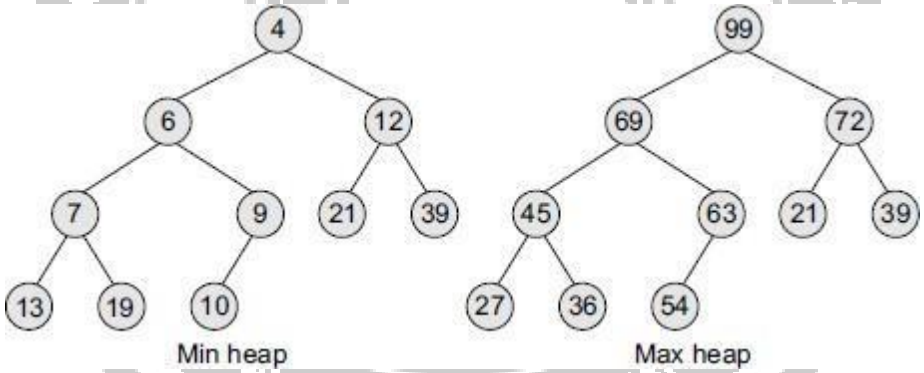
HEAPS

BINARY HEAPS

A binary heap is a complete binary tree in which every node satisfies the heap property which states that:

If B is a child of A, then key(A) ≥ key(B)

- This implies that elements at every node will be either greater than or equal to the element at its left and right child. Thus, the root node has the highest key value in the heap. Such a heap is commonly known as a max-heap.
- Alternatively, elements at every node will be either less than or equal to the element at its left and right child. Thus, the root has the lowest key value. Such a heap is called a min-heap.



The properties of binary heaps are given as follows:

Since a heap is defined as a complete binary tree, all its elements can be stored sequentially in an array. It follows the same rules as that of a complete binary tree. That is, if an element is at position i in the array, then its left child is stored at position 2i and its right child at position 2i+1. Conversely, an element at position ihas its parent stored at position i/2.

- Being a complete binary tree, all the levels of the tree except the last level are completely filled.
- The height of a binary tree is given as log2n, where n is the number of elements.
- Heaps (also known as partially ordered trees) are a very popular data structure for implementing priority queues.

OPERATIONS:

1.Insertion

2.Deletion

Inserting a New Element in a Binary Heap

Consider a max heap H with n elements. Inserting a new value into the heap is done in the following two steps:

1. Add the new value at the bottom of H in such a way that H is still a complete binary tree but not necessarily a heap.

2. Let the new value rise to its appropriate place in H so that H now becomes a heap as well. To do this, compare the new value with its parent to check if they are in the correct order. If they are, then the procedure halts, else the new value and its parent's value are

swapped and Step 2 is repeated.

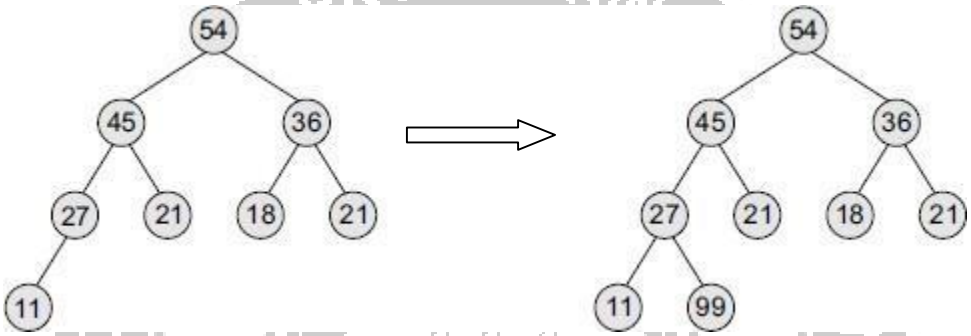
```
Step 1: [Add the new value and set its POS]
        SET N = N + 1, POS = N
Step 2: SET HEAP[N] = VAL
Step 3: [Find appropriate location of VAL]
        Repeat Steps 4 and 5 while POS > 1
Step 4:     SET PAR = POS/2
Step 5:     IF HEAP[POS] <= HEAP[PAR],
            then Goto Step 6.
            ELSE
                  SWAP HEAP[POS], HEAP[PAR]
                  POS = PAR
            [END OF IF]
      [END OF LOOP]
Step 6: RETURN
```

Algorithm to insert an element in a max heap

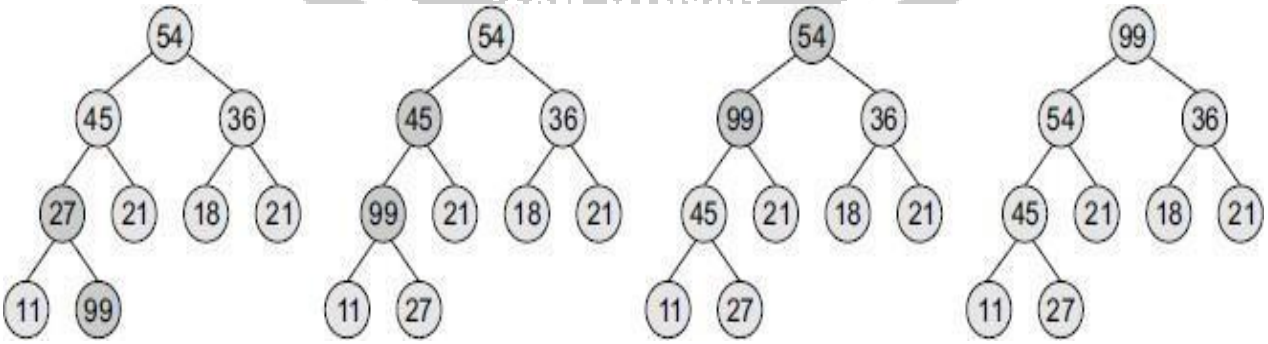
Example 1 Consider the max heap given in Fig. and insert 99 in it.

Solution

The first step says that insert the element in the heap so that the heap is a complete binary tree. So, insert the new value as the right child of node 27 in the heap. This is illustrated in Fig. Now, as per the second step, let the new value rise to its appropriate place in H so that H becomes a heap as well

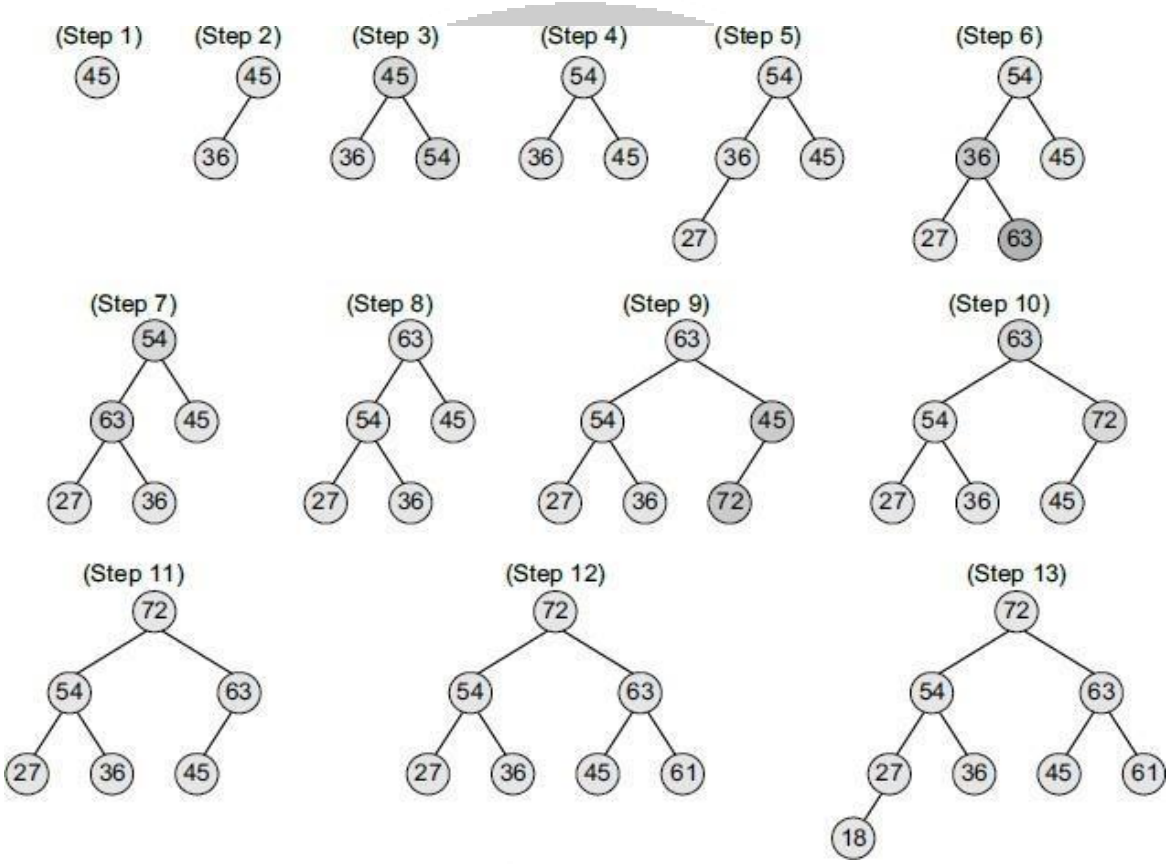


Compare 99 with its parent node value. If it is less than its parent's value, then the new node is in its appropriate place and H is a heap. If the new value is greater than that of its parent's node, then swap the two values. Repeat the whole process until H becomes a heap. This is illustrated in Fig.



Example 2 Build a max heap H from the given set of numbers: 45, 36, 54, 27, 63, 72, 61, and 18. Also draw the memory representation of the heap.

Solution



2. Deleting an Element from a Binary Heap

Consider a max heap H having n elements. An element is always deleted from the root of the heap. So, deleting an element from the heap is done in the following three steps:

1. Replace the root node's value with the last node's value so that H is still a complete binary tree but not necessarily a heap.

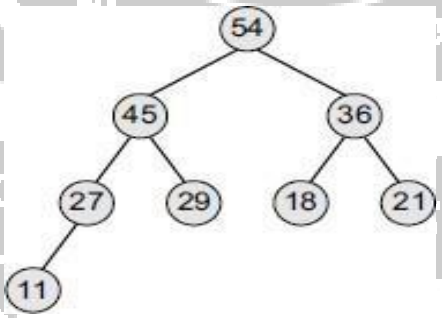
2. Delete the last node.

3. Sink down the new root node's value so that H satisfies the heap property. In this step, interchange the root node's value with its child node's value (whichever is largest among its children). Here, the value of root node = 54 and the value of the last node = 11. So, replace 54 with 11 and delete the last node.

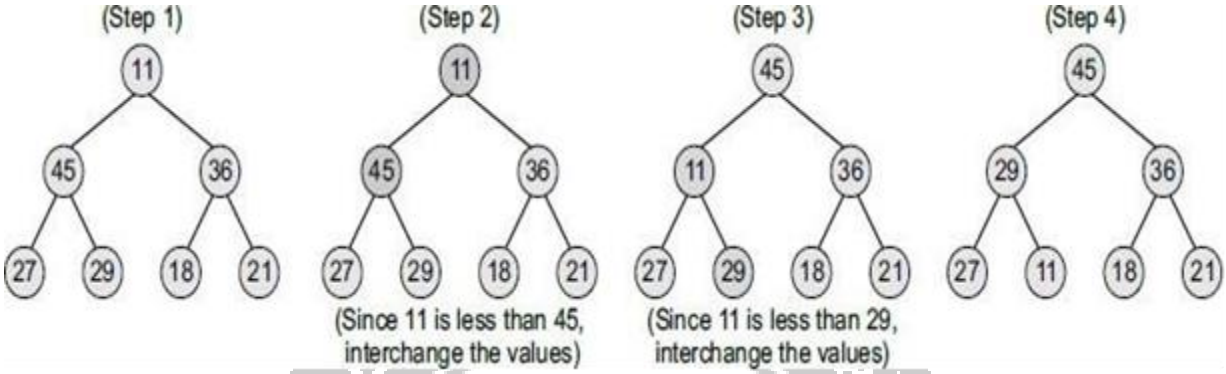
```
Step 1: [Remove the last node from the heap]
        SET LAST = HEAP[N], SET N = N - 1
Step 2: [Initialization]
        SET PTR = 1, LEFT = 2, RIGHT = 3
Step 3: SET HEAP[PTR] = LAST
Step 4: Repeat Steps 5 to 7 while LEFT <= N
Step 5: IF HEAP[PTR] >= HEAP[LEFT] AND
         HEAP[PTR] >= HEAP[RIGHT]
              Go to Step 8
        [END OF IF]
Step 6: IF HEAP[RIGHT] <= HEAP[LEFT]
              SWAP HEAP[PTR], HEAP[LEFT]
              SET PTR = LEFT
        ELSE
              SWAP HEAP[PTR], HEAP[RIGHT]
              SET PTR = RIGHT
        [END OF IF]
Step 7: SET LEFT = 2 * PTR and RIGHT = LEFT + 1
        [END OF LOOP]
Step 8: RETURN
```

Algorithm to delete the root element from a max heap

Example 1 Consider the max heap H shown in Fig. 12.8 and delete the root node's value.



Solution



Applications of Binary Heaps

Binary heaps are mainly applied for

1. Sorting an array using heapsort algorithm.

2. Implementing priority queues.

Binary Heap Implementation of Priority Queues

- A priority queue is similar to a queue in which an item is dequeued (or removed) from the front. However, unlike a regular queue, in a priority queue the logical order of elements is determined by their priority. While the higher priority elements are added at the front of the queue, elements with lower priority are added at the rear.
- Though we can easily implement priority queues using a linear array, but we should first consider the time required to insert an element in the array and then sort it. We need O(n) time to insert an element and at least O(n log n) time to sort the array. Therefore, a better way to implement a priority queue is by using a binary heap which allows both enqueue and dequeue of elements in O(log n) time.

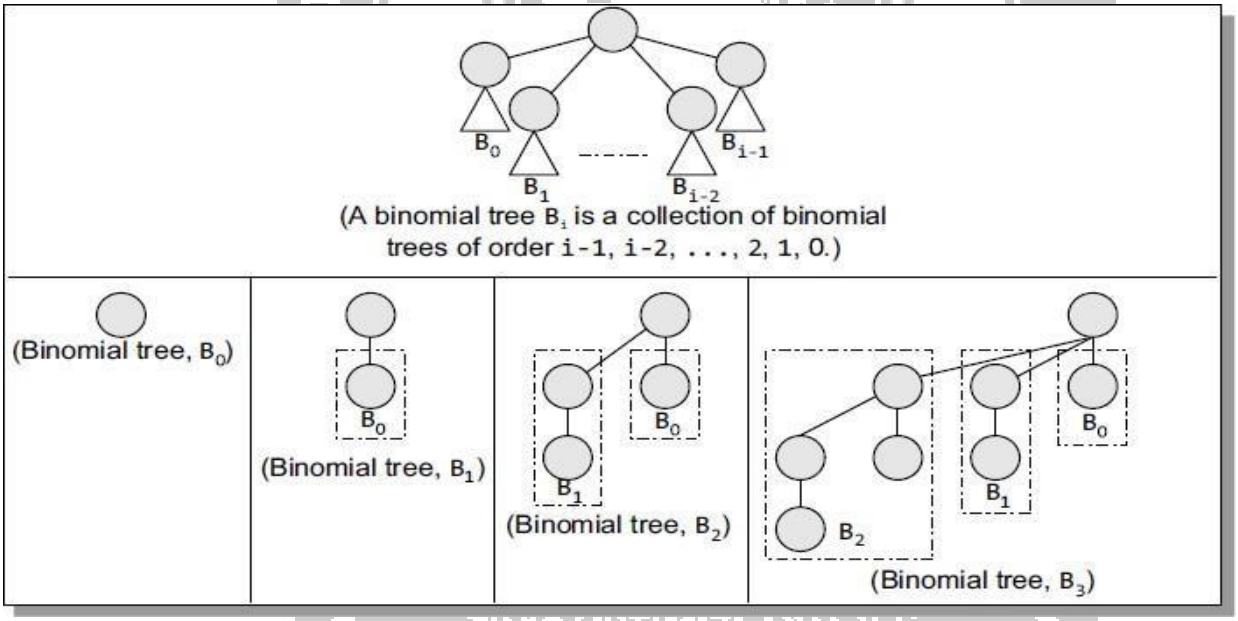
BINOMIAL HEAPS

A binomial heap H is a set of binomial trees that satisfy the binomial heap properties. First, let us discuss what a binomial tree is.

A binomial tree is an ordered tree that can be recursively defined as follows:

- A binomial tree of order 0 has a single node.
- A binomial tree of order i has a root node whose children are the root nodes of binomial trees of order i–1, i–2, ..., 2, 1, and 0.
- A binomial tree Bi has 2i nodes.
- The height of a binomial tree Bi is i.

Look at Fig. which shows a few binomial trees of different orders. We can construct a binomial tree Bi from two binomial trees of order Bi–1 by linking them together in such a way that the root of one is the leftmost child of the root of another.



A binomial heap H is a collection of binomial trees that satisfy the following properties:

- Every binomial tree in H satisfies the minimum heap property (i.e., the key of a node is either greater than or equal to the key of its parent).
- There can be one or zero binomial trees for each order including zero order. According to the first property, the root of a heap-ordered tree contains the smallest key in the tree. The second property, on the other hand, implies that a binomial heap H having N nodes contains at most log (N + 1) binomial trees.

3. FIBONACCI HEAPS

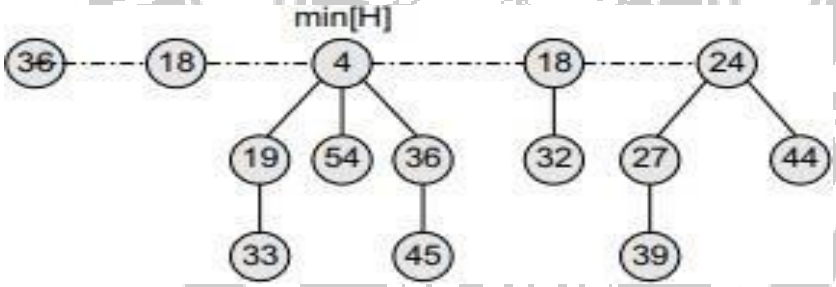
A Fibonacci heap is a collection of trees. It is loosely based on binomial heaps. If neither the decrease-value nor the delete operation is performed, each tree in the heap is like a binomial tree. Fibonacci heaps differ from binomial heaps as they have a more relaxed structure, allowing improved asymptotic time bounds.

1. Structure of Fibonacci Heaps

Although a Fibonacci heap is a collection of heap-ordered trees, the trees in a Fibonacci heap are not constrained to be binomial trees. That is, while the trees in a binomial heap are ordered, those within Fibonacci heaps are rooted but unordered.

Each node in the Fibonacci heap contains the following pointers:

- a pointer to its parent, and
- a pointer to any one of its children



Applications of Heaps

- Heap Implemented priority queues are used in Graph algorithms like Prim's Algorithm and Dijkstra's algorithm.
- Order statistics: The Heap data structure can be used to efficiently find the kth smallest (or largest) element in an array.