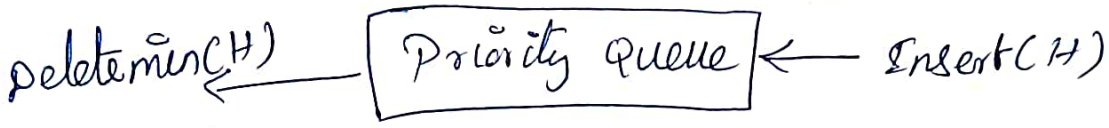
Binary Heap (Heap (or) Priority Queues):

→ It is a DS

→ allows atleast following 2 operations

insert
deletemin ← find, return, delete.

Model of Priority queue:



→ Priority queue implemented by using binary heaps (or) heap.

→ Binary heap should satisfies 2 properties

① structure property (Binary Tree)

② Heap order "

↳ Perform operations quickly.

→ Smallest element should be at root

→ Basic heap operations

Insertion
Deletemin
Deletemax

→ other heap operations

Decrease key (P, Δ, H)

Increase key (P, Δ, H)

Delete (P, H)

Deletemin

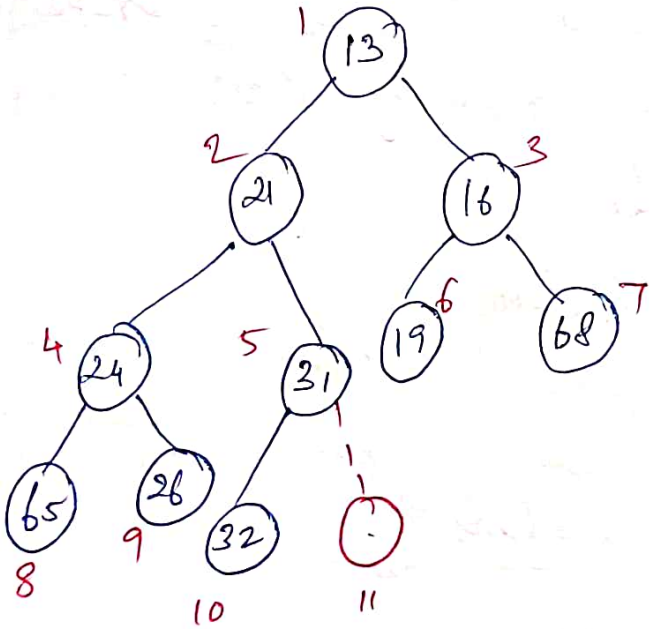
→ Applications of Priority queue.

selection pblm

Event simulation

Insertion:

Eg-, Attempt to insert 14, creating the hole, and bubbling the hole up (Percolate up)



	13	21	16	24	31	19	68	65	26	32	
0	1	2	3	4	5	6	7	8	9	10	11

Sentinel node.

$x = 14, \quad i = 10$

check isFull (H) false

$i = ++H \rightarrow size$

(iv) for ($i = 11$; $H \rightarrow Element[i/2] > x$; $i = i/2$)

$11/2$

$H \rightarrow Element[5] > 14$

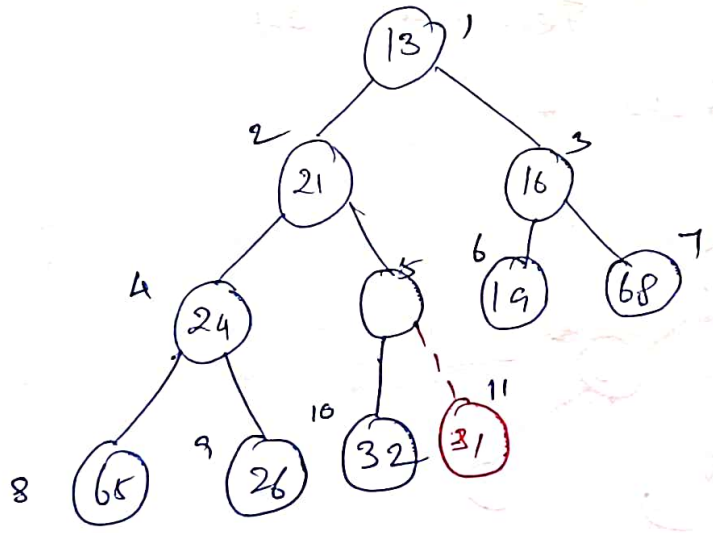
" $= 31$

$31 > 14$ True

$H \rightarrow Element[11] = H \rightarrow Element[i/2]$

5

$H \rightarrow Element[11] = 31$



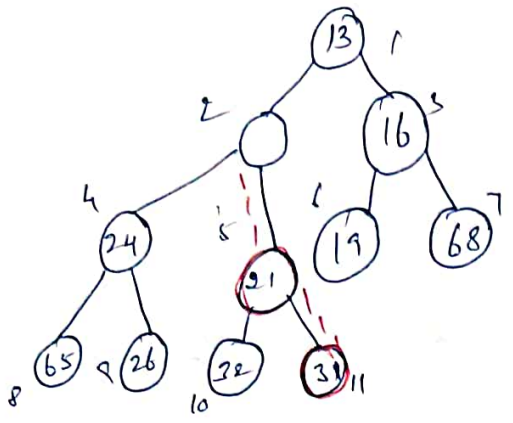
(ii) $i = i/2 \qquad i = 11/2 \qquad i = 5$

$H \rightarrow Element[5] > 14$ i.e., $21 > 14$ ✓

$H \rightarrow Element[5] = H \rightarrow Element[5/2]$

$H \rightarrow Element[2]$

$Element[5] = 21$



Step (iii)

$i = i/2 = 5/2 = 2$

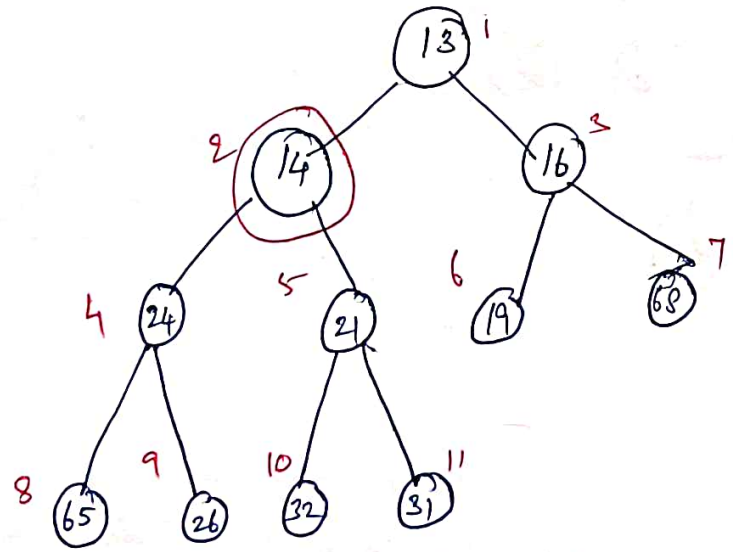
H → Element [2/2] > 14

H → Element [1] > 14

13 > 14 false.

Terminate loop.

H → Element [2] = 14



	13	14	16	24	21	19	68	65	26	32	31
0	1	2	3	4	5	6	7	8	9	10	11

Routine to Insert:

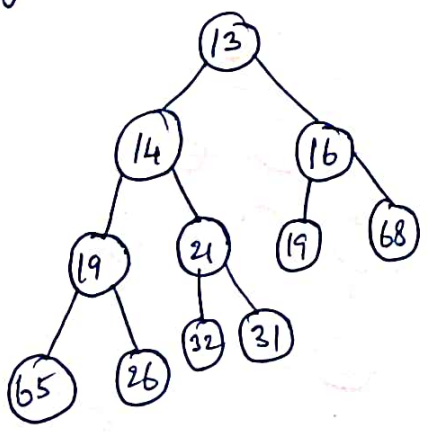
```
Void insert (ElementType x, PriorityQueue H)
{
  int i;
  if (Isfull (H))
  {
    Error ("Priority queue is full");
    return;
  }
for (i = ++H->size; H->Element[i/2] > x; i/2)
    H->Element[i] = H->Elements[i/2];
  H->Element[i] = x;
}
```

Delete Min:

→ finding the minimum is easy, the hard part is removing it.

→ when minimum is removed a hole is created at the root.

Eg:



→ After 13 is removed, try to place 31, it violate heap order Property. So place 14 in the hole, slide down the hole one level "percolate down"

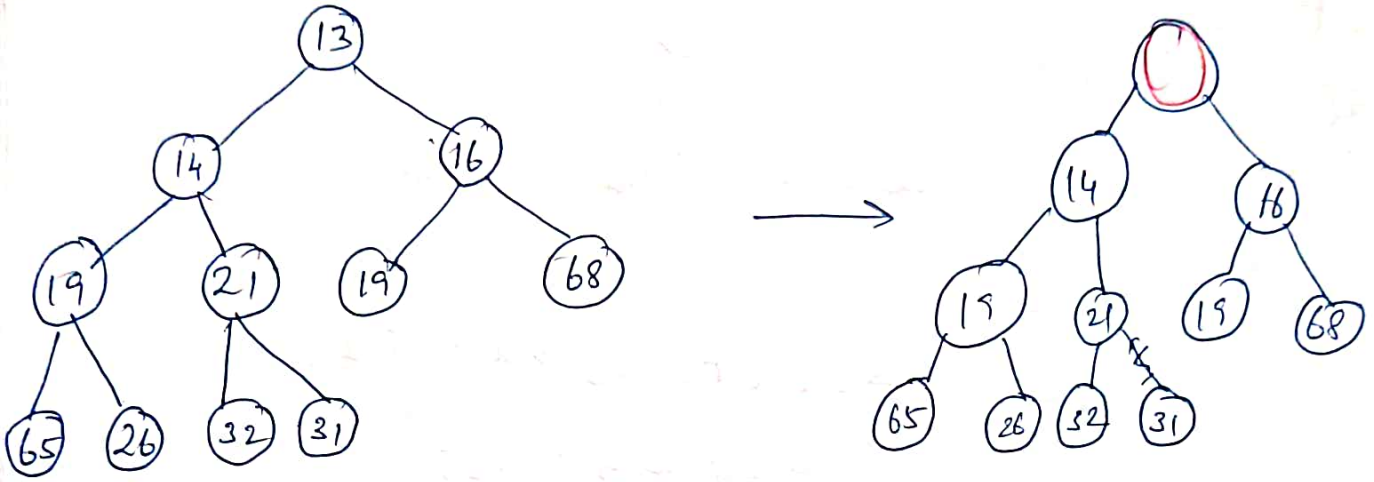
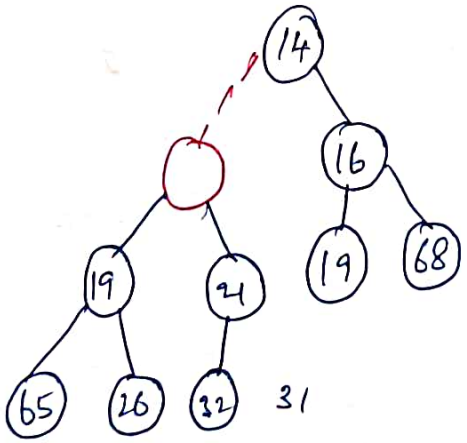
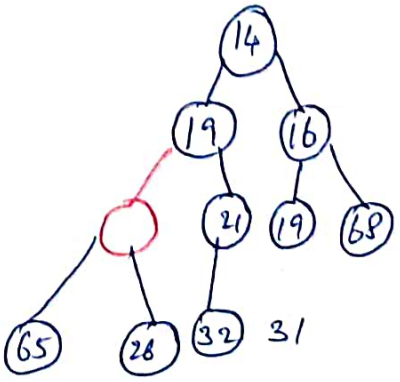


Fig: creation of hole at root

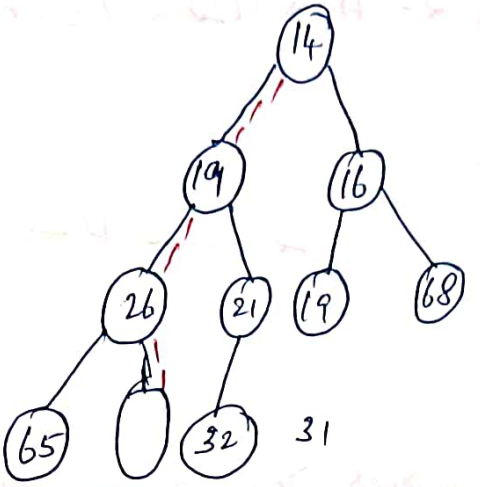
→ Compare 14 & 16, 14 is minimum, so 14 placed in the hole, created at root.



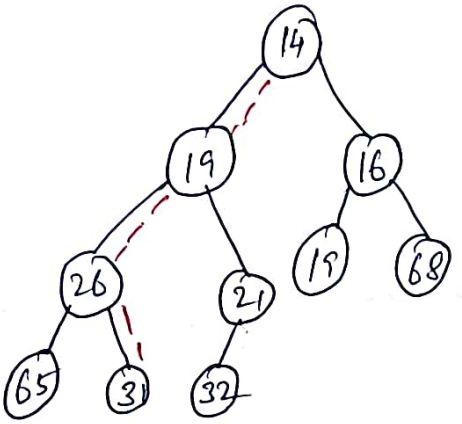
→ Compare 19 & 21, 19 & smaller 19 is filled in the hole



Compare 26 & 65,
26 placed in hole



→ finally 31 is placed in hole.



Routine:

```
ElementType deleteMin(PriorityQueue H)
{
  int i, child;
  ElementType MinElement, LastElement;
  if(IsEmpty(H))
  {
    Error("Priority Queue is Empty");
    return H->Element[0];
  }
  MinElement = H->Elements[1];
  LastElement = H->Elements[H->size--];
```

```
for (i=1; i*2 <= H->size; i=child)
{
    child = i*2;
    if (child != H->size && H->Elements[child+1] <
                                H->Elements[child])
        child++;
    if (lastElement > H->Elements[child])
        H->Elements[i] = H->Elements[child];
    else
        break;
}
H->Elements[i] = lastElement;
return MinElement;
}
```

Trace out

① Consider Previous example MinElement = 13, lastElement: 31

MinElement = H->Elements[1];

" = 13

lastElement = H->Elements [H->size--];

lastElement = H->Elements[11] = 31

for (i=1; i*2 <= H->size; i=child)

i*2 < 10 (True)

child = i*2

if (2 != 10 && H->Elements[3] < H->Element[child])

True && 16 < 14 false.

if (31 > H->Element[2])

31 > 14 True

H->Elements[1] = H->Elements[2];

H->Element[1] = 14

step 2:

i : child i=2

2*2 <=10 True

child = 4

if (4 != 10 && E[5] < E[4])

True 21 < 19 False

If (31 > 19) True

H->Element[2] = 19

step 3:

i = child i = 4 4*2 = 8

If (8 != 10 && 26 < 65) True

child ++ ; child = 9

H->Element[4] = 26

step 4:

i = 9, 9*2 <= 10 false

Terminate loop

H->Elements[9] = 31