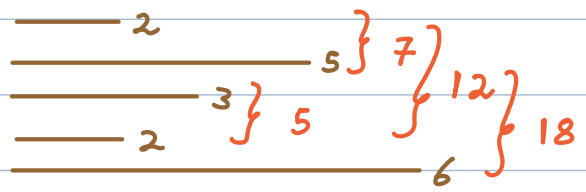
Q → Given an integer array,

A[i] → length of rope.

In one operation we can connect 2 ropes & the **cost** is **sum of length** of ropes.

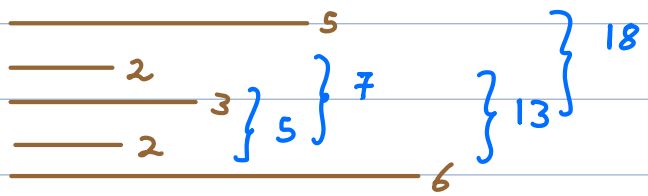
Find **min cost** to connect all ropes.

A = [2 5 3 2 6]



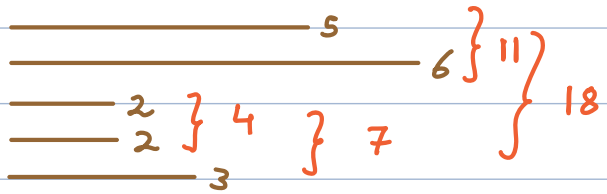
cost = 7 + 5 + 12 + 18

= 42



cost = 5 + 7 + 13 + 18

= 43



cost = 11 + 4 + 7 + 18

= 40 ✓

length of 3 ropes $x < y < z$

$x+y$ < $x+z$ < $y+z$

$(x+y)+z$ $(x+z)+y$ $(y+z)+x$

⇒ connecting small length rope gives less cost.

Sol 1 → Sort

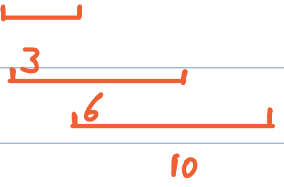
A = [2 2 3 5 6]

cost = 4 + 7 ...

insertion sort

TC = O(N²) SC = O(1)

A = [1 2 3 4]



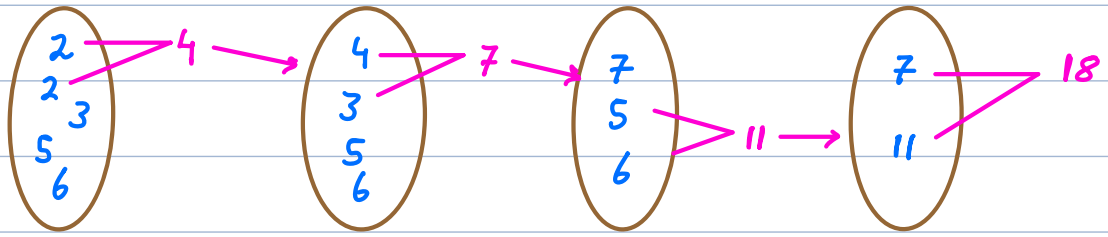
cost = 3 + 6 + 10

= 19

let say → DS with operations →

1) Insertion is O(log(N))

2) Get min element is O(log(N))



cost = 4 + 7 + 11 + 18 = 40

TC = O(N log(N))

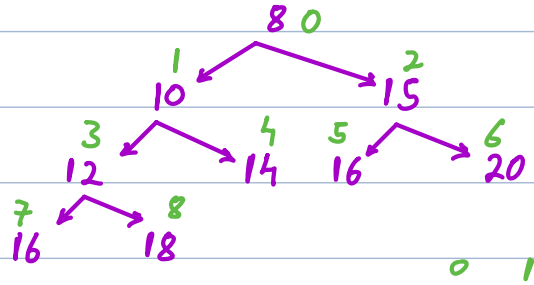
SC = O(N)

Heap Data Structure

1) Structure → complete binary tree

2) Types → Min Heap ∀ nodes, node.data <= node.left & node.right

Max Heap ∀ nodes, node.data >= node.left & node.right



	0	1	2	3	4	5	6	7	8
A = [	8	10	15	12	14	16	20	16	18]

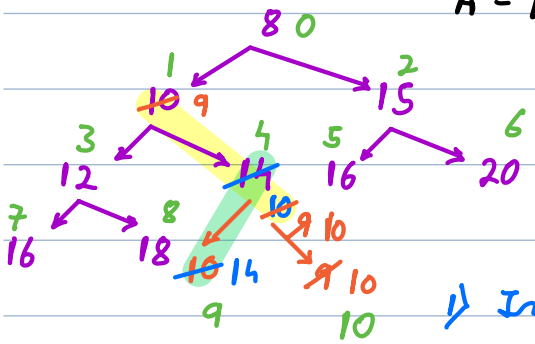
Left child → $2*i+1$

Right child → $2*i+2$

Parent → $(i-1)/2$

Insertion

	0	1	2	3	4	5	6	7	8
A = [	8	10	15	12	14	16	20	16	18]



insert (10) ✓

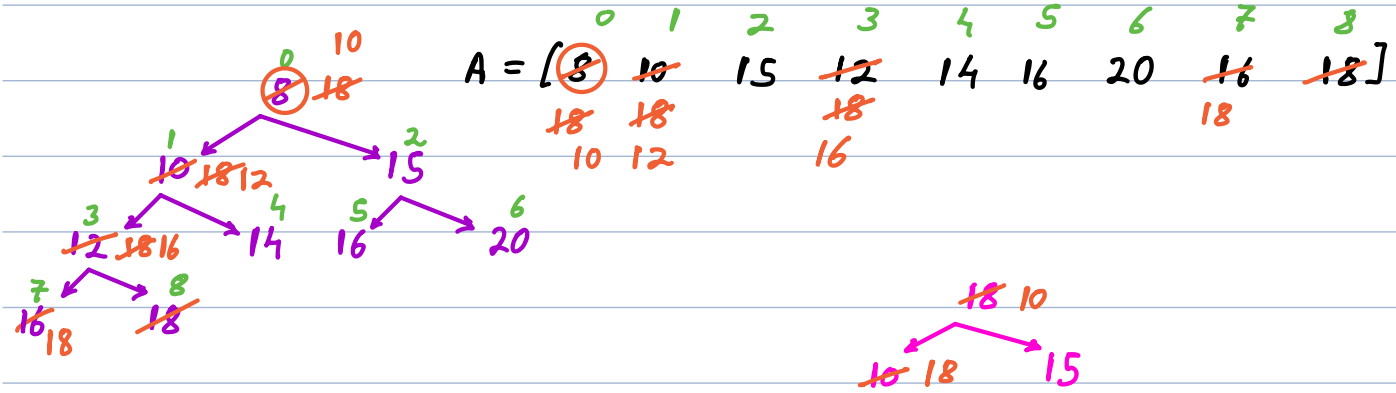
1) Insert as next node in complete binary tree.
2) check with parent node, if cur < parent ⇒ swap & repeat step 2.

insert (9)

```
// insert(x)
    i = N
    A[i] = X        N++   // count of elements
    while (i > 0) {
        p = (i-1)/2
        if (A[p] > A[i]) {        TC = O(log(N))
            swap(A, p, i)         SC = O(1)
```

```
            i = p
        }
        else  break
    }
```

Get Min




```
ans = A[0]
A[0] = A[N-1]        N--
i = 0
while ( i < N) {
    lc = 2*i + 1
    rc = 2*i + 2
    if ( lc < N && rc < N) {
        if (A[lc] <= A[rc] && A[lc] <= A[i]) {
            swap(A, i, lc)
            i = lc
        } else if ( A[rc] <= A[lc] && A[rc] <= A[i]) {
            swap(A, i, rc)
            i = rc
        } else  break
    } else if ( lc < N) {
        if ( A[lc] <= A[i]) {
```

```
            swap(A, i, lc)
            i = lc
        } else break
    }
    else break
}
```

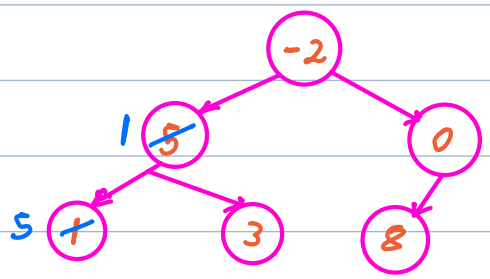
TC = O(log (N))

Build Heap

1) Insert element one by one → TC = O(N log (N))

2) Bottom up build ↓

A = [-2 0 5 1 3 8]



swaps per element

2 :

1

0

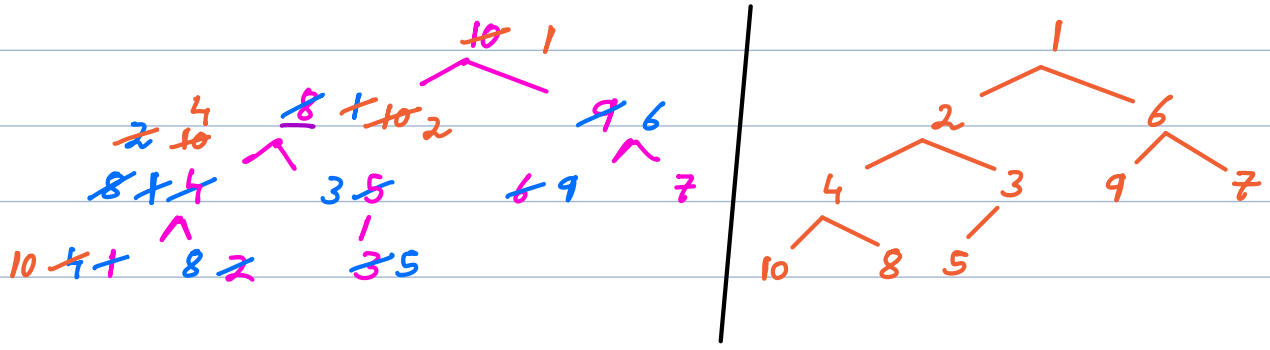
Total # steps = $\frac{N}{2} * 0 + \frac{N}{4} * 1 + \frac{N}{8} * 2 \ldots$

$= \frac{N}{2}\left(\frac{1}{2} + \frac{2}{4} + \ldots\right)$

$= \frac{N}{2}\left[\sum_{\forall i}\left(\frac{i}{2^i}\right)\right] \rightarrow \approx 2$ $= \frac{N}{2} * 2 = N$

TC = O(N)

A = [1 2 3 4 5 6 7 8 9 10]



Build Heap → TC = O(N)

Q → Given K sorted arrays, merge them into 1 sorted array.

A = [2 → 3 → 5 10]
B = [1 → 4 5]
C = [3 → 6 8 12 15]
D = [5 6 7 8]
↑

Use K pointers to merge all arrays together.

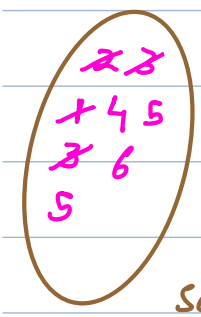
Merge 2 arrays
(Assuming length = N)

$2N + 3N + 4N \ldots + KN$

$= N(2 + 3 + \ldots + K)$

$= N * \left(\frac{K * (K+1)}{2} - 1\right)$

⇒ TC = $O(N * K^2)$



Min Heap 1 2 3 3 . . .

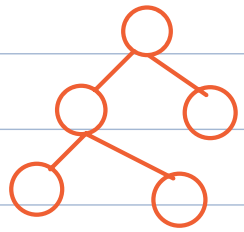
SC = O(K)

TC = $O(K * N * \log(K))$

Complete Binary Tree

N = 1 2 3 4 5 ...

#leaf = 1 1 2 2 3 ...



$(N+1)/2$