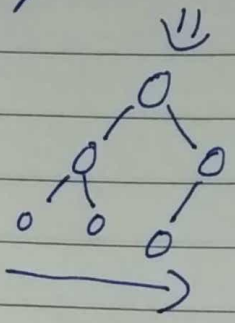
Heap

Heap is a complete binary tree that comes with a heap order property.

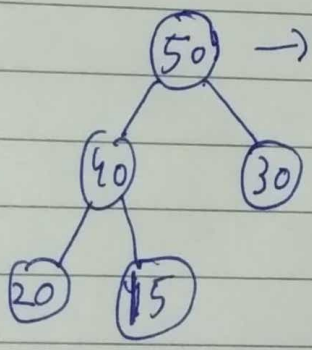
every level is completely filled except last level

nodes added from left (lean towards left)



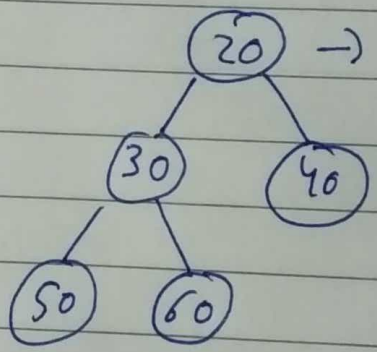
Jab bhi koi node add karni hai to left me daleye phle

Max heap



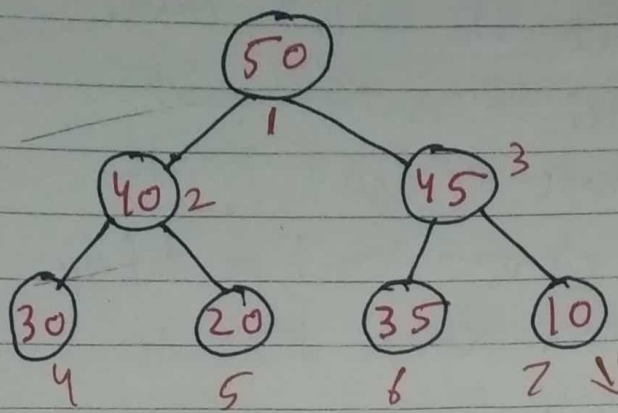
→ root node maximum hogi

Min heap

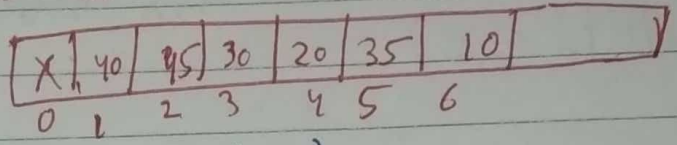


→ root node minimum

Date .....



Node → i
Parent(i) → (i/2)
left child(i) → 2*i
right child(i) → 2*i + 1



(50 insert karna bhul gya

Insertion:-

```
void insert(A[], n, value)
{
    n = n + 1;
    A[n] = value;        → element ko sabse phle last me dal do
    int i = n;
    while (i > 1)
    {
        int parent = i/2;        → element ka parent find kiya
        if (a[parent] < a[i])    → kya parent element se chota hai
        {
            swap(A, parent, i);  → to swap kardo
            i = parent;          → ab agar swap kardiya to ab phir yahi condition check karo
        }
        else
        {
            return;
        }
    }
}
```

Date .....

Deletion:-

Jab bhi delete karna hai to matlab root node ko delete karege

```
Void delete (A[], n)
{
    A[1] = A[n];   → last element ko 1 pe dal diya
    n = n-1;          ab check karege kya se
    i = 1;            max heap hai ya nahi
    while (i < n)
    {
        int left = A[2*i];
        int right = A[2*i+1];
        int larger = left > right ? 2*i : 2*i+1;
        if (A[i] < A[larger])   → agar larger bda hai
        {                          parent se to
            swap(A, i, larger);    swap kardo
            i = larger;            phir heap wali
        }                          property ko check
        else                       karte raho
        {
            return;
        }
    }
}
```

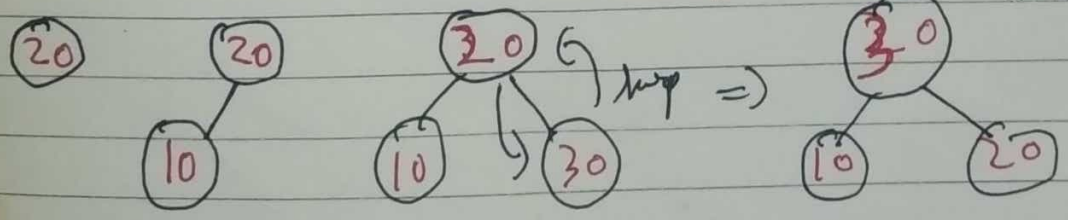
left or right me se jo larger hai wo aaya

→ Heapify method to build a Max heap

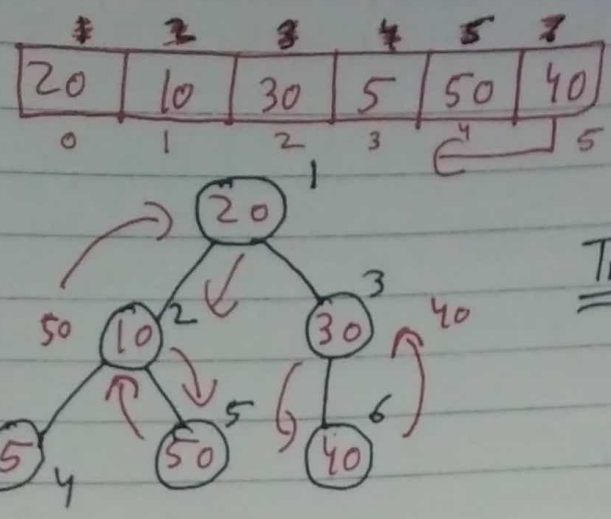
20	10	30	5	50	40

→ ap ko input me ek array milega usse heap me convert karo

Time complexity → O(n log n)



Date .....



Time complexity ⇒ O(n)

$\frac{6}{2} = 3$

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

$\frac{n}{2}$ se n tak ki jo nodes hai unhe process karne ki jarurat nahi kyoKi wo leaf node hai ham usse phle ki node ko check karege

```
Void heapify (int a[], int n, int i)
{
   int largest = i;  → node
   int l = 2*i;      → uske childrens
   int r = 2*i + 1;
   if ( l <= n && a[l] > a[largest]) {
      largest = l;   → comparing with left child
   }
   if( r <= n && a[r] > a[largest]) {
      largest = r;   → comparing with right child
   }
   if( largest != i) {
        swap (a, i, largest); → swap kardo
        heapify (a, n, largest); → phirse is function ko call karo
   }
}
```