Heap Data Structure.

a) Applications ?

1) Implementation of HeapSort

2) Implement Priority Queue.

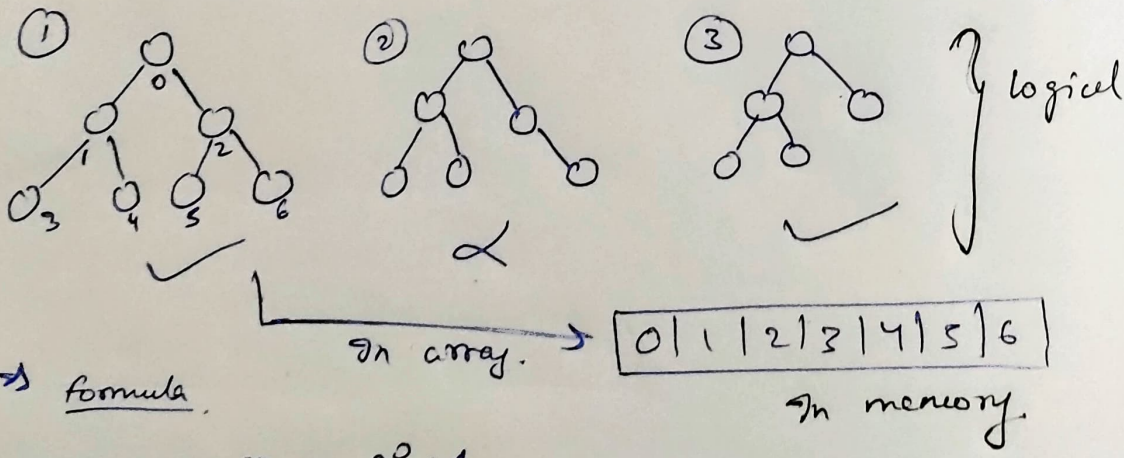
b) Types ?

Min Heap → Highest Priority item has lowest value.
[Eg:- In a Line].

Max Heap → Highest Priority item has highest value.
[Eg:- In a points table].

c) What is a Binary Heap ?

① → Binary Heap is a Complete Binary Tree.
(Stored as an array).



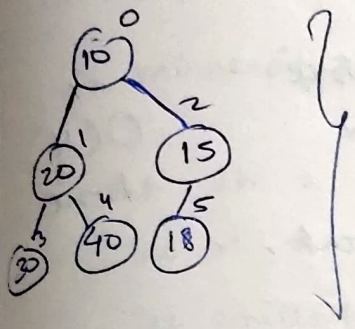
→ formula.

→ left(i) = $2i + 1$
right(i) = $2i + 2$
parent(i) = $\left\lfloor \frac{i-1}{2} \right\rfloor$ (floor)

1) Min Heap Data Structure.

→ Is a complete binary tree.

→ Every node has value smaller than it's descendants.



0	1	2	3	4	5
10	20	15	30	40	18

2) Implementation of Min Heap.

```
class MinHeap
{ int [] arr;
  int size;
  int capacity;
    MinHeap (int c)
    { arr = new int [c];
      size = 0;
      capacity = c;
    }
  int left (int i) { return (2*i + 1); }
  int right (int i) { return (2*i + 2); }
  int parent (int i) { return (i-1)/2; }
  ---- other funcs.
}
```

6) Other functions

i) Insert Operation

Step 1) size++
arr[size-1] = 12 } Complete binary tree maintained — O(H) time.

Step 2) We rearrange by comparing with parent, if less then swap.

```
void insert (int x)
{ if (size == capacity) return;
  size++;  arr[size-1] = x;

  for (int i = size-1; i!=0 && arr[parent(i)] > arr[i])
  {
      swap(arr[i], arr[parent(i)])
      i = parent(i);
  }
}
```

---

ii) Min Heapify — O(log n)

- Given a Binary heap with one possible violation, fix the heap. So (heap, node with violation) → parameters passed.
- Used as subroutine in Extract Min and Build heap

Code

- Similar compare and swap operation like insert.

Time comp
↳ Best → O(1)
worst → O(H)
Avg → O(log n) fn.

```
int arr[];
int size, capacity;

void minHeapify(int i)
{
    int lt = left(i);
    int rt = right(i);
    int smallest = i;

    if(lt < size && arr[lt] < arr[i])
        { smallest = lt; }

    if(rt < size && arr[rt] < arr[smallest])
        { smallest = rt; }

    if(smallest != i)
        { swap(arr[i], arr[smallest]);
          minHeapify(smallest);
        }
```

(iii) Extract Min

⇒ 2 fns → Get min function (i.e root)
→ Heapify function to balance after root is removed.

∴ Eg: The person with priority already visited doctor and has to be removed from queue.

$O(\log n)$

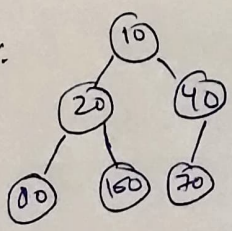
∴ In Heapify

```
→ swap(arr[0], arr[size-1]);
  size--;
  then just call Heapify(0);
```

size == 1
then size--;
return arr[0]

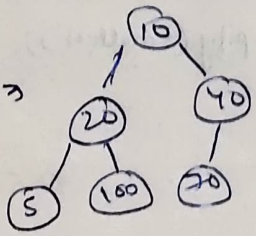
(iv) Decrease Key

Ip:

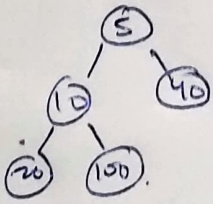


i = 3
x = 5

⇒



⇒ Min Heapify




```
void decreaseKey(int i, int x)
{
    arr[i] = x;
    while(i!=0 && arr[parent(i)] > arr[i])
    {
        swap(arr[i], arr[parent(i)]);
        i = parent(i);
    }
}
```

(v) Delete element

$O(\log n)$.

Step 1 : replace element to be deleted with $-\infty$. (Part of decrease func).

Step 2 : Call decrease function. $(3, -\infty)$

It will replace 3 with $-\infty$ & balance the heap.

Step 3 : Now call Ext Min().

(Looks $O(n \log n)$ but is $O(n)$:)

(vi) Build Heap.

: Gives a random array convert it into a minheap.

I/p : arr[] = {10, 5, 20, 2, 4, 8}

O/p : arr[] = {2, 4, 8, 5, 10, 20}

size - 1 = last ele

$\frac{size-2}{2}$ = Parent of last.

```
void buildHeap()
{
    for(int i = (size-2)/2; i>=0; i--)
        { heapify(i); }
}
```
