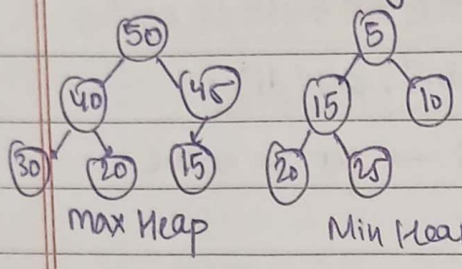
Page No.
Date 06-07-25

Heaps

* A complete binary tree that satisfies Heap properties

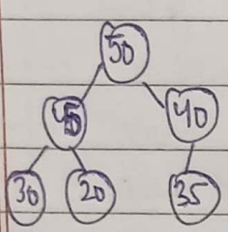
* value of children < Parent → (MAX HEAP)

ek ka samajh lo Min Heap just opposite.



Max Heap Min Heap

* n nodes → ∴ Height = log(n)



1	2	3	4	5	6
50	45	40	30	20	35

Node → i

Parent(i) = $\left(\frac{i}{2}\right)$ floor value

left child = $2 \times i$

right child = $2 \times i + 1$

for 1-indexed array

eg. i = 2 → node 45

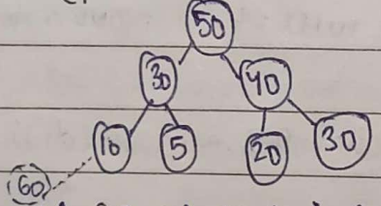
parent → $\frac{i}{2} = 1$ → 50 ✓

left child → $2 \times i = 4$ → 30 ✓

right child → $2(i+1) = 5$ → 20 ✓

* Insert (operation in such a way ki wo heap bana rahe)

O(log n)



→ A:

1	2	3	4	5	6	7	8
50	30	40	10	5	20	30	60

Insert: 60 →

1. Insert 60 at last index of array.
2. Check with its parent, if parent < 60 (value) ∴ swap

parent = $\frac{8}{2} = 4$ → 10 ⇒

1	2	3	4	5	6	7	8
50	30	40	60	5	20	30	10

Ully,

again with $\left(\frac{4}{2}\right) = 2$ → 30 → swap ⇒

1	2	3	4	5	6	7	8
50	60	40	30	5	20	30	10

again, →

60	50	40	30	5	20	30	10

Ajanta
Page No.
Date

A.length

* Easy pseudocode:

We cannot do this directly ← as arrays are immutable

∴ Method: 1 → newArr

Method: 2 → ArrayList

check code. (whatsapp).

```
void insert(A[], n, val){
    n = n+1;        → size of array ↑ when inserted
    A[n] = val;     → put val at last index
    int i = n;      → starting from last
    while(i>0){     → ∵ 1 se 0 index pe aana hua to
        int parent = Math.floor(i/2);
        if(A[parent] < A[i]){
            swap(A, parent, i);
            i = parent;   → Now i ab, earlier parent ka pos pe aa gya
        }
        else{
            return;       → To remove faltu steps till i>1 when parent is larger
        }
    }
```

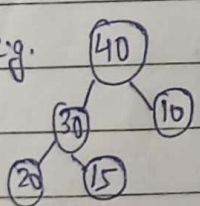
* **Delete** $O(\log n)$ → ∵ height me traversing orderly

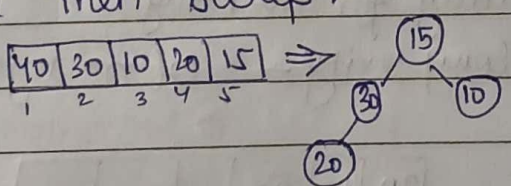
Here, in Max heap root node is the highest item.
And, " Min " " " " " lowest " .
So, when we say delete → we're deleting root node only.
(and adjust heap) → to stay complete binary tree + follow properties.

Steps:-

1. Remove root node ~~with~~ and put node with last index at there.
2. Check left and right child of new root node, if any one of them, or both (then select which one is larger) is greater than node then swap.

e.g.

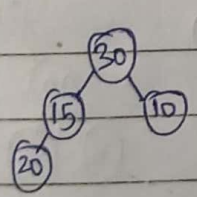




15	30	10	20
1	2	3	4

left child = 2×i = 2×1 = 2 → 30 ✓
right child = 2i+1 = 3 → 10 ×
∴ swap with left

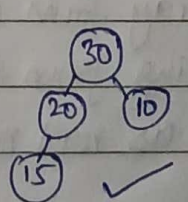
⇒



30	15	10	20
1	2	3	4

left → 2×2 = 4 → 20 ✓ → swap
right → 2×2+1 = 5 ×

⇒



30	20	10	15

left → ∄, out of bound
right → ∄, out of bound
∴ break.

pseudocode → (easy)

```
void delete(A[], n) {
    A[1] = A[n] ;  → last ko root pe rakho
    n = n-1 ;  → remove last by reducing size of array
    i = 1 ;  → start from that root node
    while (i<n) {
    int left = A[2*i];
    int right = A[2*i+1];
    int larger;
    if (left > right) {
        larger = 2*i;
    }
    else {
        larger = 2*i+1;
    }
    if (A[i] < A[larger])
        swap(A, i, larger);
        i = larger;
    }
    else {
        return;
    }
```

condition → true ↓ false ↓

```
int larger = left>right ? 2*i : 2*i+1;
```

* Heapify → creating heap

In-efficient method :- | 20 | 10 | 30 | 5 | 50 | 40 | → create ~~this~~ its Heap

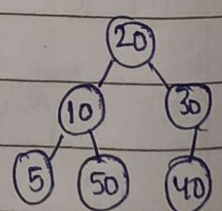
pehle 20 inserted as root → then add 10 in last index | 20 | 10 |

check → Basically perform Insert operation and traversing through given array for inserting.

∴ Time Complexity → O(n*logn);

efficient way O(n) :- | 20 | 10 | 30 | 5 | 50 | 40 |

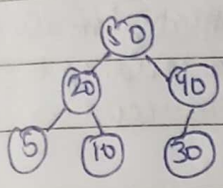
1. Based on array form a tree (complete binary tree)
2. Now starting from last index → choose that node and see it as root node that is it heap.



Here, first take (40) → ∴ it alone as root is max heap ∴ move forward

then (50) ∴ yes, then (5) ∴ yes no need to change now [Observation: Leaf node ko check karne ka jarurat nhi hai, they are always].

then (10) as root node has (5)✓ and (50)✗ as child → ∴ not a heap ∴ swap with max one → [20|50|30|5|10|40] then (30) → as root has 40 as child ∴ not Heap ∴ swap → [20|50|40|5|10|30] then (20) → " " " " and 50 as child ∴ " " ∴ " with 50 → [50|20|40|5|10|30] ✓



Heap formed → O(n) → traversing array O(n), swap O(1)

Leaf lies in range = $\left[\frac{n}{2}\right] + 1$ to n.

here, $n=6$ ∴ leaf indices are = $\left[\frac{6}{2}\right]+1 = 4$ to $6 \Rightarrow 4,5,6$ indices. ✓

∴ we can skip them starting from $\frac{n}{2}^{th}$ positions

∴ traverse array from start to $\frac{n}{2}$ index only ∴ $O(\frac{n}{2}) \Rightarrow O(n)$ ✓

easy

pseudo code → void heapify(int[] a, int n, int i){

```
int largest = i;
int l = 2*i;
int r = 2*i+1;
if(l<n && a[l]>a[largest]){
```

(check for out of range) → pehle index pata karo

```
largest = l;
if (r<n && a[r]>a[largest]){
    largest = r;
}
```

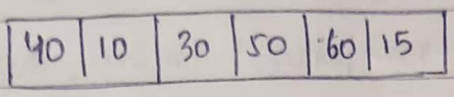
mtlb largest abhi bhi ← `if (largest != i){`
agar i hai jo start
se tha mtlb left, right
se not large, already heap

```
    swap(a, i, largest);
    heapify(a, n, largest);
}
}
```

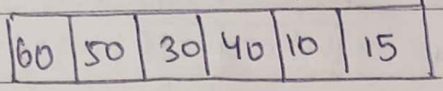
```
buildheap(int[] a, int n){
for(int i=n/2; i>=0; i--){
    heapify(a, n, i);
}
}
```

Ajanta
Page No.
Date

* Heap Sort



↓ convert into a Heap (Heapify) ∴ O(n)



↓ Delete elements one by one and storing from last in array for ascending & first in array for descending for max Heap ∴ O(log n)

Sorted

∴ Time complexity → O(n log n).

heap array inside this gives called after build Heap in psvm.

pseudo code → (understand thoroughly)

```
void heapSort(int[] a, int n){
    for(int i=n-1; i>=0; i--){
        swap(a,0,i);
        heapify(a, i, 1);
    }
}
```

* Priority Queue

code direct understanding →

```
import java.util.*;
public class Main {
    psvm {
        PriorityQueue<Integer> pq = new PriorityQueue<>();
        pq.add(5);
        pq.add(15);
        pq.add(10);
        print(pq.size());
        while(!pq.isEmpty()){
            print(pq.peek());
            pq.poll;
        }
    }
```

gives for MinHeap

→ for maxHeap → PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());

Famous
* Q. Print K^{th} largest element in an array.

| 20 | 10 | 60 | 30 | 50 | 40 |

; K=3. → Ans: 40.

code → int Klargest(int[] a, int K){

PriorityQueue<Integer> pq = new PriorityQueue<>();

for(i=0 to K){

pq.add(a[i]);

}

for(int i=K; i<a.length; i++){

if(pq.peek() < a[i]){

pq.poll();

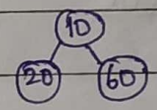
pq.add(a[i]);

}}

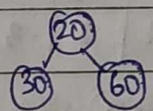
return pq.peek();

DRY RUN and check:

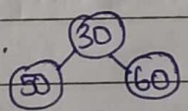
first K=3 ∴ i=0 to 2 pq.add → (0,1,2) (minHeap)



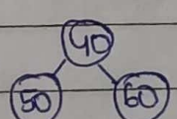
Now from 3 to n → ∴ 3 10<30 ✓ true ∴ remove 10 and add 30 on it's place and adjust heap →



i=4 → 50 ∴ 20<50 true ∴



i=5 → 40 ∴ 30<40 true ∴



↓ reached End

∴ return pq.peek(); → 40 Ans.