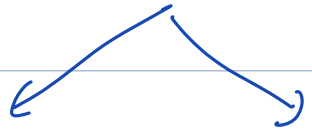
Heap → data structure



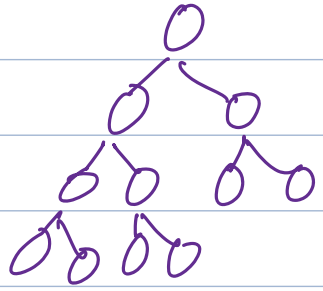
Max heap	Min heap	
• insert()	• insert()	O(log(n))
• deleteMax()	• deleteMin()	
• getMax()	• getMin()	O(1)

Heap is a binary tree

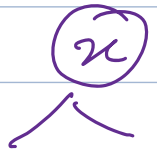
Complete binary tree

All levels are filled except the last one.

L-R



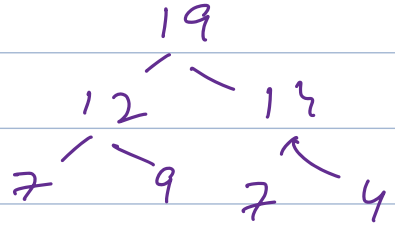
order of elem



for all nodes, both children < par

Min heap

Max heap (root is max)

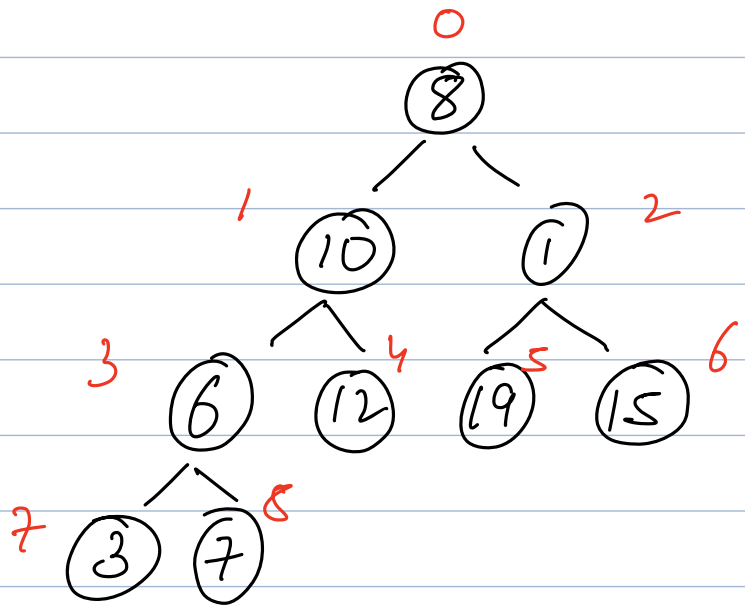


How to use heap ?

C++ ⟹ priority_queue — push()
by default, max heap — pop()
top()

Java ⟹ PriorityQueue — add()
by default min heap — poll()
peek()

Can we implement ourselves ?
Yes ! Using arrays

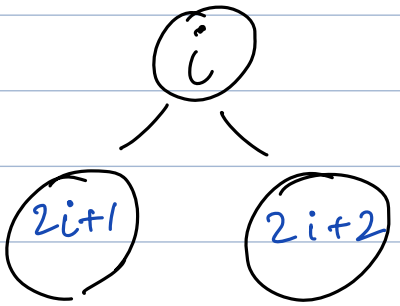


0	1	2	3	4	5	6	7	8
8	10	1	6	12	19	15	3	7

0 → 1 : 2×0 + 1
0 → 2 : 2×0 + 2

1 → 3 : 2×1 + 1
1 → 4 : 2×1 + 2

3 → 7 : 2×3 + 1
3 → 8 : 2×3 + 2



Left child = $2i+1$

Right child = $2i+2$

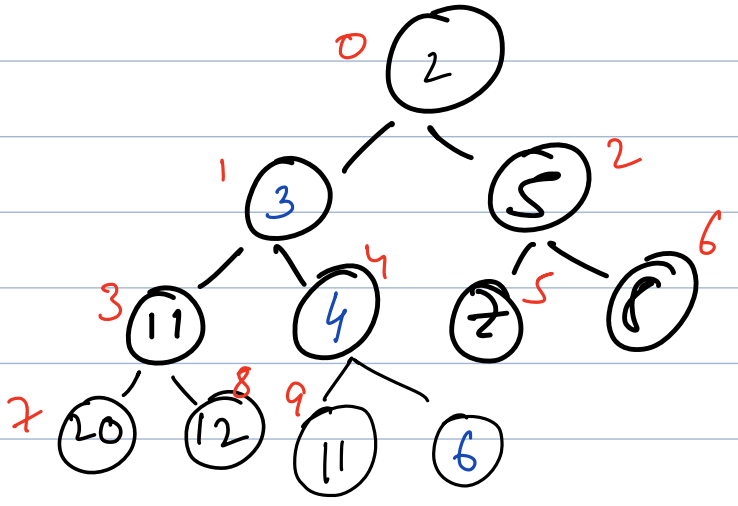
Parent = $\left(\frac{i-1}{2}\right)$

$7 \rightarrow 3$

$3 \rightarrow 1$

- Add into heap

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



TC: O(height)

In this case

O(log n)

Code

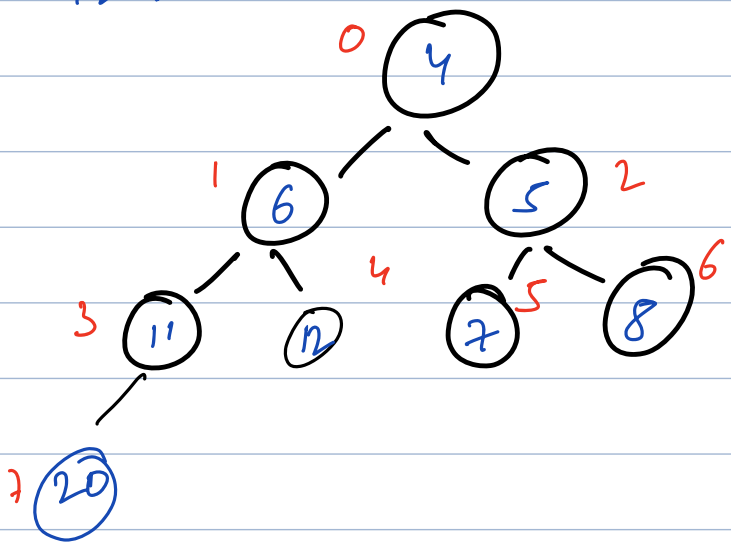
```
void add( int x, int[] arr) {
    arr.append(x)
    idx = arr.size() - 1
    while (idx != 0) {
        par = (idx-1)/2
        if( arr[par] > arr[idx]) {
            ;
```

```
        swap(arr[par], arr[idx])
        idx = par
    }
}
else
    break
}
}
```

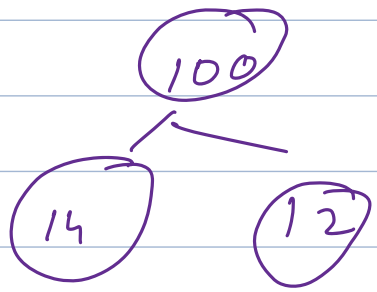
TC: $O(\log(n))$

- Delete

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



delete 2



Obs: Min is at root
We need smallest at the root.
Which child would you replace with?
smaller child

Code

```
void heapify(int idx, int ar[]){
    while (idx < n) {
        lc = 2*idx + 1
        rc = 2*idx + 2
        if ( ar[idx] < ar[lc] && ar[idx] < ar[rc])
                break
        else if ( ar[lc] < ar[rc] ) {
                swap(ar[idx], ar[lc])
                idx = lc
        }
        else {
                swap(ar[idx], ar[rc])
                idx = rc
        }
    }
}
```

TC: $\log n$

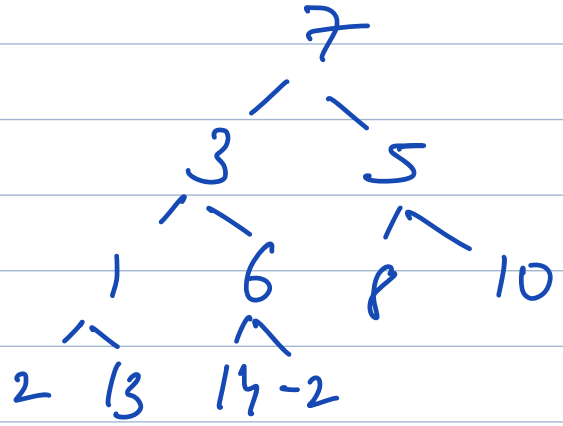
```
void deleteMin() {
    swap( ar[0], ar[n-1] )
    n--
    idx = 0
    heapify(idx)
}
```

- GetMin ⇒ return ar[0]

TC: $O(1)$

Build a heap from array

0	1	2	3	4	5	6	7	8	9	10
7	3	5	1	6	8	10	2	13	14	-2

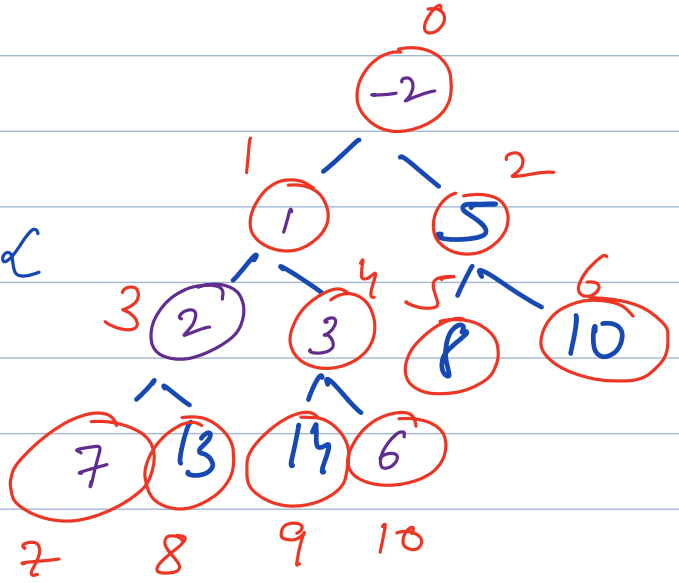


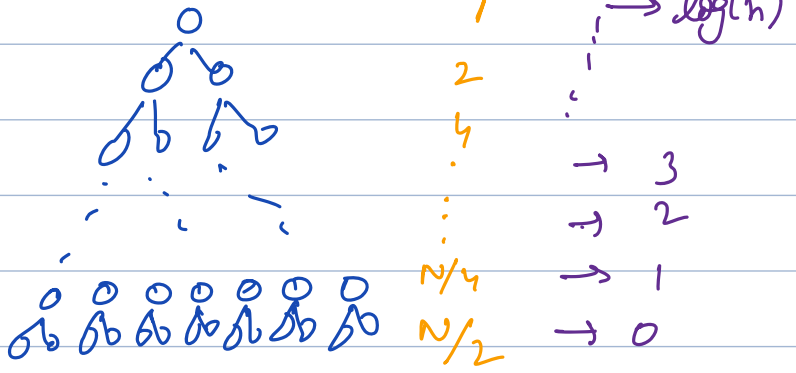
Base idea: call insert() function for all values

TC: $n \log n$

Better way:

```
for( i = n/2 - 1 ; i >= 0 ; i--){
    heapify(i, arr)
}
```





$$\frac{n-1-1}{2} = \frac{n-2}{2} = \frac{n}{2} - 1$$

$n-1$

total swaps = $0 \times N/2 + 1 \times N/4 + 2 \times N/8 + 3 \times N/16 + \dots$

$$= \frac{N}{2}\left[\underbrace{\frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \frac{4}{16} + \frac{5}{32} + \dots}_{=2}\right]$$

$$= N$$

Q2 Given multiple sorted linked lists. Merge into one sorted LL.

Eg

1 → 10 → 20
4 → 11 → 13
3 → 8 → 9 → 25

ans = 1 → 3 → 4 → 8 → 9 → 10 → 11 → 13 → 20 → 25

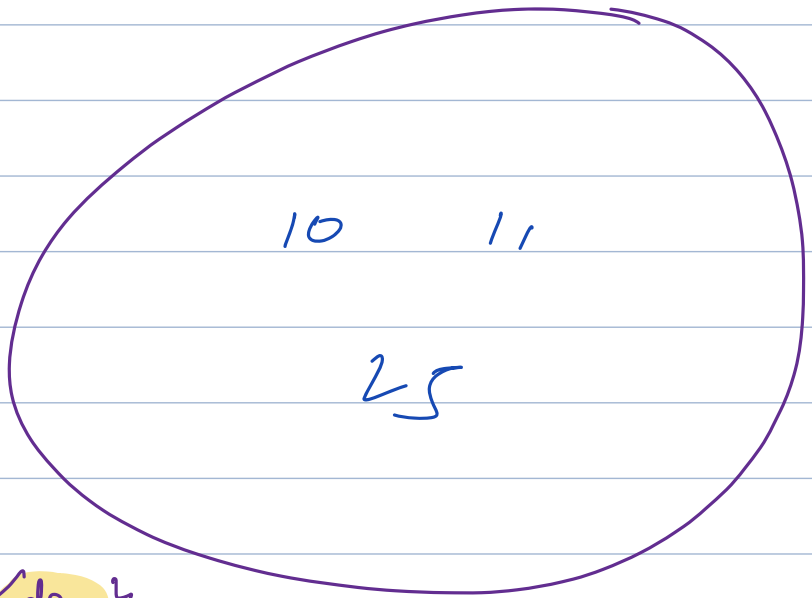
Idea: Does this type of merging sound familiar ?

⇒ Merge sort

1 → 10 → 20

4 → 11 → 13

3 → 8 → 9 → 25



1 → 3 → 4

Code

```
Node merge ( vector <Node> heads ) {
                                  value   pointer
                                    ↑       ↑
    priority_queue < pair <int, Node>> pq
                    // min heap

    for ( i=0 ; i< heads.size , i++) {
        pq.add ({ heads[i].data, heads[i] })
    }
    Node ans = cur
    while ( ! pq.empty() ) {
        pair <int, Node> p = pq.top()
        pq.pop()
        cur -> next = p.second
        cur = cur.next
        if ( p.second -> next != null ) {
            pq.add ( p.second->next.data ,
                     p.second->next )
        }
    }
    return ans.
}
```

TC = N log k

N = total nodes
k = no of lists.

Connect ropes

7 6 energy = 13

A = { 1 5 3 }

5 10 13, 9

(22)

energy = 9
energy = 13
energy = 22
44

energy = 5
energy = 10
15

10

1 2 3 4 6

3

energ = 3
energ = 6
energy = 10
19

13

6 7 9