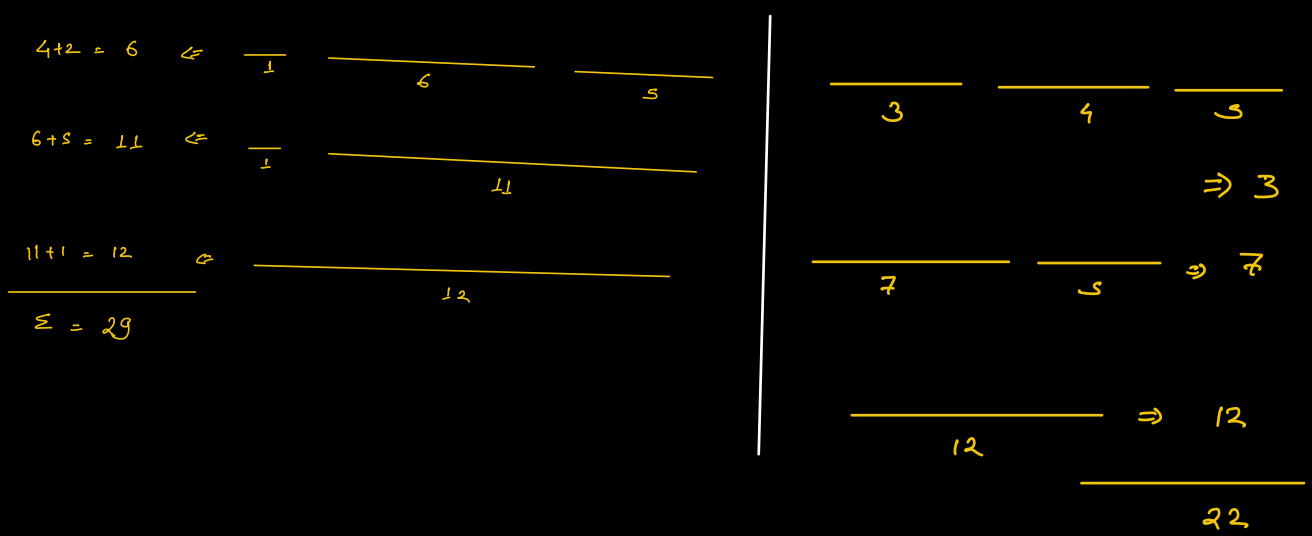
Amazon
MS
Paypal

Q. Given lengths of N ropes.
The cost of merging 2 ropes is equal to sum of the ropes getting merged.
Find the minimum cost required to merge all the ropes into a single ropes.


4, 3, 5, 8, 6, 9


merge (r1 & r2) → r1 + r2
merge (r1-r2 & r3) → r1+r2+r3
Σ = (r1+r2) + (r1+r2+r3)

merge (r1 & r3) → r1+r3
merge (r1-r3 & r2) → r1+r2+r3
Σ = (r1+r3) + (r1+r2+r3)

merge (r2, r3) → r2+r3
m (r2-r3 & r1) → r1+r2+r3
Σ = (r2+r3) + (r1+r2+r3)

Optimal strategy : Always pick two ropes of min possible size.

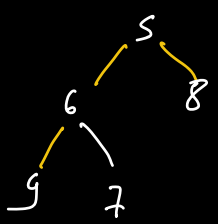
4, 3, 5, 8, 6, 9 —sort→ 3, 4, 5, 6, 8, 9

3, 4 → 7

5, 6, 7, 8, 9 → 11

7, 8, 9, 11 → 15

9, 11, 15




5, 6

TC : $O(N^2)$

$O(N) + (3 \log(N)) \times N$

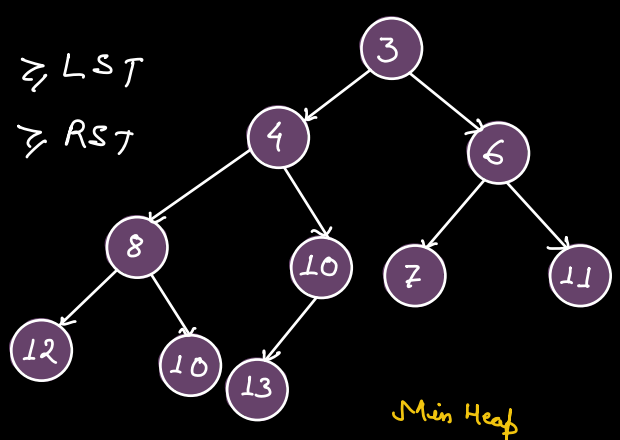
$\Rightarrow O(N) + N \log N$

## Heap

- Complete Binary Tree → All levels are completely filled except possibly the last level. Last level → left aligned. Ht → $\log N$
- Min or Max property
  - Min ↑ min Heap
  - Max ↑ max Heap

root.val ≤ LST
root.val ≤ RST

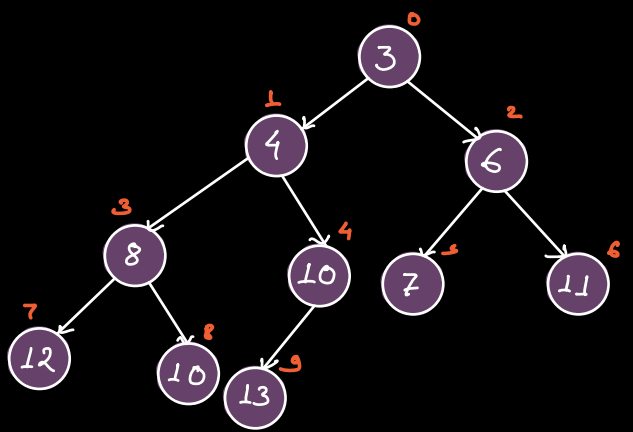
root.val ≥ LST
root.val ≥ RST



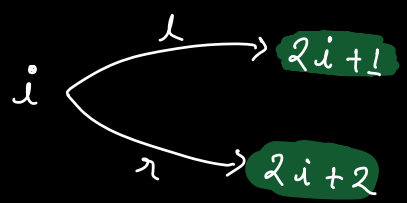

Min Heap

If we use arrays to store it:

1. No need of left & right pointer (saving space)
2. Move from child to parent.




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

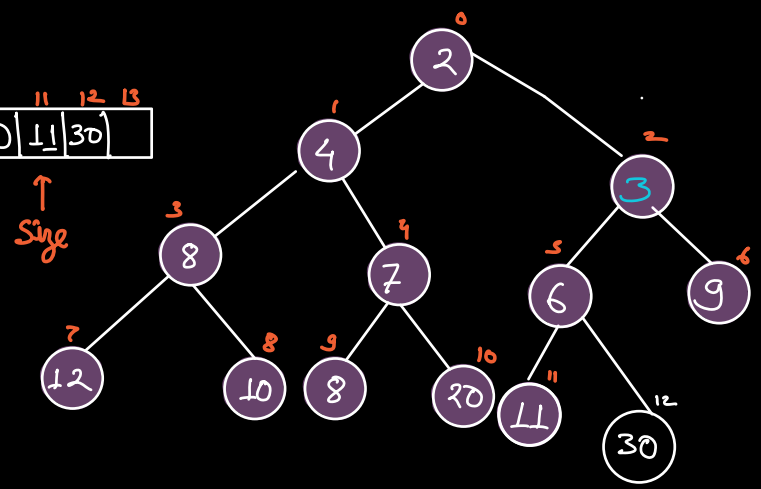



$j \xrightarrow{\text{Parent}(j)} \frac{j-1}{2}$

# Insert a val in min-Heap

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 3 | 8 | 7 | 6 | 9 | 12 | 10 | 8 | 20 | 11 | 30 | |

↑ Size




$i = 11;$

$Parent(i) \rightarrow P = \frac{11-1}{2} = 5$

$i = 5$

$P = \frac{5-1}{2} = 2$

$i = 2$

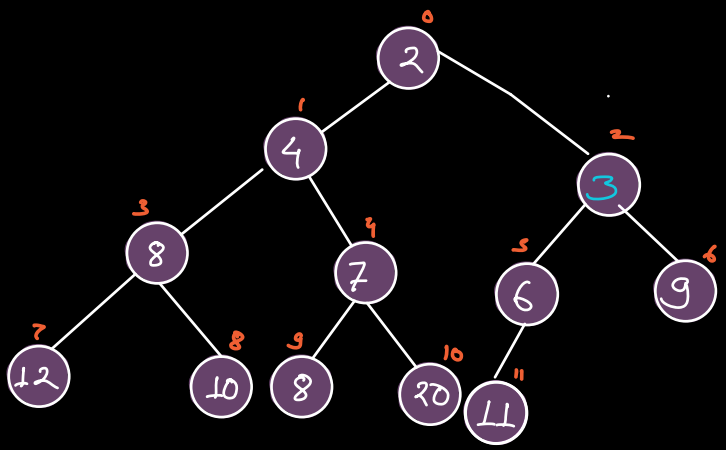
$P = \frac{2-1}{2} = 0$

TC: $O(\log N)$

| i | P | A[i] | A[P] |
|---|---|---|---|
| 12 | 5 | 30 | 6 |

Percolate up
or
shift-up

```
size++;
i = size;
A[i] = K;
while ( i > 0 ) {
    P = (i-1)/2;
    if (A[P] > A[i]) {
        swap(A[P], A[i]);
        i = P;
    }
    else {
        break;
    }
}
```



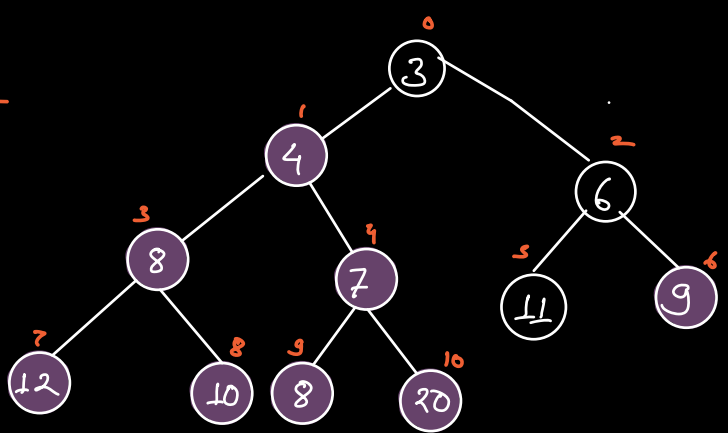

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 3 | 8 | 7 | 6 | 9 | 12 | 10 | 8 | 20 | 11 | 30 | |

↑ Size

## Deleting the last element

TC : O(1)

## Delete the min from min-Heap



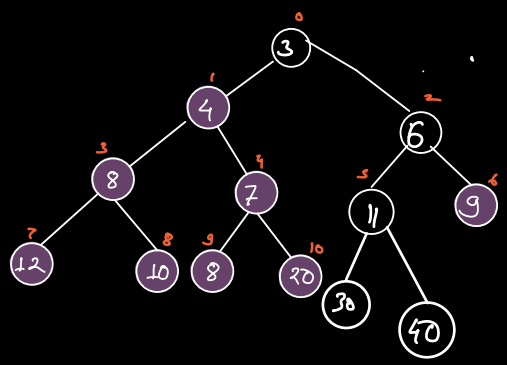

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3 | 4 | 6 | 8 | 7 | 11 | 9 | 12 | 10 | 8 | 20 | 2 | | |

↑ size

TC : O(logN)

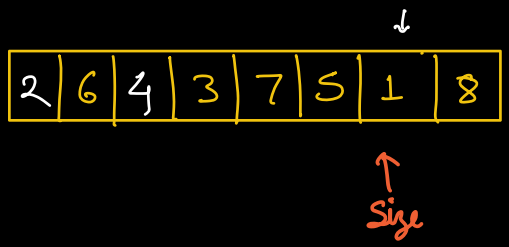
size

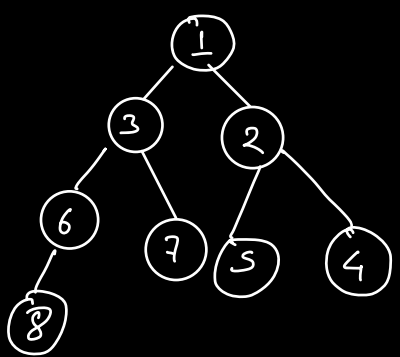
Percolate down
or
Shift-down




```
swap(size, 0);
size--;
i = 0;
while ( i < N ) {
    int min = i;
    int l = 2*i+1, r = 2*i+2;
    if( l<N && A[l] < A[min])
        min = l;
    if ( r<N && A[r] < A[min])
        min = r;
    if ( i == min)
        break;
    swap (A[i], A[min]);
    i = min;
}
```

Given an array. Convert it to a min heap.

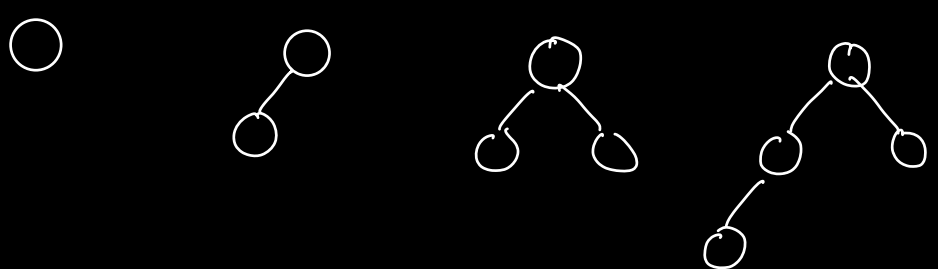


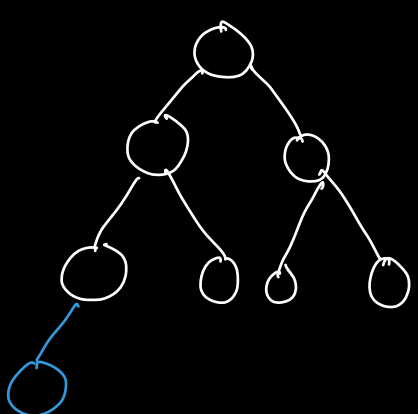



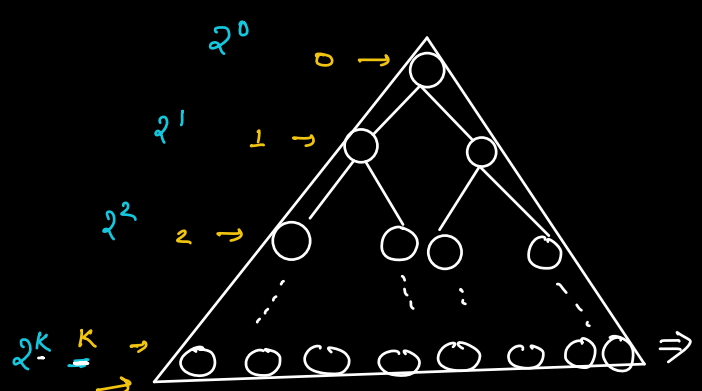

Iterate over the array

keep inserting the values in the heap.

$\Rightarrow O(N \log N)$



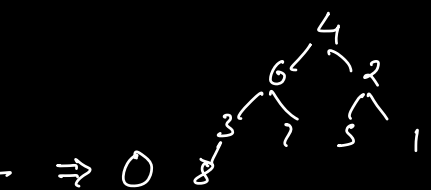


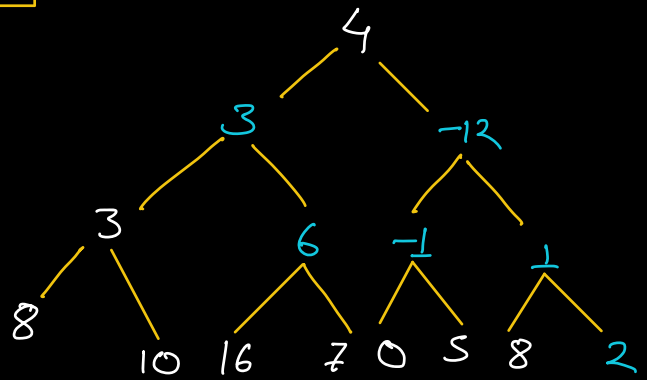



$$2^K = \frac{N+1}{2} \approx \frac{N}{2}$$

$2^0 + 2^1 + 2^2 + \cdots 2^K = N$

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 4 | 6 | 2 | 3 | 7 | 5 | 1 | 8 |






$\frac{N}{2} \Rightarrow 0$

$\frac{N}{4} \Rightarrow 1$

$\frac{N}{8} \Rightarrow 2$

$\frac{N}{16} \Rightarrow 3$

$\frac{N}{32} \Rightarrow 4$

$\vdots$

$1 \Rightarrow \log N$ (root)

Total no of swaps (iterations)

$$\frac{N}{2} \times 0 + \frac{N}{4} \times 1 + \frac{N}{8} \times 2 + \frac{N}{16} \times 3 + \dots + 1 \times \log_2 N$$

$$\frac{N}{2^1} \times 0 + \frac{N}{2^2} \times 1 + \frac{N}{2^3} \times 2 + \frac{N}{2^4} \times 3 + \dots + 1 \times \log_2 N$$

$$\left\{ \sum_{d=0}^{\log N} \frac{N}{2^{d+1}} \times d \right.$$

$$= N \left[ \frac{0}{2^1} + \frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + \dots \right] \to \infty$$

$$= \frac{N}{2} \left[ \frac{1}{2} + \frac{2}{2^2} + \frac{3}{2^3} + \frac{4}{2^4} + \dots \right] \to \infty$$

$= 2$

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

TC: O(N)

Java → Priority Queue

C++ → priority_queue

Python → heapq

JS → ? google