Priority Queues

A Priority queue is an important abstract data type that supports insertion of a new item (*Insert*) and deletion of the smallest item (*DeleteMin*). These operations find extensive use in applications such as, the implementation of CPU schedulers in computer operating systems, sorting applications, selection of k^{th} largest or k^{th} smallest elements, and the efficient implementation of several graph algorithms.

Min-Heaps and Max-Heaps

A sequence of n elements $a_1, a_2, \ldots, a_n$ is called a min-heap if $a_i \le a_{2i}$ and $a_i \le a_{2i+1}$ for all values $1 \le i \le n/2$.

Likewise a sequence of elements $a_1, a_2, \ldots, a_n$ is called a max-heap if $a_i \ge a_{2i}$ and $a_i \ge a_{2i+1}$ for all values $1 \le i \le n/2$.

Heaps are a very useful in the efficient implementation of priority queues. A min-heap supports the *insert* and *deletemin* operations while a max-heap supports the *insert* and *deletemax* operations.

We deal with min-heaps. The discussion is similar for max-heaps.

Heaps and Complete Binary Trees

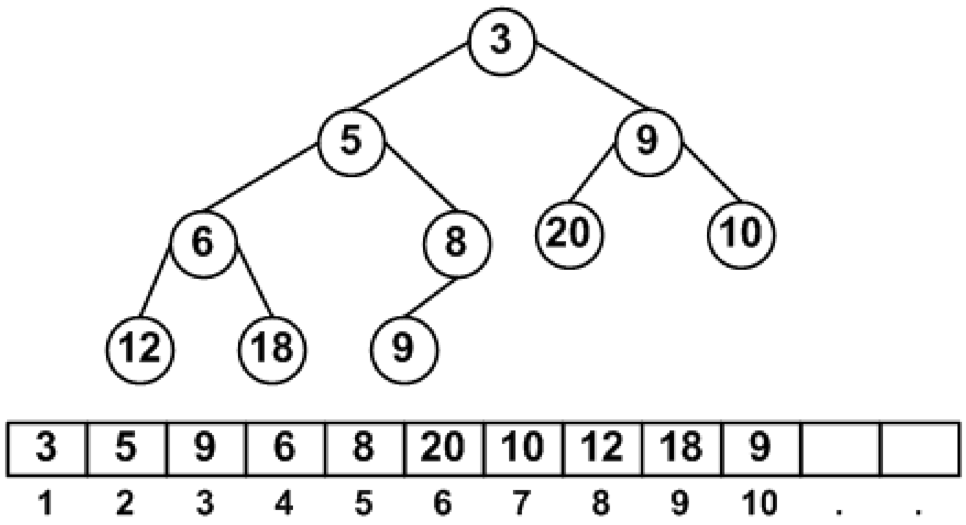
Suppose that we have a set or a multiset P of n elements such that any two elements in the set can be compared with the ≤ relation.

Now consider a complete binary tree having n nodes in which each node is associated with a unique element from P. Moreover, suppose that the element at every node is less than (or equal to) the element at its left child and the element at its right child. Such a tree is referred to as a *partially ordered complete binary tree*.

Recall that in a complete binary tree, for any node i, the left and right children, if they exist, are the nodes $2i$ and $2i + 1$ respectively. Moreover, the parent of any non-root node i is $[i/2]$.

Thus a partially ordered complete binary tree represents a heap; consider the elements of a heap stored in an array. If an element is at position i in the array, then the left child will be in position $2i$ and the right child will be in position $2i + 1$. By the same token, a non-root element at position i will have its parent at position $[i/2]$.

Figure 1 shows an example of a partially ordered complete binary tree and the corresponding min-heap represented as an array of elements.



3	5	9	6	8	20	10	12	18	9		
1	2	3	4	5	6	7	8	9	10	.	.

Figure 1: An example of a partially ordered complete binary tree and its array representation

Thus a heap of elements and a partially ordered complete binary tree are two ways of representing the same thing.

We have seen before that a complete binary tree with height k will have between 2^k and $2^{k+1} - 1$ elements. Therefore a complete binary tree with n elements will have height = $[\log_2 n]$.

Because of the partial ordering property, the minimum element will always be present at the root of the tree. Thus the finding the smallest element takes only constant time.

Inserting an element into a min-heap

To insert an element say x, into a partially ordered complete binary tree with n elements, we first add a new node in position $(n + 1)$ and then associate element x to that node. We then check if the partial ordering property is violated by associating x to node $(n + 1)$. If the partial ordering property is not violated, then we have found the correct position for x. Otherwise, let $i = (n + 1)$. The element x in position i is smaller than the element that is in position $[i/2]$. So we swap these two element values. We continue this process with i set equal to $[i/2]$, until the element x in position i is greater than or equal to the element in position $[i/2]$ or i is equal to 1, that is the element x is now associated with the root node.

Figure 2 illustrates inserting element 4 into a partially ordered complete binary tree.

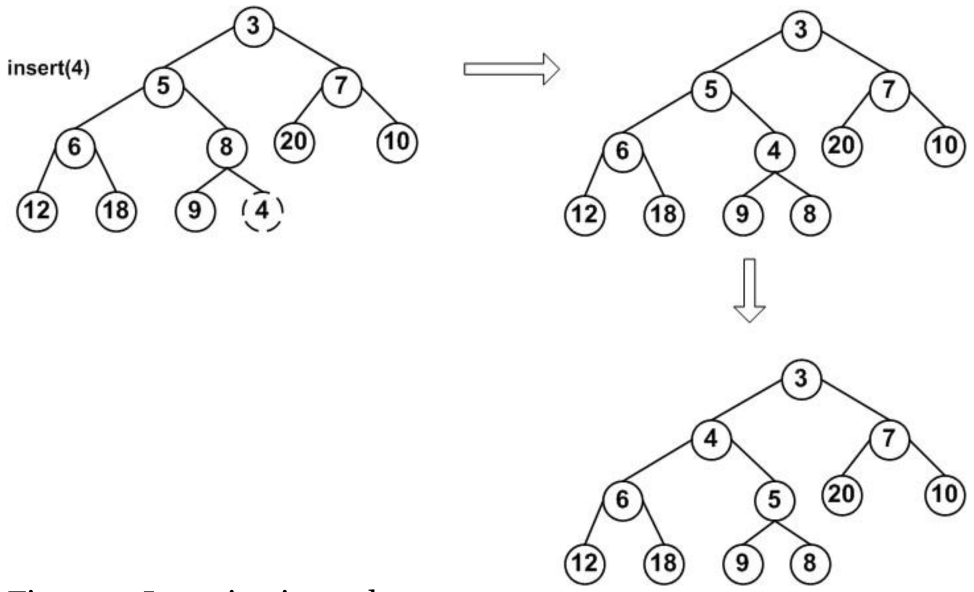


Figure 2: Insertion into a heap

Deleting the smallest element from a min-heap

When the minimum is deleted, a hole is created at the root level. Since the tree now has one less element and the tree is a complete binary tree, the element in the least position is to be relocated. This we first do by placing the last element in the hole created at the root. This will leave the heap property possibly violated at the root level. We now push-down the element associated at the root until the partial ordering property is restored. While pushing down the element, it is important to swap the element with the smallest of the elements among its two children. The deletion operation is illustrated in Figure 3. It is easy to see that the worst-case running time of deletemin is O (log n) where n is the number of elements in the heap.

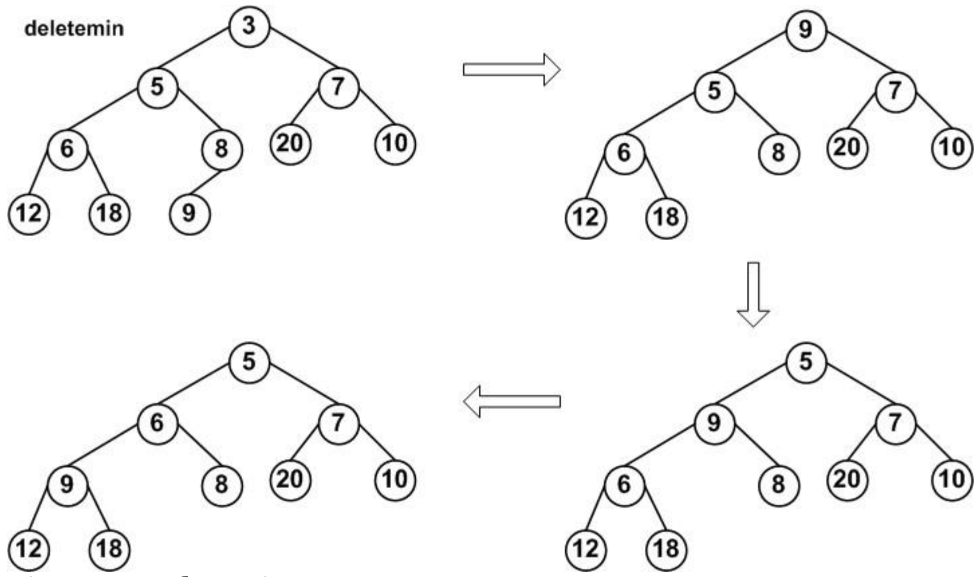


Figure 3: Deletemin

Implementation of Priority Queues

PriorityQueue class implements the max-heap using vectors.

The PriorityQueue class

A priority queue is a sequence container in which the item immediately available for use is the largest of those in the container, (also known as the highest priority item). The PriorityQueue class uses a container of type vector<T> for storing items of type T. If the item type T is a user defined type then T must contain the operations of copying, assignment and the comparison operator <.

1. Header file to be included in programs that use this class.

   #include "PriorityQueue.h"

2. PriorityQueue item access operations.

   bool IsEmpty() const
   Returns true if the priority queue is empty, false otherwise.

   int GetSize() const
   Returns the number of items currently stored in the priority queue.

3. PriorityQueue item insert and remove operations.

   void Insert(const T &newItem)
   Inserts the new item into the queue.

   ItemType DeleteMax()
   Removes the highest priority item of type ItemType from the queue and returns it.