Heaps - 1

Question:

N ropes each has some length

Cost of connecting 2 ropes = sum of lengths of the ropes

Minimum cost to join all the ropes

N = 5

A = 2 5 2 6 3

6 3 → 6 3 → 6 3 → 3 → 18

15

Cost = 7 + 9 + 15 + 18

= 49

Strategy 1: Always combine the 2 smallest ropes

A = 2 5 2 6 3

→ 4 5 6 3

7 5 6 , 7 11

(x+y, z) ↔ (x+z, y)

Cost = 4 + 7 + 11 + 18

= 40

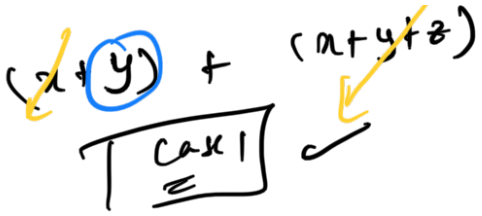
3 Ropes: x, y, z where $x < y < z$

Case 1: (x+y) + (x+y+z)

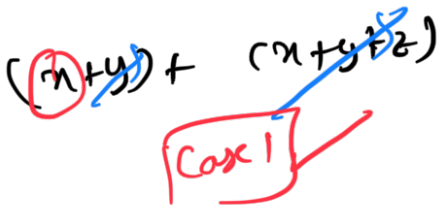
Case 2: (x+z) + (x+y+z)

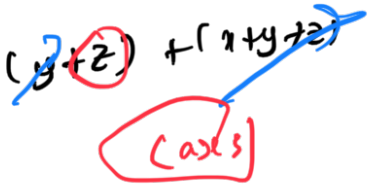
Case 3: (y+z) + (x+y+z)

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



case 2:





Implement:

1) Always join the 2 smallest ropes which are available

Cost = 4

A = 2 5 2 6 3

sort(A) = 2 2 3 5 6 $O(n \log n)$

(A[0], A[1]) $\Rightarrow O(1)$

(sorted)?

A = 4 3 5 6

Insert 4 in the correct place

sorted insert: $O(n)$

A = 3 4 5 6

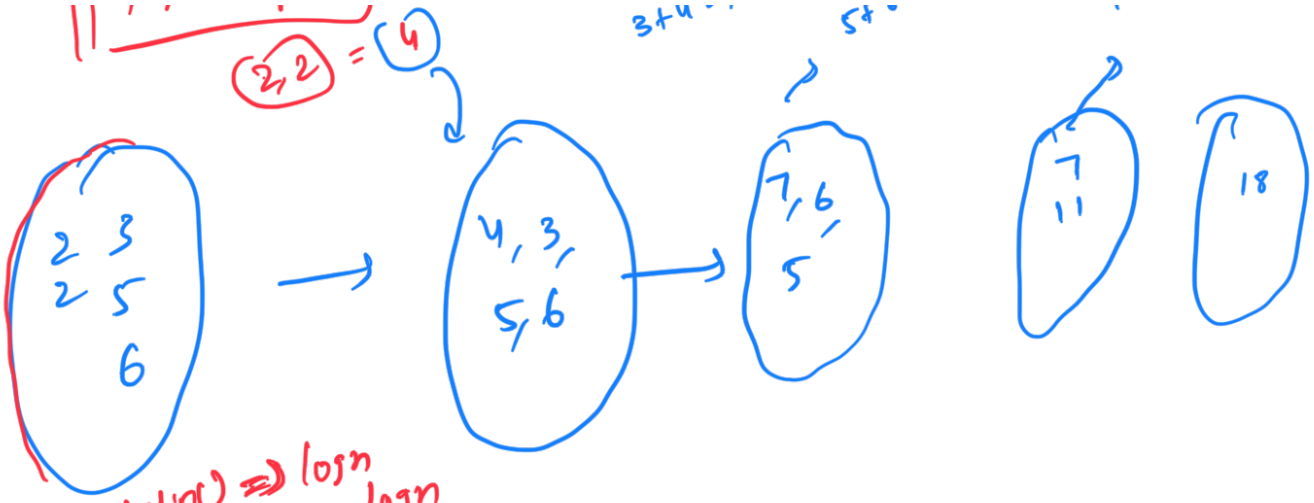
T.C: $O(1) + O(n) \Rightarrow O(n) \Rightarrow$ 1 step

(n-1) steps to join all ropes

T.C: $(n-1) \times O(n) \Rightarrow O(n^2) + O(n \log n)$

$\Rightarrow O(n^2)$

Can we do better ?? Heaps $\Rightarrow$ one operation/step in T.C: $O(\log n)$



2 extract Min() ⇒ logn
Insert Operation : logn
3logn

Cost = 4 + 7 + 11 + 14 = 40

1) Extract Min() ⇒ O(logn)
2) Insertion ⇒ O(logn)

Heap

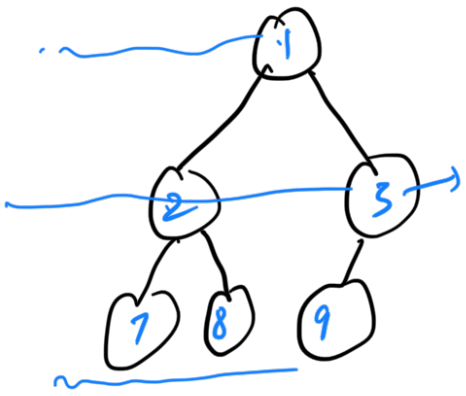
Its a specialized tree Data structe.

→ Complete Binary Tree

→ All the levels are filled except possibly the last level.

→ In last level, nodes are as left as possible

CBT



Height : O(logn)

n is no of nodes

Heaps (CBT)

Types of

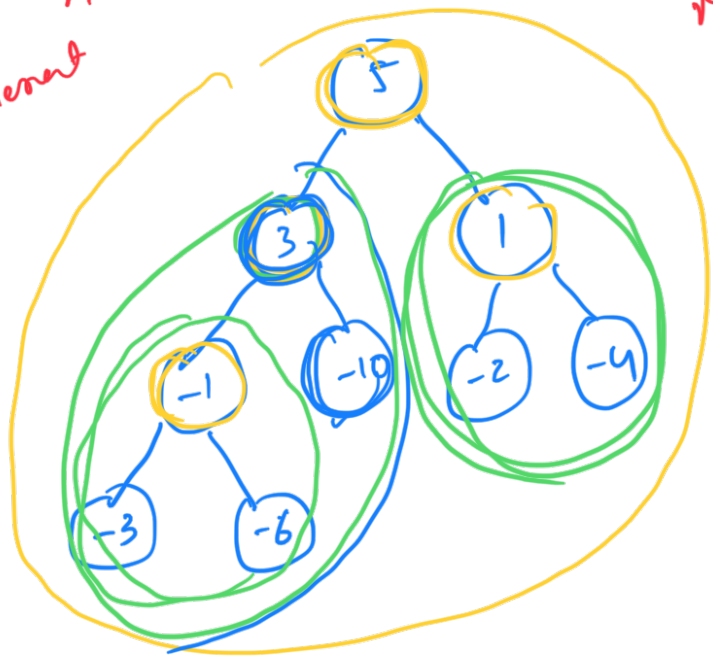
1) Max Heap :

the value present at the root node must be greatest among all the values present in its children. the same property must recursively true for all its childrens

∀ node, node.val ≥ node.left.value
node.val ≥ node.right.value

Root Node: Max Element

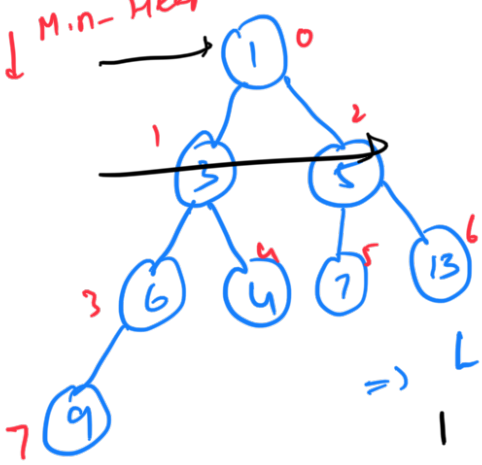
Max-Heap!



CBT ?

2) Min-Heap.

Root node has the min element.



Min heap

=> Level order traversal:

1 3 5 6 4 7 13 9

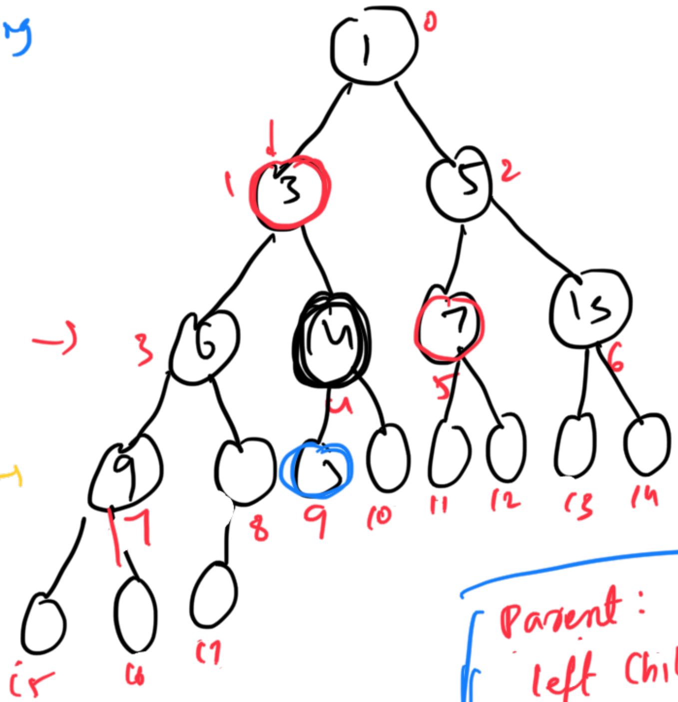
Representation:

heap =

1	3	5	6	4	7	13	9
0	1	2	3	4	5	6	7

0-indexing

Level-order



log(i+1)

i = 5 = 2

if condn

ind(3) = 1

i i (2i+1, 2i+2)

4 : 9, 10

6 : 13, 14

2 : 5 6

Parent : i
left Child = 2i+1
Right child = 2i+2

Exactly 1 parent

i ((9-1)/2)

par(9) = 4   (i-1)/2

par(13) = 6 → ((13-1)/2)

par(16) = 7 → ((16-1)/2) = 15/2 = 7

For node 'i' : 2i+1, 2i+2 are children

For node 'i' : (i-1)/2 is the parent (Percolate-Up) (Bottom-Up)

Operations

1) Insert 3

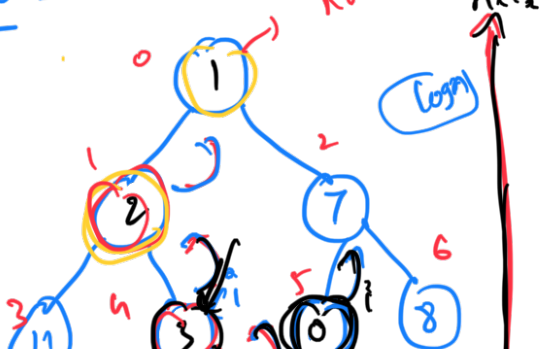
2) Insert 1

... 100

Root

height

log(n)



Min-Heap

1) CBT

2) Heap property

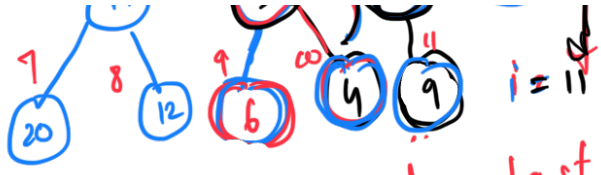
n log n

T.C :

3) Insert

i = 11

i = (i-1)/2



1) Inserting node at last ⇒ O(1)

2) Trace logn level

T.C: O(logn)

Min-Heap

Best Case: O(1)

```
void insert ( vector<int> heap , int ele){
    heap.push_back(ele);
    ind = heap.size() - 1;          // n-1
    while ( i >= 0 && heap[(i-1)/2] > heap[i] ) {
        temp = heap[i];
        heap[i] = heap[(i-1)/2];
        heap[(i-1)/2] = temp;
        i = (i-1)/2;
    }
}
```

if (heap[(ind-1)/2] > heap[i])

heap[-1]

Insertion: O(logn)

ExtractMin()

Delete element and returns the maximum

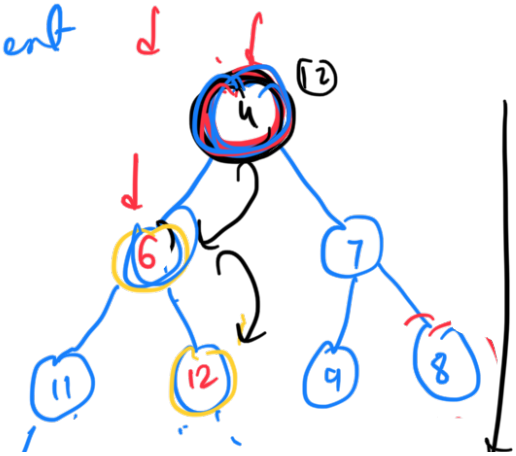
O(1)

min = heap[0]

parent < child

height = log n

Percolate down



1) CBT
2) heap property

min-heap

TOP

20

1) Swap heap[0] & heap[n-1] => O(1)
2) heap.pop_back(); O(1)
3) Keep swapping with min child

T.C: O(logn)

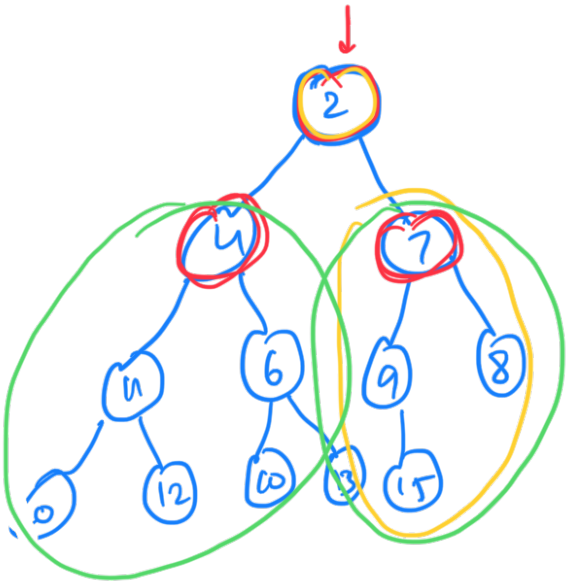
```
int extractMin ( vector<int> heap) {
    if( heap.size() == 0) {
        return INT_MAX;
    }
    if( heap.size() == 1) {
        min_ele = heap[0];
        heap.pop_back();
        return min_ele;
    }
    min = heap[0];
    swap ( &heap[0], &heap[n-1]);
    heap.pop_back();
    minHeapify(0);
}

// Percolate Down Function
int minHeapify (int i) {
    l = 2i+1;
    r = 2i+2;
    int minm = i
    if ( l < n && heap[l] < heap[minm])
        minm = l;
```

```
if ( r < n && heap[r] < heap[minm] )
    minm = r;
if ( minm != i ) {
    swap ( &heap[i], &heap[minm] );
    heapify ( minm );
}
}
```

O(log(n))

search (10)



2) O(n)

B.S.T

O(logn)

STL

Empty Heap

```
priority_queue <int> pq;
```

Max-heap

Language

1) push()
2) pop()
3) size()
4) top()

pq.push (3)
pq.push (1)

Size

Delete Max Element

Return Max Ele

Min-Heap

Min-Heap

(priority_queue <int, vector<int>, greater<int>>)

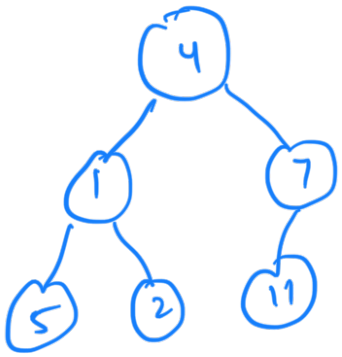
(comparator)

```
struct Node {
    int key;
    int value;
};
```

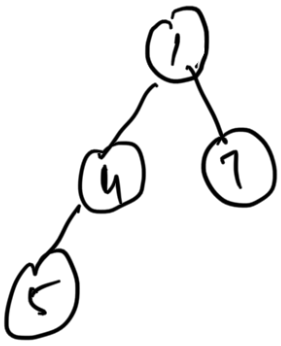
Question: Build a heap

A = 4 1 7 5 2 11

Min-heap



Approach 1:



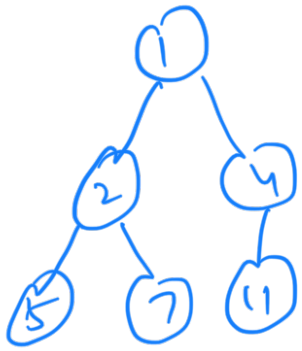
N insertion operation

T.C: O(nlogn)

Approach 2:

A = 4 1 7 5 2 11

Sort(A)z 1 2 4 5 7 11



T.C: O(nlogn)

T.C: O(n)

Better Approach: Build from the leaf level

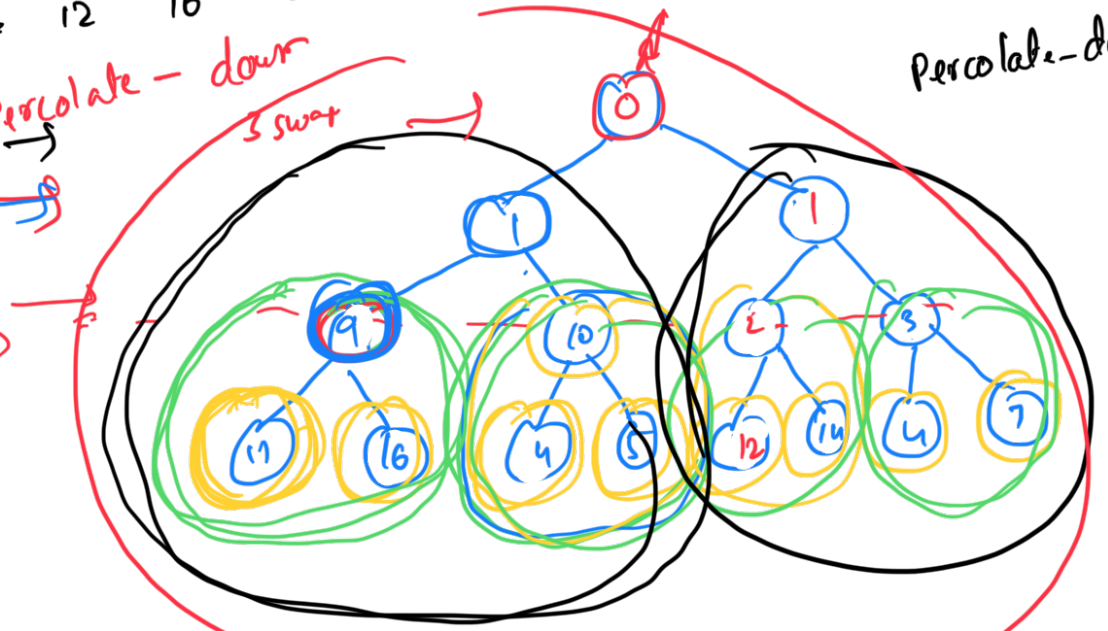
A= 12 16 14 9 10 2 7 11 1 4 5 0 1 4 3

h=3

Percolate-down

2 swaps

3 swaps



Percolate_down

1

2^1

2^2

2^3

minHeapify()

for(i= n-1; i ≥ 0; i--) {

MinHeapify(i);

(Iterate)?

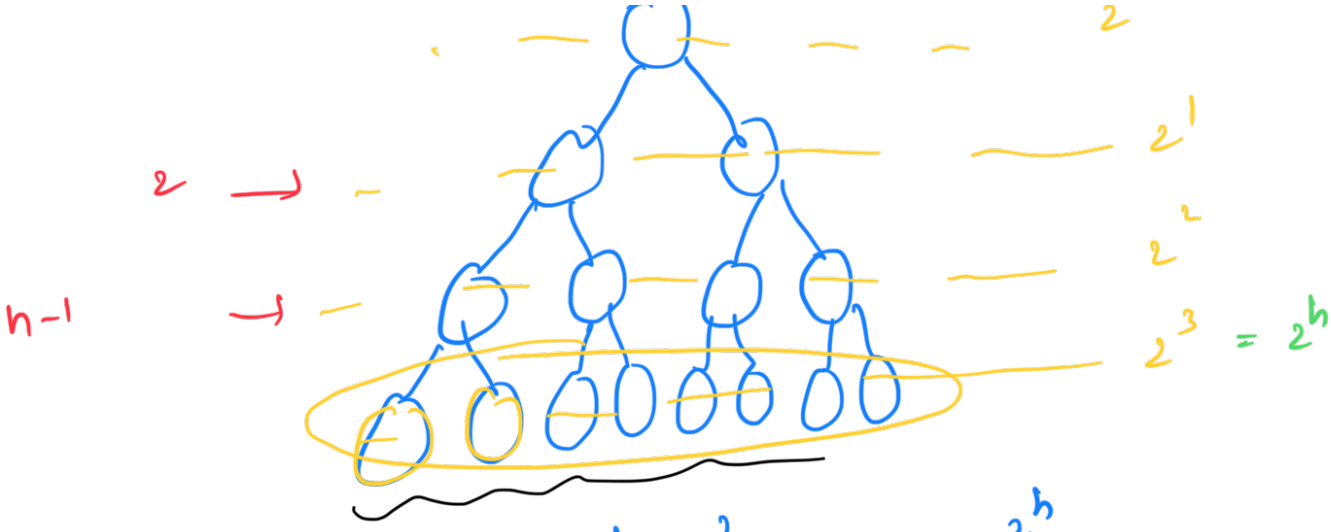
}

n times

T.C

T.C: O(nlogn)

h= 3



$N = 2^0 + 2^1 + 2^2 + \cdots 2^h$ (G.P)

$\frac{a(r^n - 1)}{r-1}$

$N = \frac{1(2^{h+1} - 1)}{2-1} = 2^{h+1} - 1$

$N = 2^{h+1} - 1$ — (1)

$2^{h+1} = N+1$

operations $= 2^h \times 0 + 2^{h-1} \times 1 + 2^{h-2} \times 2 + 2^{h-3} \times 3$

$= \sum_{i=0}^{h} 2^{h-i} \times i = 2^h \sum_{i=0}^{h} \frac{i}{2^i}$

$\frac{0}{1} + \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots = 2$

$\sum_{i=0}^{h} \frac{i}{2^i} = 2$

operations $= 2^h \times 2 = 2^{h+1}$ — (2)

of nodes $= N+1$

T.C.: $O(N)$

$N = 2^{h+1} - 1$ — (1)

operations = $2^{...}$ = [illegible]

T.C of building heap = O(N)

S.C:

S.C: Extra space required to convert input to output