Sairam INSTITUTIONS

Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

YEAR	SEM
II	III

CS8391

DATA STRUCTURES
(COMMON TO CSE &IT)

UNIT No. 2

LINEAR DATA STRUCTURES
STACKS,QUEUES
2.5.1 PRIORITY QUEUE

2.5.1 PRIORITY QUEUE

A priority queue is a collection of elements such that each element has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

An element of higher priority is processed before any element of lower priority.

Two elements with the same priority are processed according to the order in which they were added to the queue.

Two types of queue are

Ascending Priority Queue

Descending Priority Queue

Ascending Priority Queue

Collection of items into which item can be inserted arbitrarily & from which only the smallest item can be removed.

Descending Priority Queue

Collection of items into which item can be inserted arbitrarily & from which only the largest item can be removed.

APPLICATIONS OF QUEUE

Batch processing in an operating system To implement Priority Queues.

Priority Queues can be used to sort the elements using Heap Sort. Simulation and Mathematics user Queuing theory.

Computer networks where the server takes the jobs of the client as per the queue strategy.

It is a data structure which determines the priority of jobs. The Minimum the value of Priority, Higher is the priority of the job. The best way to implement Priority Queue is Binary Heap. A Priority Queue is a special kind of queue data structure. It has zero or more collection of elements, each element has a priority value. Priority queues are often used in resource management, simulations, and in the implementation of some algorithms (e.g., some graph algorithms, some backtracking algorithms).

Several data structures can be used to implement priority queues.

Implementation of Priority Queue

1. Linked List.
2. Binary Search Tree.
3. Binary Heap.

Linked List :

A simple linked list implementation of priority queue requires o(1) time to perform the insertion at the front and o(n) to delete at minimum element.

Binary Search tree :

This gives an average running time of o(log n) for both insertion and deletion.(delete min).

The efficient way of implementing priority queue is Binary Heap (or) Heap.

Heap has two properties:

1. Structure Property.
2. Heap Order Property.

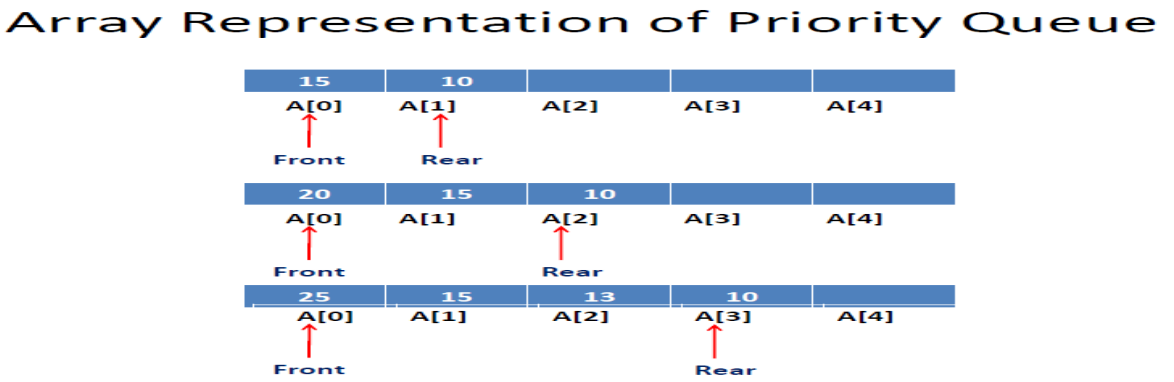
1.Structure Property :

The Heap should be a complete binary tree, which is a completely filled tree, which is a completely filled binary tree with the possible exception of the bottom level, which is filled from left to right. A Complete Binary tree of height H, has between 2h and (2h+1 - 1) nodes.

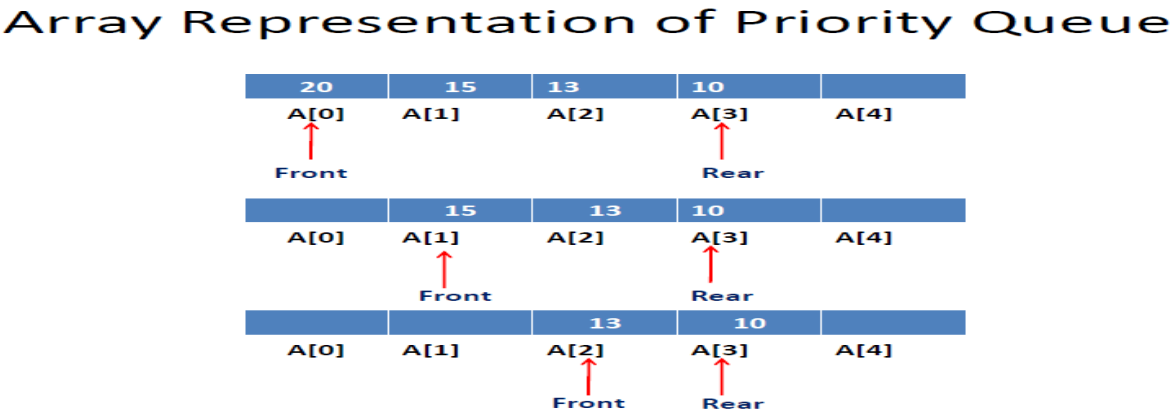
Sentinel Value : The zero th element is called the sentinel value. It is not a node of the tree. This value is required because while addition of new node, certain operations are performed in a loop and to terminate the loop, sentinel value is used. Index 0 is the sentinel value. It stores ir related value, in order to terminate the program in case of complex codings.

Structure Property : Always index 1 should be starting position.

Heap Order Property : The property that allows operations to be performed quickly is a heap order property



.



HEAP OPERATIONS:

There are 2 operations of heap

Insertion

Deletion

Insert:

Adding a new key to the heap

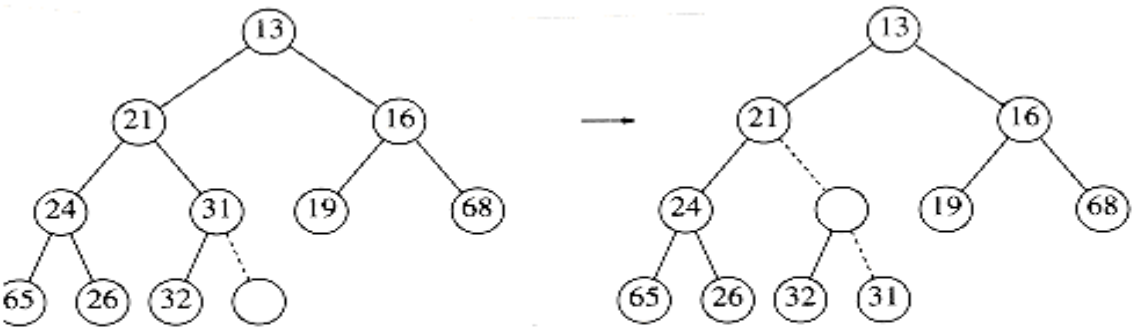
Rules for the insertion:

To insert an element X, into the heap, do the following:

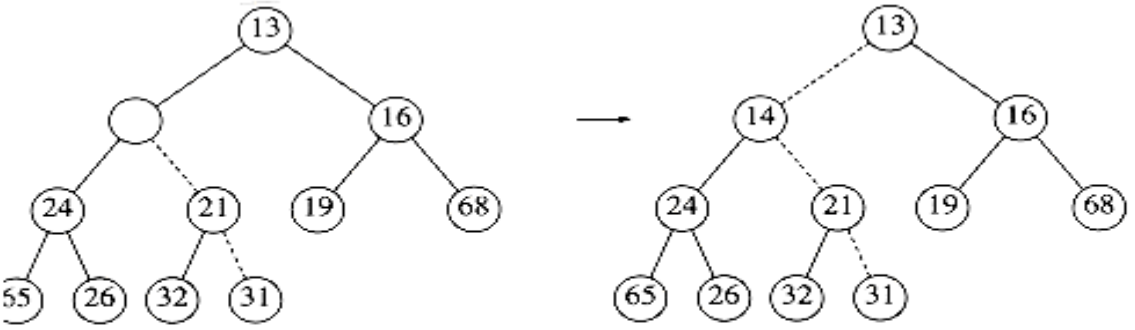
Step1: Create a hole in the next available location , since otherwise the tree will not be complete.

Step2: If X can be placed in the hole, without violating heap order, then do insertion, otherwise slide the element that is in the holes parent node, into the hole, thus, bubbling the hole up towards the root.

Step3: Continue this process until X can be placed in the hole.



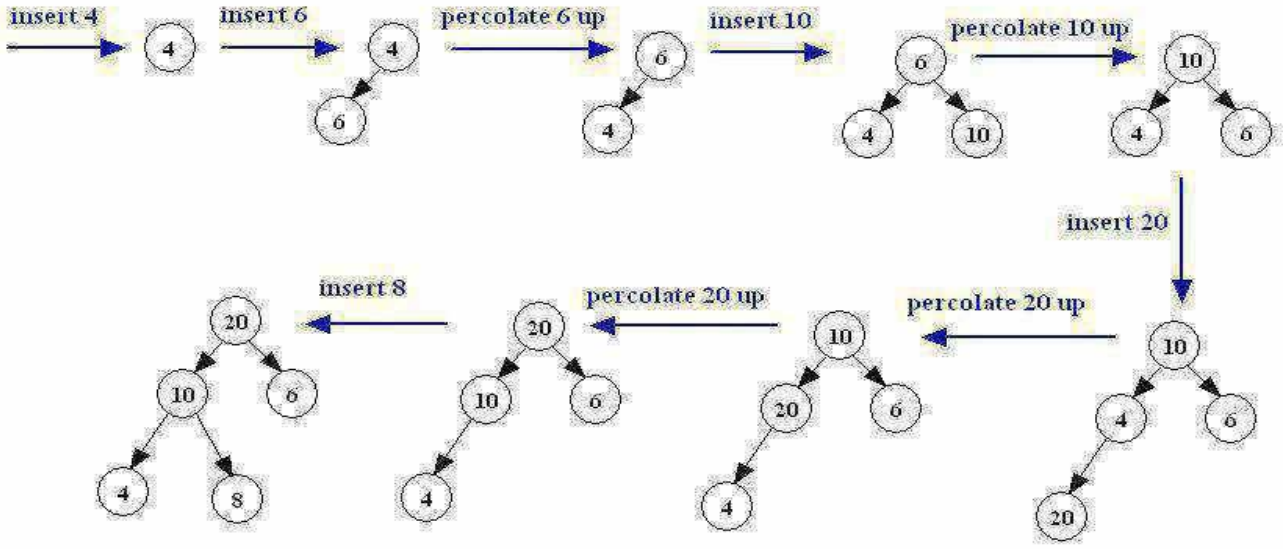
Attempt to insert 14: creating the hole, and bubbling the hole up



The remaining two steps to insert 14 in previous heap

Example Problem :

Insert the keys 4, 6, 10, 20, and 8 in this order in an originally empty maxheap



Delete-max or Delete-min:

Removing the root node of a max- or min-heap, respectively

Procedure for Delete min :

* Delete min operation is deleting the minimum element from the loop.

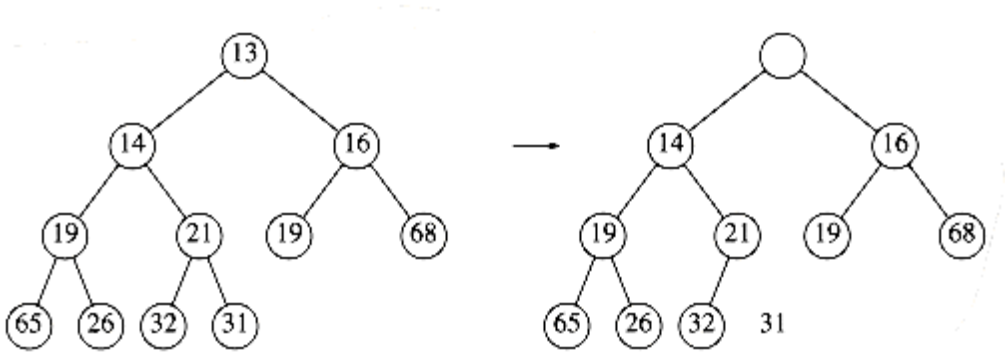
* In Binary heap | min heap the minimum element is found in the root.

* When this minimum element is removed, a hole is created at the root.

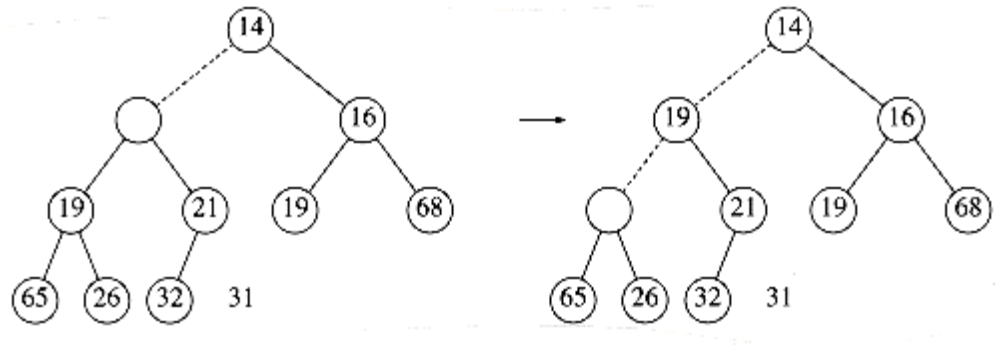
* Since the heap becomes one smaller, make the last element X in the heap to move somewhere in the heap.

* If X can be placed in the hole, without violating heap order property, place it , otherwise slide the smaller of the holes children into the hole, thus , pushing the hole down one level.

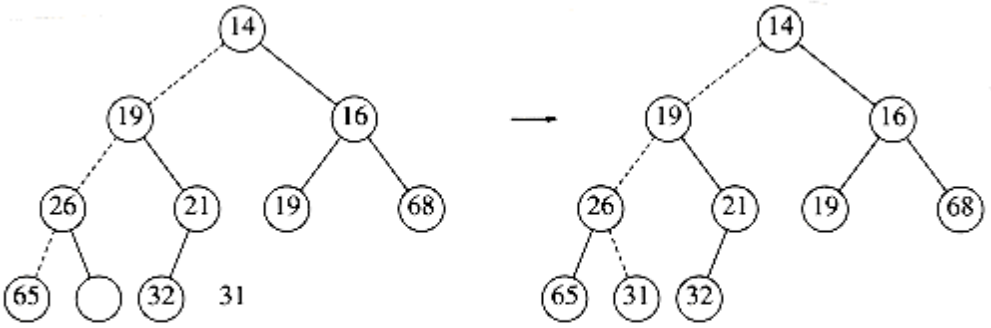
* Repeat this process until X can be placed in the hole.This general strategy is known as Percolate Down.



Creation of the hole at the root

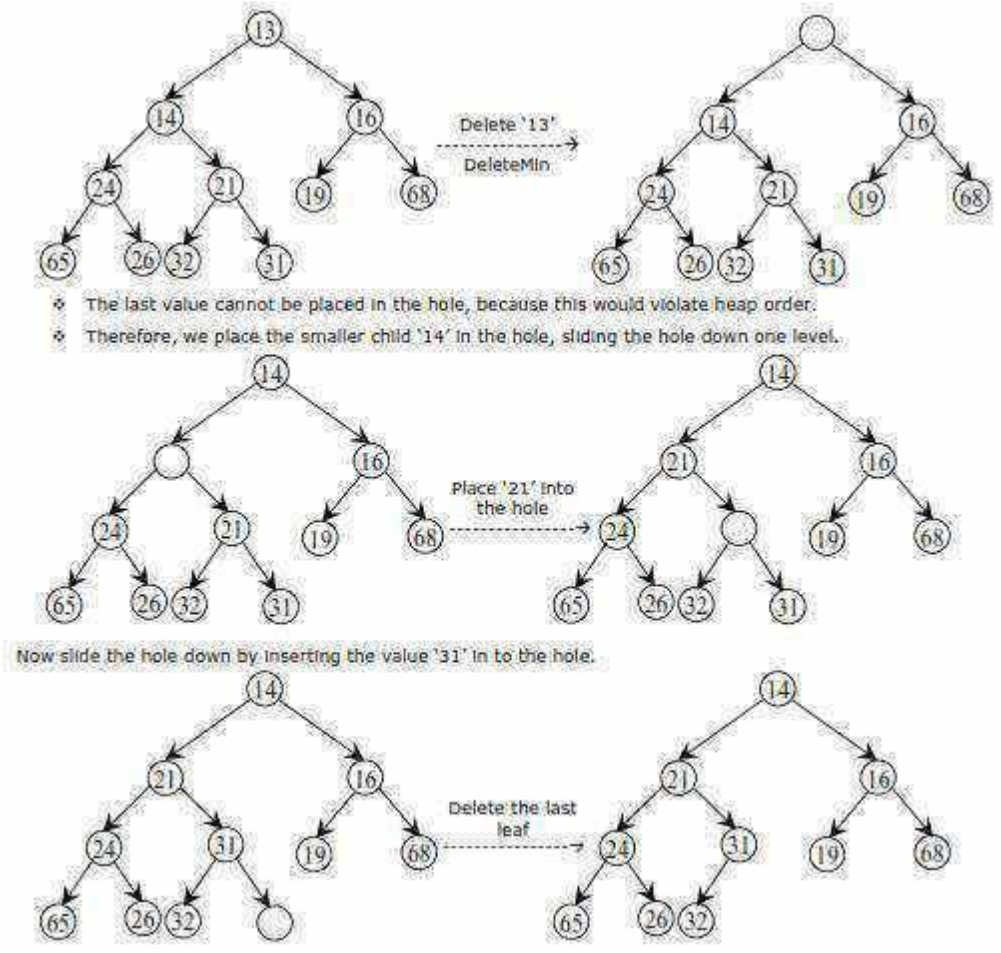


Next two steps in delete_min



Last two steps in delete_min

Delete Min -- 13



BINARY HEAP ROUTINES [Priority Queue]

```
Typedef struct heapstruct *priorityqueue;
Typedef int elementype;
Struct heapstruct
{
int capacity;
int size;
elementtype *element;
};
```

Declaration of Priority Queue

```
Priorityqueue initialize(int maxelement)
{
Priorityqueue H;
If(minpsize<maxelements)
Error("Priority queue size is too small);
H=malloc(sizeof(struct heapstruct));
If(H=NULL)
Fatalerror("Out of space");
/ * Allocate the array plus one extra for sentinel * /
H-->elements=malloc((maxelements+1)*sizeof(elementtype));
If(H-->elements==NULL)
Fatalerror("out of space");
H-->capacity=maxelements;
H-->size=0;
H-->elements[0]=mindata;
Return H;
}
/ * H-->elements[0]=sentinelvalue * /
```

Insert Routine

```
Void insert(elementtype X, priorityqueue H)
{
int i;
if(isfull(H))
```

```
{
Error("Priority queue is full");
Return;
}
For(i=++H-->size;H-->elements[i/2]>X;i=i/2)
H-->elements[i]=H-->elements[i/2];
H-->elements[i]=X;
}
```

Delete 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-->element[1];
Lastelement=H-->element[H-->size--];
For(i=1;i*2<=H-->size;i=child)
{
/ *Find smaller child */
Child=i*2;
If(child!=H-->size && H-->elements[child++]<H-->elements[child])
{
Child++;
}
/ * Percolate one level * /
If(lastelement>H-->elements[child])
H-->element[i]=H-->elements[child];
```

Else

Break;

}

H-->element[i]=lastelement;

Return minelement;

}

Other Heap Operations

Decrease Key.

The Decrease key(P,Δ,H) operation decreases the value of the key at position P, by a positive amount Δ.

This may violate the heap order property, which can be fixed by percolate up.

Ex : decreasekey(2,7,H)

Increase Key.

The Increase Key(P,Δ,H) operation increases the value of the key at position P, by a positive amount Δ. This may violate heap order property,

which can be fixed by percolate down. Ex : increase key(2,7,H)

Delete.

The delete(P,H) operation removes the node at the position P, from the heap H. This can be done by,

Step 1: Perform the decrease key operation, decrease key(P,∞,H).

Step 2: Perform deletemin(H) operation.

Step 1: Decreasekey(2, ∞,H)

Build Heap.

ADVANTAGE

The biggest advantage of heaps over trees in some applications is that construction of heaps can be done in linear time. It is used in

o Heap sort

o Selection algorithms

o Graph algorithms

DISADVANTAGE

Heap is expensive in terms of safety ,maintenance, performance ,Performance

Allocating heap memory usually involves a long negotiation with the OS.

APPLICATIONS

The heap data structure has many applications

Heap sort

Selection algorithms

Graph algorithms

Heap sort :

One of the best sorting methods being in-place and with no quadratic worst-case scenarios.

Selection algorithms:

Finding the min, max, both the min and max, median, or even the *k*-th largest element can be done in linear time using heaps.

Graph algorithms: By using heaps as internal traversal data structures, run time will be reduced by an order of polynomial. Examples of such problems are Prim'sminimal spanning tree algorithm and Dijkstra's shortest path problem