PRIORITY QUEUE (HEAPS) – BINARY HEAP

- In a priority queue, an element with high priority is served before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

Two types of priority queue

1. Max Priority Queue
2. Min Priority Queue

Max Priority Queue

In Max Priority Queue, elements are inserted in the order in which they arrive they queue and always maximum value is removed first from the queue.

E.x : insert in order 8, 3, 2, 5 removed in the order 8, 5, 3, 2

Min Priority Queue

Min Priority Queue is similar to Max priority queue except removing maximum elements first, we remove min. element first in min priority queue

BINARY HEAP

- The efficient way of implementing priority queue is Binary Heap.
- Binary heap is merely referred as Heaps

Heap have two properties namely

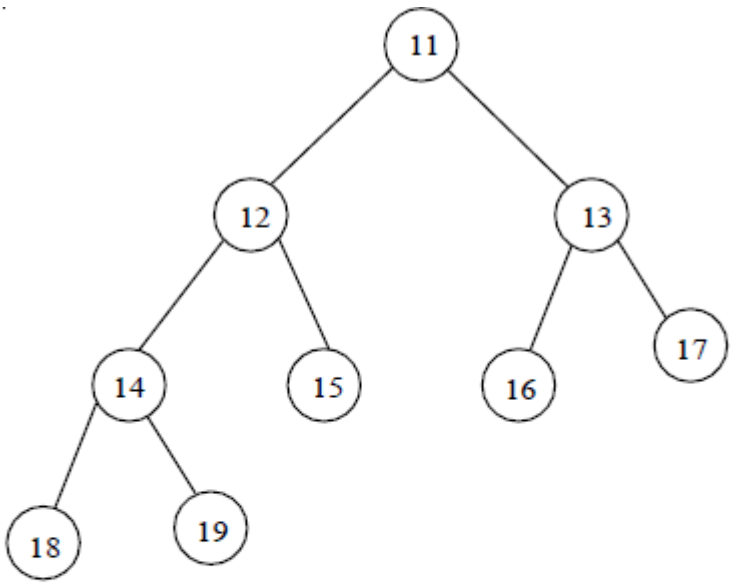
- Structure property
- Heap order property.

Structure Property

- A heap should be complete binary 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 2^H and 2^{H+1} -1 nodes.
- For example, if the height is 3. Then the number of nodes will be between 8

and 15. (ie) (2^3 and 2^4-1).

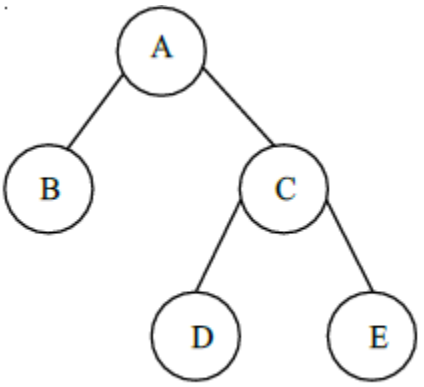
- For any element in array position i, the left child is in position 2i, the right child is in position 2i + 1, and the parent is in i/2.
- As it is represented as array it doesn't require pointers and also the operations required to traverse the tree are extremely simple and fast.
- But the only disadvantage is to specify the maximum heap size in advance.



A complete Binary Tree

	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

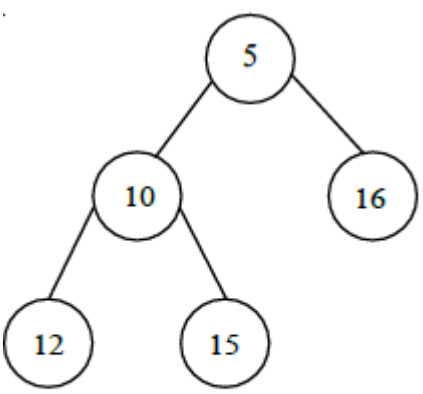
Array implementation of complete binary tree



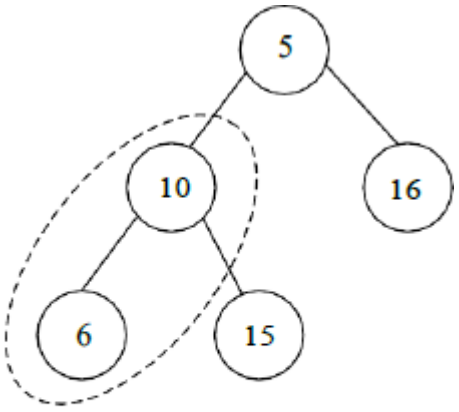
Not a Complete Binary Tree

Heap Order Property

- In a heap, for every node X, the key in the parent of X is smaller than (or equal to) the key in X.
- This property allows the deletemin operations to be performed quickly has the minimum element can always be found at the root.
- Thus, we get the FindMin operation in constant time.



(a) Binary tree with structure and heap order property.



(b) Binary tree with structure but violating heap order property

Basic Heap Operations

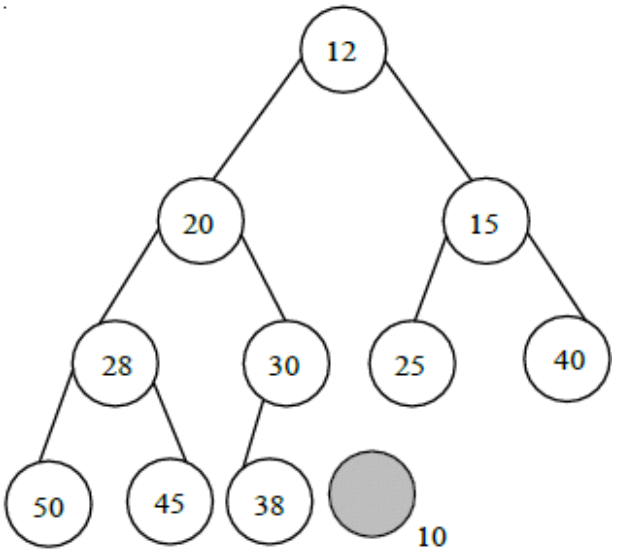
To perform the insert and DeleteMin operations ensure that the heap order property is maintained.

Insert Operation

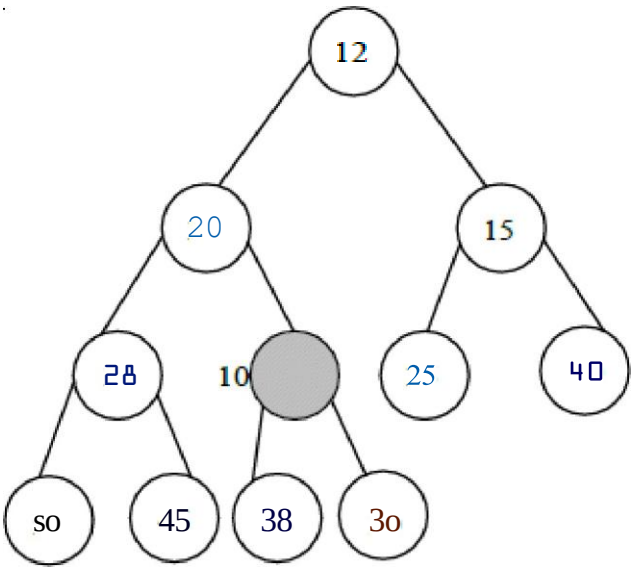
- To insert an element X into the heap, we create a hole in the next available location, otherwise the tree will not be complete.
- If X can be placed in the hole without violating heap order, then place the element X there itself.
- Otherwise, we slide the element that is in the hole's parent node into the hole, thus bubbling the hole up toward the root.
- This process continues until X can be placed in the hole.
- This general strategy is known as Percolate up, in which the new element is percolated up the heap until the correct location is found.

Example :

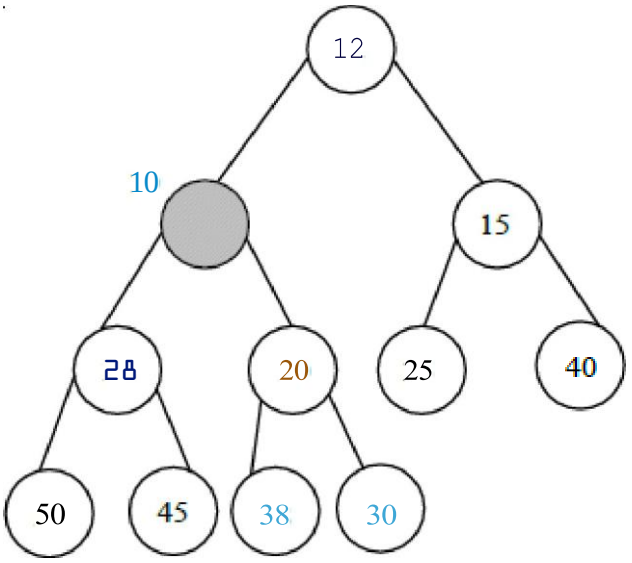
To Insert 10 :



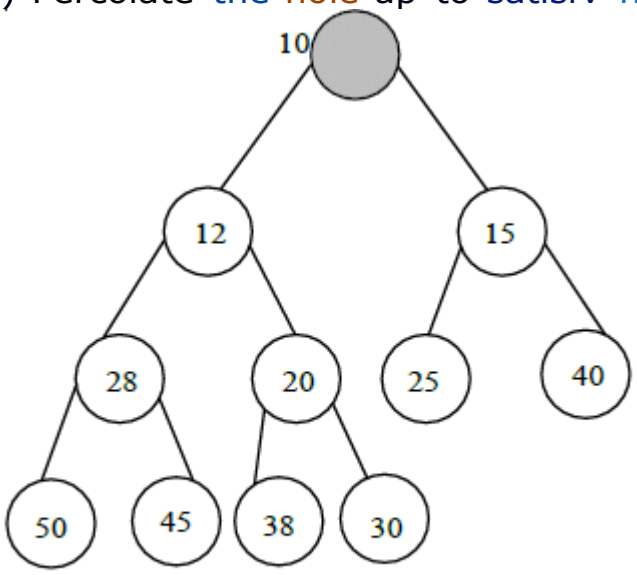
(a) A hole is created at the next location



(b) Percolate the hole up to satisfy heap order



(c) Percolate the hole up to satisfy heap order



(d) Percolate the hole up to satisfy heap order

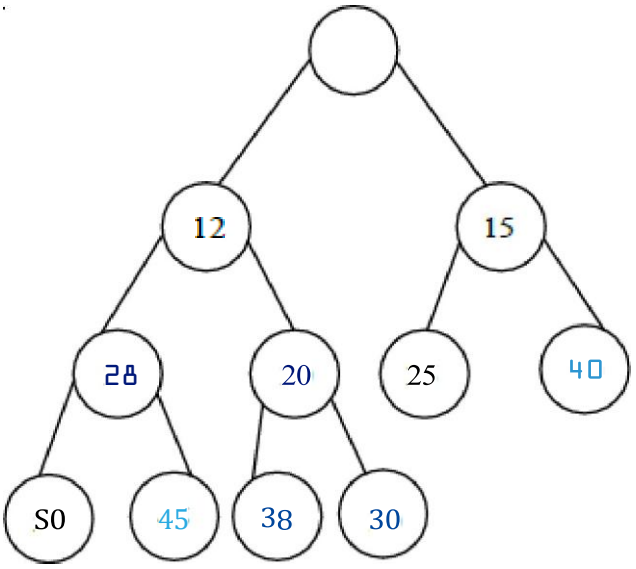
In Fig (d) the value 10 is placed in its correct location.

DeleteMin

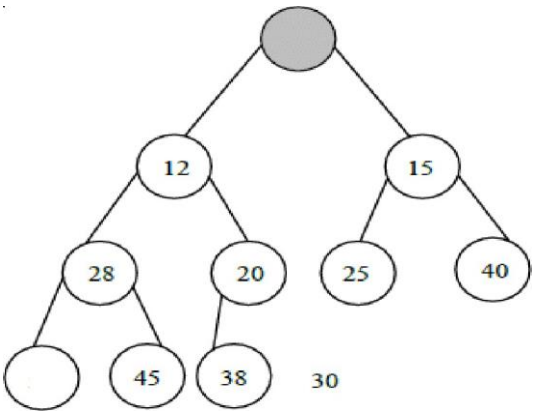
- DeleteMin Operation is deleting the minimum element from the Heap.
- In Binary heap the minimum element is found in the root.
- When this minimum is removed, a hole is created at the root.
- Since the heap becomes one smaller, makes the last element X in the heap to move somewhere in the heap.

- If X can be placed in hole without violating heaporder property place it.
- Otherwise, we slide the smaller of the hole's children into the hole, thus pushing the hole down one level.
- We repeat until X can be placed in the hole.
- This general strategy is known as percolate down.

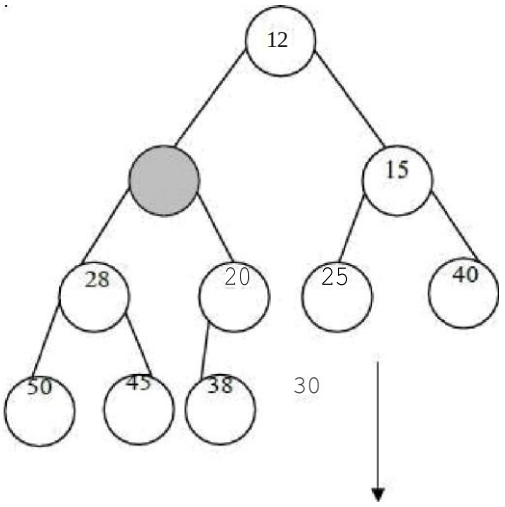
Example: To delete the minimums element 10



Delete minimum element 10, creates the hole at the root.

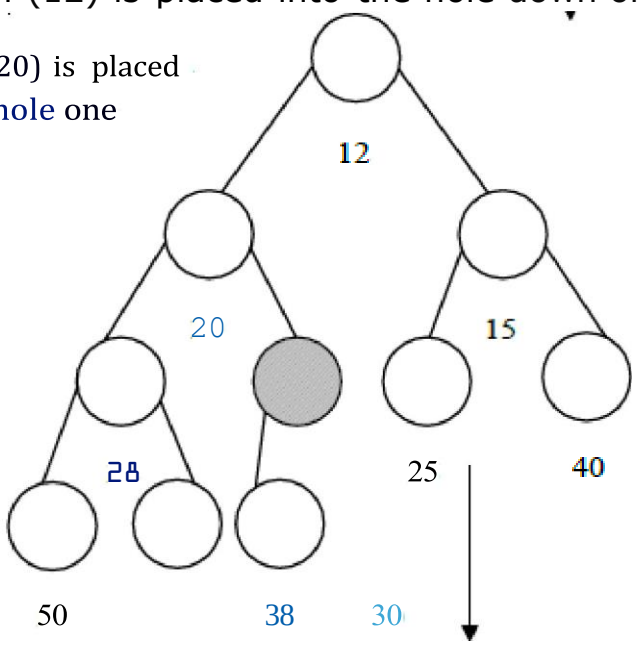


The last element '30' must be moved somewhere in the heap

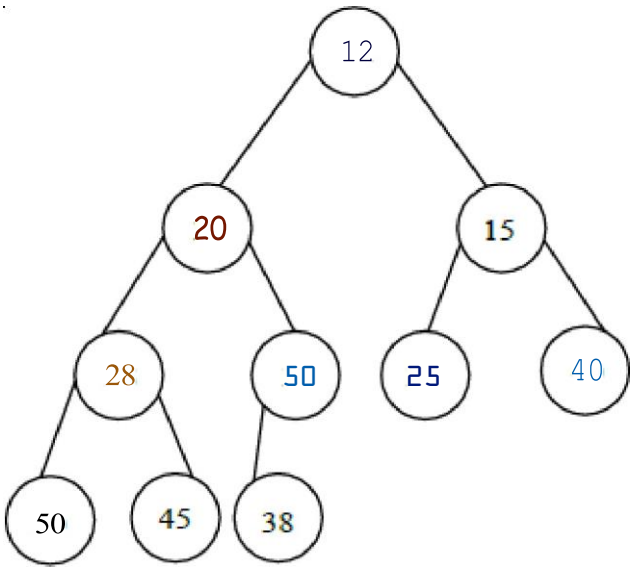


The Hole's smallest children (12) is placed into the hole down one level.

The hole's smallest children (20) is placed into the hole by pushing the hole one level down.



The last element '30' :is placed in the correct hole.


12
20
15
28
50
25
40
50
45
38

```
#include <iostream.h>
#include <conio.h>

#define MAX 20

int heap[MAX];
int size = 0;

// Function to heapify
void heapify(int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && heap[l] > heap[largest])
        largest = l;

    if (r < n && heap[r] > heap[largest])
        largest = r;

    if (largest != i) {
        int temp = heap[i];
        heap[i] = heap[largest];
        heap[largest] = temp;

        heapify(n, largest);
    }
}

// Function to insert an element
void insert(int value) {
    heap[size] = value;
    size++;

    for (int i = (size / 2) - 1; i >= 0; i--) {
        heapify(size, i);
    }
}

// Function to delete a given element
void deleteNode(int value) {
    int i;
    for (i = 0; i < size; i++) {
        if (heap[i] == value)
            break;
```

```
    }

    if (i == size) {
        cout << "Element not found\n";
        return;
    }

    heap[i] = heap[size - 1];
    size--;

    for (i = (size / 2) - 1; i >= 0; i--) {
        heapify(size, i);
    }
}

// Function to display heap
void display() {
    for (int i = 0; i < size; i++) {
        cout << heap[i] << " ";
    }
}

void main() {
    clrscr();

    insert(3);
    insert(4);
    insert(9);
    insert(5);
    insert(2);

    cout << "Max-Heap array: ";
    display();

    deleteNode(4);

    cout << "\nAfter deleting an element: ";
    display();

    getch();
}
```

Output:

Max-Heap array: 9 5 4 3 2
After deleting an element: 9 5 2 3

APPLICATIONS OF HEAP

- To quickly find the smallest and largest element from a collection of items or array.
- In the implementation of Priority queue in graph algorithms like Dijkstra's algorithm (shortest path), Prim's algorithm (minimum spanning tree) and Huffman encoding (data compression).
- In order to overcome the Worst Case Complexity of Quick Sort algorithm from O(n^2) to O(nlog(n)) in Heap Sort.
- For finding the order in statistics.
- Systems concerned with security and embedded system such as Linux Kernel uses Heap Sort because of the O(nlog(n)) .