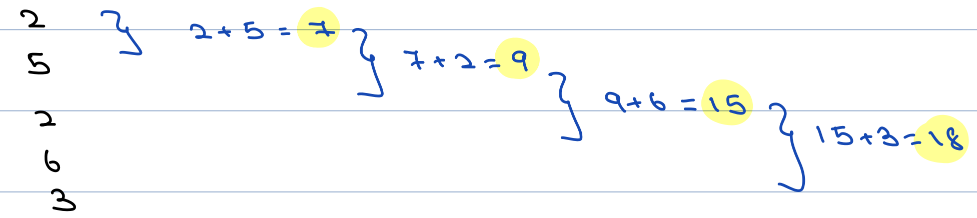
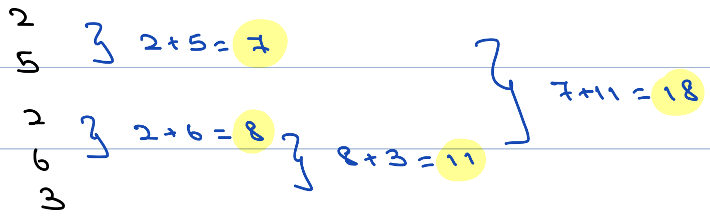
Ques Connecting the ropes →

- We are given an array that represents the size of different ropes.
- In a single operation, you can connect two ropes.
- Cost of connecting two ropes -> sum of the length of ropes you are connecting.
- Find the minimum cost of connecting all the ropes.

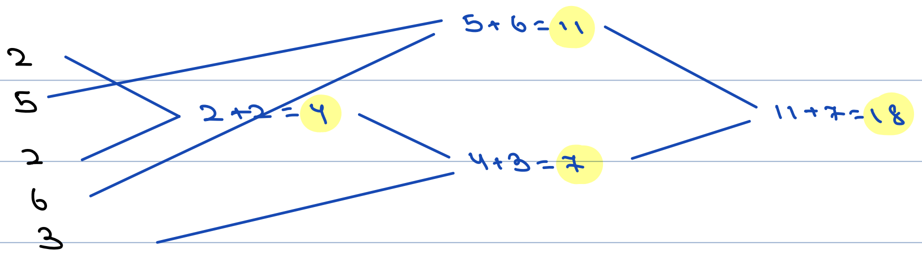
e.g,



Overall Cost = 7 + 9+15+ 18 = 43



Total Cost = 7+8+ 11 + 18 = 44



Total Cost = 4+ 11+7+18 = 40

Let's say $x < y < z$

$$\begin{matrix}(x+y) \\ + \\ (x+y+z)\end{matrix} < \begin{matrix}(x+z) \\ + \\ (x+y+z)\end{matrix} < \begin{matrix}(y+z) \\ + \\ (x+y+z)\end{matrix}$$

idea :- Always pick smallest two ropes all the time.

Soln1 :- Sorting → Insertion Sort.

→ T.C → $O(n^2)$.

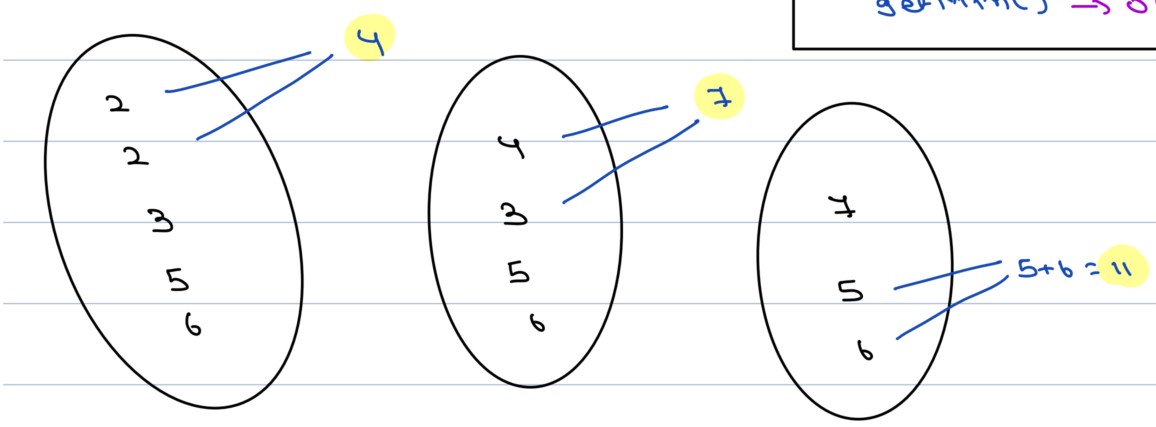
$$\underbrace{\underbrace{2 \quad 2}_{4} \quad 3}_{7} \quad 5 \quad 6 \quad 10$$

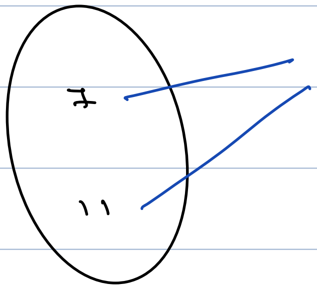
Soln 2 :- Data Structure

heap

insert(x) } deletemin() } → log(n)

getmin() → O(1)





11+7 = 18

Overall Cost :- 4+7+11+18

= 40

Initial insertions → n*logn .

n-1 { 2 deleteMin() + 1 insertion()] → O(nlogn)

T.C → O(nlogn) .

S.C → O(n)

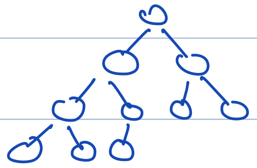
Heap Data Structure (Binary Heap) .

condn :

① Structure → Complete Binary Tree .

↓

all the levels are completely filled, except maybe the last level and that should also be filled from left to right .



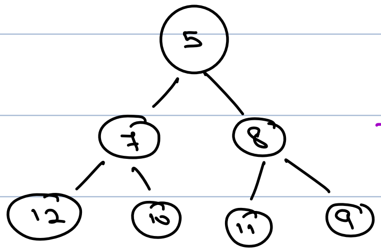
Condn 2)

2) Heap Order Property.

↓

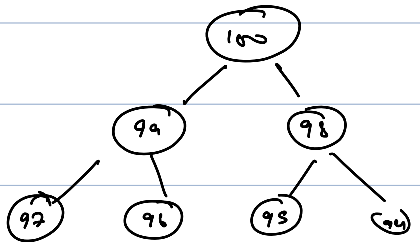
Parent nodes should have higher priority from with children .

Min heap



→ min heap.

No relation b/w left & right child.



→ max heap.

fyi :-

Types of heap :-

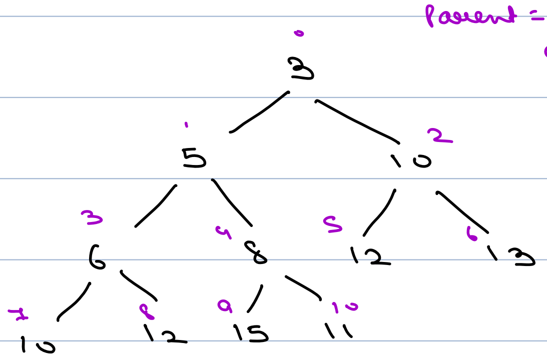
min heap

max heap

Array implementation of Heaps :-

	0	1	2	3	4	5	6	7	8	9	10
arr[] =	{ 3,	5,	10 ,	6,	8 ,	12,	13,	10,	12,	15,	11 }

↳ CBT .



Parent = $\frac{(i-1)}{2}$

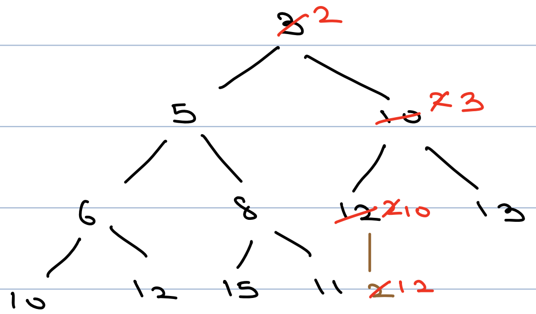
c ⟶ : ⟶ LC (2*i+1)

⟶ RC (2*i+2)

→ min heap .

1) Insert

	0	1	2	3	4	5	6	7	8	9	10
arr[] =	{ 3,	5,	10 ,	6,	8 ,	12,	13,	10,	12,	15,	11 }



insert (2)

0 1 2 3 4 5 6 7 8 9 10 11

arr[] = { ~~8~~, 5, ~~10~~, 6, 8, ~~12~~, 13, 10, 12, 15, 11, ~~2~~ }

2   ~~2~~ 3   ~~2~~ 10   12

insert (2)

11 → parent → $\frac{(11-1)}{2} = 5$

swap (11, 5)

5 → parent → $\frac{(5-1)}{2} = 2$

swap (2, 5)

2 → parent → $\frac{(2-1)}{2}$ ⇒ 0

swap (0, 2)

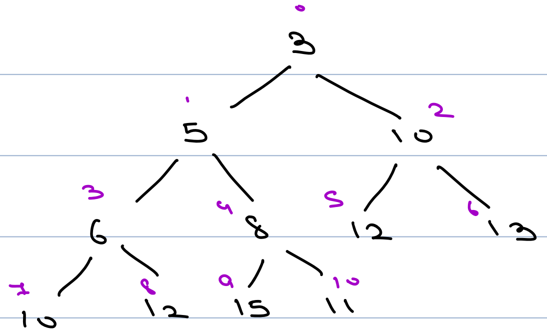
```
// Given heap[]
heap.insert (val);      // insert at last .
i = heap.size () - 1;
while (i > 0) {
        pi = (i-1)/2 ;
        if (heap[pi] > heap[i]) {
                swap( heap, pi, i);
                i = pi;
        }
        else {
                break;
        }
}
```

T.C → O(logn)

→ Get Min :-

arr[] =

0	1	2	3	4	5	6	7	8	9	10
3	5	10	6	8	12	13	10	12	15	11



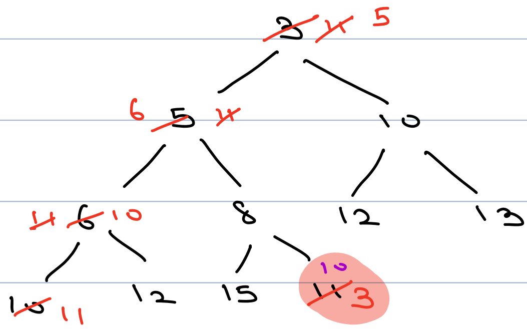
return arr[0].

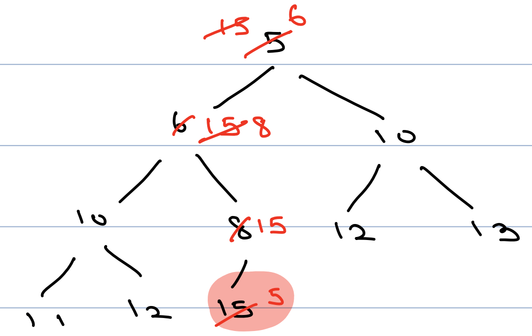
T.C → O(1).

→ Delete Min :-

arr[] =

0	1	2	3	4	5	6	7	8	9	10
3	5	10	6	8	12	13	10	12	15	11





	0	1	2	3	4	5	6	7	8	9	10
arr[] = {	~~8~~ ~~11~~ 5,	~~5~~ ~~11~~ 6,	10,	~~6~~ ~~11~~ 10,	8,	12,	13,	~~10~~ 11,	12,	15,	~~11~~ 3 }

i

0 → LC → 1
  → RC → 2

swap (0,1)

1 → LC → 3
  → RC 4

swap (1,3)

3 → LC → 7
  → RC → 8

swap (3,7)

7 → LC → 15
  → RC → 16

```
// given heap arr[];
swap ( heap, 0, heap.size()-1);
heap.remove ( heap.size() - 1);
heapify ( heap[], 0);
```

```
void  heapify (heap[], i) {

while( 2*i+1 < N) {
        x = Min( heap[i], heap[2*i+1], heap[2*i+2] );

        if (x = = heap[i]) {
              break;
        }
        else if (x = heap[2*i+1]) {
              swap( heap, i, 2*i+1);
              i=2*i+1;
        }
        else {
              swap (heap, i, 2*i+2);
              i=2*i+2;
        }
}
```

Edge case when right child is invalid.

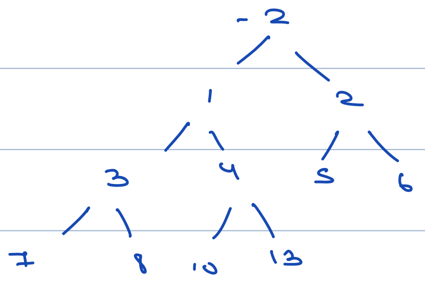
T.C → $O(\underline{\log n})$.

Build a heap :-

arr[] = 7, 3, 5, 1, 6, 8, 10, 2, 13, 4, -2

idea1 :- sort the array.

-2, 1, 2, 3, 4, 5, 6, 7, 8, 10, 13



T.C -> O(nlogn)

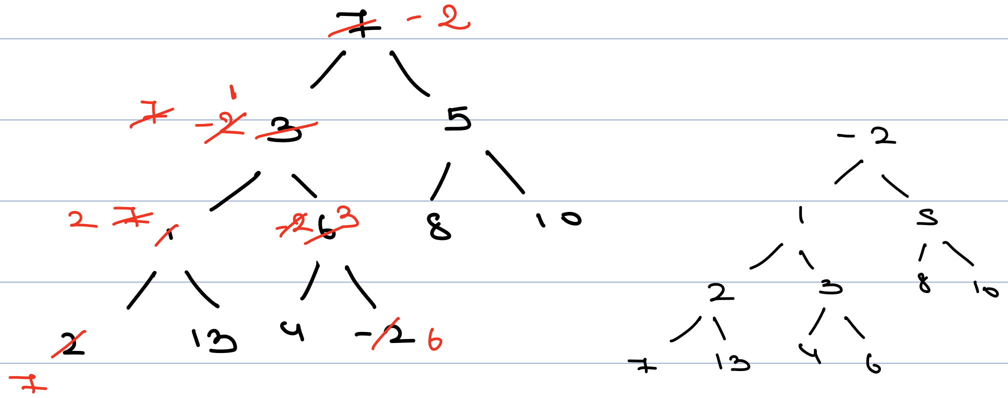
idea 2 :- call insert function for all elements.

T.C -> O(nlogn).

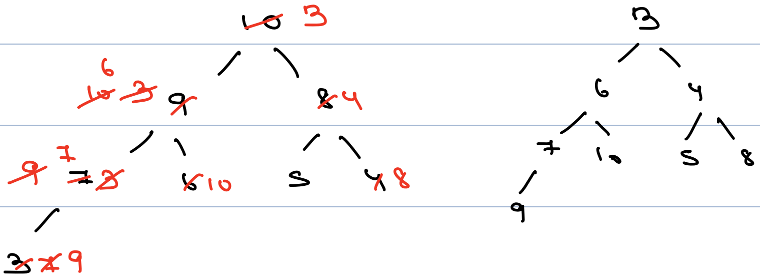
idea 3 :-

arr[] = 7, 3, 5, 1, 6, 8, 10, 2, 13, 4, -2

leaf elements maintain the heap order property.



ex 2   arr[] → 10, 9, 8, 7, 6, 5, 4, 3



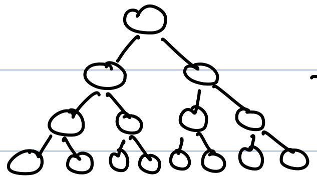
last node = n-1

↓ parent

$\frac{n-1-1}{2} \Rightarrow \frac{n-2}{2} \Rightarrow \frac{n}{2}-1$

first non leaf node.

```
for (i=(n/2 -1) ; i>=0; i--) {
        heapify (heap[], i);
}
```



nodes.	swap
1 ⋮	3
→ n/8	2
→ n/4	1
→ n/2	0

T.C → $\left[\cancel{\frac{n}{2} \times 0}^{\,0} + \frac{n}{4} * 1 + \frac{n}{8} \times 2 + \frac{n}{16} * 3 \cdots \right]$

$\Rightarrow \frac{n}{2} \left[\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots \right]$

↓

AGP ⇒ 2.

$\Rightarrow \frac{n}{\cancel{2}} (2) \Rightarrow n$

T.C → O(n) → creating a heap from arr/list directly.

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

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

$$\frac{S}{2} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots \rightarrow g.p.$$

$$\frac{S}{2} = \frac{\frac{1}{2}}{1-\frac{1}{2}} \Rightarrow \frac{S}{2} = 1 \Rightarrow \underline{S = 2}.$$

Ques Merge N sorted LL into 1 sorted LL.

H_1 1 → 3 → 7 → 12 → N

H_2 2 → 6 → 18 → N

H_3 5 → 10 → 20 → N

H_4 7 → 19 → N

Brute force :-

Let's say, we have K lists of len N.

N, N, N . . . N

2N

3N

4N

Total = 2N + 3N + 4N + . . KN

$\Rightarrow N(2+3+4+\dots K) \Rightarrow \underline{N * K^2}$.

Priority _ Queue.

idea :- Min heap

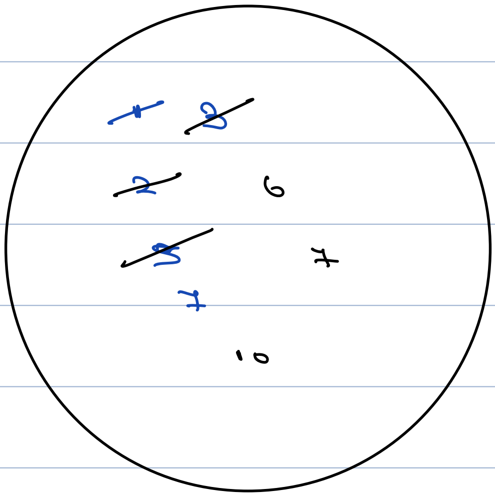
H1 1 → 3 → 7 → 12 → N

H2 2 → 6 → 18 → N

H3 5 → 10 → 20 → N

H4 7 → 19 → N

min heap.



1 → 2 → 3 → 5

we have k list of len N

T.C → n*k (log k)

S.C → O(k)

→ initially insert head of every list

→ pick minimum from the heap add it to your answer list, which ever list element you consumed add the next element of that list in the Min heap