Priority Queue using Binary Heap

Difficulty Level : Medium • Last Updated : 03 Apr, 2023

Priority Queue is an extension of the queue with the following properties:

1. Every item has a priority associated with it.
2. An element with high priority is dequeued before an element with low priority.
3. If two elements have the same priority, they are served according to their order in the queue.

A Binary Heap is a Binary Tree with the following properties:

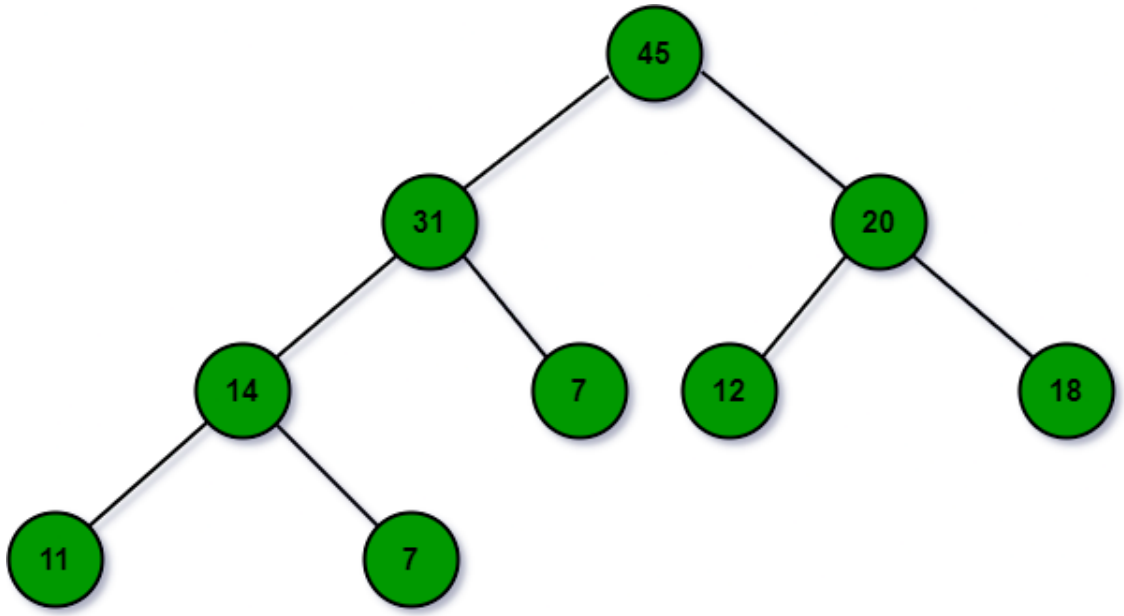
1. It is a Complete Tree. This property of Binary Heap makes them suitable to be stored in an array.
2. A Binary Heap is either **Min Heap** or **Max Heap**.
3. In a **Min Binary Heap**, the key at the root must be minimum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree.
4. Similarly, in a **Max Binary Heap**, the key at the root must be maximum among all keys present in Binary Heap. The same property must be recursively true for all nodes in Binary Tree.

Operation on Binary Heap

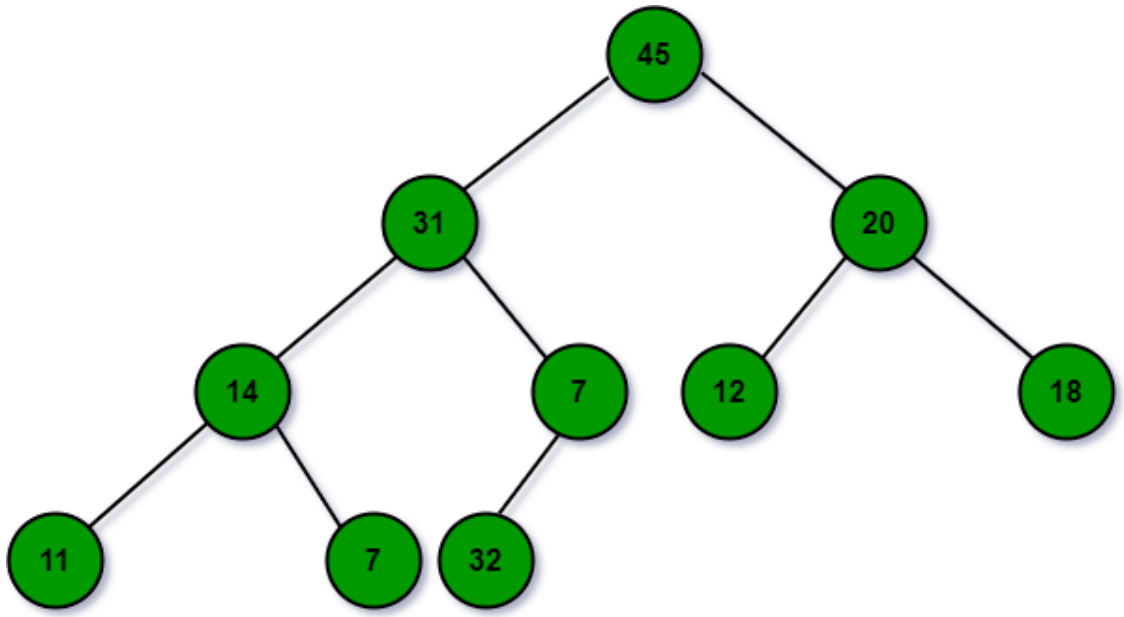
- **insert(p):** Inserts a new element with priority **p**.
- **extractMax():** Extracts an element with maximum priority.
- **remove(i):** Removes an element pointed by an iterator i.
- **getMax():** Returns an element with maximum priority.
- **changePriority(i, p):** Changes the priority of an element pointed by **i** to **p**.

Example of A Binary Max Heap

- Suppose below is the given Binary Heap that follows all the properties of Binary Max Heap.

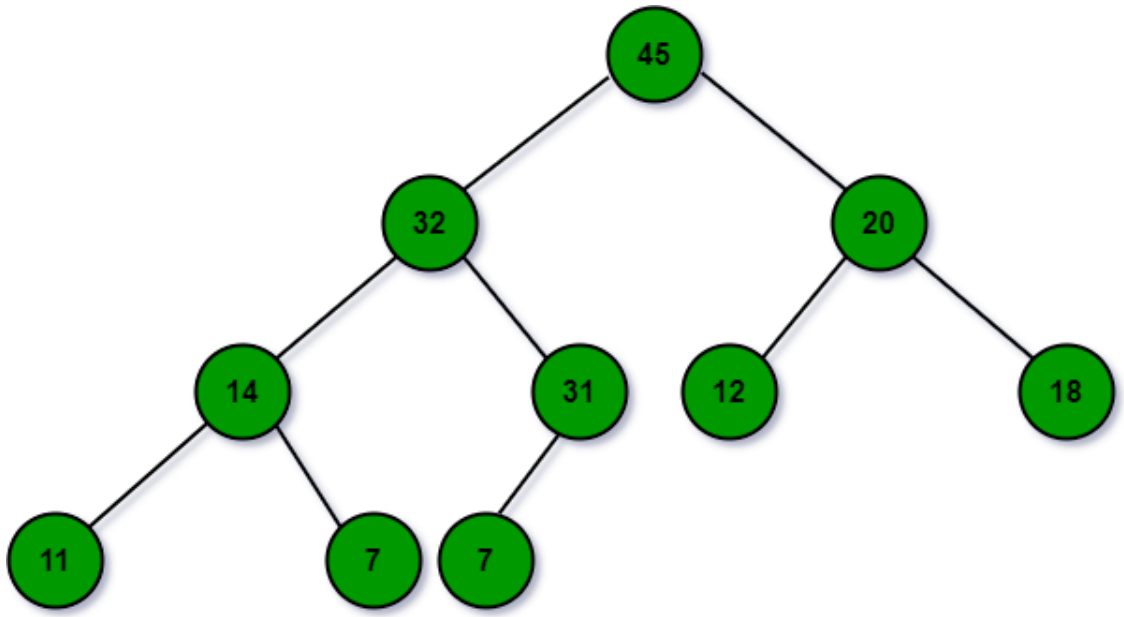


- **Now a node with value 32 need to be insert in the above heap:** To insert an element, attach the new element to any leaf. **For Example** A node with priority **32** can be added to the leaf of the node **7**. But this violates the heap property. To maintain the heap property, shift up the new node **32**.

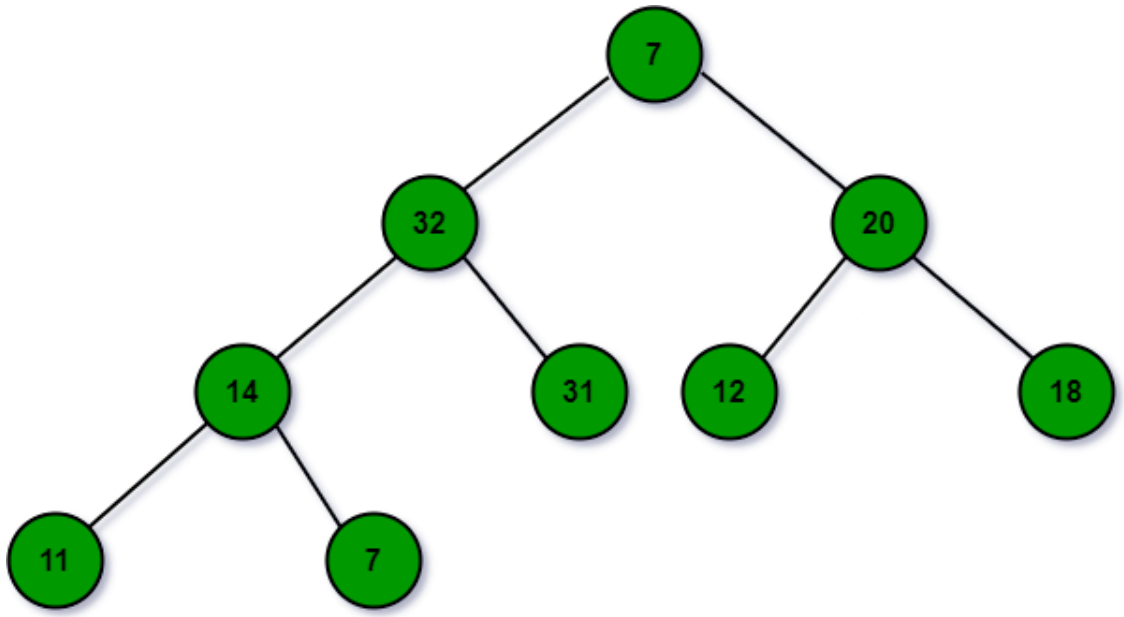


- **Shift Up Operation get node with 32 at the correct position:** Swap the incorrectly placed node with its parent until the heap property is satisfied. **For Example:** As node **7** is less

than node **32** so, swap node **7** and node **32**. Then, swap node **31** and node **32.**

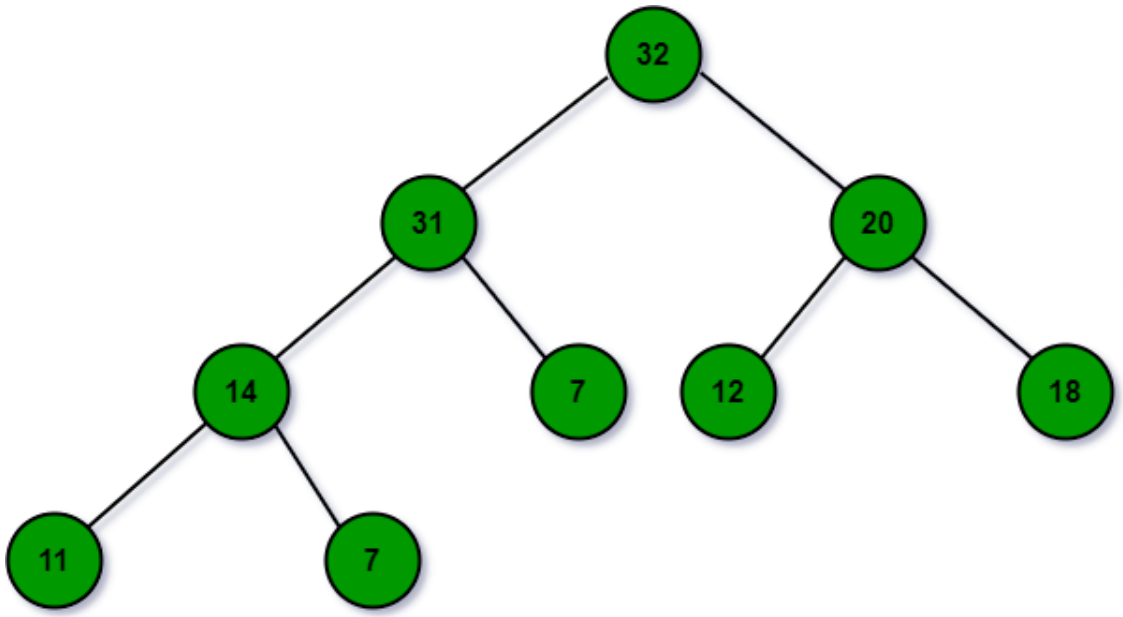


- **ExtractMax:** The maximum value is stored at the root of the tree. But the root of the tree cannot be directly removed. First, it is replaced with any one of the leaves and then removed. **For Example:** To remove **Node 45**, it is first replaced with node **7**. But this violates the heap property, so move the replaced node down. For that, use shift-down operation.



- **ShiftDown operation:** Swap the incorrectly placed node with a larger child until the heap property is satisfied. **For Example** Node **7** is swapped with node **32** then, last it is

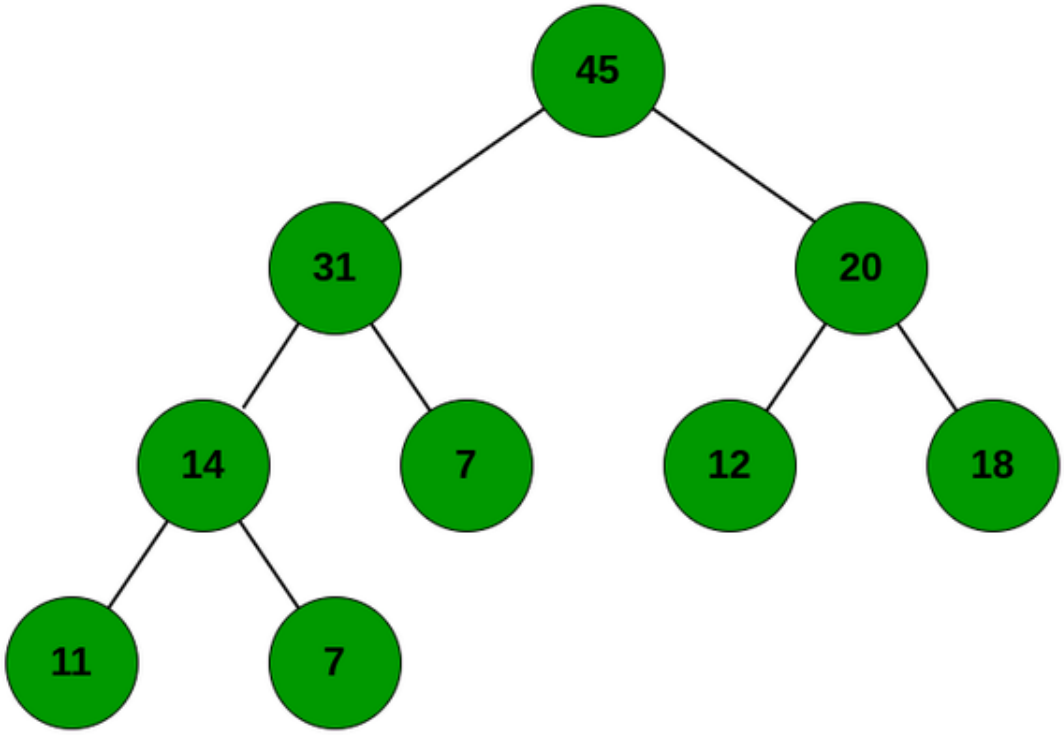
swapped with node **31**.

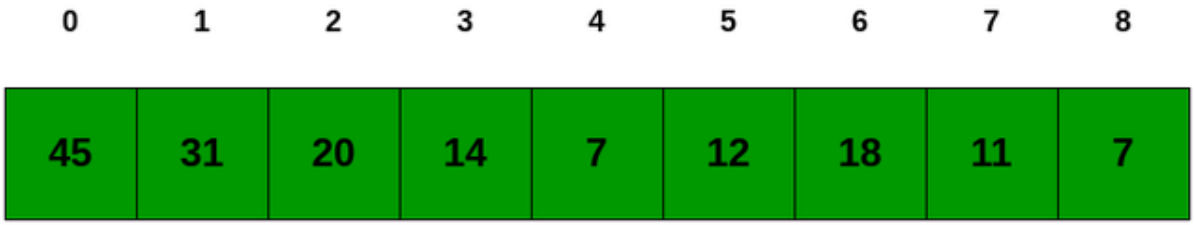


- **ChangePriority:** Let the changed element shift up or down depending on whether its priority decreased or increased. **For Example:** Change the priority of nodes **11 to 35**, due to this change the node has to shift up the node in order to maintain the heap property.
- **Remove:** To remove an element, change its priority to a value larger than the current maximum, then shift it up, and then extract it using extract max. Find the current maximum using getMax.
- **GetMax:** The max value is stored at the root of the tree. To getmax, just return the value at the root of the tree.

Array Representation of Binary Heap

Since the heap is maintained in form of a **complete binary tree**, because of this fact the heap can be represented in the form of an array. To keep the tree complete and shallow, while inserting a new element insert it in the leftmost vacant position in the last level i.e., at the end of our array. Similarly, while extracting maximum replace the root with the last leaf at the last level i.e., the last element of the array. Below is the illustration of the same:





Below is the program to implement Priority Queue using Binary Heap:

C++

```
// C++ code to implement priority-queue
// using array implementation of
// binary heap

#include <bits/stdc++.h>
using namespace std;

int H[50];
int size = -1;

// Function to return the index of the
// parent node of a given node
int parent(int i)
{

    return (i - 1) / 2;
}

// Function to return the index of the
// left child of the given node
int leftChild(int i)
{

    return ((2 * i) + 1);
}

// Function to return the index of the
// right child of the given node
int rightChild(int i)
{

    return ((2 * i) + 2);
}

// Function to shift up the node in order
// to maintain the heap property
void shiftUp(int i)
{
    while (i > 0 && H[parent(i)] < H[i]) {

        // Swap parent and current node
        swap(H[parent(i)], H[i]);

        // Update i to parent of i
        i = parent(i);
    }
}

// Function to shift down the node in
// order to maintain the heap property
void shiftDown(int i)
{
    int maxIndex = i;
```

```
        // Left Child
        int l = leftChild(i);

        if (l <= size && H[l] > H[maxIndex]) {
            maxIndex = l;
        }

        // Right Child
        int r = rightChild(i);

        if (r <= size && H[r] > H[maxIndex]) {
            maxIndex = r;
        }

        // If i not same as maxIndex
        if (i != maxIndex) {
            swap(H[i], H[maxIndex]);
            shiftDown(maxIndex);
        }
    }

    // Function to insert a new element
    // in the Binary Heap
    void insert(int p)
    {
        size = size + 1;
        H[size] = p;

        // Shift Up to maintain heap property
        shiftUp(size);
    }

    // Function to extract the element with
    // maximum priority
    int extractMax()
    {
        int result = H[0];

        // Replace the value at the root
        // with the last leaf
        H[0] = H[size];
        size = size - 1;

        // Shift down the replaced element
        // to maintain the heap property
        shiftDown(0);
        return result;
    }

    // Function to change the priority
    // of an element
    void changePriority(int i, int p)
    {
        int oldp = H[i];
        H[i] = p;
```

```
        if (p > oldp) {
            shiftUp(i);
        }
        else {
            shiftDown(i);
        }
    }

    // Function to get value of the current
    // maximum element
    int getMax()
    {

        return H[0];
    }

    // Function to remove the element
    // located at given index
    void remove(int i)
    {
        H[i] = getMax() + 1;

        // Shift the node to the root
        // of the heap
        shiftUp(i);

        // Extract the node
        extractMax();
    }

    // Driver Code
    int main()
    {

        /*         45
                 /      \
               31        14
             /  \      /  \
           13   20   7    11
         /  \
        12   7
        Create a priority queue shown in
        example in a binary max heap form.
        Queue will be represented in the
        form of array as:
        45 31 14 13 20 7 11 12 7 */

        // Insert the element to the
        // priority queue
        insert(45);
        insert(20);
        insert(14);
        insert(12);
        insert(31);
        insert(7);
        insert(11);
        insert(13);
```

```
    insert(7);

    int i = 0;

    // Priority queue before extracting max
    cout << "Priority Queue : ";
    while (i <= size) {
        cout << H[i] << " ";
        i++;
    }

    cout << "\n";

    // Node with maximum priority
    cout << "Node with maximum priority : "
         << extractMax() << "\n";

    // Priority queue after extracting max
    cout << "Priority queue after "
         << "extracting maximum : ";
    int j = 0;
    while (j <= size) {
        cout << H[j] << " ";
        j++;
    }

    cout << "\n";

    // Change the priority of element
    // present at index 2 to 49
    changePriority(2, 49);
    cout << "Priority queue after "
         << "priority change : ";
    int k = 0;
    while (k <= size) {
        cout << H[k] << " ";
        k++;
    }

    cout << "\n";

    // Remove element at index 3
    remove(3);
    cout << "Priority queue after "
         << "removing the element : ";
    int l = 0;
    while (l <= size) {
        cout << H[l] << " ";
        l++;
    }
    return 0;
}
```

Java