HEAP

Introduction

- A **heap** is a specialized **tree-based data structure** that satisfies the ***heap property*** which states that
  - **If B is a child of A, then key(A) ≥ key(B).**

    Or
  - **If B is a child of A, then key(A) ≤ key(B).**
- This implies that the **root node** has the **highest key value** in the heap. Such a heap is commonly known as a **max-heap**. (Alternatively, if the root has the **lowest key value**, then the resultant heap is called a **min-heap**.)

BINARY HEAP

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).**

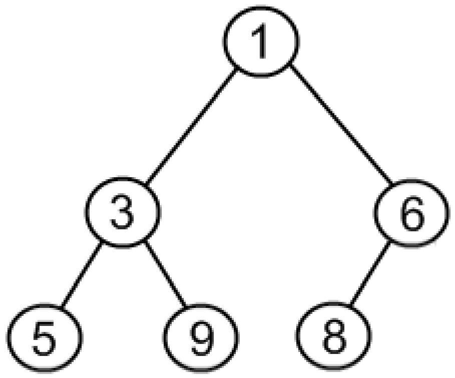
Or

- **If B is a child of A, then key(A) ≤ key(B).**

BINARY HEAP

MIN HEAP

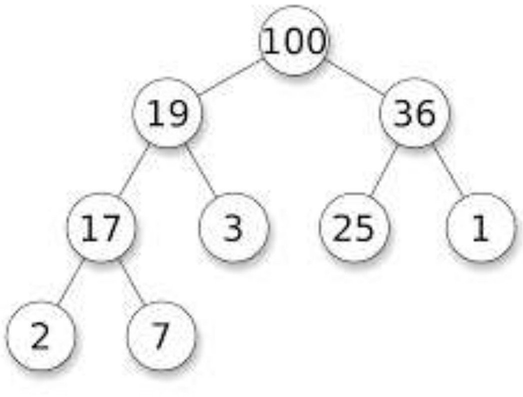
A binary heap with **elements at every node** being **either less than or equal** to the element at its **left and the right child**. Thus the root has the lowest key value. Such a heap is called **min-heap**



BINARY HEAP

MAX HEAP

A binary heap with **elements at every node** being either **greater than or equal** to the element at its **left and the right child**. Thus the root has the highest key value. Such a heap is called **max-heap**



Insertion in a Binary Heap

- Consider a max heap H having n elements. Inserting a new value into the heap is done in two major steps.

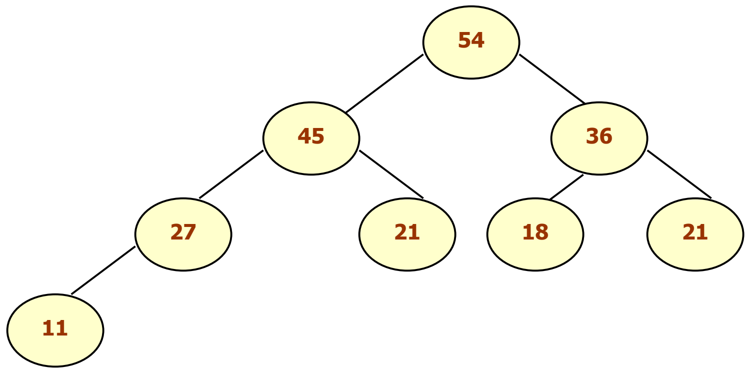
- Add the new value at the bottom of H in such a way that H is still a complete binary tree but not necessarily a heap.

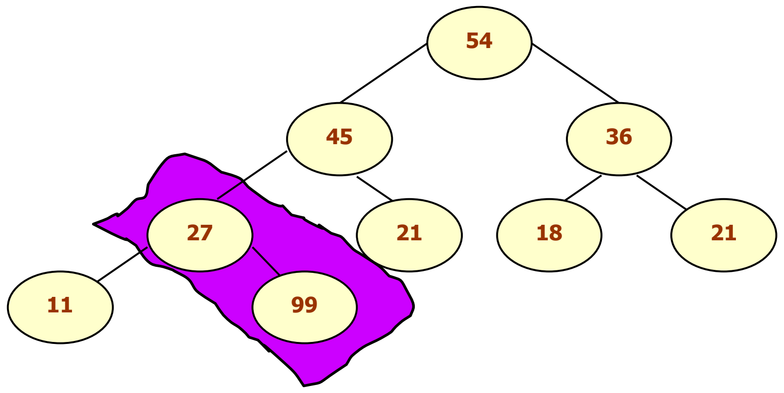
- Let the new value rise to its appropriate place in H so that H now becomes a heap as well.

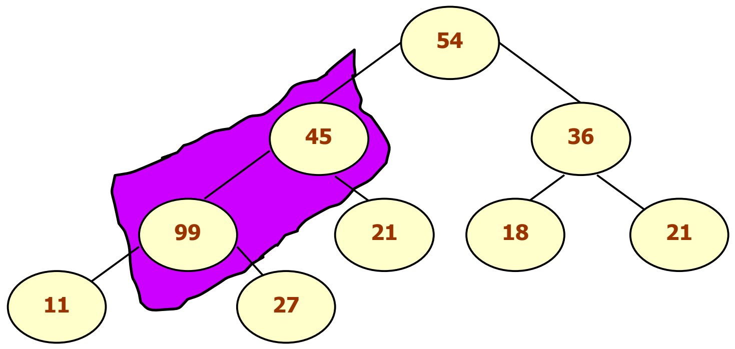
  - To do this, compare the new value with its parent; to check if they are in the correct order. If they are in the correct order, then the procedure halts else, the new value and its parent's value is swapped and step 2 is repeated.

Insertion in a Binary Heap (MAX HEAP)

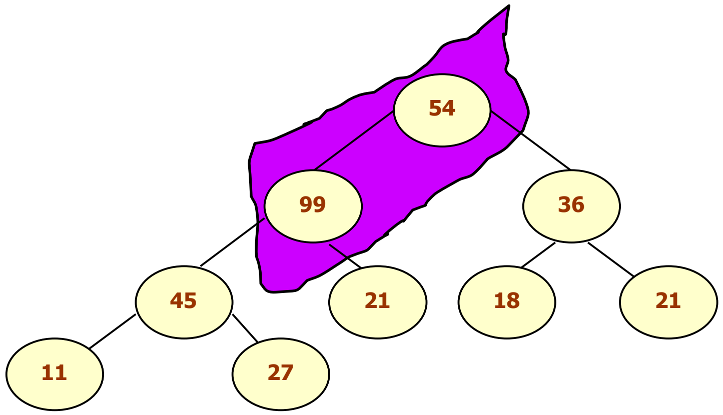
- Consider the Max Heap given below and insert 99 in it

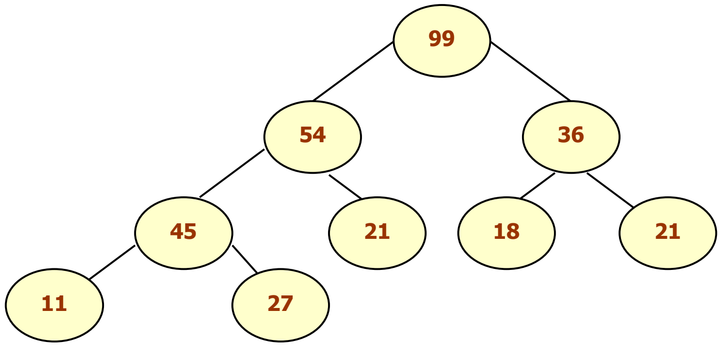




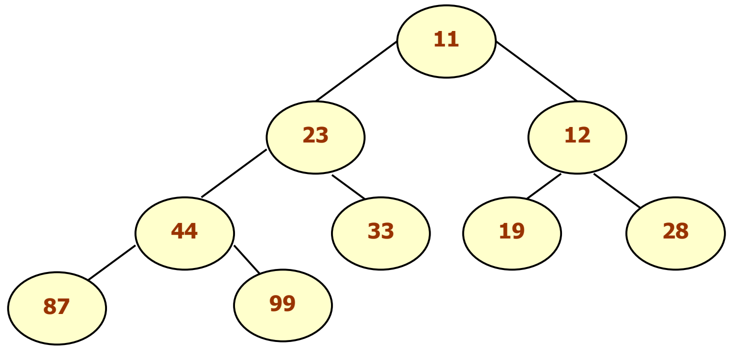


Insertion in a Binary Heap (MAX HEAP)

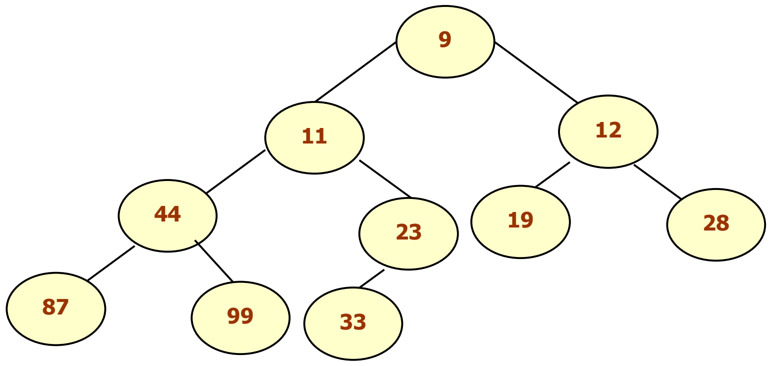




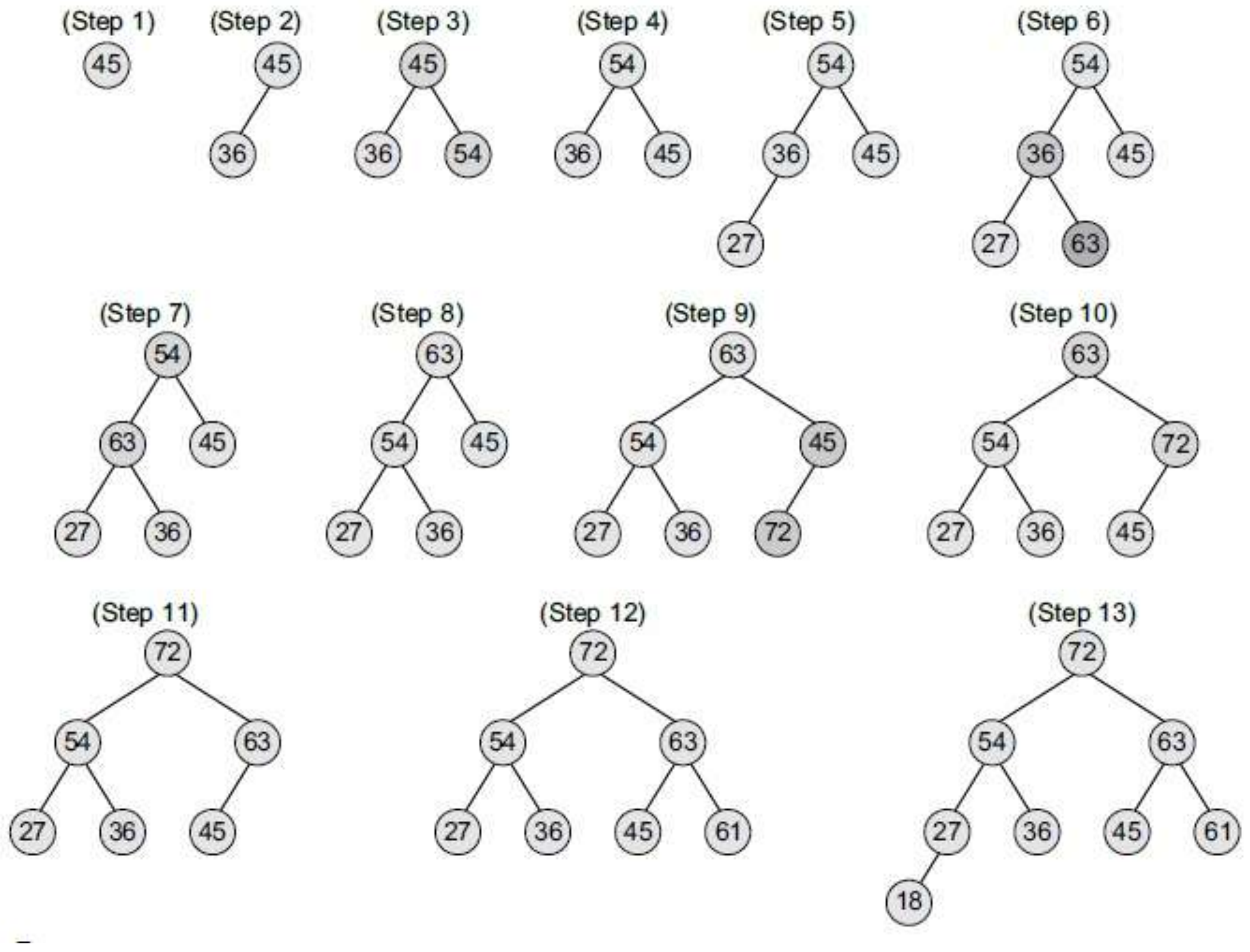
Insertion in a Binary Heap (MIN HEAP)



Insert 9
?



BOOK EXAMPLE
MAX HEAP INSERTION



Insertion in a Binary Heap

```
Algorithm to insert a value in the heap

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 < 0
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
```

Deletion in a Binary Heap
(DeleteMin / DeleteMax)

- 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 two major steps.

- 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.

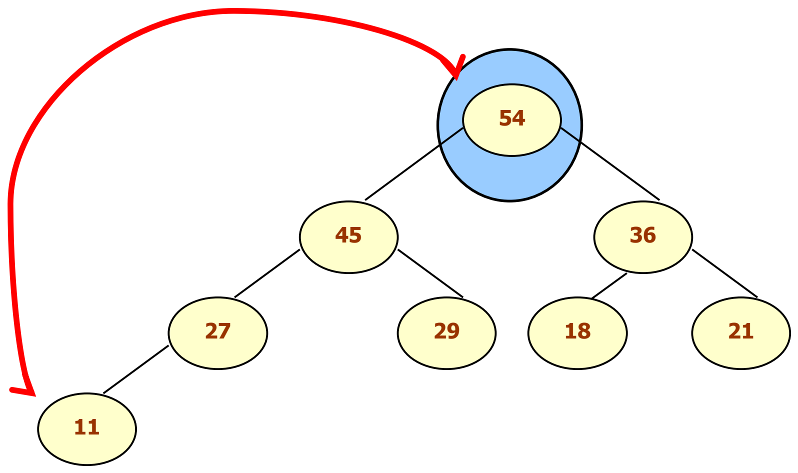
- Delete the last node.

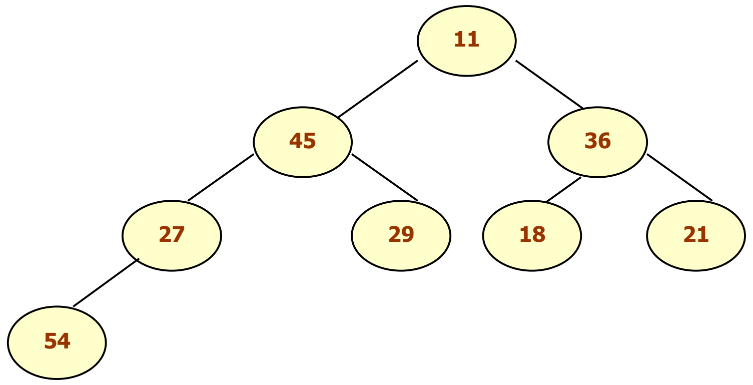
- Sink down the new root node’s value so that H satisfies the heap property. In this step, interchange the root node’s value with its child node’s value (whichever is largest among its children).

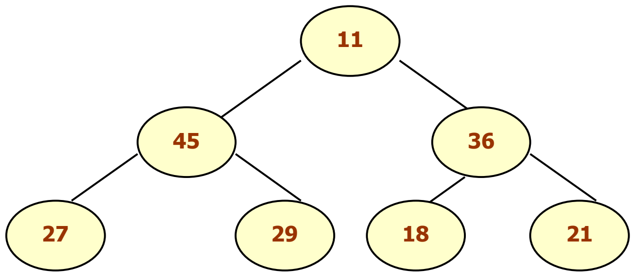
Deletion in a Binary Heap

Consider the Max Heap H given below and delete the root node's value.

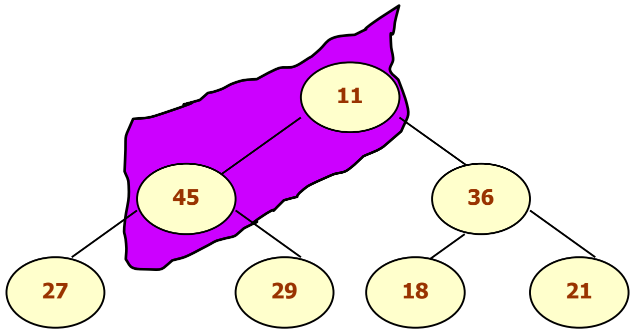
DELETE 54

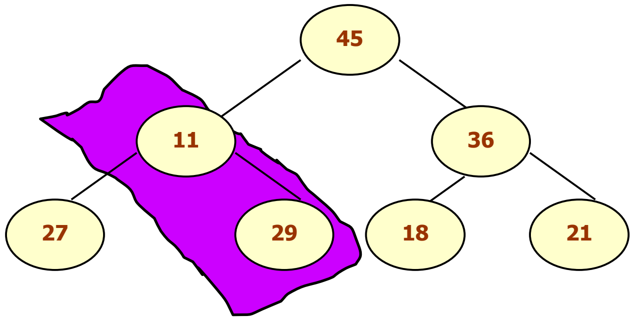


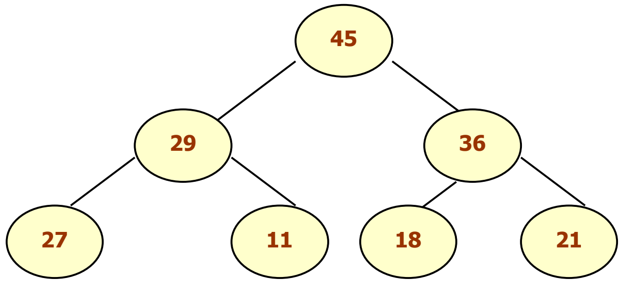




Deletion in a Binary Heap







Deletion in a Binary Heap

```
Algorithm to delete the root element from the heap- DELETE_HEAP(HEAP, N, VAL)

Step 1: [ Remove the last node from the heap]
           SET LAST = HEAP[N], SET N = N - 1
Step 2: [Initialization]
           SET PTR = 0, LEFT = 1, RIGHT = 2
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], then
                      Go to Step
           [END OF IF]
Step 6:    IF HEAP[RIGHT] <= HEAP[LEFT], then
                      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
```

Applications of Binary Heap

Binary heaps are mainly applied for

- Sorting an array using **heapsort** algorithm
- Implementing **priority queues**