Priority Queue

In a priority queue all the elements are assigned some priority. The order in which the elements could be deleted or processed from the priority queue depends upon this priority. The element with the highest priority is accessed then the element with the second priority and so on. The elements with the same priority are accessed in the order in which they were added to the queue.

Operating system uses priority queue for scheduling jobs where jobs with higher priority are processed first. Priority queue is also used in time-sharing systems where the programs with high priority are processed first and a standard queue is formed for the programs with the same priority. A priority queue can be implemented using a heap as we would see in a later section.

Heap

Heap is a complete binary tree. There are two types of heaps. If the value present at any node is greater than all its children then such a tree is called as the **max-heap** or **descending heap**. In case of a min-heap or **ascending heap** the value present in any node is smaller than all its children. Figure 9-55 shows a descending heap.

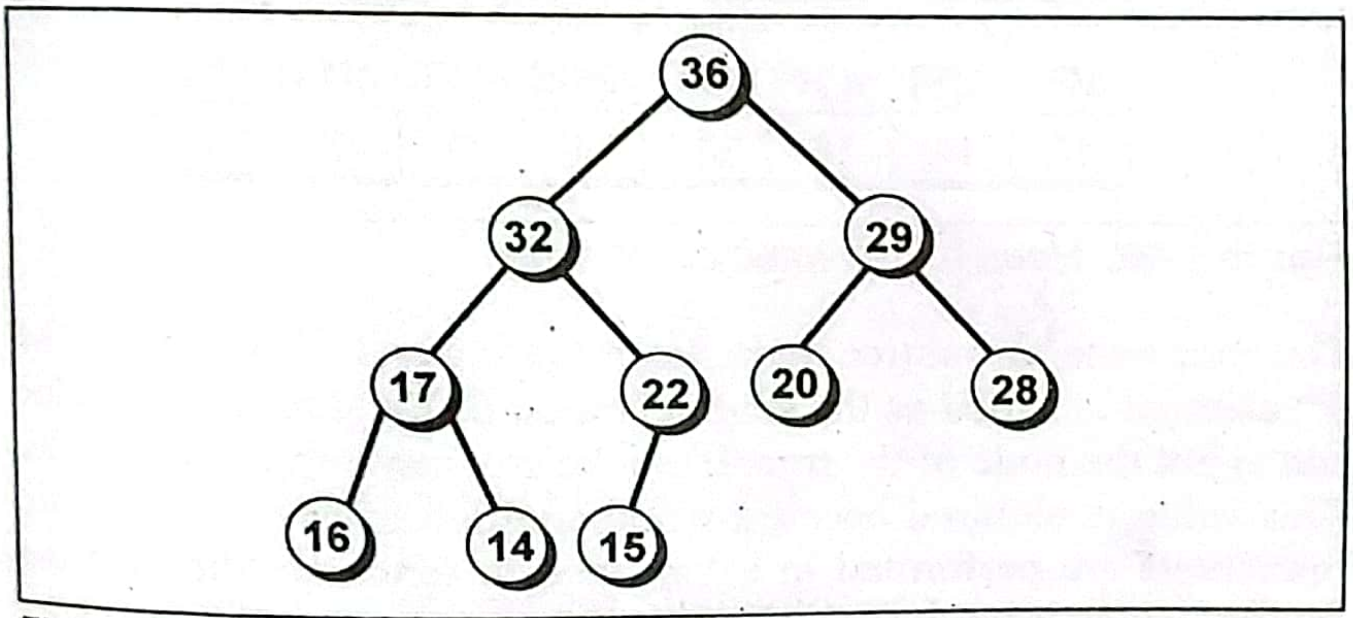

Figure 9-55. *Max-heap or descending heap.*

Priority Queue Represented As A Heap

We have seen how a binary tree can be represented by one-dimensional array. The nodes are numbered as **0** for root node, then from left to right at each level as **1**, **2**, etc. For an **i**th node its left and right child exist at **(2i + 1)**th and **(2i + 2)**th position respectively. Same is the case with heap, if the index of the root is considered as **1** then the left and right child of **i**th node are present at **(2i)**th and **(2i + 1)**th position respectively and the parent node is present at **(i/2)**th index in the array. Figure 9-56 shows an array **a** that represents the heap shown in Figure 9-55.

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
a	1000	36	32	29	17	22	20	28

a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]	a[15]
16	14	15	'\0'	'\0'	'\0'	'\0'	'\0'

Figure 9-56. *Heap represented as an array.*

The root node of the tree starts from the index **1** of the array. The **0**th element is called as the sentinel value that is a maximum value and is not the node of the tree. It can be any number, like say 1000. This value is required because while addition of new node certain operations are performed in a loop and to terminate the loop the sentinel value is used.

The operations that can be performed on a heap are insertion of a node, deletion of a node and replacement of a node. On performing these operations the tree may not satisfy the heap properties and hence must be re-constructed.

Insertion Of A Node In A Heap

To insert an element to the heap, the node is inserted after the last element in the array and is compared with its parent that is present at **(i/2)**th position. If it is found to be greater than its parent then they are exchanged. This procedure is repeated till the element is placed at its appropriate place. Let us understand this with the help of an example. Suppose a node that contains a value **24** is inserted in the tree that is shown in Figure 9-55.

Firstly, the value is inserted at index **11** in the array, as the tree has **10** nodes. Then **24** is compared with its parent **22** and since **24** is greater than **22** they are exchanged. This is shown in Figure 9-57.

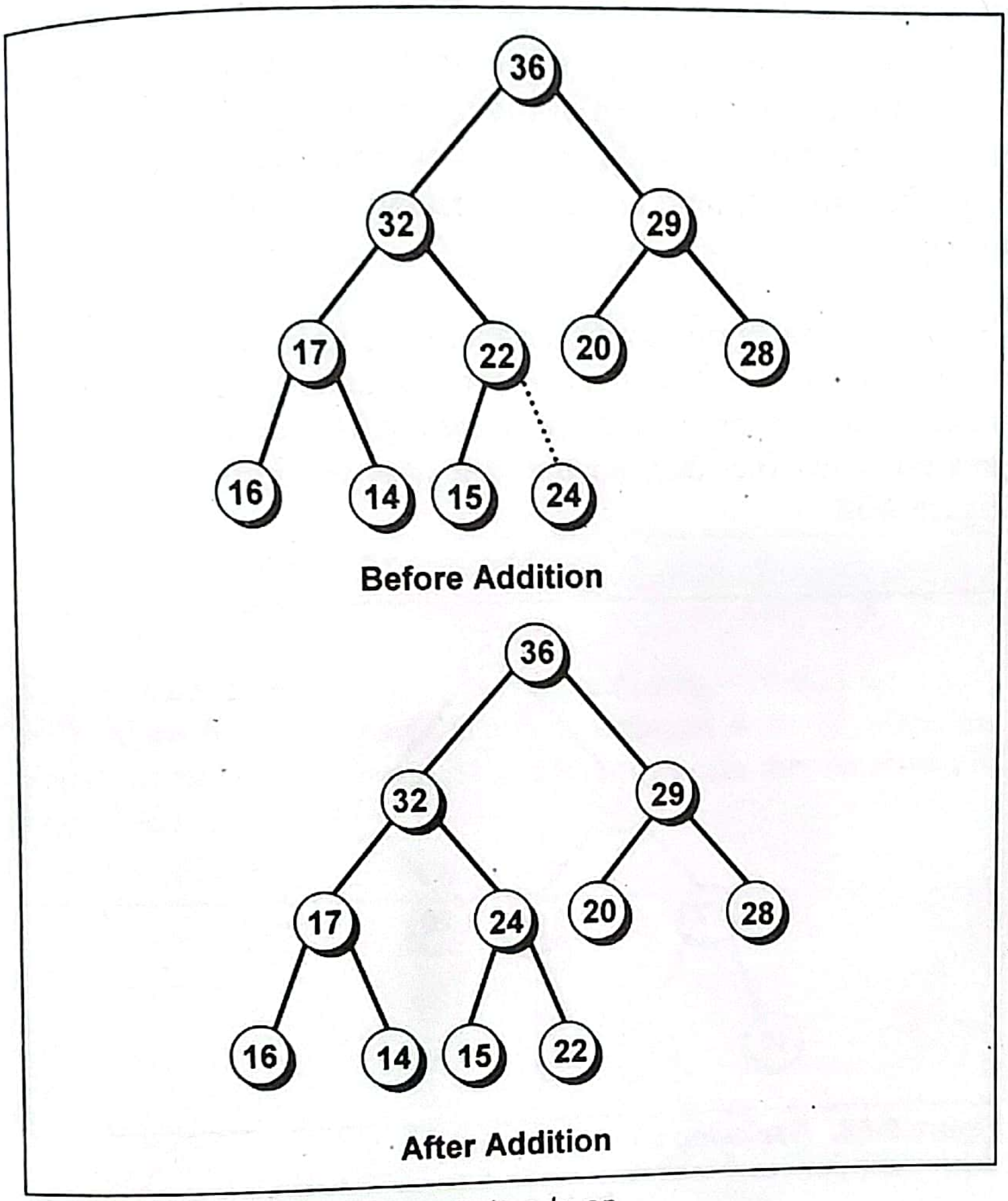

Figure 9-57. *Addition of a node to a heap.*

Now the element **24** is compared with its new parent **32** and since **24** is less than **32** they are not exchanged. The insertion process ends here and it is the final position of the node **24** in the tree.

Replacement Of A Node In A Heap

To replace the node of highest priority (i.e. the root node), the new value is inserted at the root of the node. Then it is compared with its children. If it is smaller, then it is exchanged with the child that is greater among them. Let us understand this with the help of an example.

Suppose the value present in the root node in Figure 9-55 (i.e. 36) is replaced with **5**. Then it is compared with its children 32 and 29. Since **5** is smaller than these values it is exchanged with the greatest value (i.e. **32**), so the tree now becomes as shown in Figure 9-58.

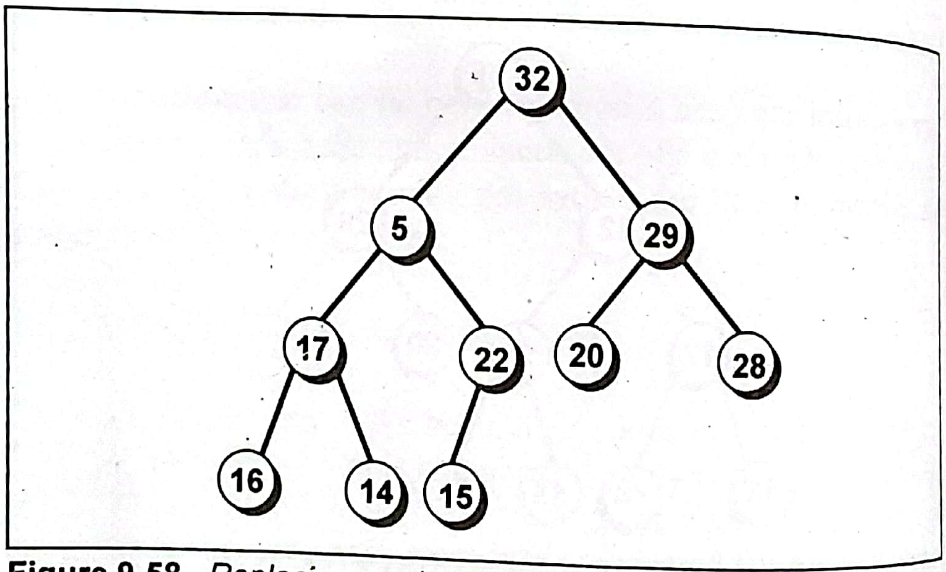

Figure 9-58. *Replacing a value in a heap.*

Now **5** is compared with its current children **17** and **22** and since **5** is less than these values it is exchanged with the greatest value (i.e. **22**), so the tree now becomes as shown in Figure 9-59.

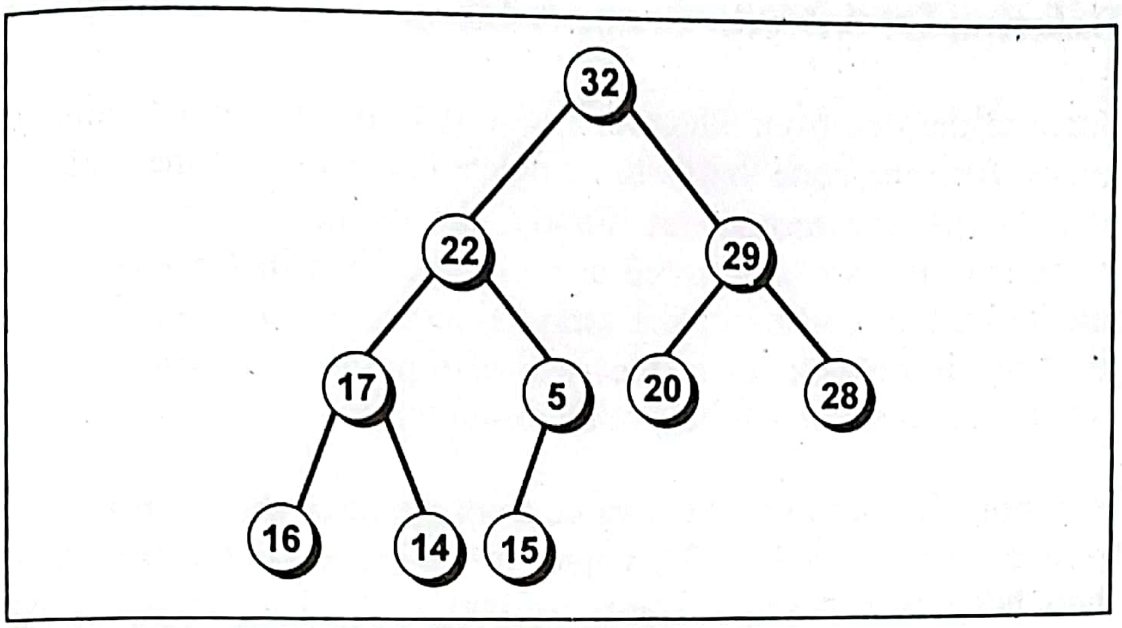

Figure 9-59. *Exchanging the values.*

Now **5** is compared with its current children—**15** and an empty node. Since **5** is less than **15** it is exchanged with **15**. Here the process of replacement ends. So the tree now becomes as shown in Figure 9-60.

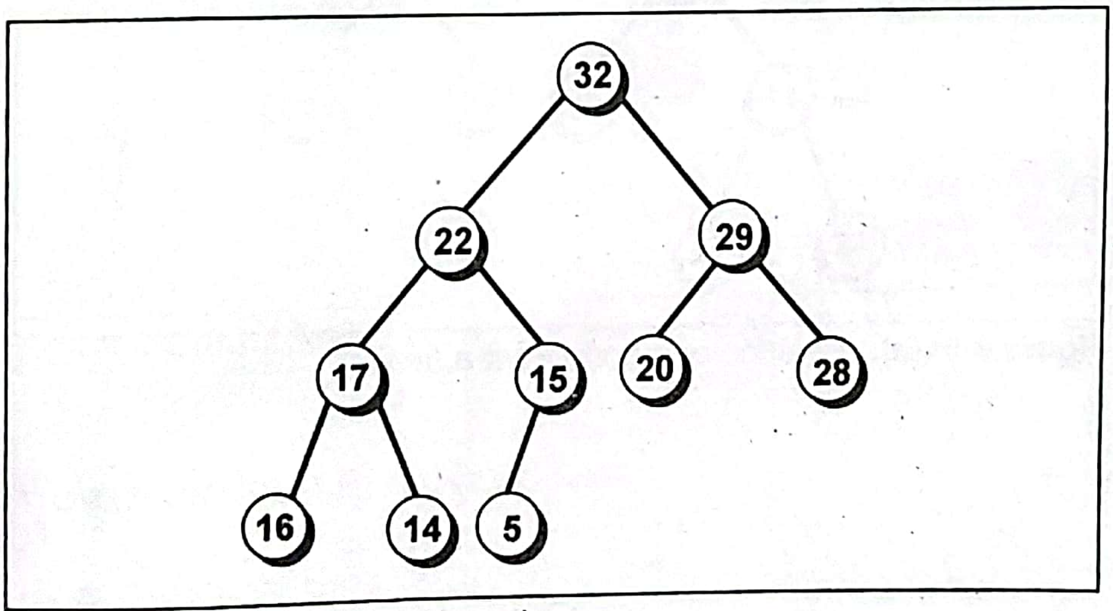

Figure 9-60. *Exchanging the values.*

Deletion Of A Node From A Heap

Suppose the tree from which the node is to be deleted contains n nodes. Also the node that is to be deleted is always of the highest priority (i.e. the root node). Firstly, the element present at the index **n** in the array is stored at the 1st position in the array and maximum index value of the array is decremented by one. Then the heap is restored as in the case of replace operation. Let us understand this with the help of an example.

Suppose we want to delete a value from the heap shown in Figure 9-55. For this the value **15** is copied at the index **1** in the array and then heap is restored. Figures 9-61(a) to 9-61(c) shows steps involved in restoring the heap.

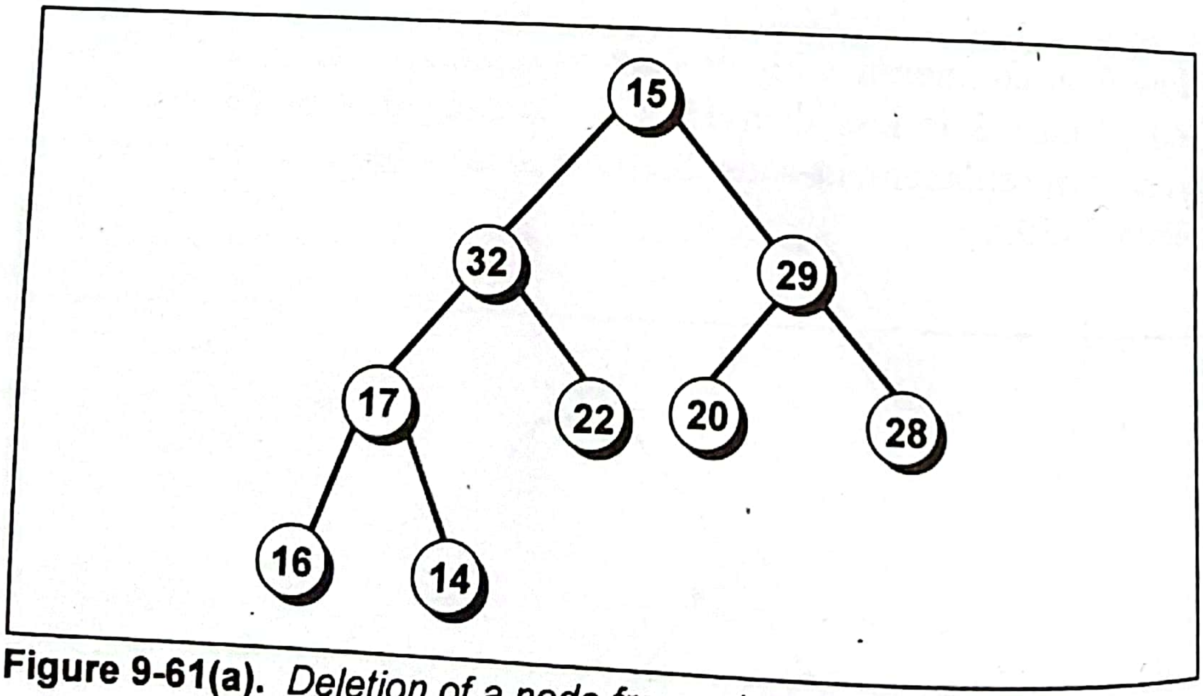

Figure 9-61(a). *Deletion of a node from a heap.*

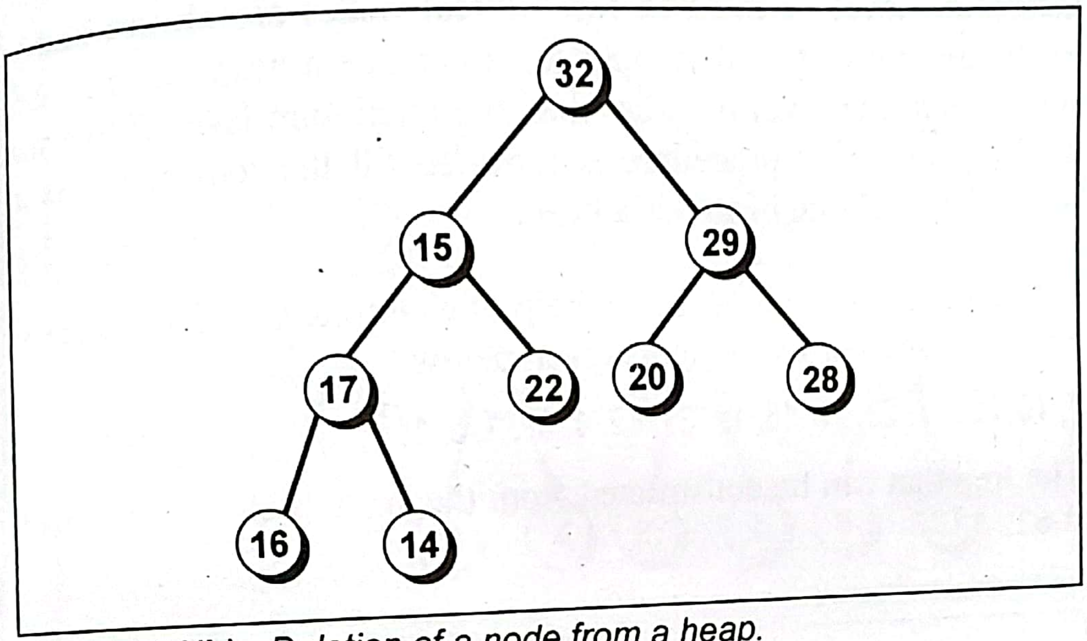

Figure 9-61(b). *Deletion of a node from a heap.*

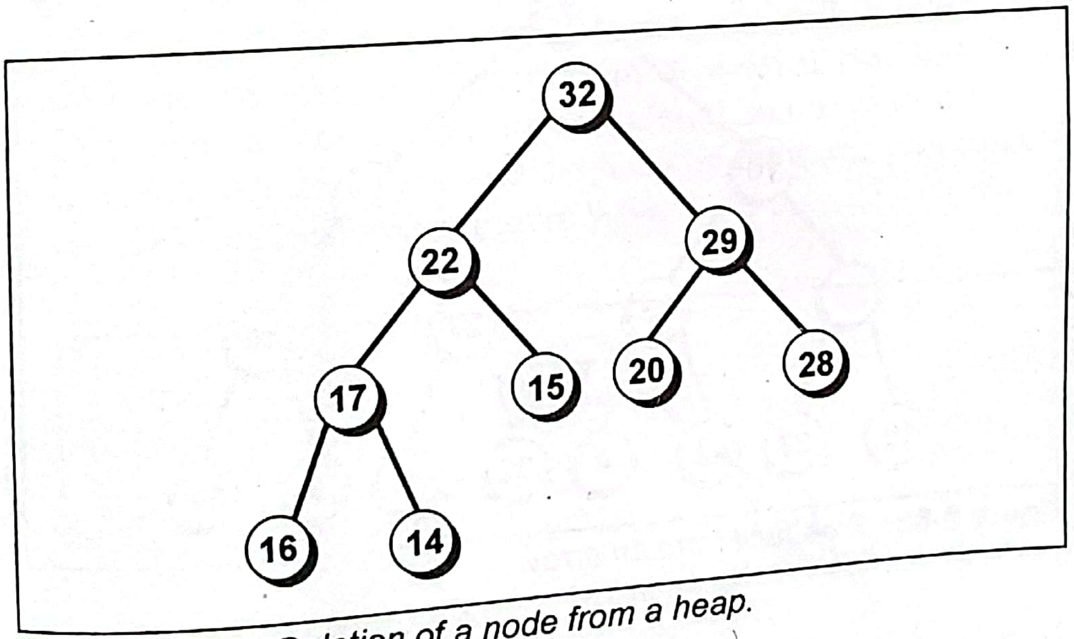

Figure 9-61(c). *Deletion of a node from a heap.*

Construction Of A Heap

Heap can be constructed from an array. To begin with a tree is constructed. If the tree doesn't satisfy heap properties then it is converted into a heap by adjusting nodes.

Adjustment of nodes starts from the level one less than the maximum level of the tree (as the leaf nodes are always heap). Each sub-tree of that particular level is made a heap. Then all the sub-trees at the level two less than the maximum level of tree are made heaps. This procedure is repeated till the root node. As a result the final tree becomes a heap.

Let us understand this with the help of example. Consider an array **arr** that contains **15** elements given below:

7, 10, 25, 17, 23, 27, 16, 19, 37, 42, 4, 33, 1, 5, 11

The tree that can be constructed from the array is shown in Figure 9-62.

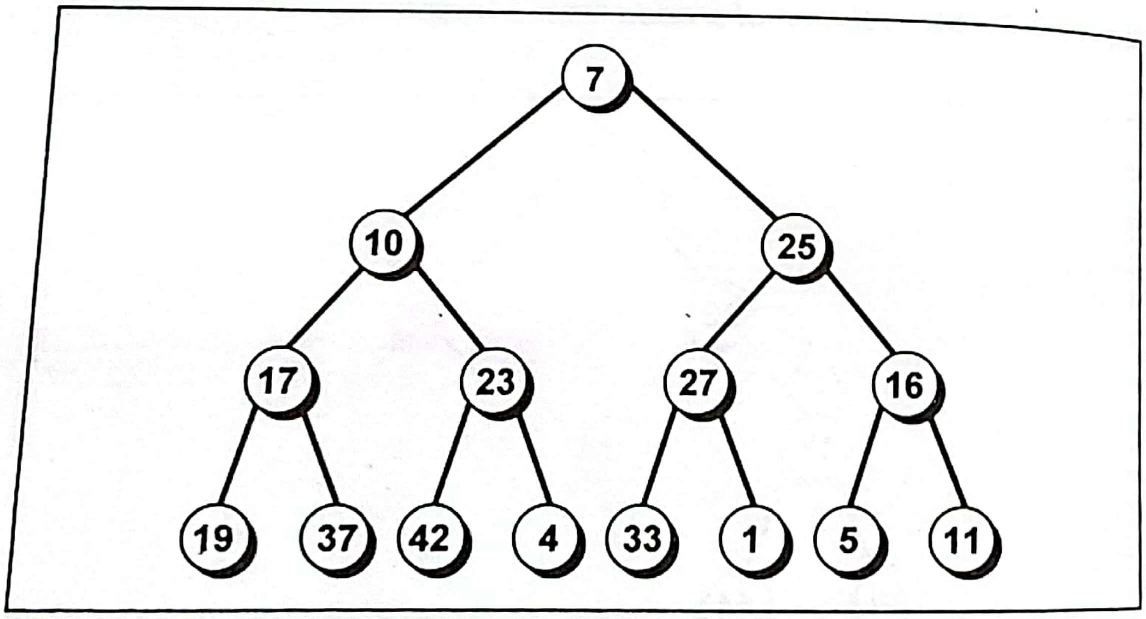

Figure 9-62. *Tree built from an array.*

To make it a heap initially the elements that are present at a level one less than the maximum level of the tree are taken into consideration. In our case, these are **17**, **23**, **27** and **16**. They are converted to heaps in the same way as replacing the node of a heap. The resultant tree is shown in Figure 9-63.

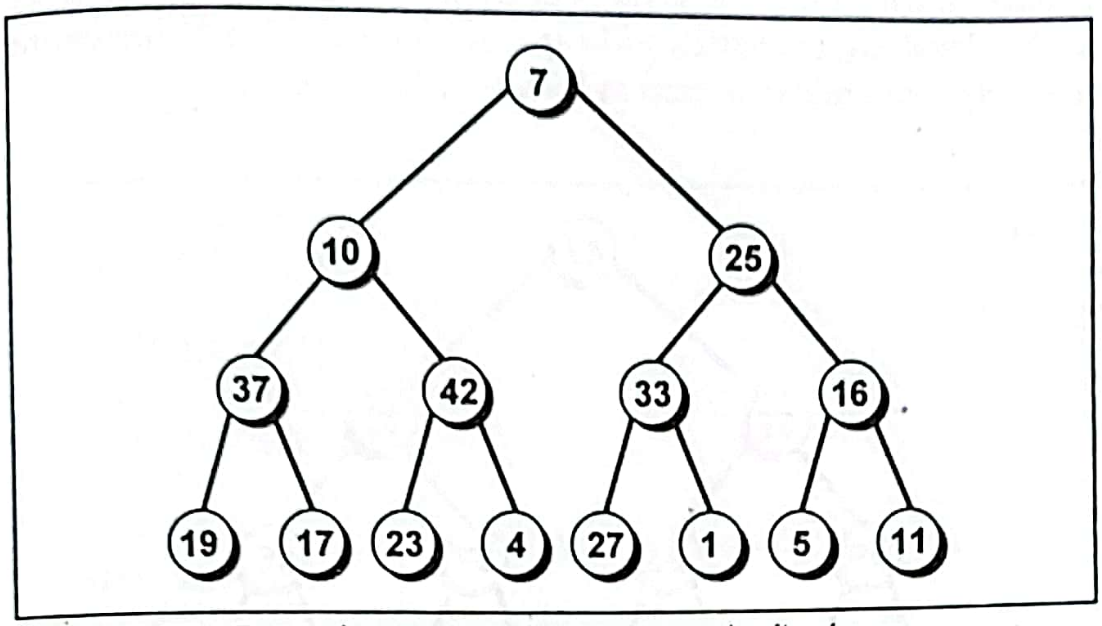

Figure 9-63. *Restoring nodes of a tree to make it a heap.*

Now the elements that are present at a level two less than the maximum level of the tree are considered. In our case, these are **10** and **25**. These are also made the heap by the same procedure. The resultant tree is shown in Figure 9-64.

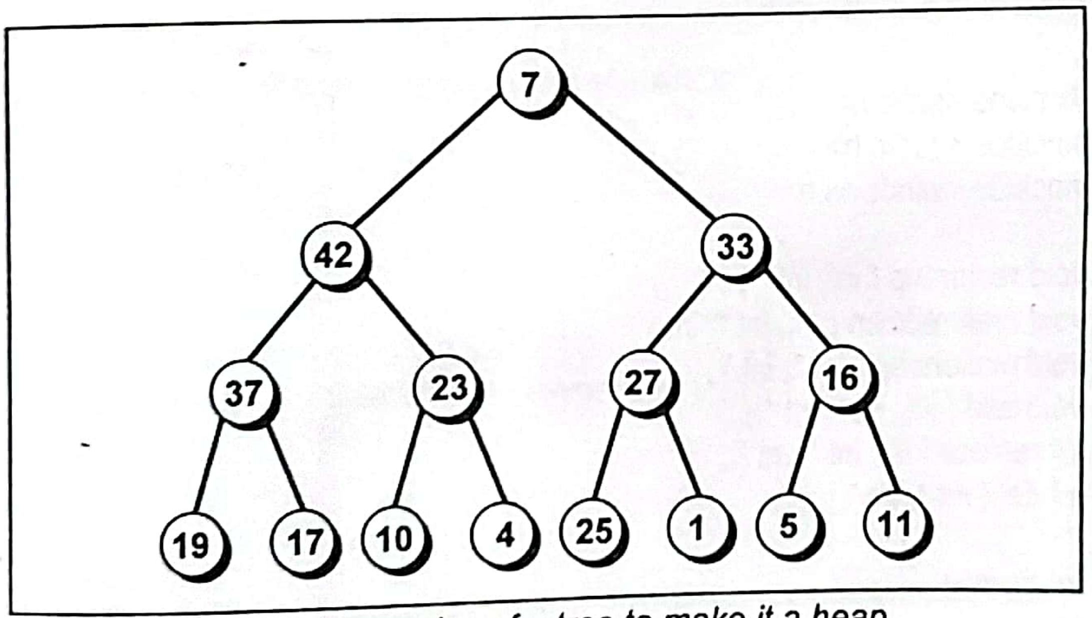

Figure 9-64. *Restoring nodes of a tree to make it a heap.*

Similarly, each time one level is decremented and all the sub-trees at that level are converted to heaps. As a result, finally the entire tree gets converted to a heap as shown in Figure 9-65.

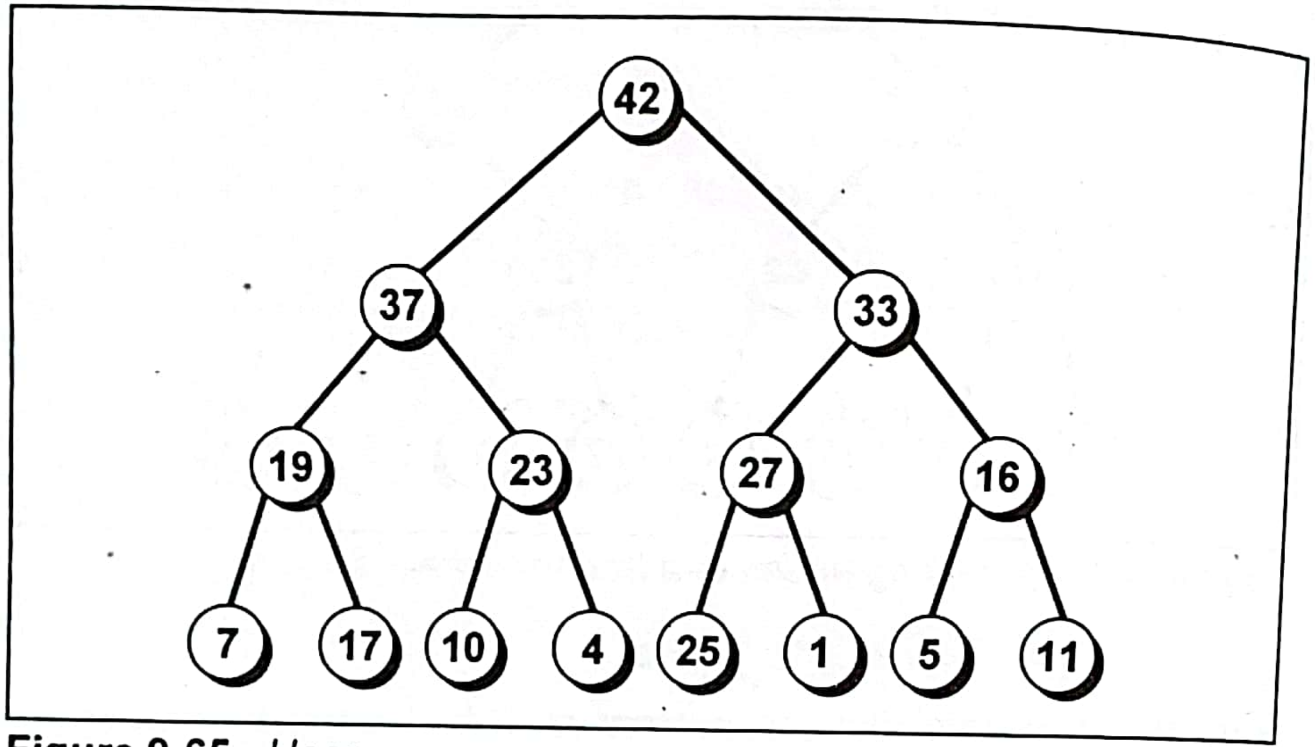

Figure 9-65. *Heap.*