Heaps - 1

Question:

→ N ropes each with a length

cost of connecting 2 ropes ⇒ sum of their length

R1 ⇒ 5
R2 ⇒ 7

Cost = 5 + 7 = 12

→ min cost to join all the Ropes

N = 5

A = 2 5 2 6 [18 3]

Cost = 4 + 9 + 15 + 18 ⇒ 46

App1: Sort the array & Join

A = 12 6

Cost = 4 + 7 + 12 + 18 = 41

App2: Always join the 2 smallest Ropes

A = 2 5 2 6 3

Cost = 4 + 7 + 11 + 18 = 40

(N ropes)

Proof - 3 Ropes

R1 R2 R3

x < y < z

s1 ≥ s2

Strategy 1: ((R1, R2), R3)

c1 = (x+y) + (x+y+z) → z

Strategy 2: ((R2, R3), R1)

c2 = (y+z) + (x+y+z) → z

Strategy 3: ((R1, R3), R2)

c3 = (x+z) + (x+y+z)

y z

A = 2 5 2 6 3

Implement:

1) Sort the array

A = [3 4 5 6 _]

ele1=2 A[0], A[1]

ele2 = 2.

Insert(ele1 + ele2)

cost = 4 + ...

T.C for 1 operation : O(n)

logn

Arrays

A = 1 3 7 8 10

6

O(n)

1) Fetch the smallest elmt: O(1) → O(n)

2) Fetch the 2nd smallest : O(1) → O(n) A[0] A[1]

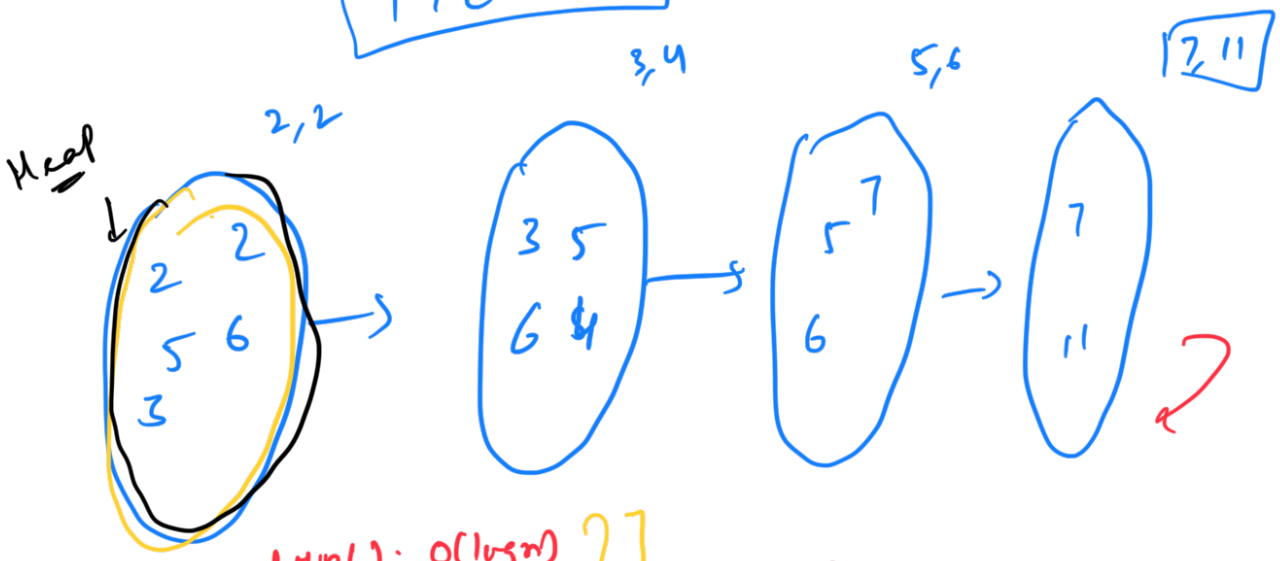
3) Insert new elmt in sorted Array: O(n)

1 operation : O(n)

Total T.C : $O(n^2)$ + $O(n \log n)$ R_1 R_2 R_3 R_4 R_5 (n-1) opn.

→ If we have a DS, which does all these operations in $O(\log n)$?

HEAPS



extractMin() : O(log n)
extractMin() : O(log n)
Insert : O(log n)

T.C :

1 Operation : O(log n)

Total T.C : N log n

N = 6

Operations : 1 + 1 + 1 + 1 + 1 + 1 + 1

21

(N-1) operations

Heaps

Heap is a specialized tree-based DS

Structure : Complete Binary Tree

Height of a CBT : O(log n)

Types:

Max-Heap: The key present at the root node must be the greatest element among all the keys present in the children. this is recursively true of the Left & Right subtree.

Complete B-T

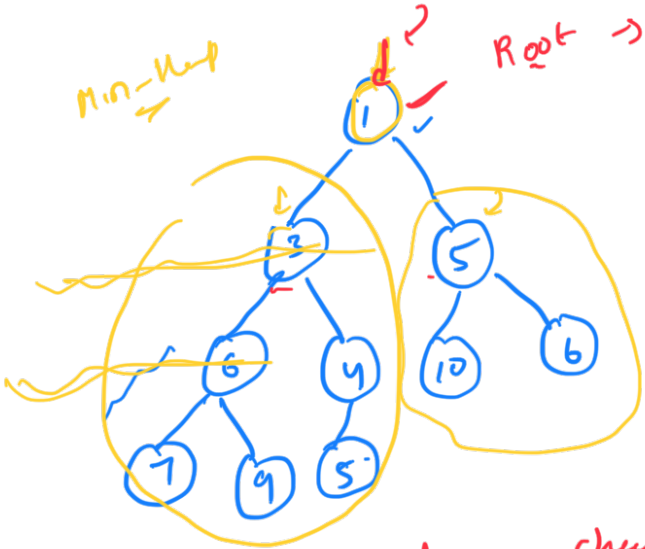
For all the nodes of Heap;

node.data > node.left.data
> node.right.data

Min Heap: Root is the smallest element.

LST & RST are themselves min Heaps.

node.data < node.left.data
node.right.data



2nd Min

>4

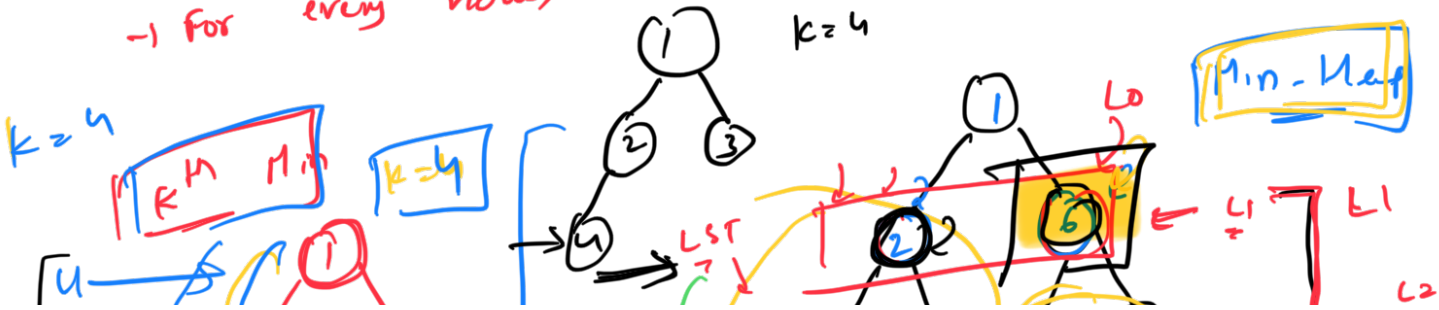
[-INF, INF]

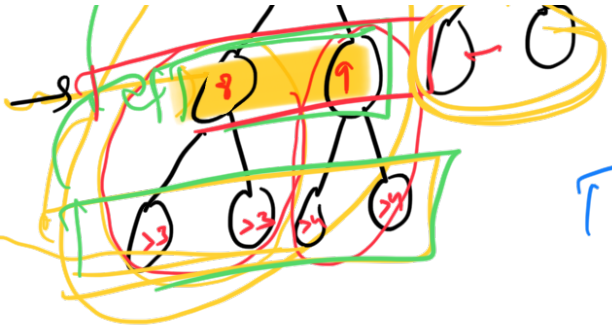
Min

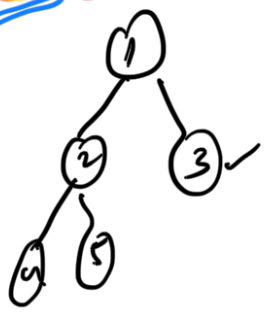
Min

min(root.left, root.right) & 2nd smallest = O(1)

→ For every node, check if parent < children



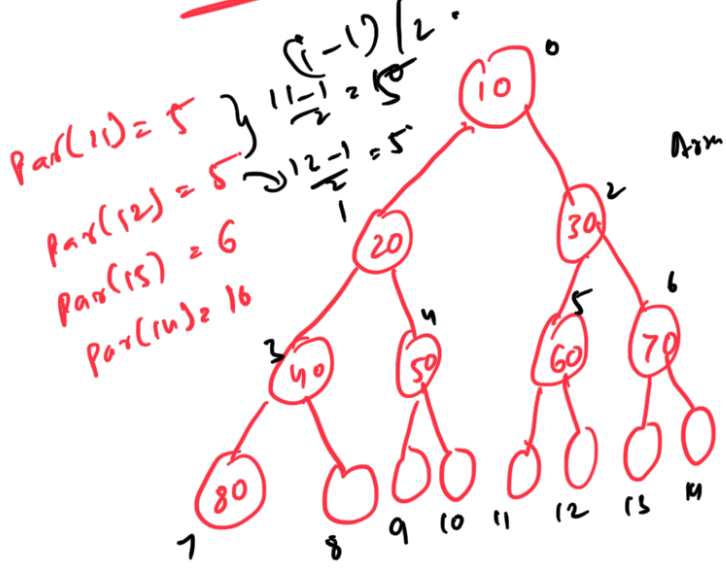




node.left < node.right

3rd Min : min (node.right,
node.left.left,
node.left.right)

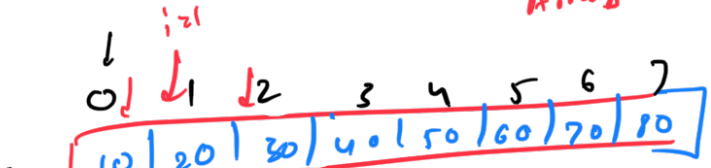
Representation:



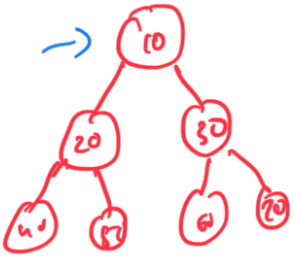
par(11) = 5
par(12) = 5
par(15) = 6
par(16) = 16

Heap => Binary Tree

2*0+1, 2*0+2



Arr

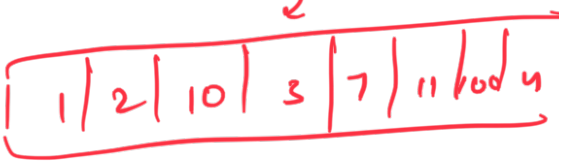


parent index

0 => 1, 2
1 => 3, 4
2 => 5, 6
3 => 7, 8
4 => 9, 10
5 => 11, 12
6 => 13, 14

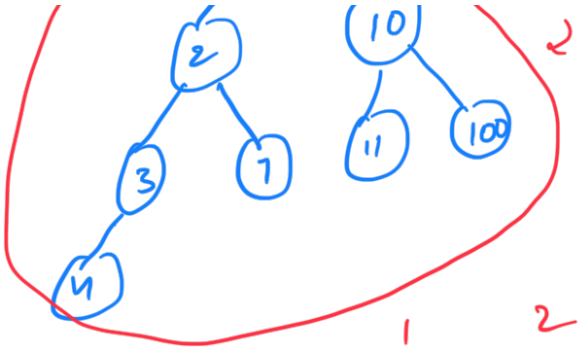
2nd(50) = 2
children(30) = 5, 6

i => parent
2i+1, 2i+2 => children



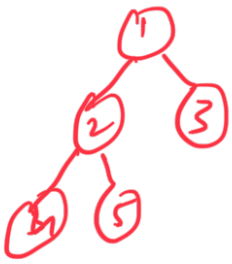
child index = i
i-1

parent = $\frac{i}{2}$



Min-Heap

1 2 3 4 5 6



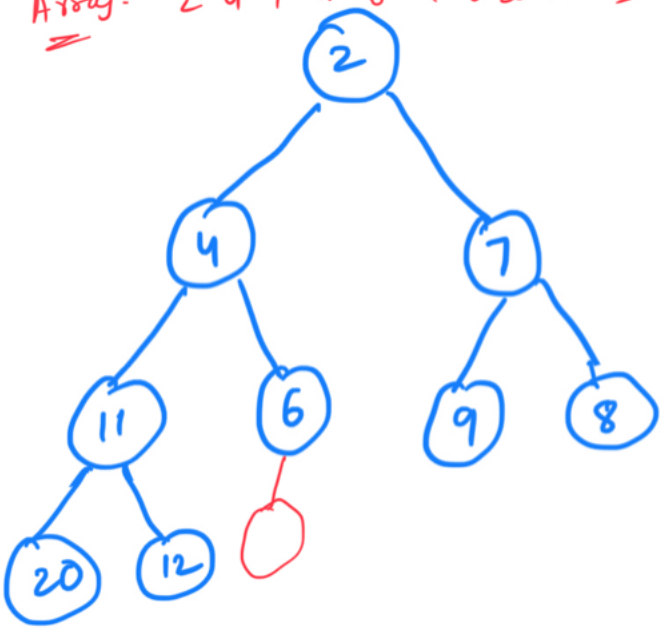
Operation:

Insertion:

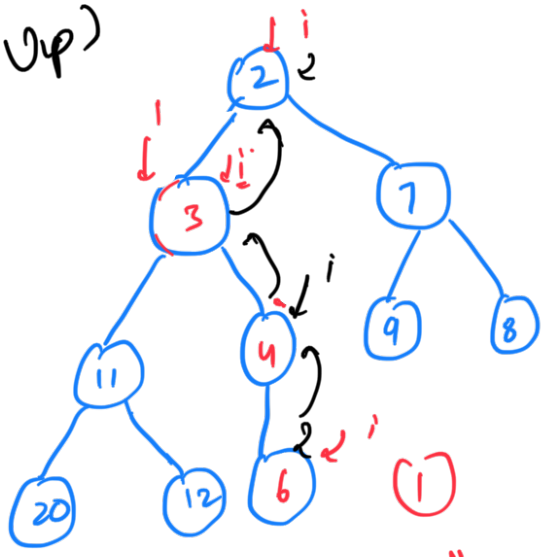
Insert(3)

Array: 2 4 7 11 6 9 8 20 12 3

Complete BT ✓



[Percolate Up]



i > 0

Insert 3

(i − 1) / 2

1) Insert Element at the end of the array

2) Keep swapping while parent > children

```
void insert ( vector<int> heap, int ele ) {
    heap.push_back(ele);
    i = heap.size()-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;
    }
}
```

heap = [ ]

heap = [2]

i = 0

Percolate Up

T.C: O(log n)

i > 1

i/2

1) Min ele in Min Heap: Root O(1)

2) Max ele in Min Heap: Max ele ⇒ Leaf

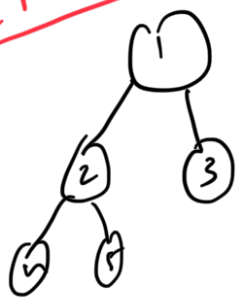
N nodes

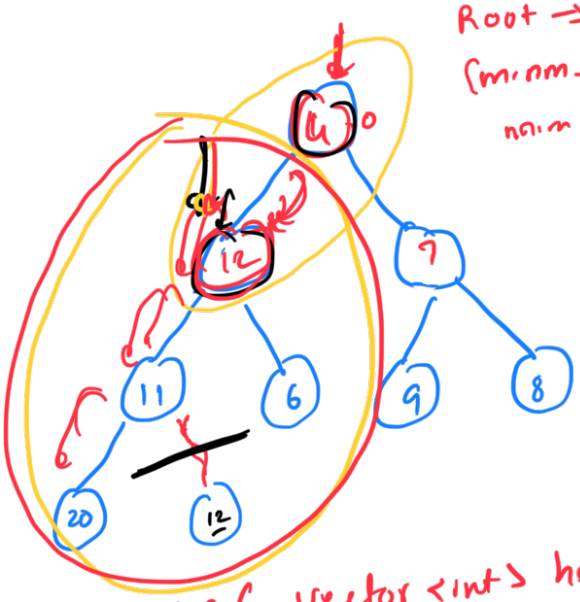
N/2 Leaf

3) Search element: O(n)

Deletion: (ExtractMin())

[Percolate Down]

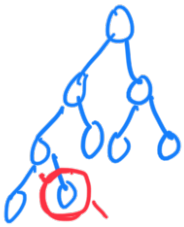




Root → leafs

(minm_ind) = 0

minm = i



Worst Case: O(log n)

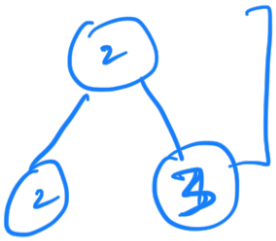
(minm == x)

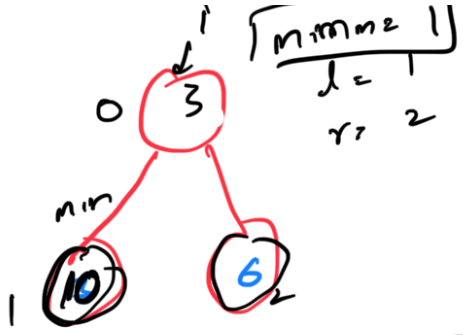
```
int extractMin ( vector<int> heap) {
    int min_ele = heap[0];
    heap[0] = heap[heap.size()-1];
    heap.pop_back();
    percolateDown(0);
    return min_ele;
}

void percolateDown (heap, i) {
    l = 2i+1;
    r = 2i+2;
    minm = i
    if(l<n && heap[l] < heap[minm]) {
        minm = l;
    }
    if(r<n && heap[r] < heap[minm]) {
        minm = r
    }
    if(minm != i) {
        temp = heap[i];
        heap[i] = heap[minm];
        heap[minm] = temp;
        percolateDown(minm);
    }
}
```

minm = i, l, r

T.C: O(log n)

heap[2] < heap[1]



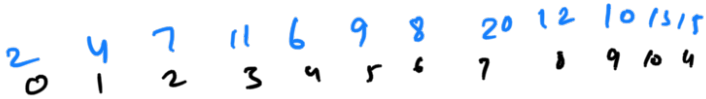


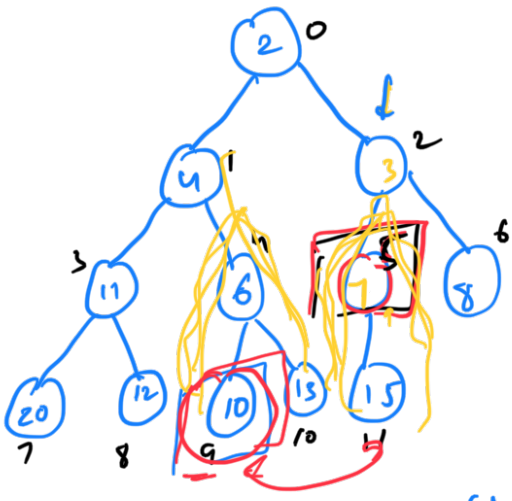
heap[1] < heap[0]

Graphs => Prim's

Min-Heap

Decrease key:





Parent < child

Percolate Down

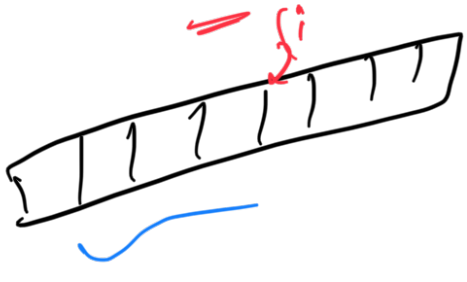
decreaseKey(5, 3)

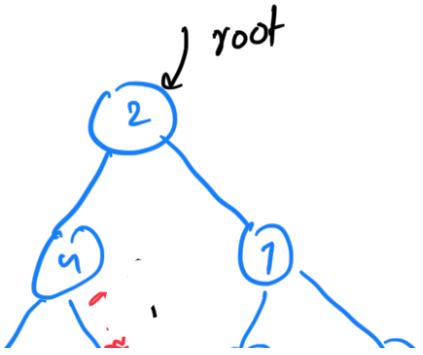
T.C: O(log n)

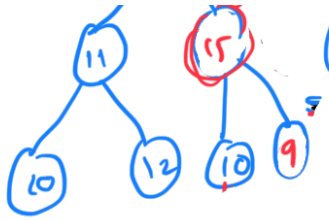
Delete a key:

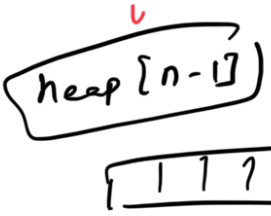
deleteKey(9) → ind

=> { 1) decreaseKey(ind, -INF)
2) Delete the root









swap(heap[i], heap[n-1])

heap.pop_back()

C++: priority_queue()

- top() → Min/Max
- push => Insert
- pop => Extract the Min

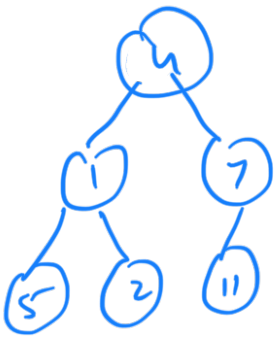
Question: Build a heap

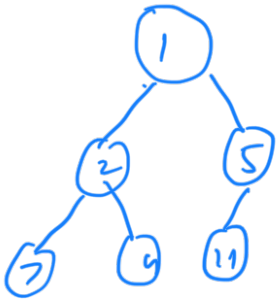
Given an array, Convert this to a MinHeap

A = [4, 1, 7, 5, 2, 11]

A' = 1 2 5 7 4 11 Not a MinHeap

→ O(n)





Approach1: Sort the array

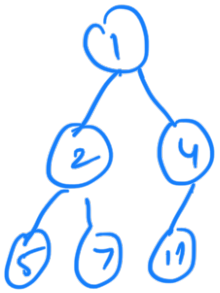
A = 4 1 7 5 2 11

1 4 5 7 11

Sort(A): 1 2 ...

T.C: O(nlogn)

S.C: O(1)

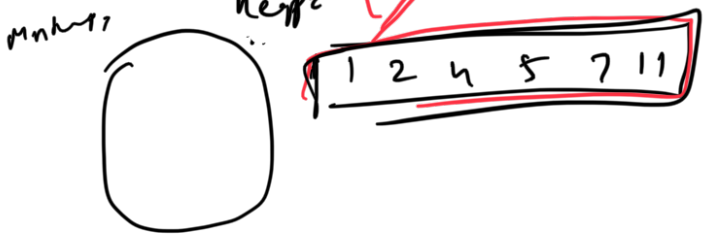


Approach2 : A- 1 2 4 5 7 11

Input

4 1 7 5 2 11

A =



N => Insert operation

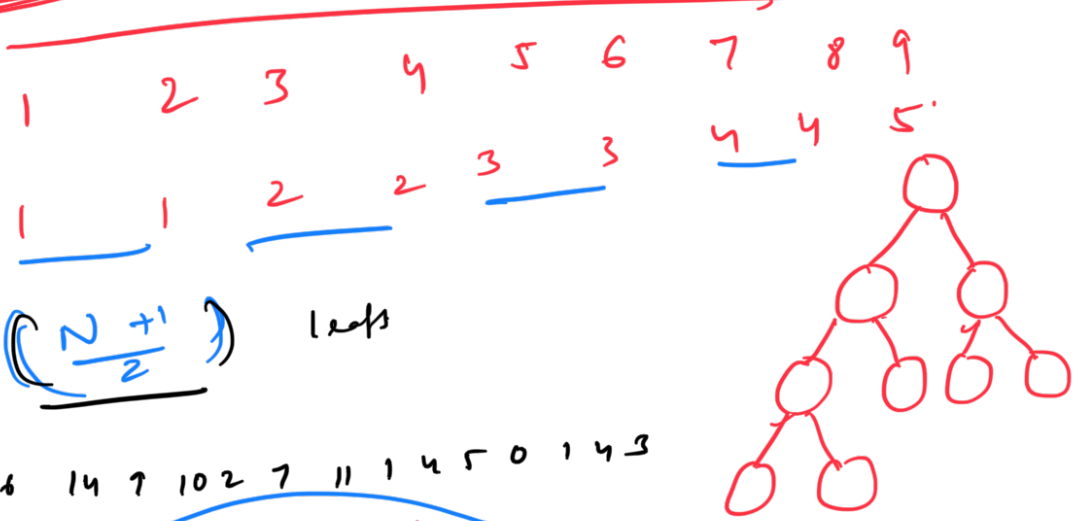
T.C: O(nlogn)

S.C: O(n)

Min-Heap

Approach3:

Perfect B.T of nodes of N nodes $\leq \frac{N}{2}$.



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

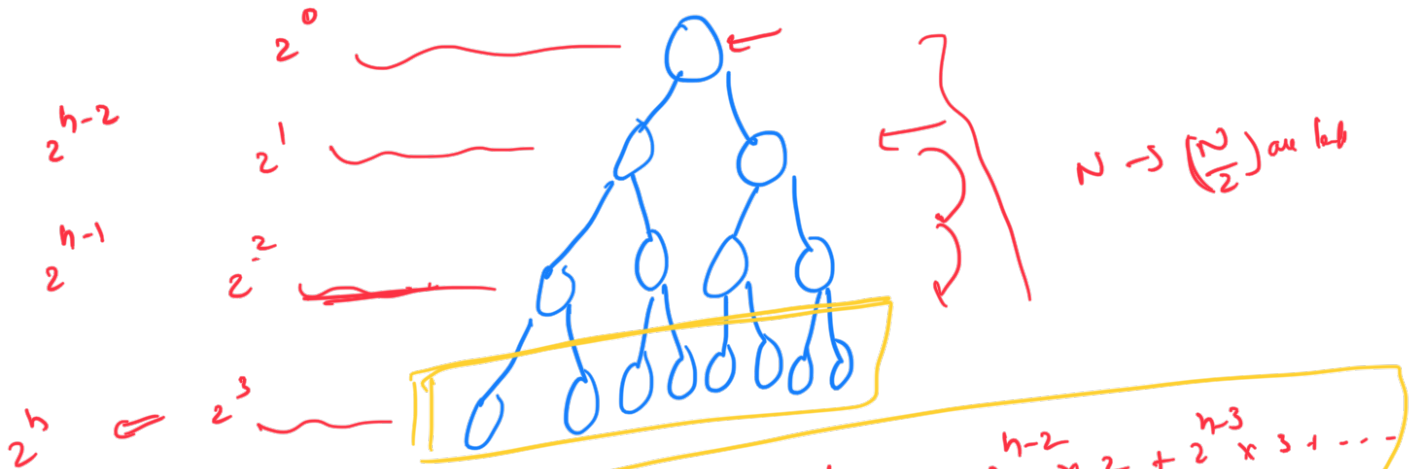
Bottom-up Ap

Percolate Dow

Percolate

```
void buildHeap(arr){
    for(i = n/2; i >= 0; i--){
        percolateDown(i);
    }
}
```

h = 3



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

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

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

$$\#\text{operations} = k \cdot 2^h$$

$$= k \cdot \frac{(N+1)}{2} = O(N)$$

S.C: O(1)

N in terms of h

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

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

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

Construct a B B-S-T : $O(n \log n)$

Heap : $\Theta(n)$

Percolate Up : Iteration [Recursion]

Recursion [Iteration]

Percolate Up :