$$= \frac{N}{2} * 2 = N$$

$$S = \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \dots$$

$$- \quad \frac{S}{2} = \quad \frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \dots$$

$$S/2 = \frac{1/2 + 1/4 + 1/8 + 1/16 + \dots}{}$$

GP, $r = \frac{1}{2}$

$$\text{Sum} = \frac{a}{1-r}$$

$$S/2 = \frac{1/2}{1 - 1/2}$$

$$S/2 = 1 \quad \Rightarrow S = 2$$

Merge N Sorted Arrays

0 - [2, 3, 11, 15, 20]
1 - [1, 5, 7, 9]
2 - [0, 2, 4]
3 - [-2, 5, 10, 20]

We've to merge these sorted arrays.

Idea:

- If we want to merge 2 sorted arrays, then we need 2 pointers.
- If we want to merge 3 sorted arrays, then we need 3 pointers.
- If we want to merge n sorted arrays, then we need n pointers. ⇒ complexity becomes very high we need to keep track of N pointers.

Optimized: