Lecture-2

Data Structure

- structure to store your data.
- This helps in performing the operations/algorithm on data easily.

Binary Heap → Heap (Priority Queues)

Set of operations
- add elements
- remove minimum element.

1) Array

```
insert(x):
    a[n] = x
    n++
```
O(1)

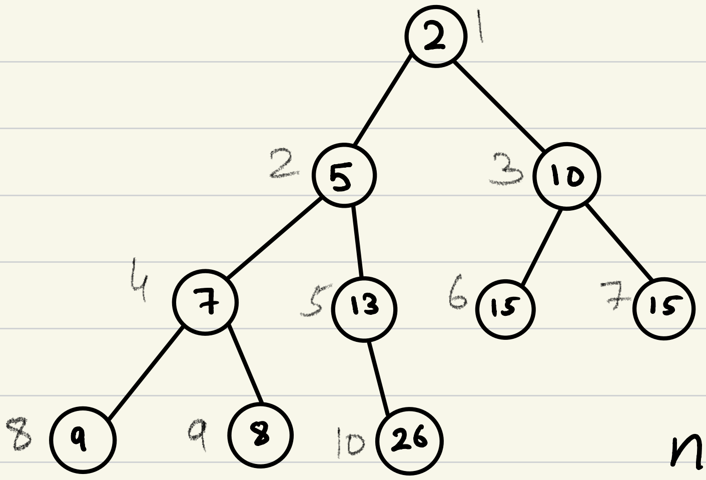
```
remove_min():
    j=0
    for i= 1...n-1:
        if a[i] < a[j]:
            j=i
    swap (a(j), a(n-1))
    n--
    return a[n]
```
O(n)

2) Sorted Array:

```
insert (x):
    a(n) = x
    n++
    i= n-1
    while i>0 and a(i) > a(i-1)
        swap (a(i), a(i-1))
        i--
```
O(n)

```
remove_min():
    n--
    return a(n)
```
O(1)

3. Binary Heap

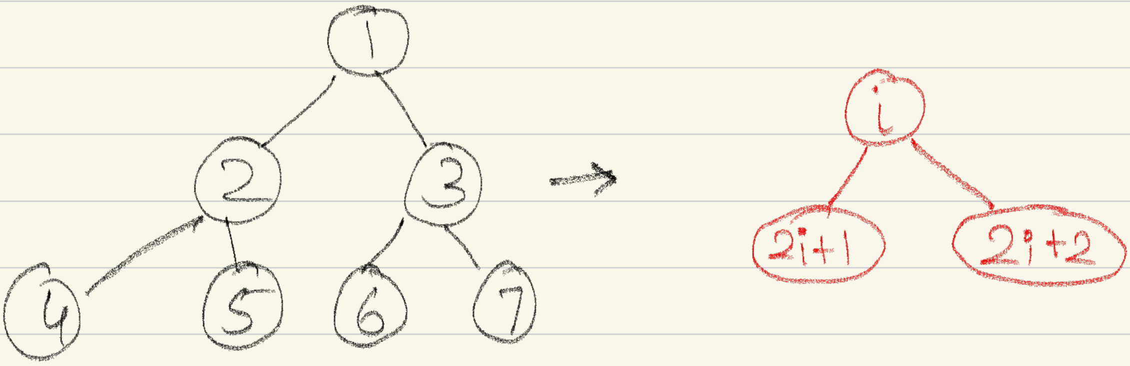


Greater Than or equal to

$n = 10$

Array →

2	5	10	7	13	15	15	9	8	26
1	2	3	4	5	6	7	8	9	10



Parent to left child → $2i+1$

Parent to Right child → $2i+2$

Child to Parent → $\left\lfloor \frac{i-1}{2} \right\rfloor$ $\qquad \frac{4-1}{2} = 1.5 \approx 1$

$\lfloor\ \rfloor$ → Lower value $\qquad \frac{3-1}{2} = 1$

$\lceil\ \rceil$ → Upper value

```
insert(x):
    a(n) = x
    n++
    i = n-1
    while i > 0 and a[i] < a[(i-1)/2]:
        swap(a[i], a[(i-1)/2])
        i = (i-1)/2
```

shift up

log n

the max height of Binary heap is log n

```
remove_min():
    swap(a(0), a(n-1))
    n--
    i = 0
while (2i+1<n) & (a(i) > a(2i+1) or a(i) > a(2i+2)):
        if a(2i+1) < a(2i+2)
            swap(a(i), a(2i+1))
        else
            swap(a(i), a(2i+2))
```

child exist

optional because the node will settle at the bottom.

log n

max height is log n

```
while 2i+1 < n:
    j = 2i+1
    if (2i+2 < n) and (a(2i+2) < a(j))
        j = 2i+2
    if (h(j) ≥ h(i))
        break
    swap(h(i), h(j))
    i = j
```

shift down

insertion(x) → log n

remove_min() → log n

	insert	remove_min
Array	1	n
Sorted Array	n	1
Binary heap	log n	log n

n-insertion & n-removal

→ $n + n^2$

→ $n^2 + n$

→ $n \log n + n \log n$

Algorithm

Heap Sort

```
Sort (array):
    for i=0...n-1
        insert (a[i])      → log n
    for i=0...n-1
        remove_min()       → log n
```

n log n

n log n

$O(n \log n)$

Instead of using 2 arrays
- Input array
- Binary Heap

we can decide the initial of input array as binary heap and then we can increase the size of binary heap by one