# Agenda

1) Heap DS
2) Serialisation of Tree
3) Insertion / Deletion
4) Build a Heap
5) Merge N sorted arrays.

---

**Q** Given an integer array represents the length of N ropes.

In one operation you can connect two ropes.

Cost of connecting 2 ropes = Sum of length of both ropes.

Find the min cost to connect all ropes.

Eg: $A = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ [\underline{2}, & \underline{5}, & 3, & 2, & 6] \end{array}$

Way 1    1) $2 + 5 = 7$

$[7, 3, 2, 6]$

2) $7 + 3 = 10$

$[10, 2, 6]$

3) $10 + 2 = 12$

$[12, 6]$

4) $12 + 6 = 18$

$7 + 10 + 12 + 18 = \underline{\underline{47}}$

$$A = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ \end{array}$$
$$A = [2, 5, 3, 2, 6]$$

1) $2 + 2 = 4$

$[4, 5, 3, 6]$

2) $4 + 3 = 7$

$[7, 5, 6] \qquad = \quad \underline{\underline{40}}$

3) $\underline{5} + \underline{6} = \boxed{11}$

$[11, 7]$

4) $11 + 7 = \boxed{18}$

## $\underline{\underline{\text{Quiz}}}$

$A = [1, 2, 3, 4]$

1) $1 + 2 = 3$

$[3, 3, 4]$

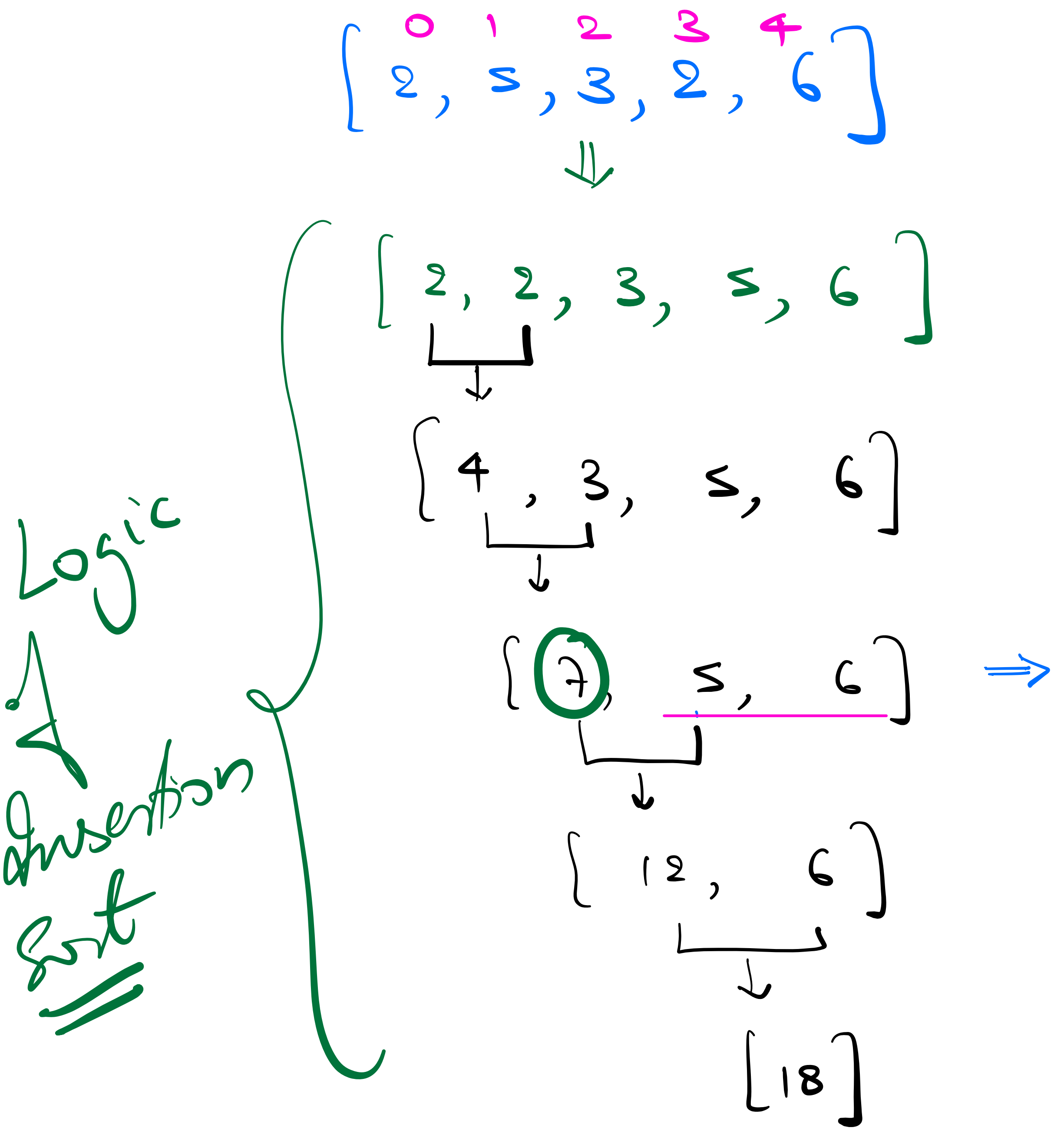
2) $3 + 3 = 6$

$[6, 4]$

$\boxed{\underline{\underline{19}}}$

3) $6 + 4 = 10$

$\underline{\underline{\text{Sol}^n}}$ 1) Sorting and adding



$T.C. = O(N^2)$

Idea ⇒ Need the min & 2nd min everytime ⇒ $O(1)$

⇒ Insert a new element after every step ⇒ O(N)

3 ropes : $x$, $y$, $z$ $(x < y < z)$

| | (I) | (II) | III |
|---|---|---|---|
| Step 1 | $x + y$ min | $x + z$ | $y + z$ |
| Step 2 | $(x+y) + z$ | $(x+z) + y$ | $(y+z) + x$ |

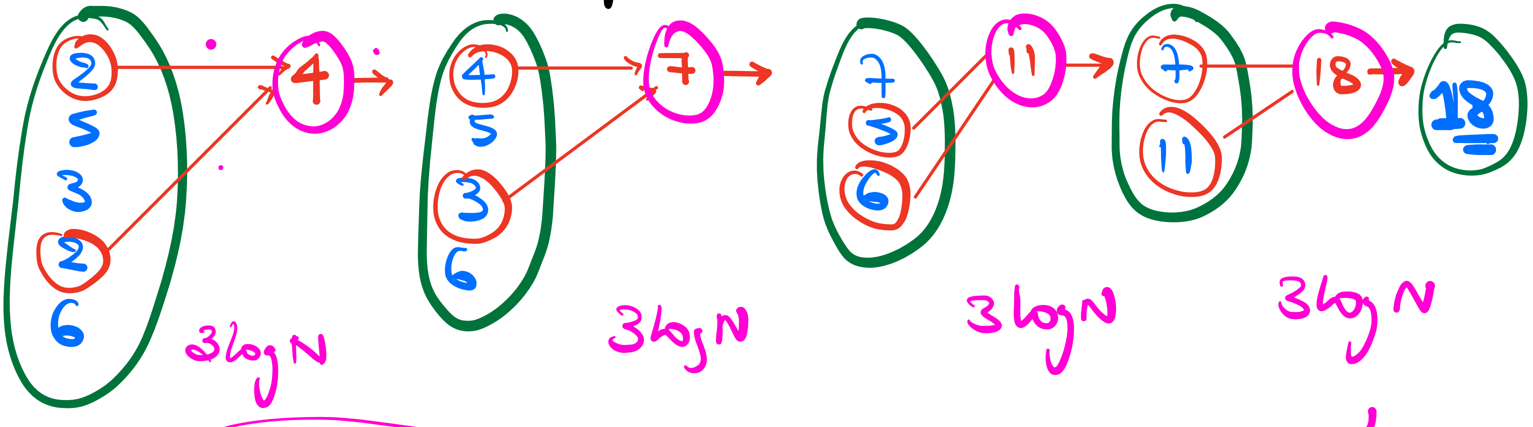
min

Same for all combi.

Let's image if we have a DS.

1) Insertion of elements ⇒ O(log N)

2) Extraction of min element ⇒ O(log N)



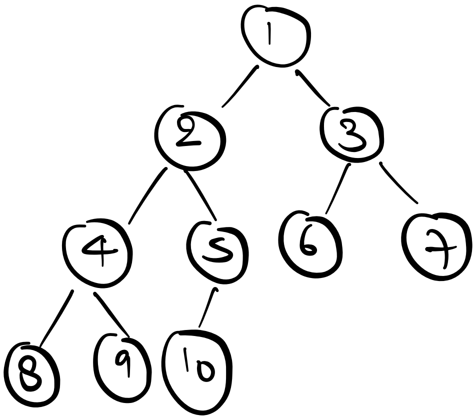
(N-1) operation

T.C = O(N log N)

# Heaps

1) Heap is a Complete Binary Tree

⇓



→ All levels are filled except last level

→ Last level is filled from left

2) Heap Order Property.

(Max & Min)

Min Heap.

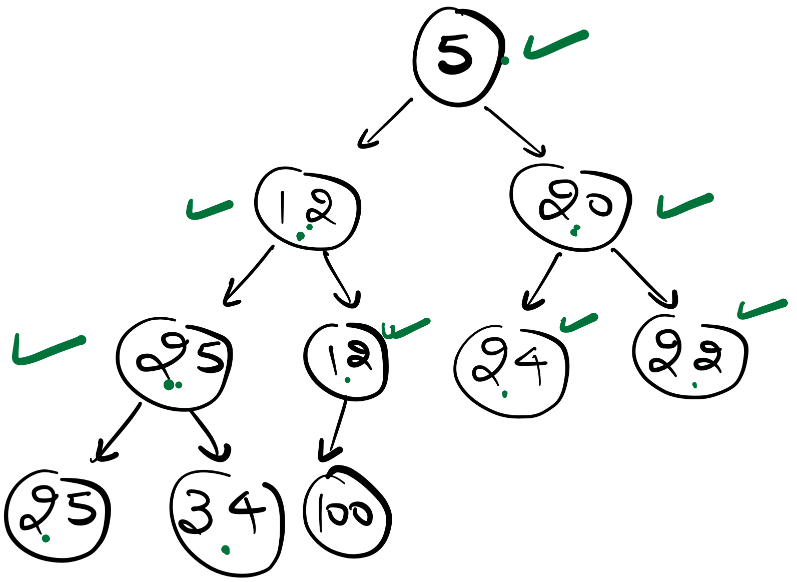
∀nodes ⇒ node.data ≤ node.left.data

&

node.data ≤ node.right.data

# Max Heap.

∀nodes ⇒ node.data >= node.left.data
&
node.data >= node.right.data

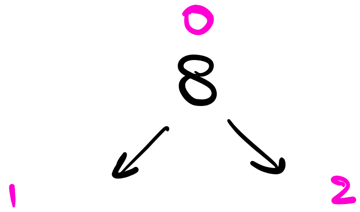


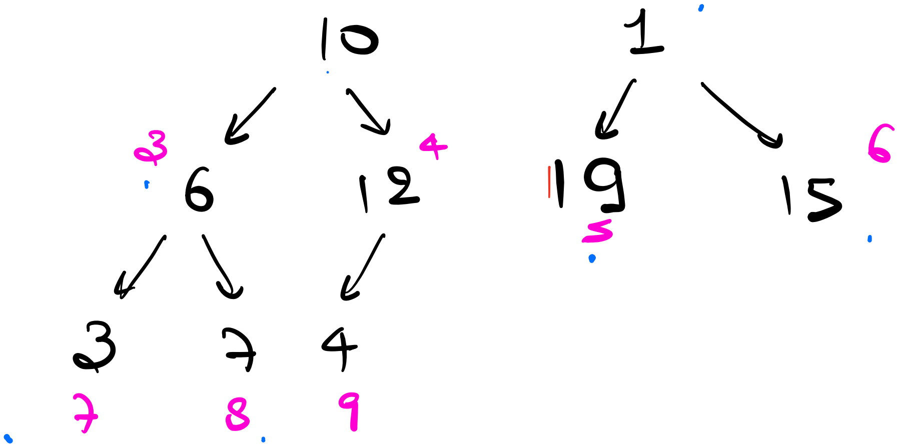
⇒ Min Heap

**NOTE** : In a Heap, there is no relation b/w the left & right of a node

---

# Serialisation of Binary Tree





$$A = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 8, & 10, & 1, & 6, & 12, & 9, & 15, & 3, & 7, & 4 \end{array}\right]$$

Serialisation of Tree.

Representing tree in a level order wise traversal.

| Node | Left Child | Right Child |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 4 |
| 2 | 5 | 6 |
| 3 | 7 | 8 |
| 4 | 9 | × |

Index of Parent Node = $i$

↳ Left Child = $2 \times i + 1$

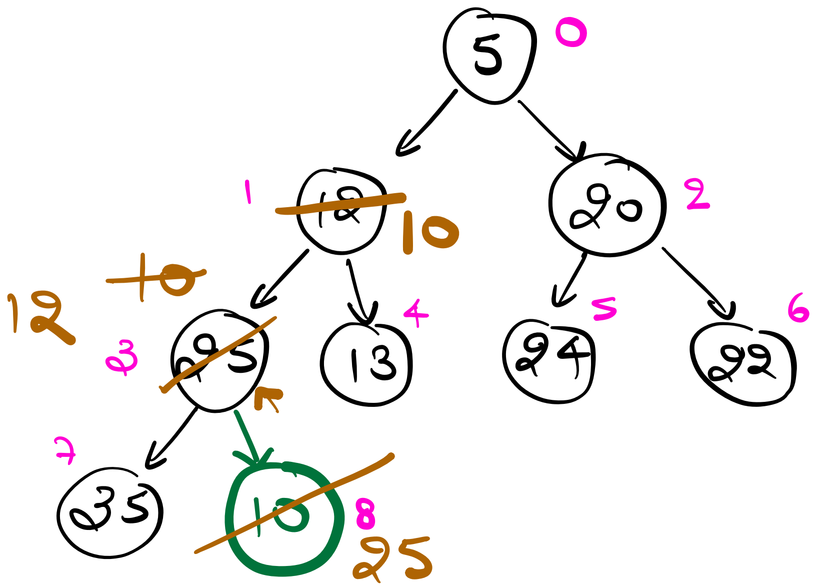
↳ Right Child = $2 \times i + 2$

Index of Child Node = $i$

↳ Parent Node = $(i - 1)/2$

---

## Insertion in Min Heap.



$$A = \left[\begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 5, & \cancel{12}, & 20, & \cancel{25}, & 13, & 24, & 22, & 35 & \cancel{10} \\ & 10 & & \cancel{10} & & & & & 25 \\ & & & 12 & & & & & \end{array}\right]$$

Insert $\boxed{10}$

## Heapify

⇒ Process of maintaining Heap property after insertion/deletion.

# Up- Heapify.

## Code

Heap ↑ index of element ↑

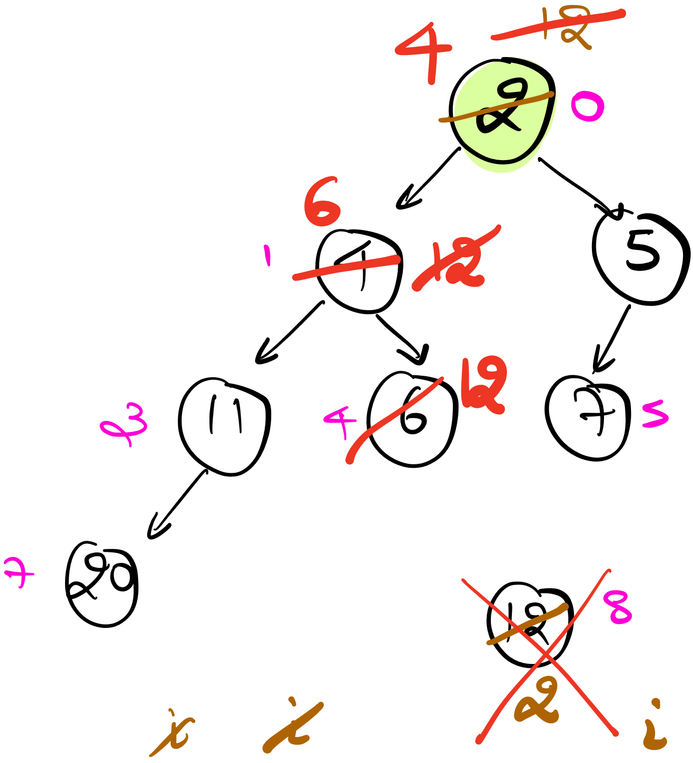
```
void upHepify(A, index) {
    int parent = (index -1)/2
    while (index != 0 && (A[parent] > A[index])) {
        temp = A[parent];
        A[parent] = A[index];
        A[index] = temp;

        index = parent;
        parent = (index -1)/2
    }
}
```

```
            Dynamic Array
void insert (A, value) {
        A.add(value);
        upHepify(A, A.size()-1);
}
```

$T.C. = O(H) = O(\log_2 N)$

---

## Extract Min



Heap =

| ~~i~~ | ~~i~~ | | | i | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ~~8~~ |
| [~~2~~, | ~~4~~, | 5, | 11, | ~~6~~, | 7, | 8, | 20] | ~~12~~ |
| ~~12~~ | ~~12~~ | | | 12 | | | | 2 |

4 6

1) Swap first & last element

## Code

```
swap(Heap, 0, Heap.size() - 1);
Heap.remove(Heap.size() - 1);
downHeapify(Heap, 0);

                                    0
                                    ↑
void downHeapify(Heap, i) {
    int N = Heap.size();

    int lc = 2×i + 1;
    int rc = 2×i + 2;

    while ( lc < N) {

        if (rc == N) {
                if ( Heap[i] > Heap[lc]) {
                        swap;
                }
                break;
}
```

lc is min

```
if (Heap[lc] < Heap[i] && Heap[lc] <=
    Heap[rc]) {

        swap(i, lc);
        i = lc;
        lc = i*2+1
        rc = i*2+2;
}
```

rc is min

```
else if (Heap[rc] < Heap[i] &&
    Heap[rc] < Heap[lc]) {

        swap(i, rc);
        i = rc;
        lc = i*2+1
        rc = i*2+2;
```

root is min

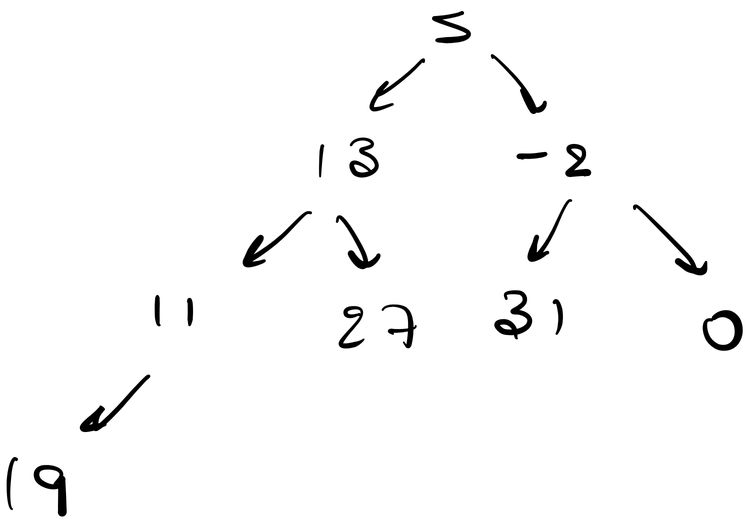
```
} else {
        break;
}
    }
}
```

T.C. = $O(H) = O(\log N)$

---

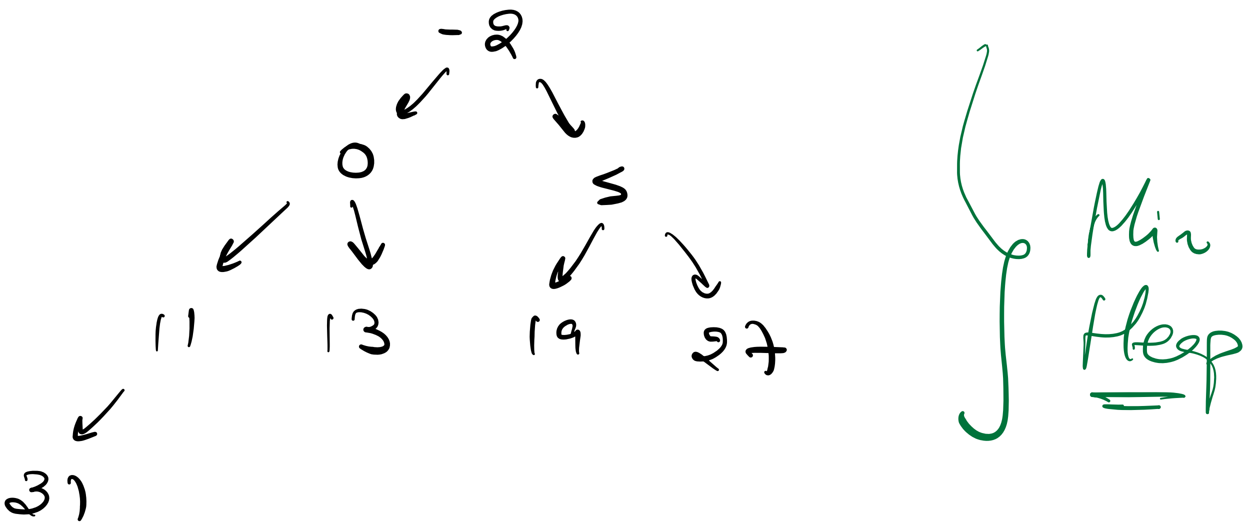
# Build a Heap

$A = [5, 13, -2, 11, 27, 31, 0, 19]$



Idea 1 : Sort the Array.

$A = [-2, 0, 5, 11, 13, 19, 27, 31]$



$T.C. = O(N \log N)$

$S.C. = O(\log N)$ → Recursive

**Idea 2** : Insert all array elements in Empty Heap

$T.C. = O(\log N \times N)$

$S.C. = O(N)$

**Idea 3** : H.W. Read about it

$T.C. = O(N)$

$S.C. = O(1)$

---

# Merge N Sorted Array

A: [2, 3, 11, 15, 20]

B: [1, 5, 7, 9]

C: [0, 2, 4]

D: [3, 4, 5, 6, 7, 8]

E: [-2, 5, 10, 20]

min of all first elements.

2, 1, 0, 3, ~~-2~~

5

⇑ Max size of

Heap = $\underline{\underline{N}}$

T.C. of o/p = $O(\underline{\log N})$

```
MinHeap < Node >

class Node {
    int arr[];
     int ind;
}

new Node ( oldNode.arr,
           oldNode.ind+1 );
```