SCALER

Heaps - 1

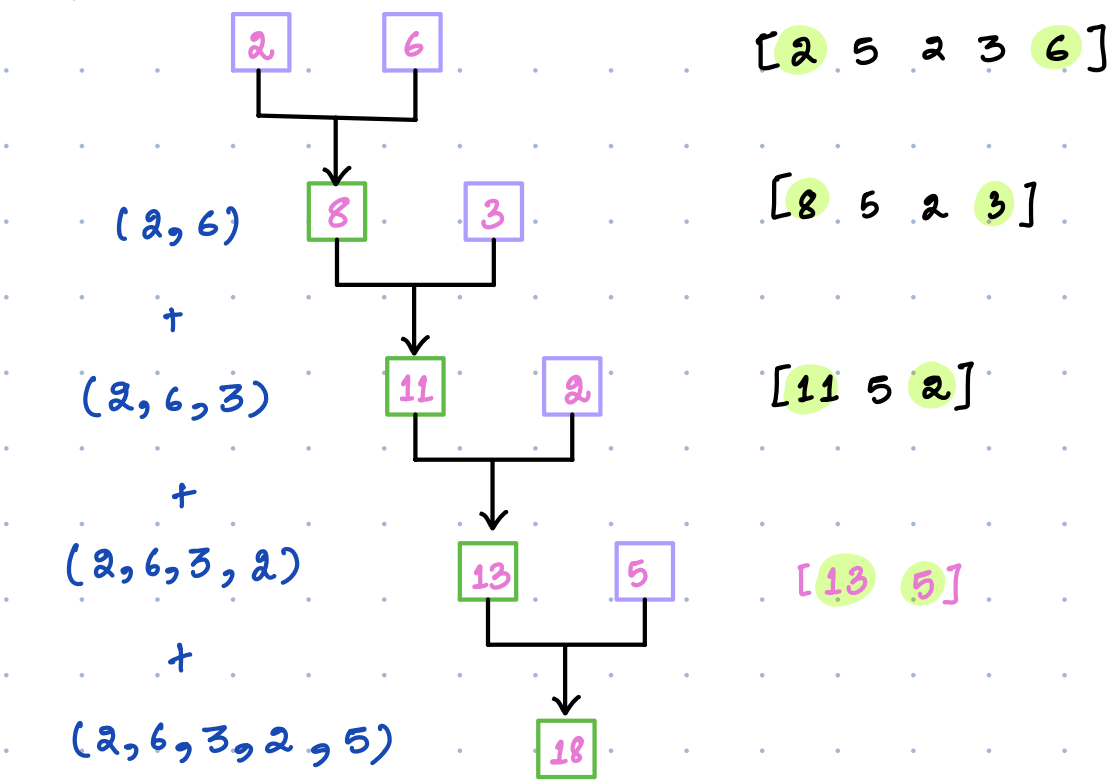
TABLE OF CONTENTS

Connecting the Ropes

Given an array that represents the size of different ropes. In a single operation, you can connect two ropes. Cost of connecting 2 ropes is sum of the ropes of length of the ropes you are connecting. Find the minimum cost of connecting all the ropes.

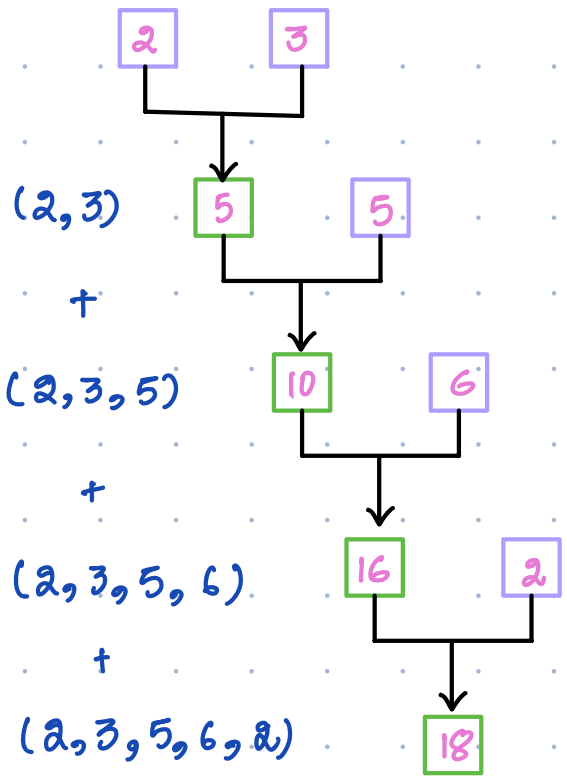
arr[ ] → [2 , 5 , 3 , 2 , 6]

Way 1 :



cost = 8 + 11 + 13 + 18 = 50

Way 2 :



[2 5 2 3 6]

[5 5 2 6]

[10 2 6]

[16 2]

[18]

cost = 5 + 10 + 16 + 18 = 49

[2 3 2 5 6] → [2 2 3 5 6]

[4 3 5 6] → [3 4 5 6]

cost = 4 + 7

[7 5 6]

T.C : $O(N^2 \log N)$

18
12
(7)
4
2
2
3
5
6
(4)

cost = 4 + 7 + 12 + 18 = 41

QUIZ

[1, 2, 3, 4]

[3, 3, 4]

[4, 6]

cost = 3 + 6 + 10 ⇒ 19

Observation : The earlier the element is selected, the more no of times it is contributing in the sum.

To Minimise sum, select 2 smallest elements & combine them.

Approach 1 : Sort the array in asc order → Pick 2 small elements → combine them → Insert the new rope → Sort the array

```
[2 3 2 5 6] → [2 2 3 5 6]
[4 3 5 6]   → [3 4 5 6]
[7 5 6]     → [5 6 7]
[11 7]      → [7 11]
```

Cost = 4 + 7 + 11 + 18 = 40

T.C : $O(N^2 \log N)$

Approach 2 : Use Insertion Sort

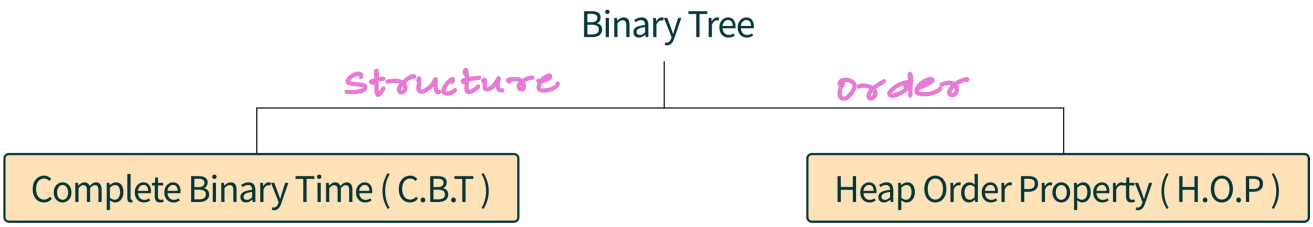
T.C : $O(N^2)$

Requirement:
- Get 2 Min → T.C. $O(\log N)$
- Insert new value → T.C $O(\log N)$

Heap / Priority Queue → DS is used whenever we have to do frequent extraction of MIN/MAX & insertion.

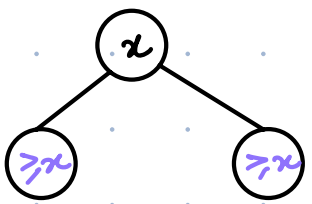
Heap Data Structure

Binary Tree with 2 properties :-

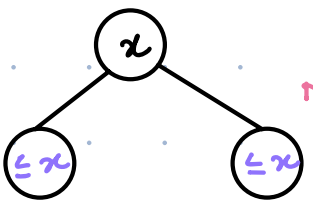


- All levels except last should be completely filled.

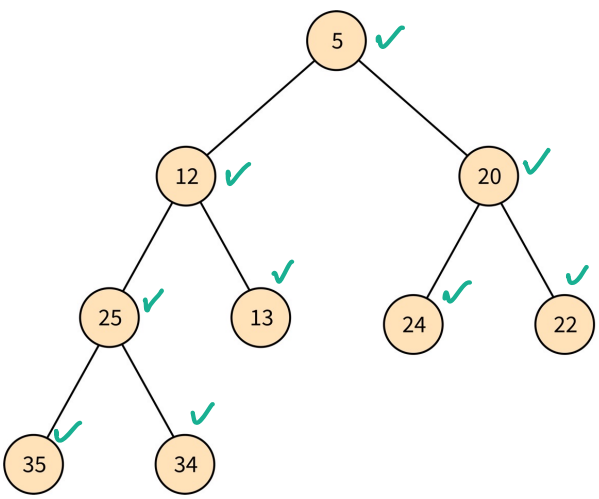
- Last level should be filled from L → R.



MIN HEAP



MAX HEAP



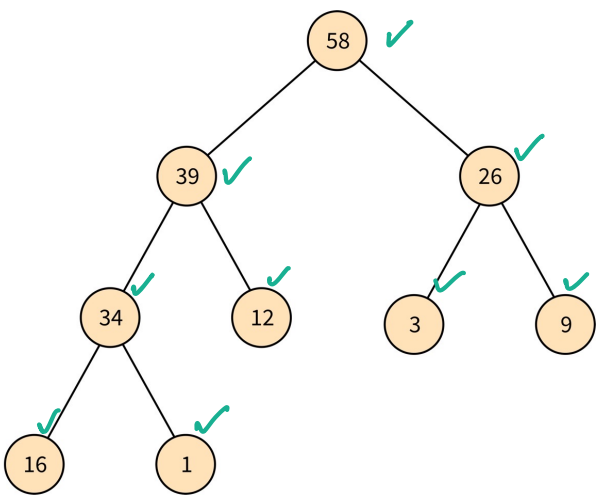
CBT ✓

H.O.P ✓

MIN HEAP.

get Min → O(1)

Access Root value



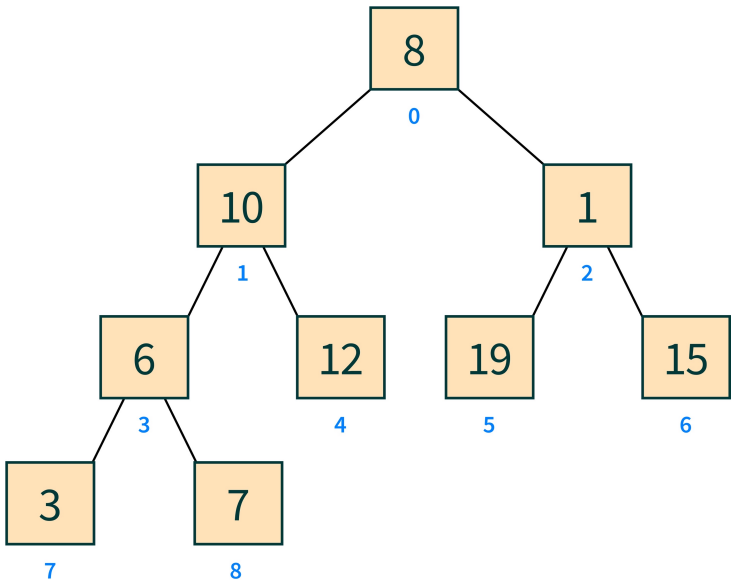
CBT ✓

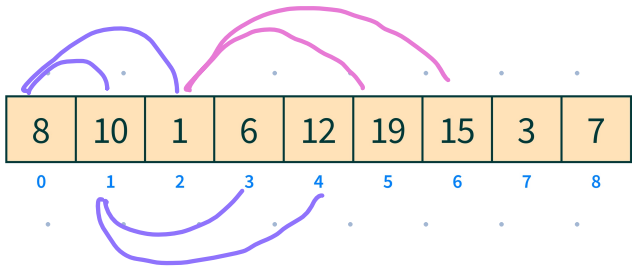
H.O.P ✓

Max Heap.

get Max → O(1)

Visualise Arrays as Complete Binary Tree





i C_i

$0 \rightarrow 1 : 2i+1$

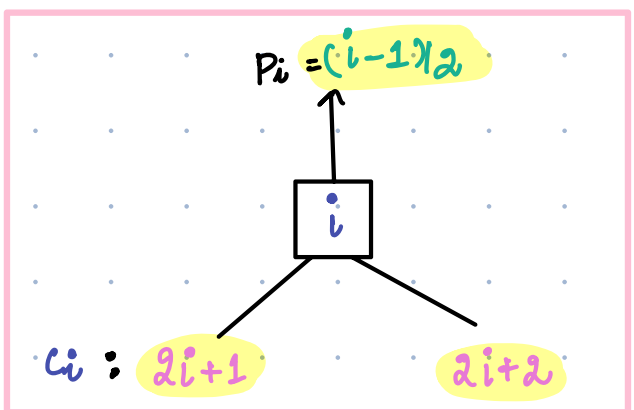
$0 \rightarrow 2 : 2i+2$

$0 \leftarrow 1 \rightarrow 3 : 2*1+1 = 3$

$0 \leftarrow 1 \rightarrow 4 : 2*1+2 = 4$

$0 \leftarrow 2 \rightarrow 5 : 2*2+1 = 5$

$0 \leftarrow 2 \rightarrow 6 : 2*2+2 = 6$

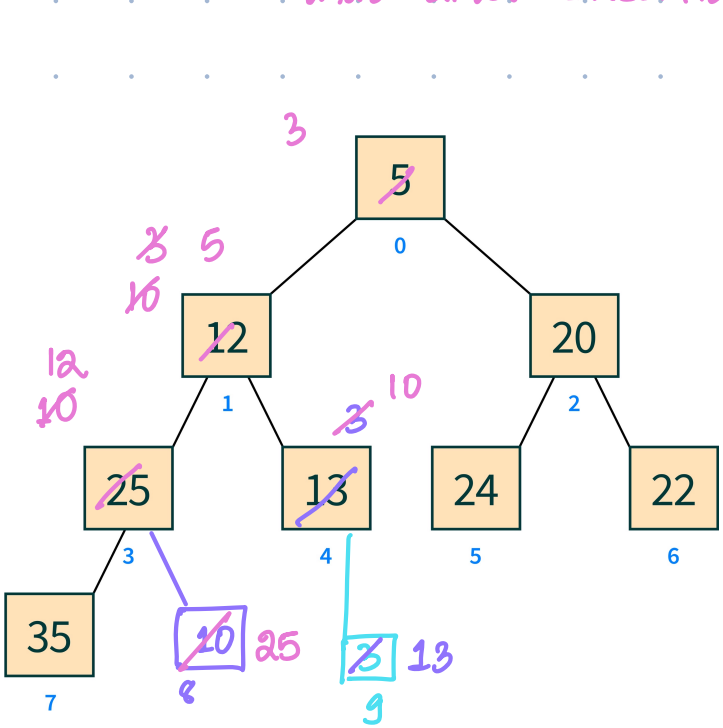


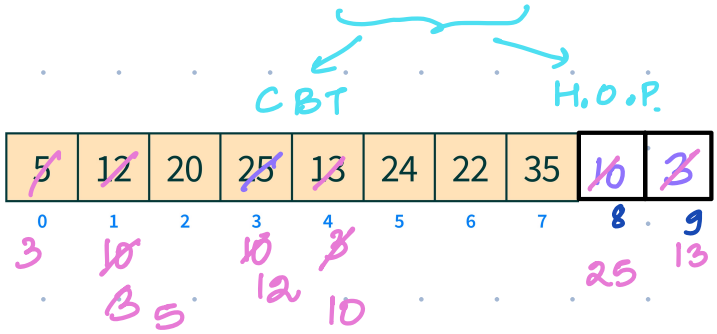
$1 \leftarrow 3$

$1 \leftarrow 4$

Min Heap

Insertion → we should Insert the new value in such a way that after INSERTION it still remains MIN HEAP.





→ Insert(10)

→ Insert(3)

MIN HEAP.

i	parent = (i-1)/2	A[P] ≤ A[i]
8 → 3 → 1	3 → 1 → 0	No → Swap No → Swap Yes → Stop
9 → 4 → 1 → 0	4 → 1 → 0	No → Swap No → Swap No → Swap Stop.

T.C : O(1) + O(log N) ⇒ O(log N)
S.C : O(1)

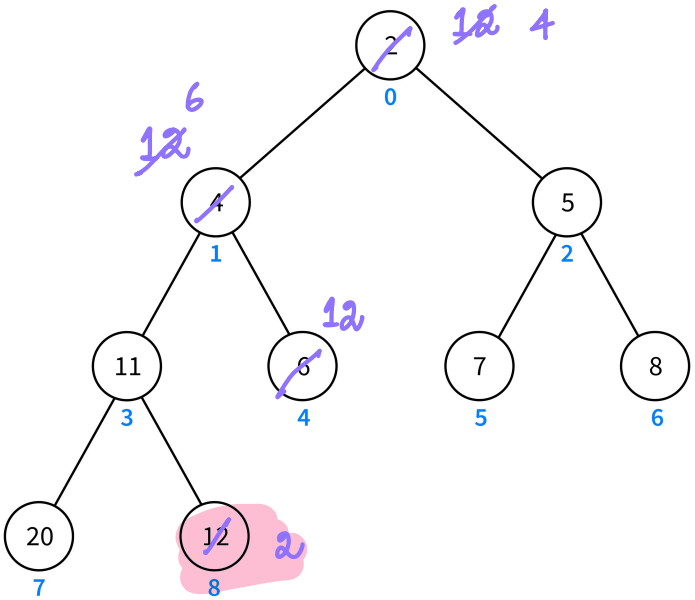
Approach : Insert at the last & then satisfy H.O.P from bottom → top.

```
void InsertInHeap( vector<int> heap, int K) {

    // Insert new value at the last.
    heap.push_back(K);
    i = heap.size() - 1;

    // Satisfy H.O.P.
    while (i > 0) {
        pi = (i - 1) / 2;
        if ( heap[pi] > heap[i]) {
            swap (heap[pi], heap[i]);
            i = pi;
        }
        else
            break;
    }
}
```

Extract-Min / Remove - Min → O(log N)



0	1	2	3	4	5	6	7	8
2	4	5	11	6	7	8	20	12

- Swap last index value with 0th index value
- Delete last index
- Satisfy H.O.P from Top → Bottom.
  (Swap with smallest child)

i	$c_1 = 2i+1$	$c_2 = 2i+2$	
0	1	2	Swap(12,4)
1	3	4	Swap(12,6)
4			Stop.

< / > Code

```
int ExtractMin ( vector<int> heap)
    n = heap.size();
    swap( heap[0], heap[n-1]);
    int ans = heap[n-1];
    heap.pop_back();
    heapify(heap,0);
    return ans;
}
```

T.C : O(log N)
S.C : O(1)

```
void heapify( vector<int> heap, int i) {
    int n = heap.size();

    while( 2i+1 < n ) {    // Until Node has atleast 1 child.
        x = min(heap[i], heap[2i+1], heap[2i+2]); // Handle the case when right child is not there
        if (x == heap[i])    return;
        else if (x == heap[2i+1]) {
            swap (heap[i], heap[2i+1]);
            i = 2i+1;
        }
        else {
            swap (heap[i], heap[2i+2]);
            i = 2i+2;
        }
    }
}
```

Built-in Heaps in different programming languages.

C++ :

```
priority_queue<int> maxHeap;
priority_queue<int, vector<int>, greater<int>> minHeap;
```

Oper^n :

- push() → Add Element
- pop() → Remove Min / Max element
- top() → Get Min / Max element

Default is MAXHEAP.

Java :

```
PriorityQueue<Integer> minHeap = new PriorityQueue<>();
PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
```

Oper^n :

- add() → Add Element
- poll() → Remove Min / Max element
- peek() → Get Min / Max element

Default is MINHEAP.

Js : doesn't have built in heap. Implement Manually.

Python : Use heapq module.

Default is MIN-HEAP.

For MAX-HEAP use (-ve) values.

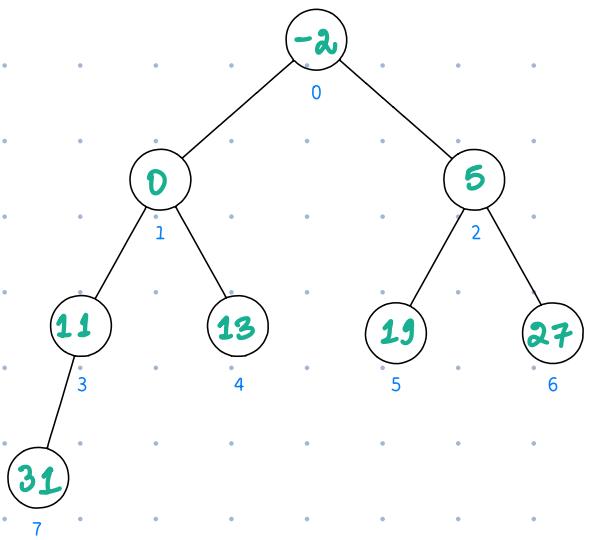
✪ Build a Heap [Interview Problem]

arr[ ] → [5 , 13 , -2 , 11 , 27 , 31 , 0 , 19]

[-2 0 5 11 13 19 27 31]

Idea -1

T.C : O(N log N)
S.C : O(1)



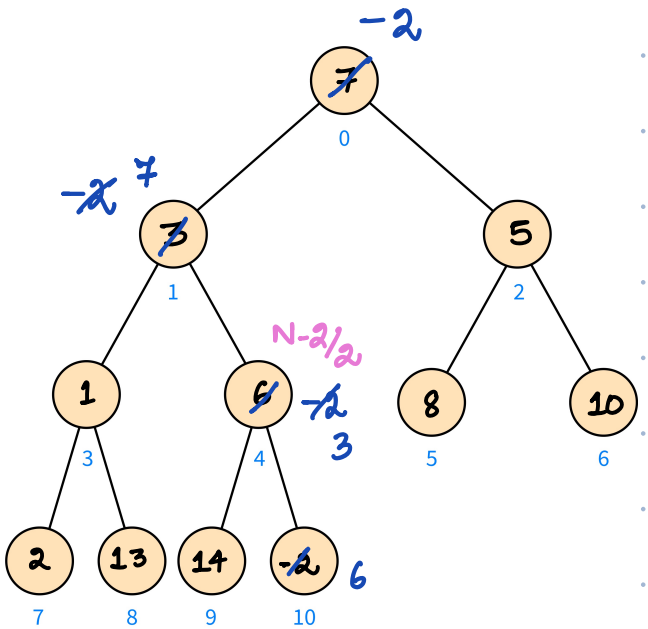
Idea -2

Pick an element & insert into heap.

T.C : O(N log N)
S.C : O(1)

Idea -3

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



N = 10

$\frac{i-1}{2} = \frac{10-1}{2} = \frac{9}{2} = 4$

Observation :

- H.O.P is automatically valid for leaf Nodes.
- If we satisfy the H.O.P for non-leaf Nodes also then we get MIN-HEAP.

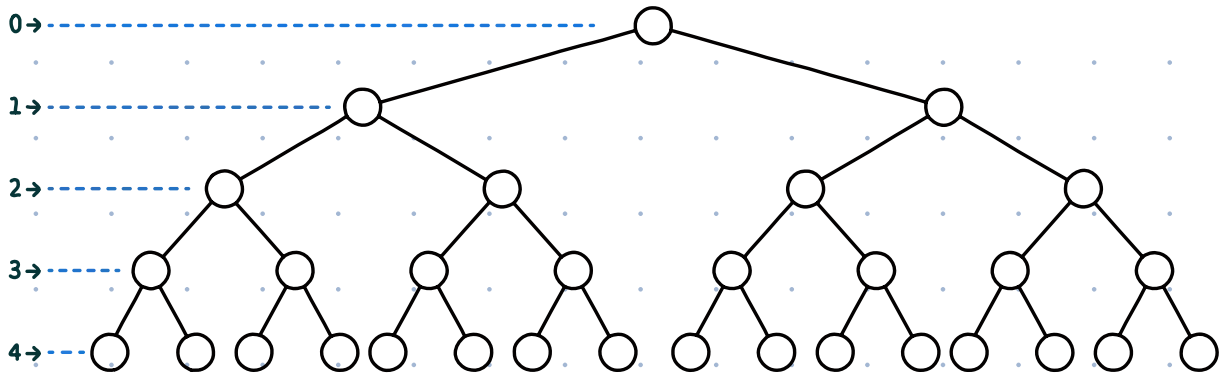
Idx of last non-leaf Node : Parent of last Node (N-1)

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

< / > Code

```
for (i = (N-2)/2 ; i >= 0 ; i--) {
    heapify(heap, i);
}
```

Time Complexity Analysis



Nodes	swaps.
1	Ht
N/16	3
N/8	2
N/4	1
N/2	0

$$\text{Total swaps} = \left[\frac{N}{2}*0 + \frac{N}{4}*1 + \frac{N}{8}*2 + \frac{N}{16}*3 \text{ ------ }\right]$$

$$= N\left[\frac{1}{4} + \frac{2}{8} + \frac{3}{16} + \text{ - - - - - -}\right]$$

$$= N/2\left[\underbrace{1/2 + 2/4 + 3/8 + \text{ - - - - - -}}_{S}\right] \quad \Rightarrow \quad N/2 \times 2 = N$$

T.C : O(N)

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

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

(1) − (2)

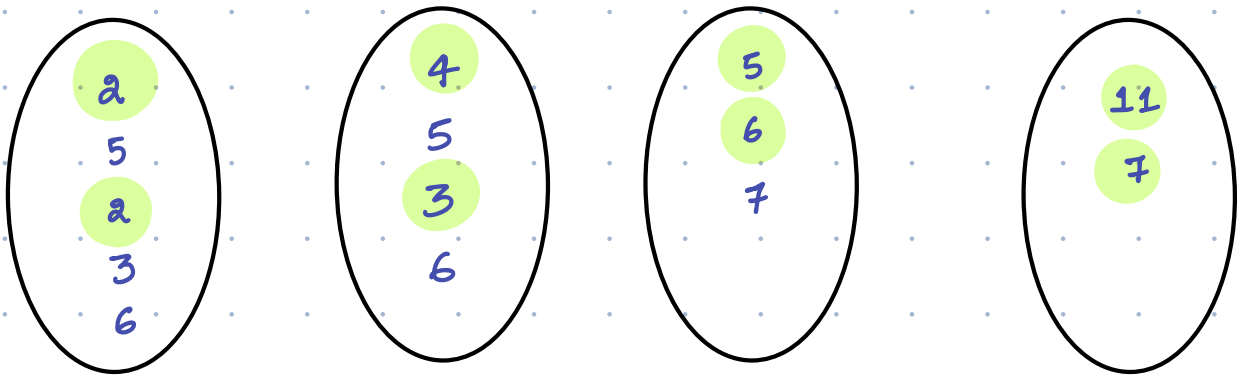
$$\frac{S}{2} = \underbrace{\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \text{ - - - - - -}}_{GP}$$

$$\text{Sum}_\infty = \frac{a}{1-r} = \frac{1/2}{1-1/2} \Rightarrow \frac{1/2}{1/2} = 1$$

$$\frac{S}{2} = 1 \quad \Rightarrow \quad S = 2$$

Connect Ropes Solution

arr[ ] → [2 , 5 , 3 , 2 , 6]



Remove 2 min elements : 2 log N

Insert 1 new element : log N

3 log N

T.C : O(log N)

Code :

```
int ConnectRopes(vector<int> A) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    // Insert all elements in heap
    for(i=0; i<n; i++)
        pq.add(A[i]);

    cost = 0;
    while(pq.size() > 1) {
        int v1 = pq.remove();
        int v2 = pq.remove();
        cost += (v1 + v2);
        pq.add(v1 + v2);
    }
    return cost;
}
```