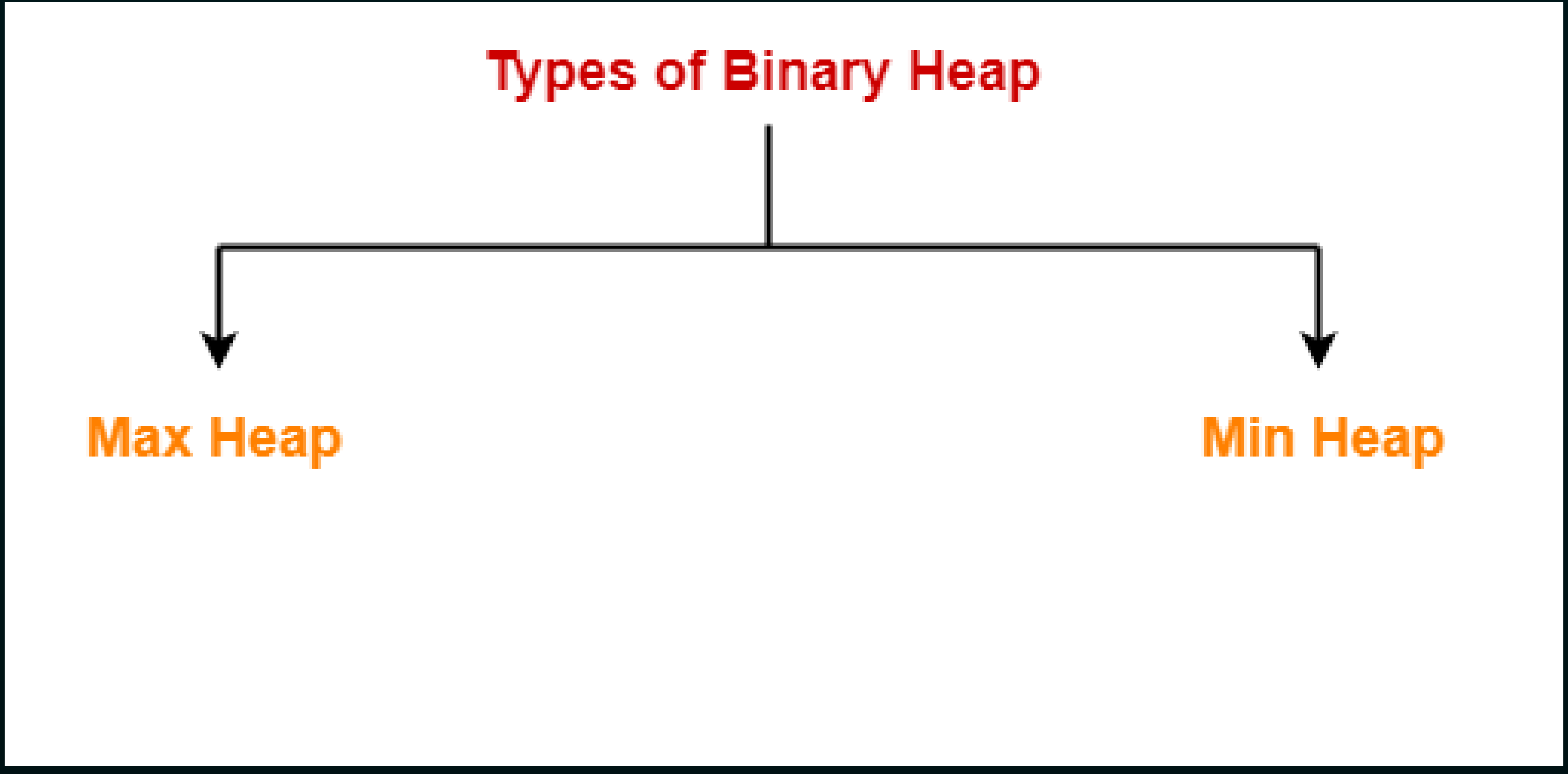
Introduction

- Heap
  - A specialized data structure and has special characteristics.
  - May be implemented using a n-ary tree.
- Binary Heap
  - Binary Tree with the following two properties
    1. Ordering Property: Elements in the heap are arranged in specific order.
       - Two types of heaps- min heap & max heap
    2. Structural Property: Binary heap is an almost complete binary tree
       - All its levels completely filled except possibly the last level
       - The last level is strictly filled from left to right

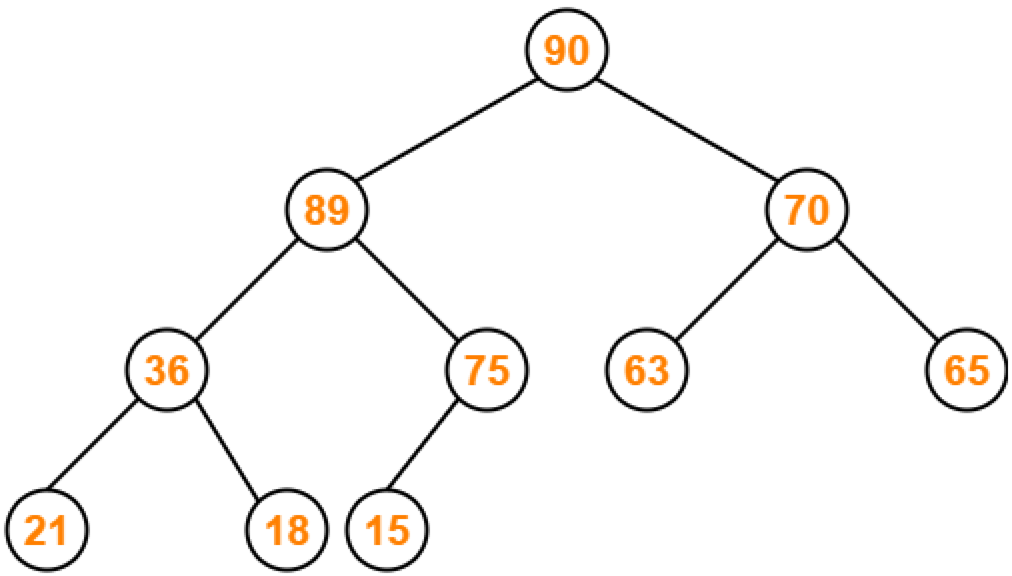
Cont…

- Types of Binary Heap
  - Depending on the arrangement of elements, a binary heap may be of following two types-



Cont…

- Max Heap
  - In max heap, every node contains greater or equal value element than its child nodes
  - Thus, root node contains the largest value element

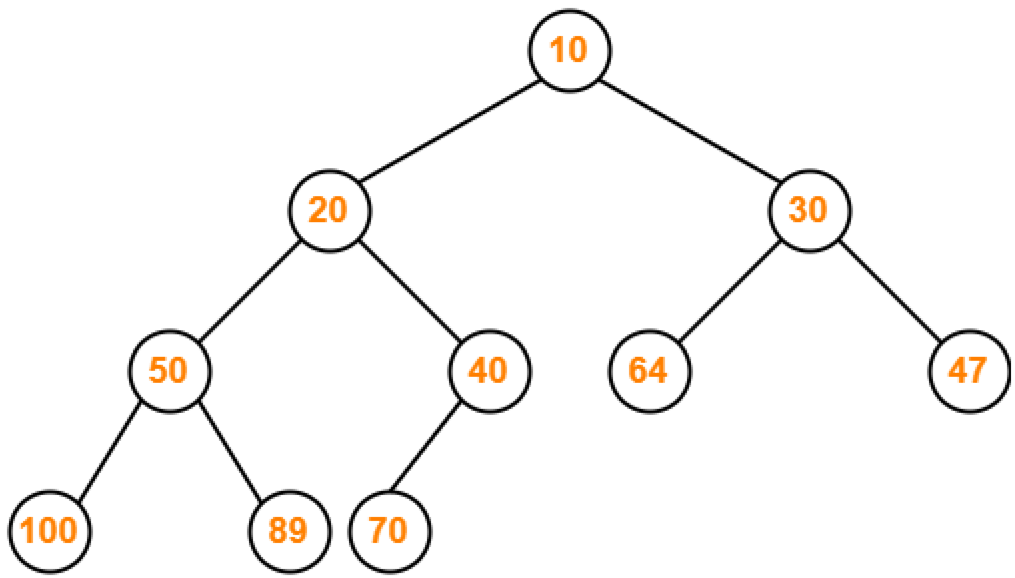


Max Heap Example

- Every node contains greater or equal value element than its child nodes.
- It is an almost complete binary tree with its last level strictly filled from left to right.

Cont…

- Min Heap
  - In min heap, every node contains lesser value element than its child nodes
  - Thus, root node contains the smallest value element.

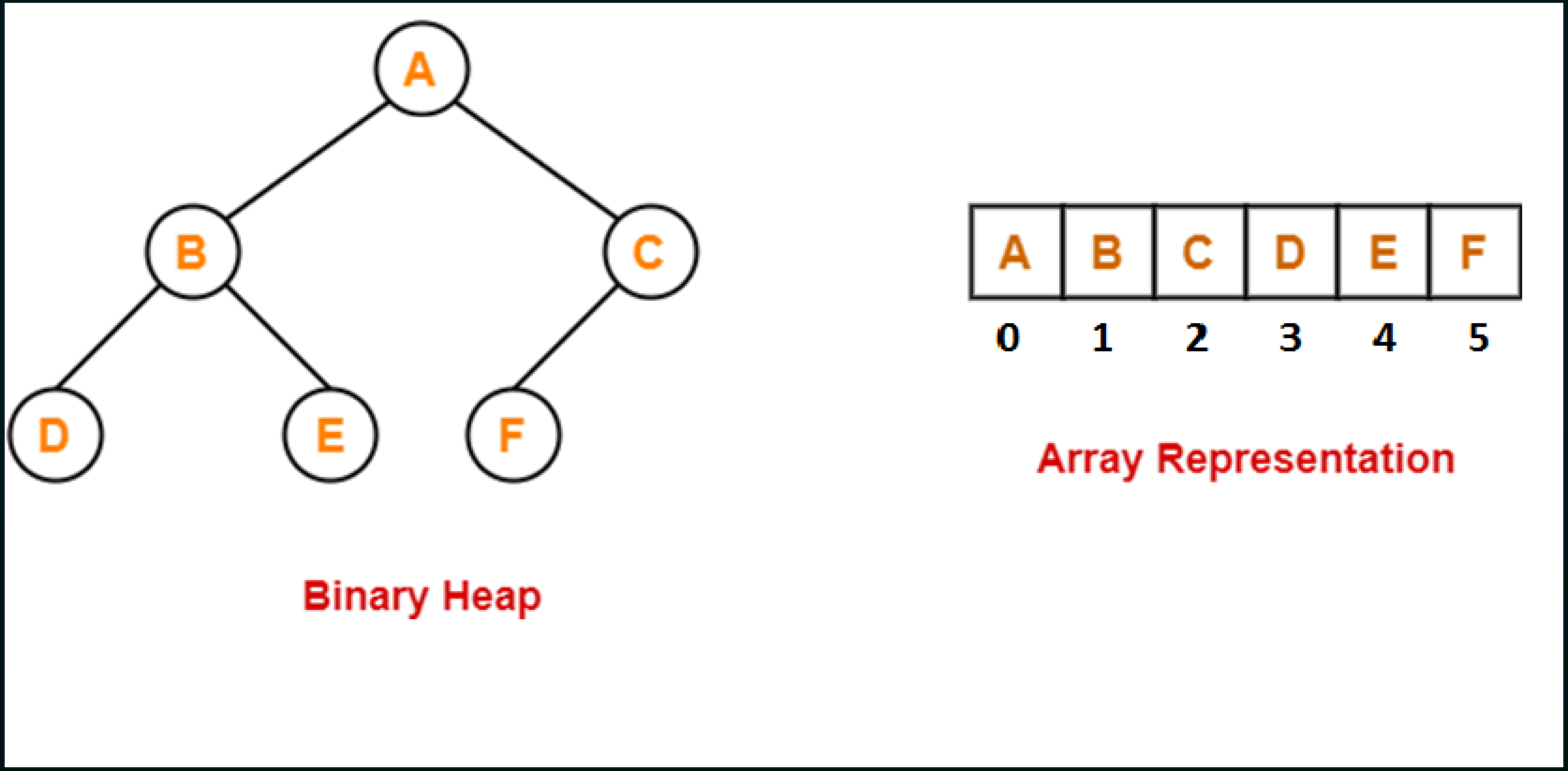


Min Heap Example

- Every node contains lesser value element than its child nodes.
- It is an almost complete binary tree with its last level strictly filled from left to right.

Array Representation of Binary Heap

- A binary heap is typically represented as an Array



Cont…

- For a node present at index ‘i’ of the array Arr[ ]
  - If indexing starts with 0,
    - Its parent node - present at array location = Arr[(i-1)/2]
    - Its left child node - present at array location = Arr[2i+1]
    - Its right child node - present at array location = Arr[2i+2]
  - If indexing starts with 1,
    - Its parent node - present at array location = Arr $[\lfloor i/2 \rfloor]$
    - Its left child node - present at array location = Arr [2i]
    - Its right child node - present at array location = Arr[2i+1]

Cont…

- Note-01:
  - Level order traversal technique may be used to achieve the array representation of a heap tree
  - Array representation of a heap never contains any empty indices in between
- Note-02:
  - Given an array representation of a binary heap,
    - If all the elements are in descending order, then heap is definitely a max heap
    - If all the elements are in ascending order, then heap is definitely a min heap

Cont…

- Consider a binary max-heap implemented using an array. Which one of the following array represents a binary max-heap?
  a) 25, 14, 16, 13, 10, 8, 12
  b) 25, 12, 16, 13, 10, 8, 14
  c) 25, 14, 12, 13, 10, 8, 16
  d) 25, 14, 13, 16, 10, 8, 12

Max Heap Construction

- Given an array of elements, the steps involved in constructing a max heap
  - Step-01: Convert the given array of elements into an almost complete binary tree
  - Step-02: Ensure that the tree is a max heap
    - Check that every non-leaf node contains a greater or equal value element than its child nodes
    - If there exists any node that does not satisfies the ordering property of max heap, swap the elements
    - Start checking from a non-leaf node with the highest index (bottom to top and right to left)

Example: Max Heap Construction

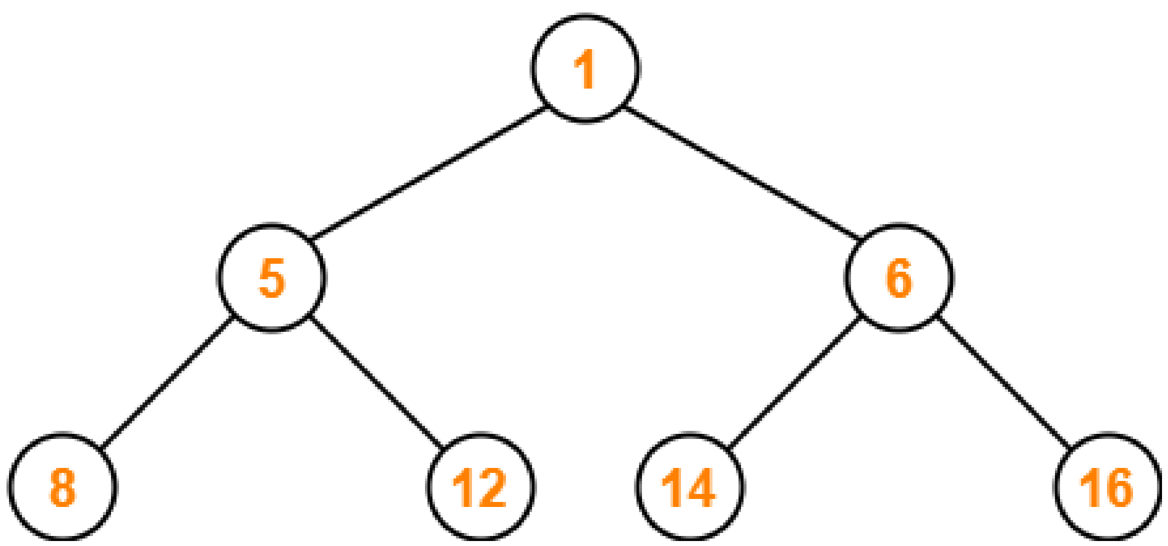
- Construct a max heap for the given array of elements-

  1, 5, 6, 8, 12, 14, 16

Cont…

- Construct a max heap for the given array of elements-

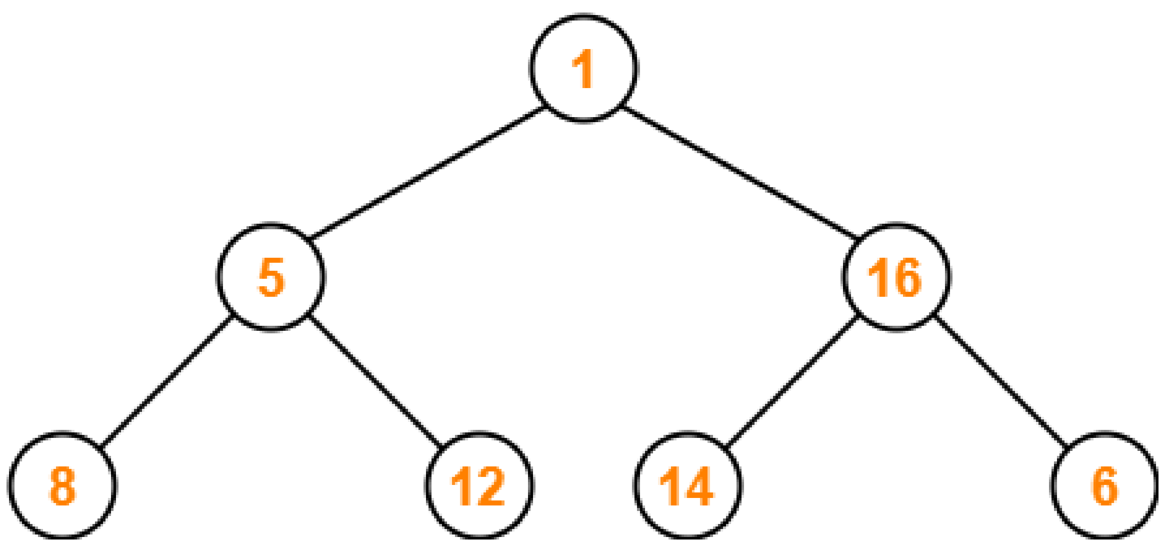
  1, 5, 6, 8, 12, 14, 16
- Step-01: Convert the given array of elements into an almost complete binary tree



Cont…

- Construct a max heap for the given array of elements-

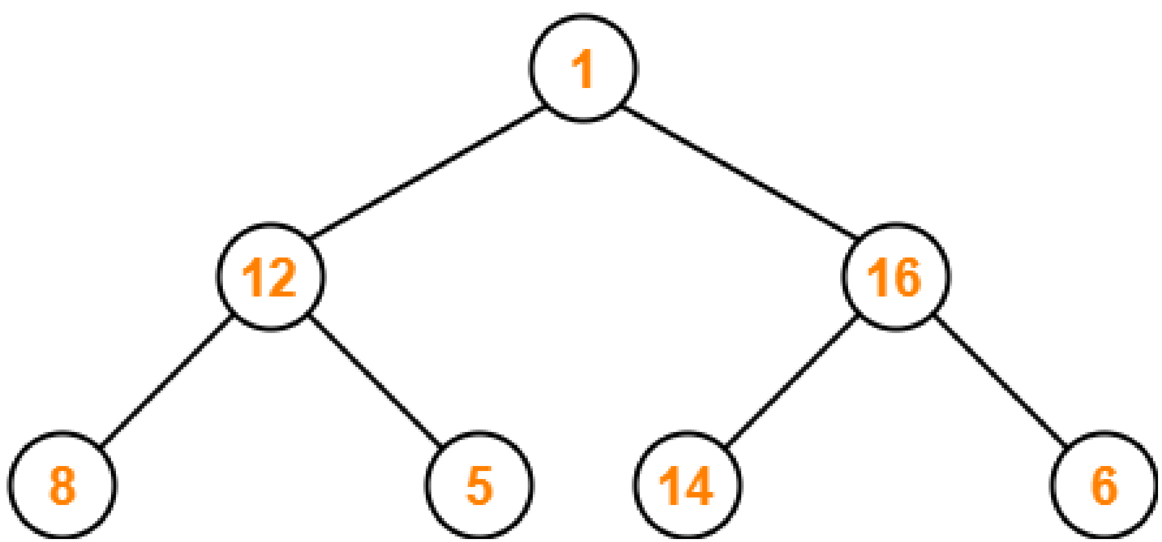
  1, 5, 6, 8, 12, 14, 16
- Step-02:



Cont…

- Construct a max heap for the given array of elements-

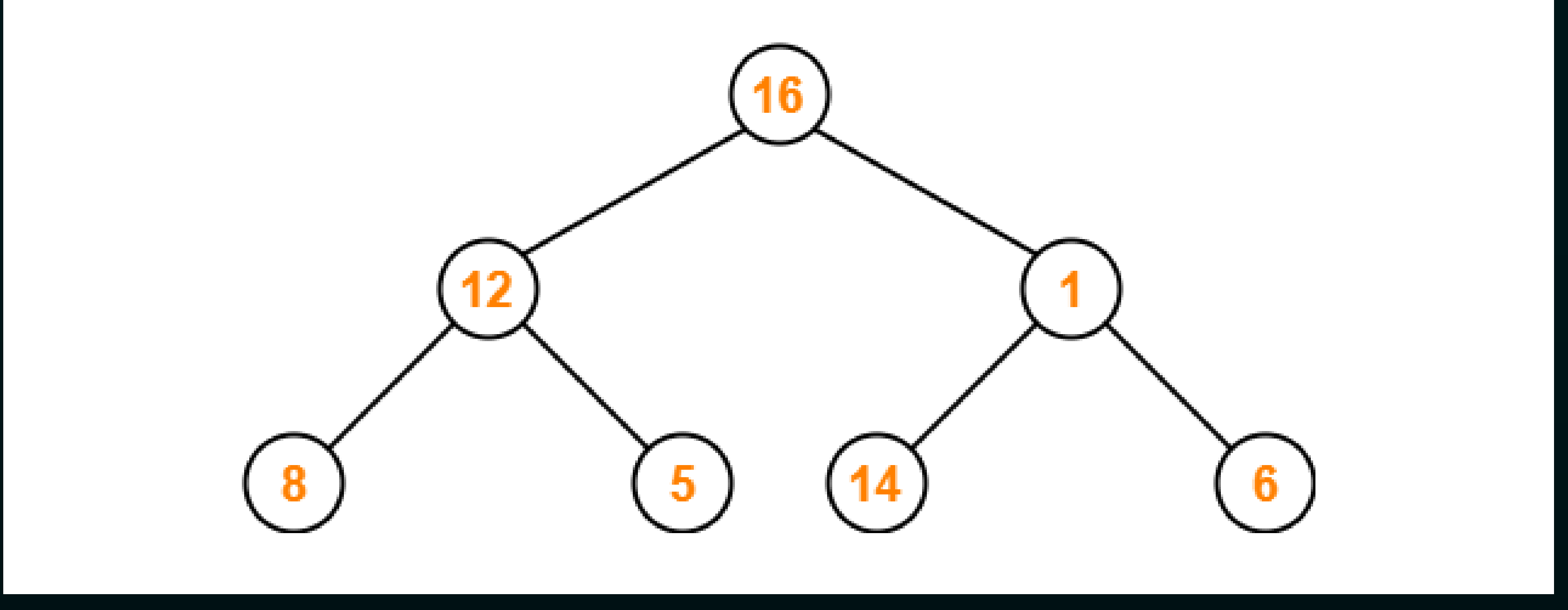
  1, 5, 6, 8, 12, 14, 16
- Step-03:



Cont…

- Construct a max heap for the given array of elements-

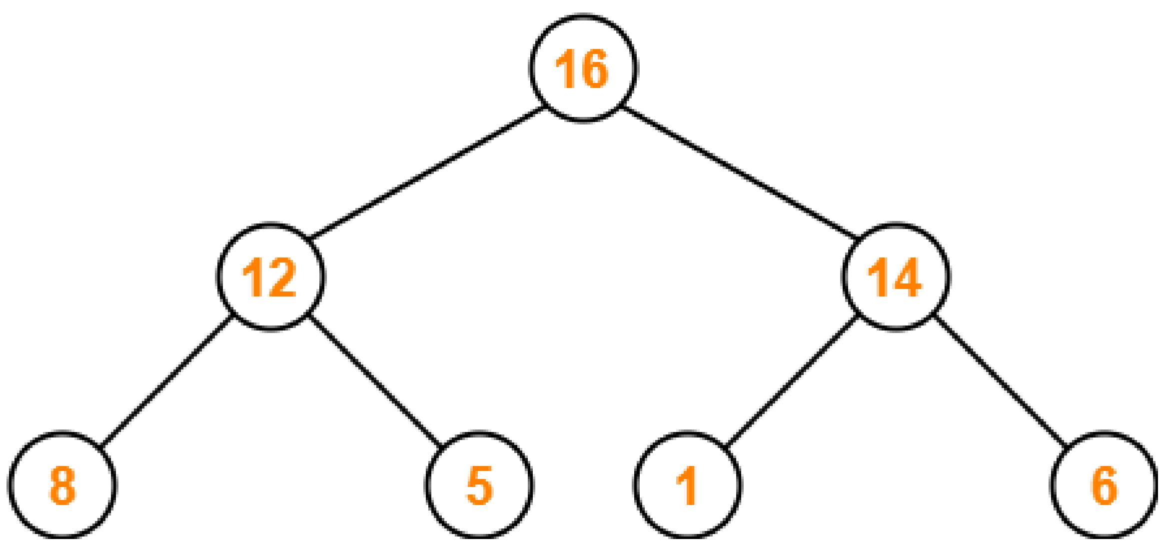
  1, 5, 6, 8, 12, 14, 16
- Step-04:



Cont…

- Construct a max heap for the given array of elements-

  1, 5, 6, 8, 12, 14, 16
- Step-04:



Cont…

- Algorithm for Building Heap

```
BUILD_HEAP(A, n)
{
    last_parent = floor(n/2)-1;
    for( i = last_parent; i>=0; i--)
            MAX_HEAPIFY(A, i);
}
```

Cont...

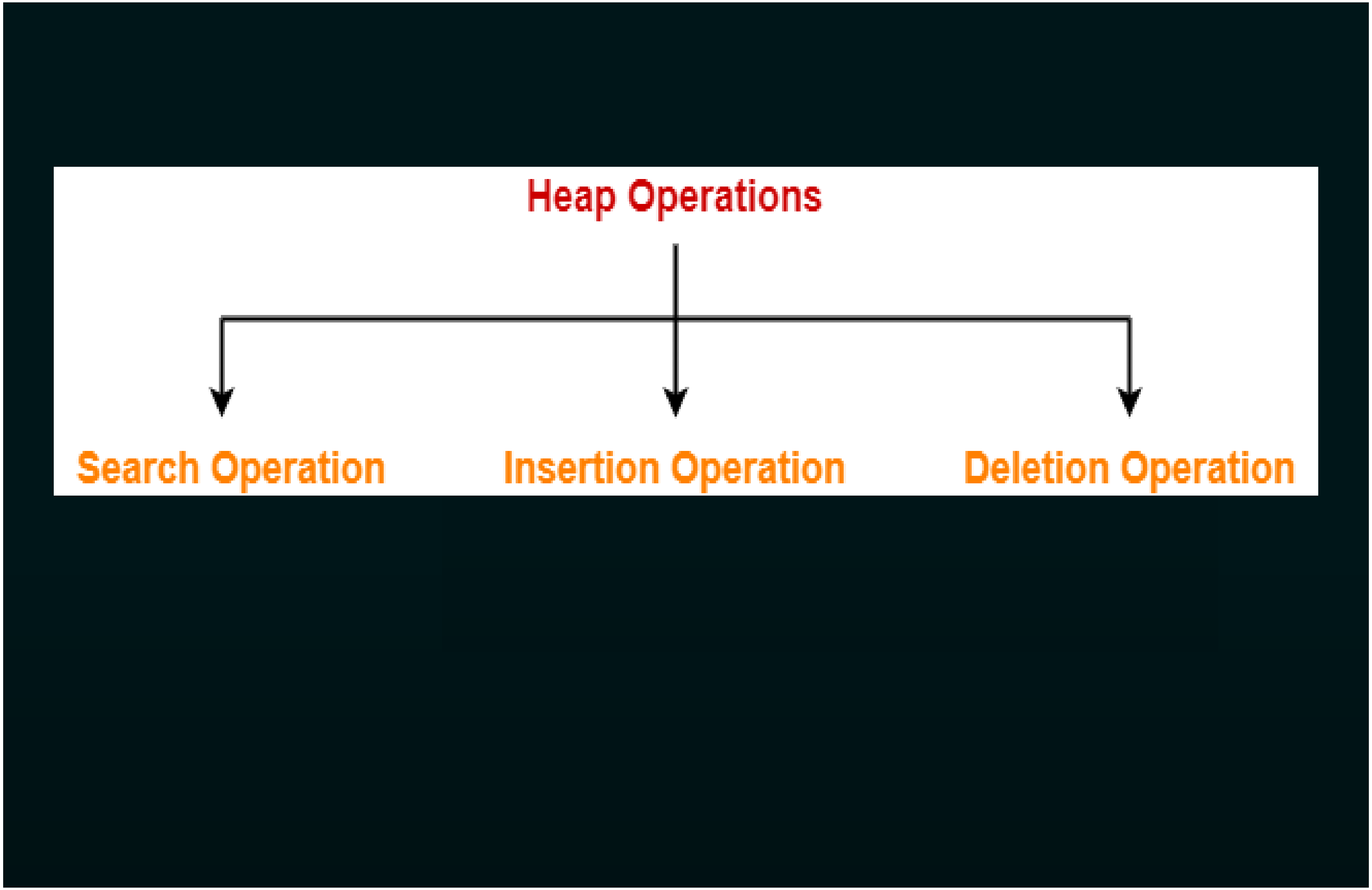
- Algorithm for Maintaining the heap property

```
MAX_HEAPIFY(A, i)
{
    largest = i;
    l = 2*i +1;
    r = 2*i +2;
    if  (l < A.heapsize and A[l] > A[i])
            largest = l;

    if  (r < A.heapsize and A[r] > A[largest])
            largest = r;

     if (largest != i)
            exchange A[i] with A[largest]
            MAX_HEAPIFY(A, largest)
```

Heap Operations



Insertion Operation(heapify_up)

- Step-01:
  - Insert the new element as a next leaf node from left to right
- Step-02:
  - Ensure that the tree remains a max heap
    - Check that every non-leaf node contains a greater or equal value element than its child nodes
    - If there exists any node that does not satisfies the ordering property of max heap, swap the elements
    - Start checking from a non-leaf node with the highest index (bottom to top and right to left)

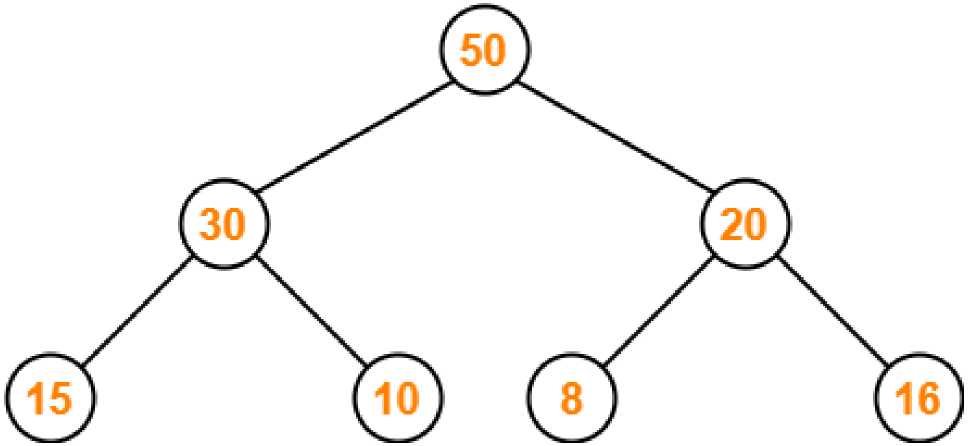
Example: Insertion of Node

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Insert a new node with value 60.

Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Insert a new node with value 60.

Step: 01

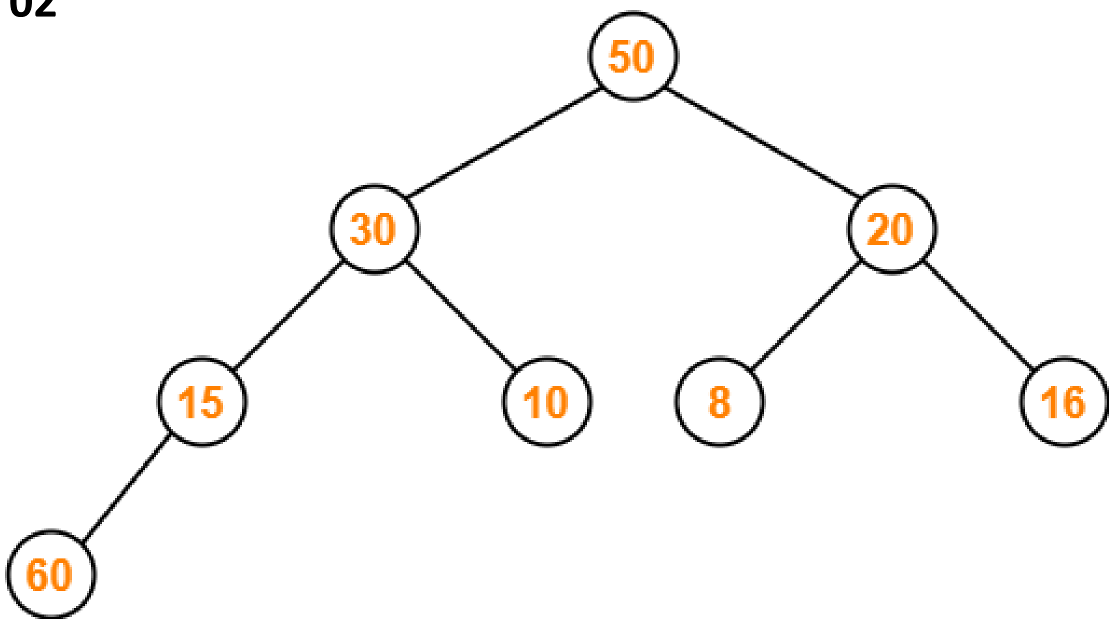


Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16

  Insert a new node with value 60.

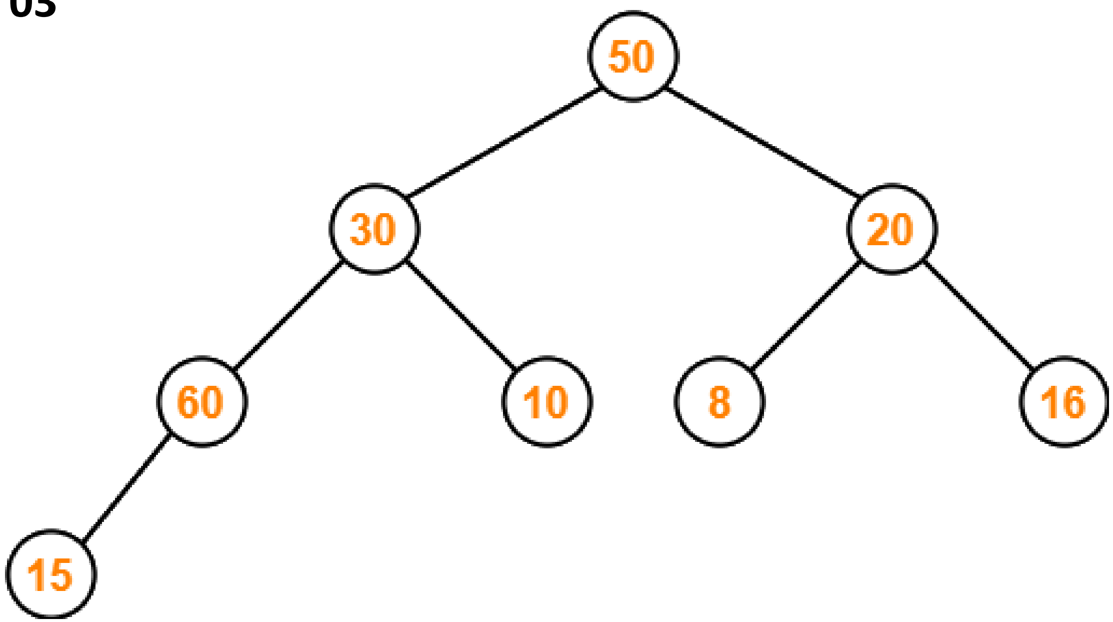
Step: 02



Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Insert a new node with value 60.

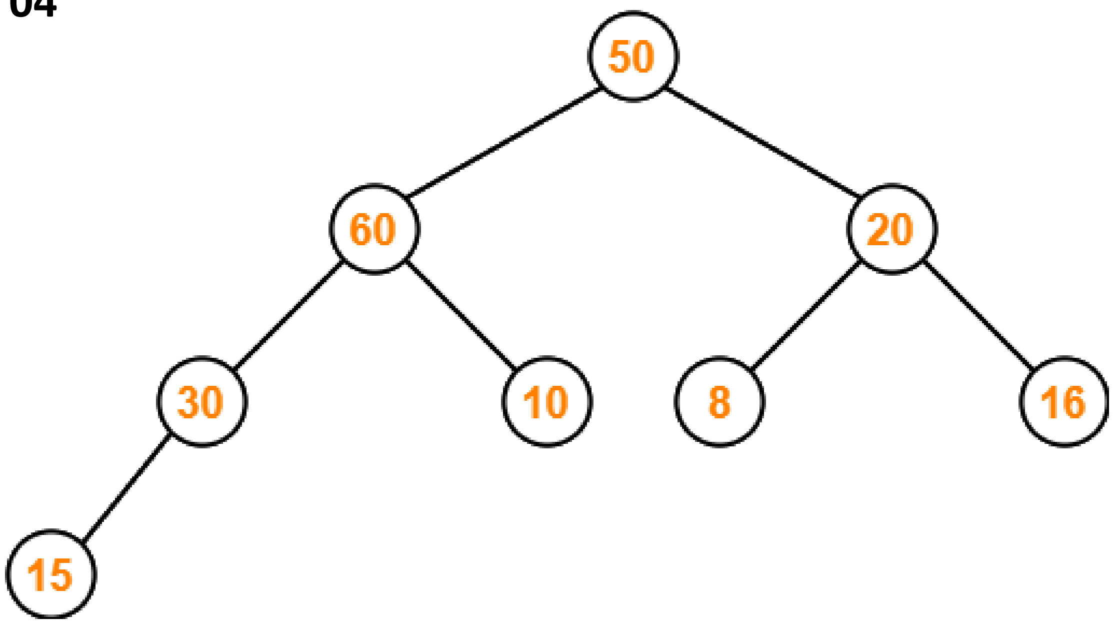
Step: 03



Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Insert a new node with value 60.

Step: 04

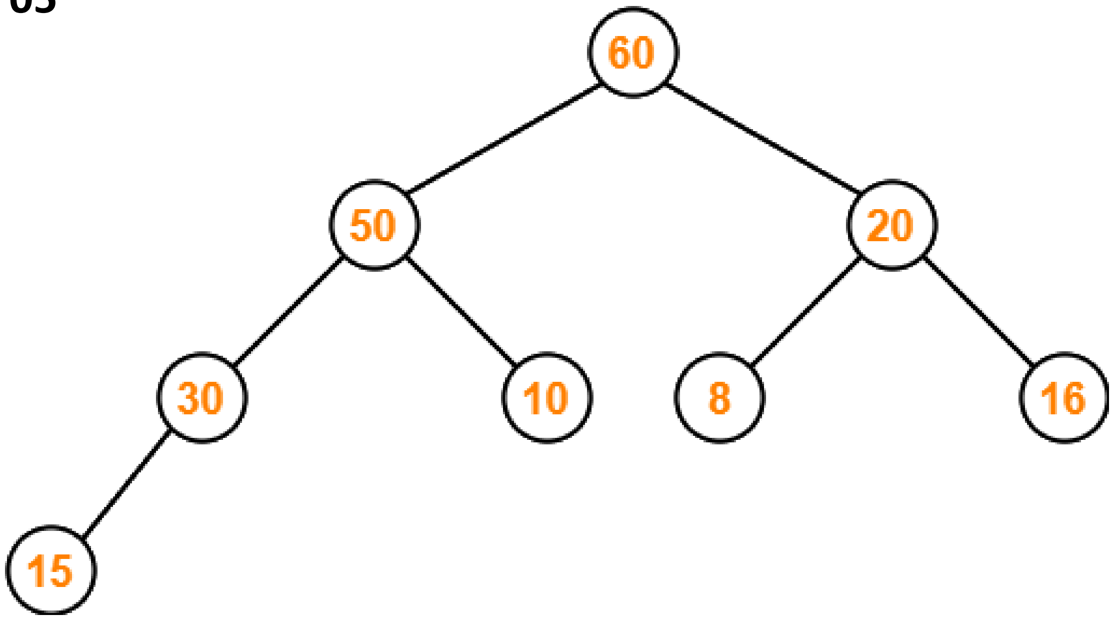


Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16

  Insert a new node with value 60.

Step: 05



Deletion Operation

- Case-01: Deletion of Last Node
  - Just remove/disconnect the last leaf node from the heap
- Case-02: Deletion of Some Other Node – disturbs heap properties
  - Step-01:
    - Delete the desired element from the heap tree.
    - Pluck last node and put in place of the deleted node
  - Step-02:
    - Ensure that the tree remains a max heap

Example: Deletion of a Node

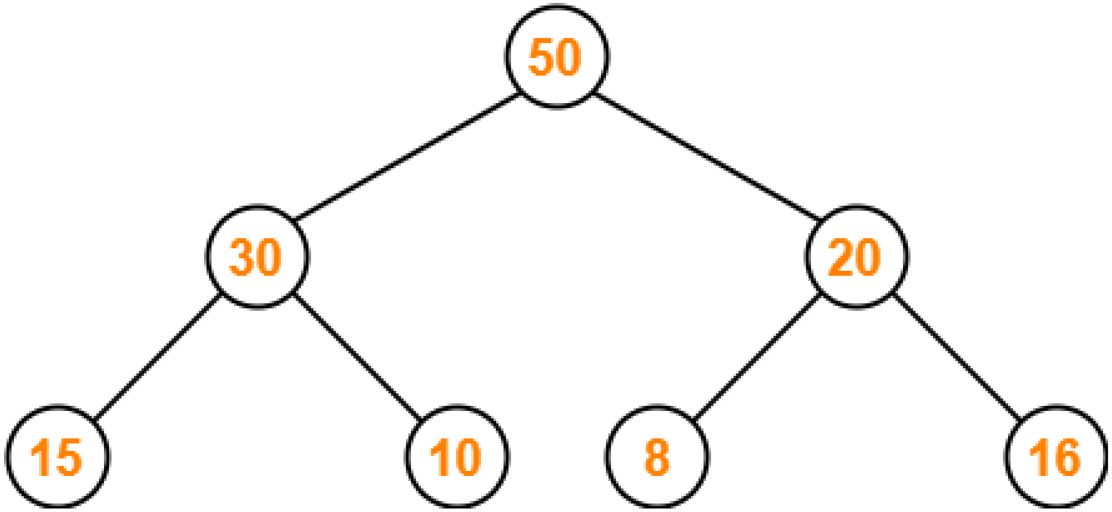
- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16

  Delete a node with value 50.

Cont...

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Delete a node with value 50.

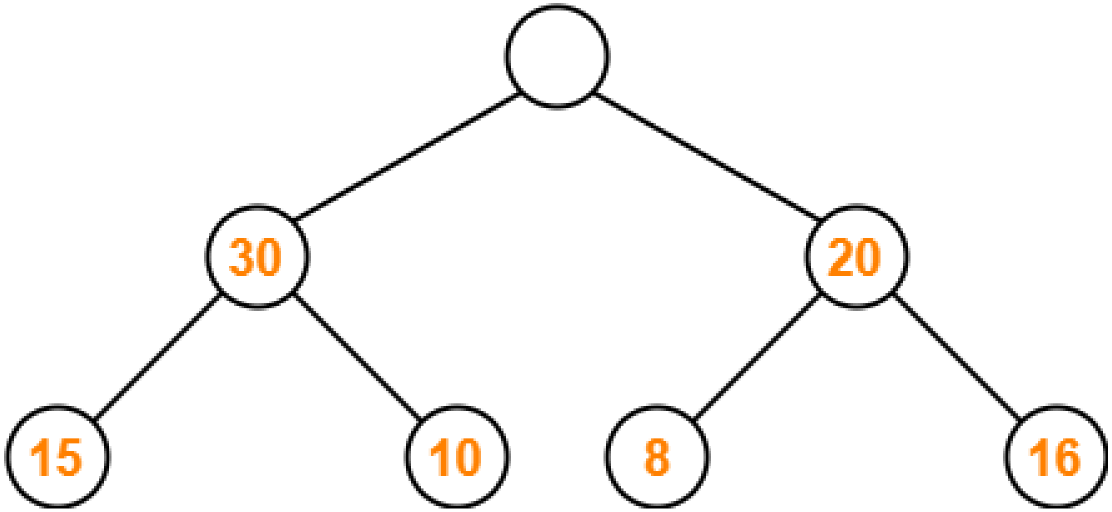
Step: 01



Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Delete a node with value 50.

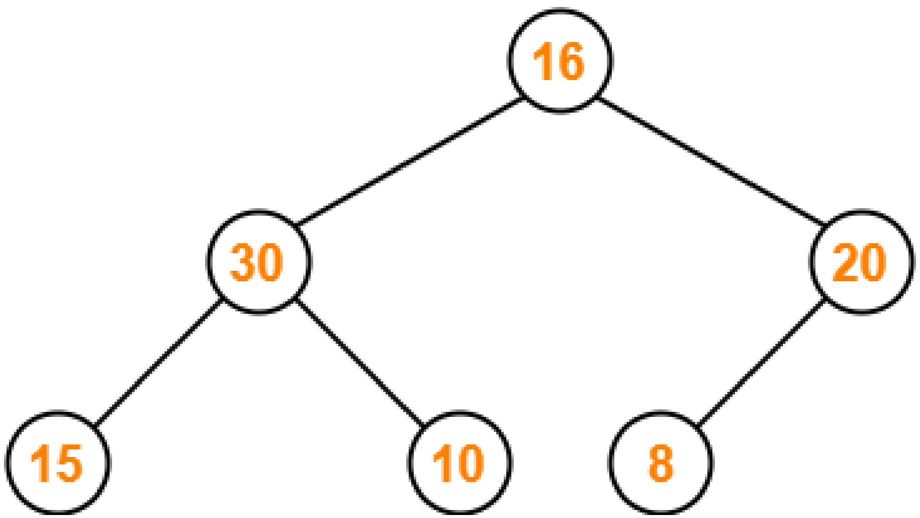
Step: 02



Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Delete a node with value 50.

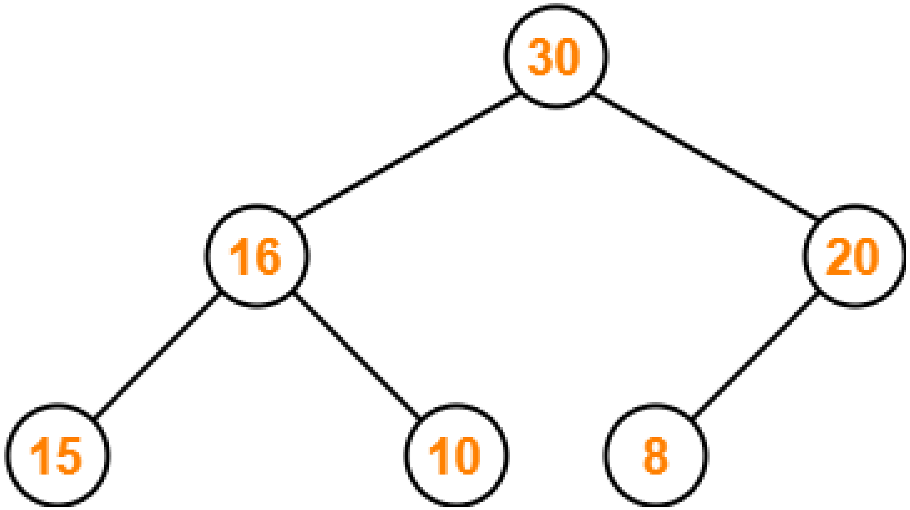
Step: 03



Cont…

- Consider the following max heap- 50, 30, 20, 15, 10, 8, 16
  Delete a node with value 50.

Step: 04

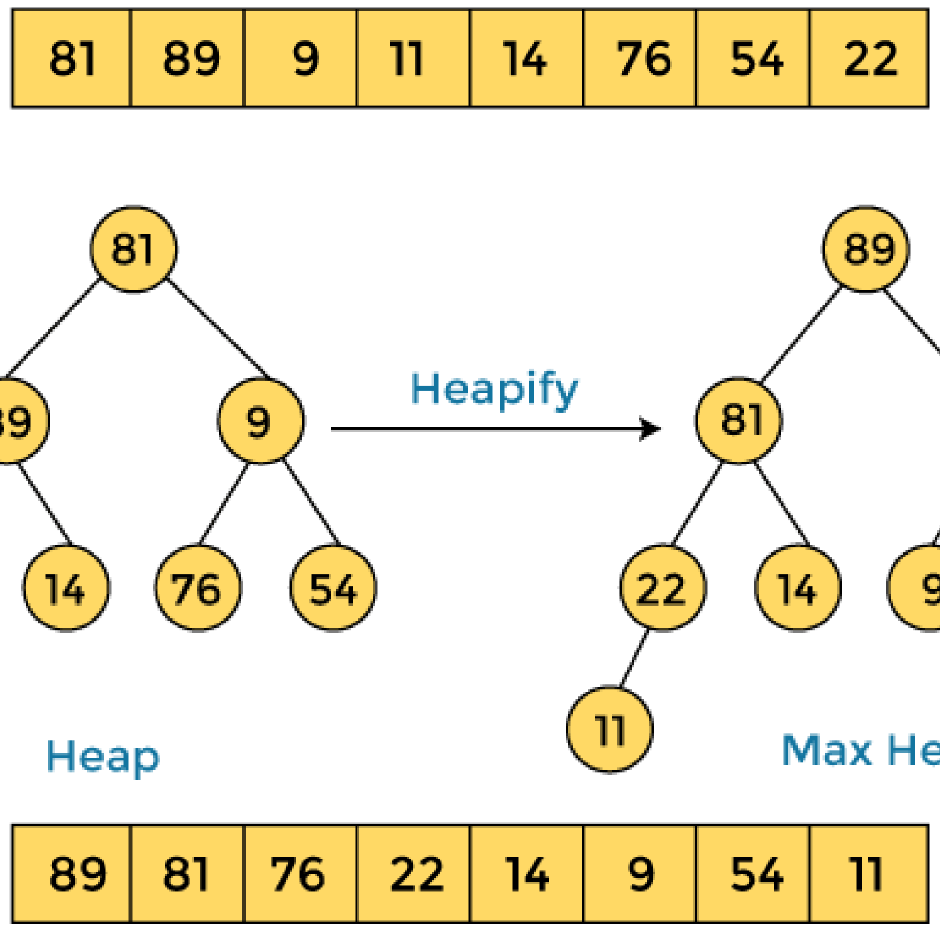


Heap Sort

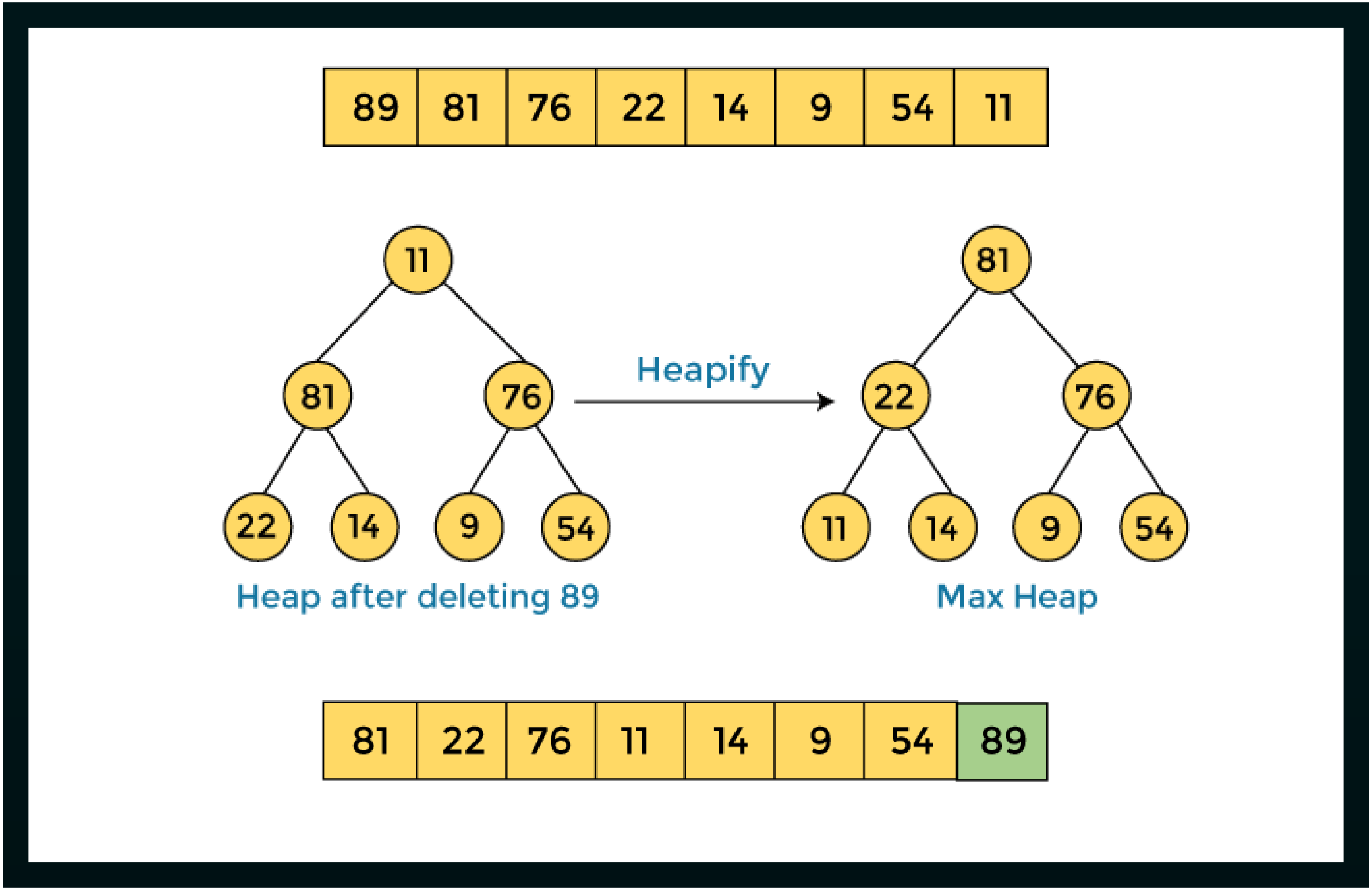
- Heap sort processes the elements by creating the min-heap or max-heap using the elements of the given array
- Concept of heap sort
  - Eliminate the elements one by one from the heap part of the list
  - and then insert them into the sorted part of the list
- Heap sort - recursively performs two main operations
  - Build a heap H, using the elements of array
  - Repeatedly delete the root element of the heap formed in 1st step

Cont…

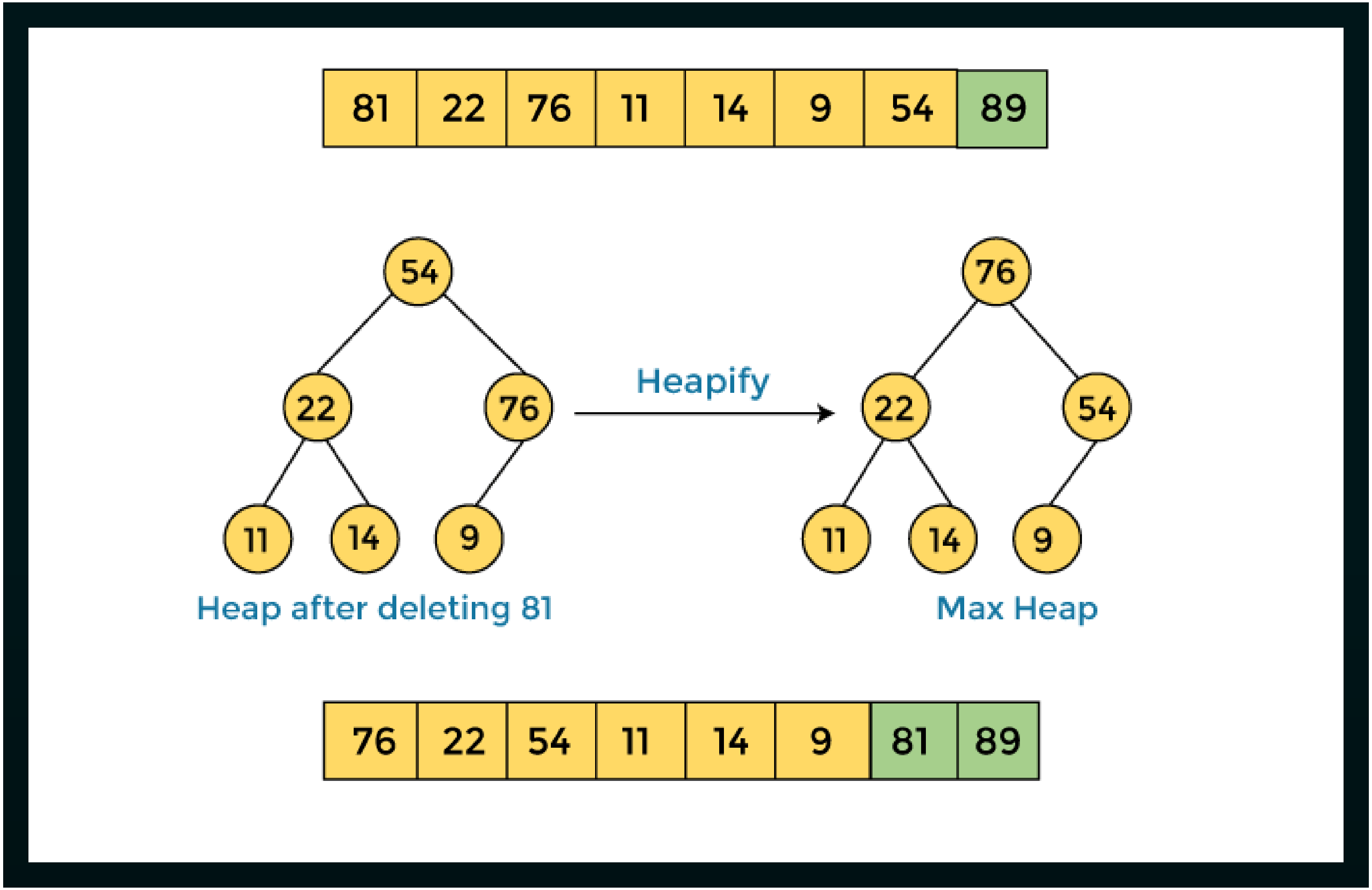
- Example



Cont...

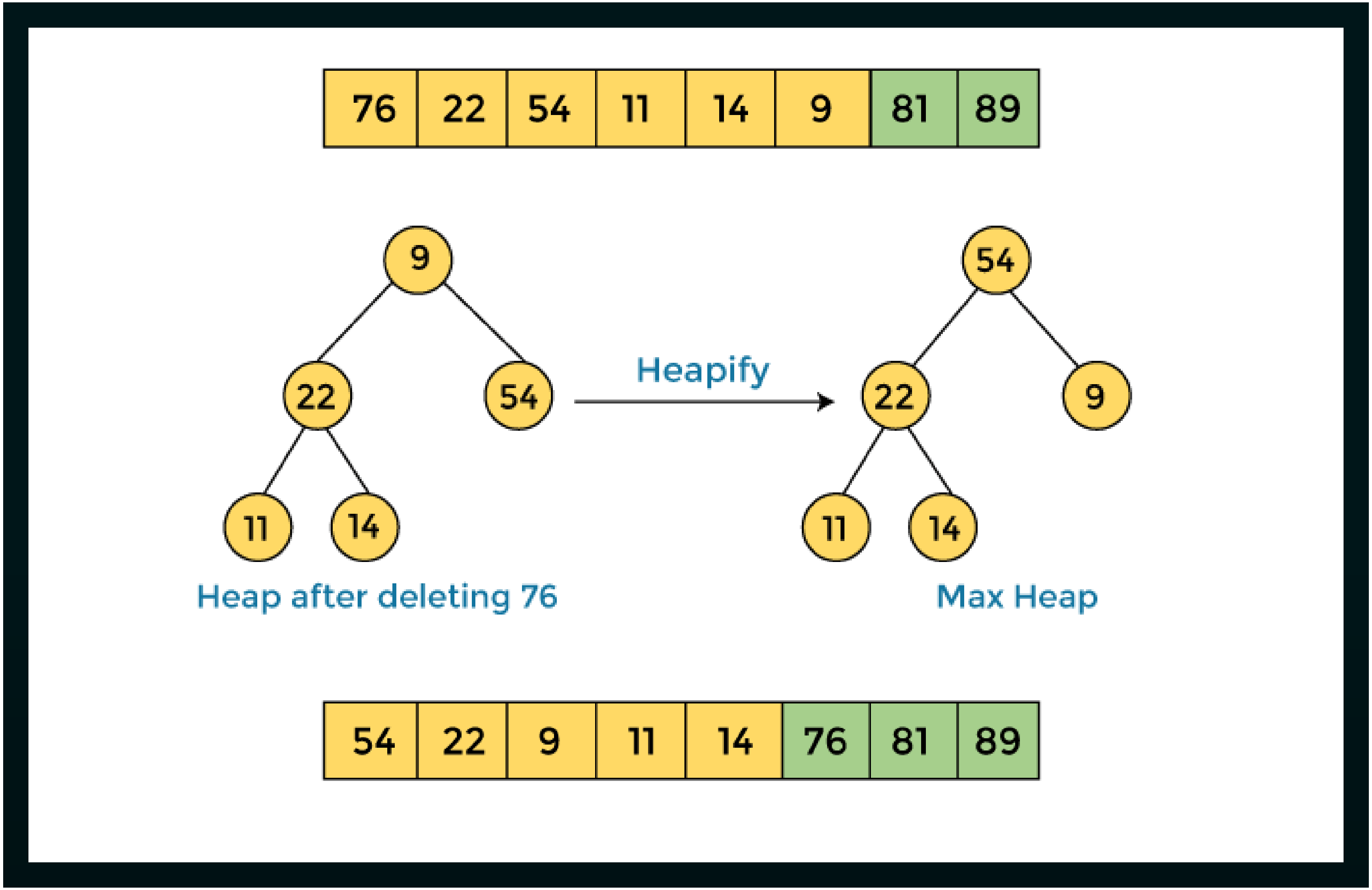

89 81 76 22 14 9 54 11
11
81
76
22
14
9
54
Heapify
81
22
76
11
14
9
54
Heap after deleting 89
Max Heap
81 22 76 11 14 9 54 89

Cont...



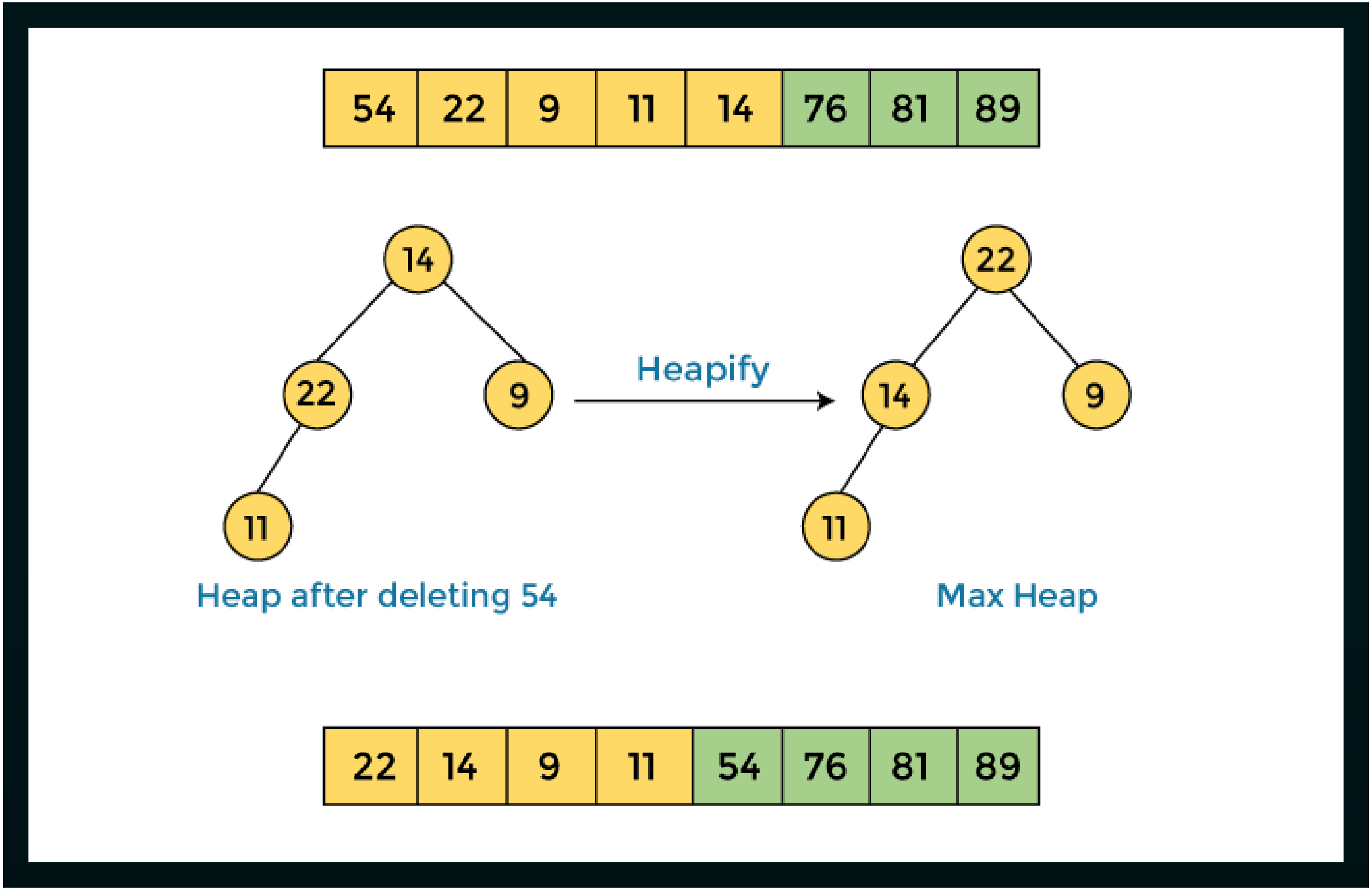
81
22
76
11
14
9
54
89
54
22
76
11
14
9
Heapify
76
22
54
11
14
9
Heap after deleting 81
Max Heap
76
22
54
11
14
9
81
89

Cont...



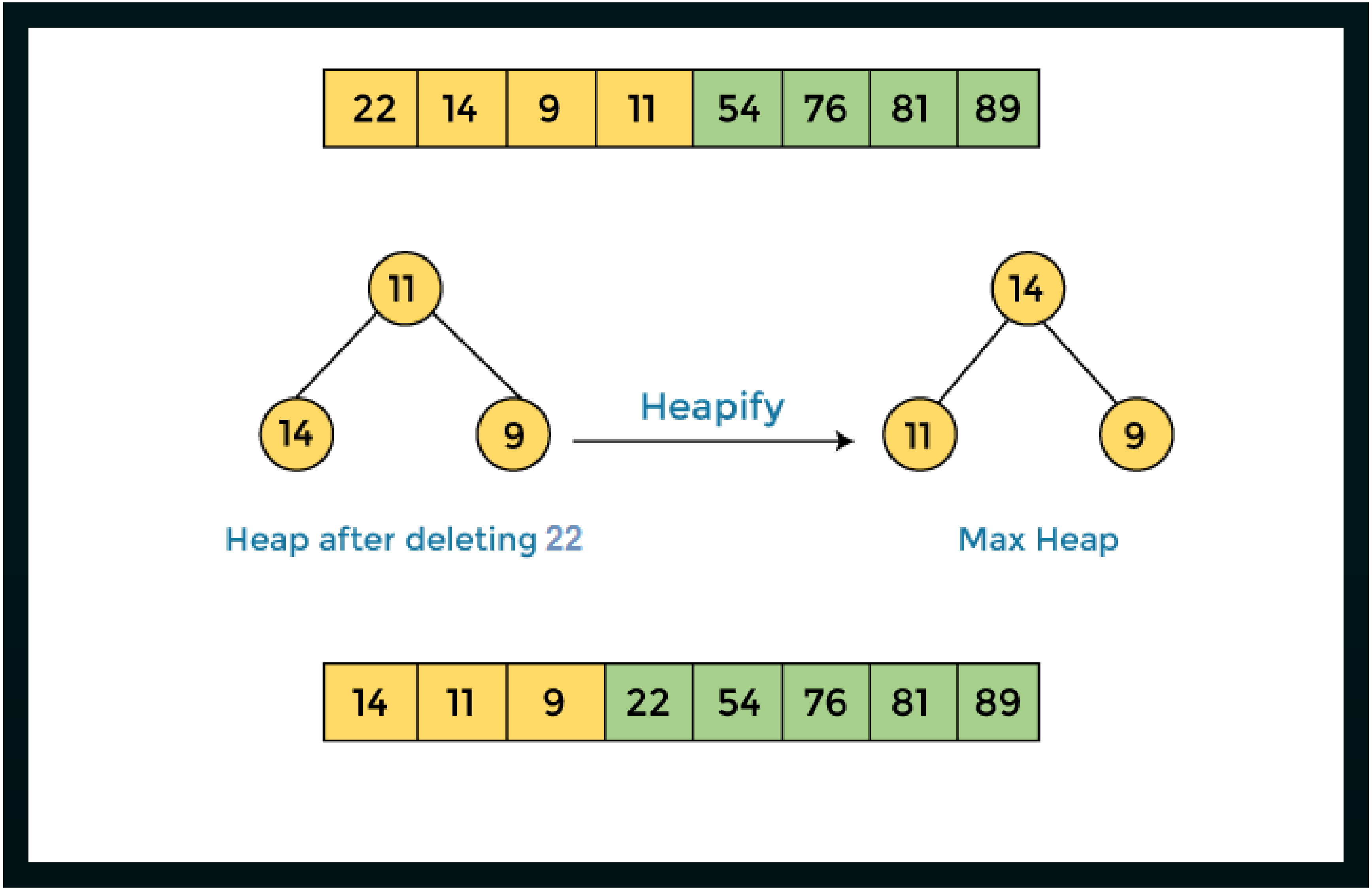
76 22 54 11 14 9 81 89
9
22
54
11
14
Heapify
54
22
9
11
14
Heap after deleting 76
Max Heap
54 22 9 11 14 76 81 89

Cont...

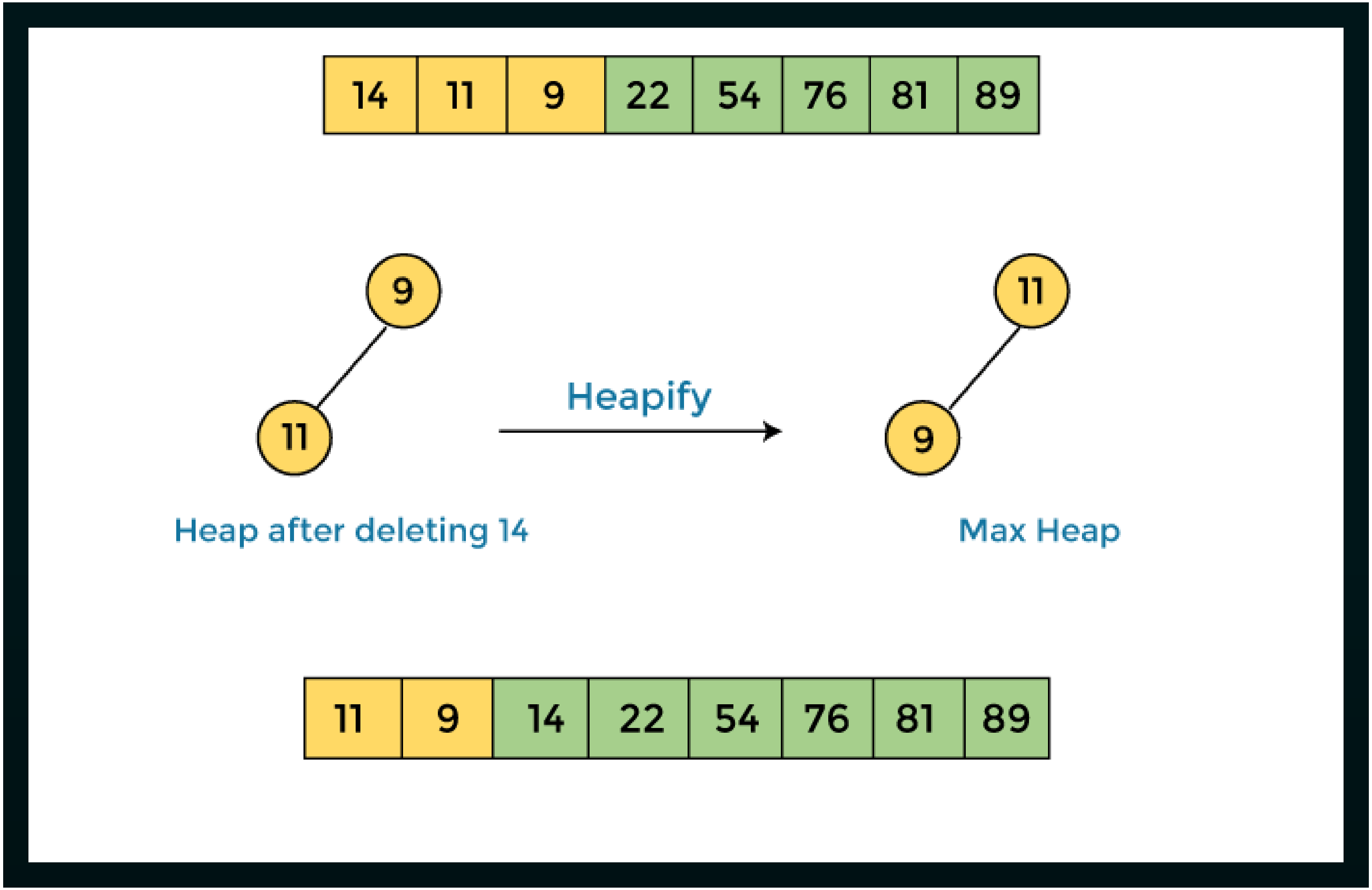


54
22
9
11
14
76
81
89
14
22
9
11
Heapify
22
14
9
11
Heap after deleting 54
Max Heap
22
14
9
11
54
76
81
89

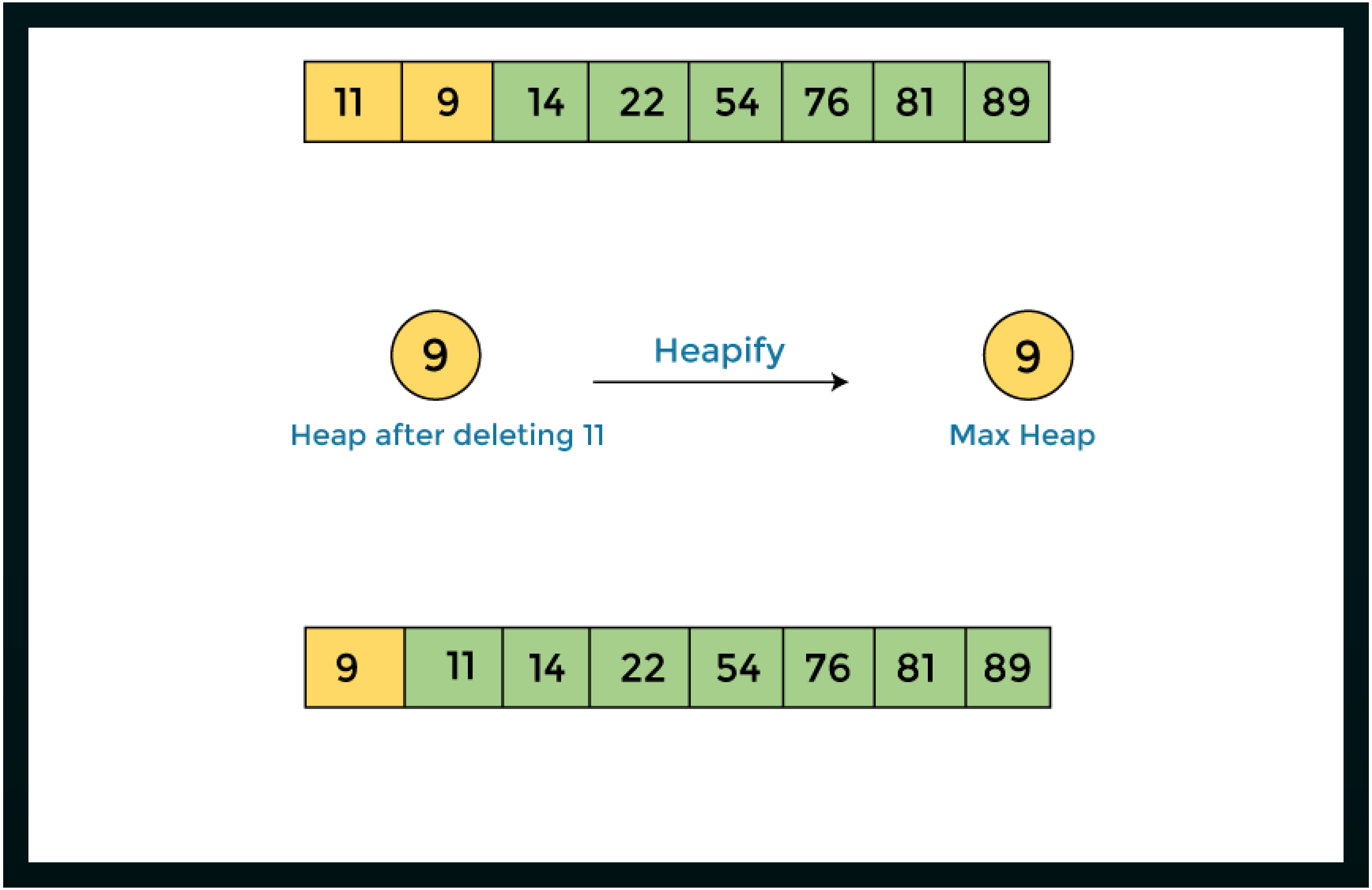
Cont...


22 14 9 11 54 76 81 89
11
14
9
Heapify
14
11
9
Heap after deleting 22
Max Heap
14 11 9 22 54 76 81 89

Cont...

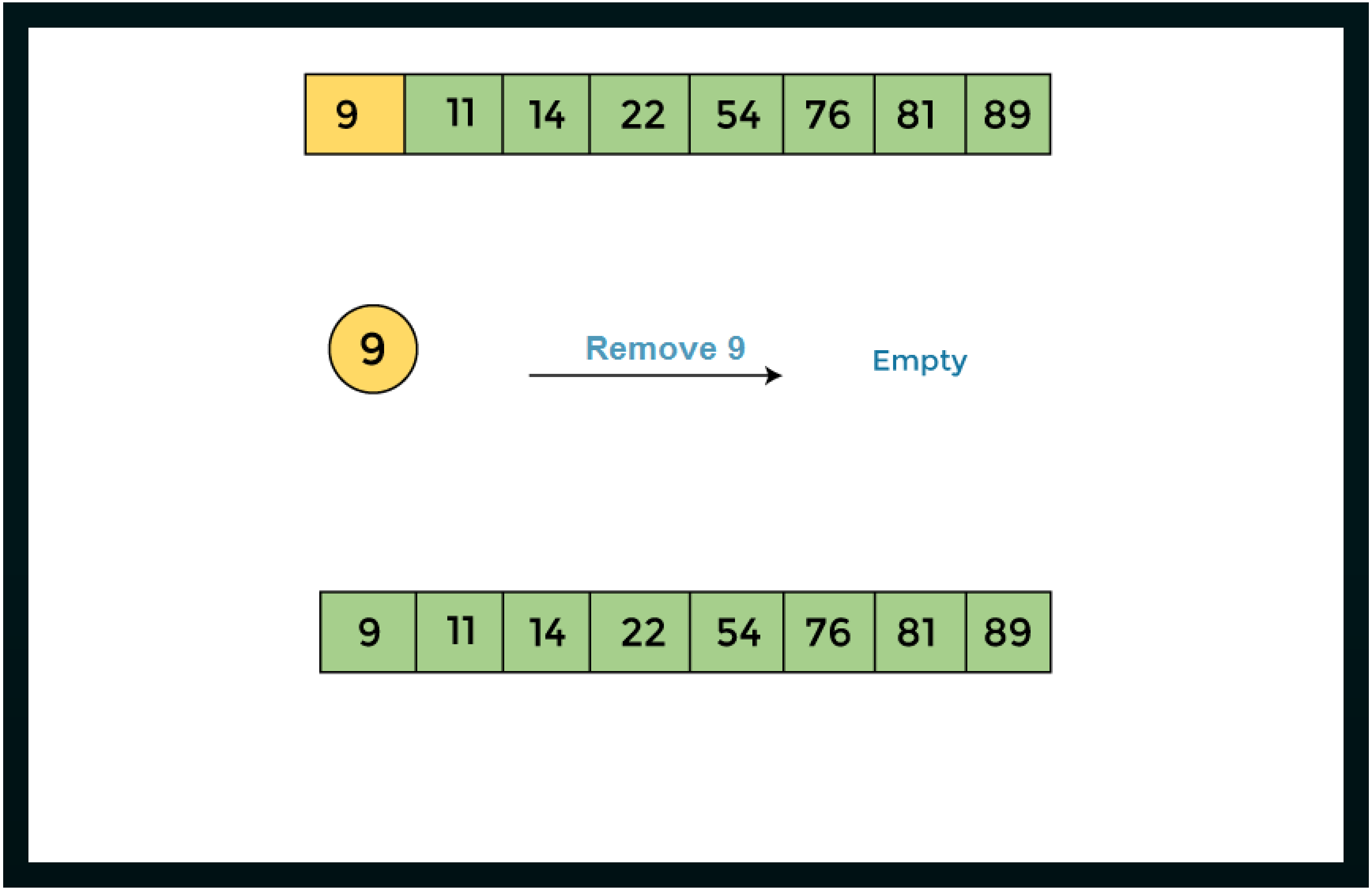

14
11
9
22
54
76
81
89
9
11
Heapify
11
9
Heap after deleting 14
Max Heap
11
9
14
22
54
76
81
89

Cont...



11
9
14
22
54
76
81
89
9
Heapify
9
Heap after deleting 11
Max Heap
9
11
14
22
54
76
81
89

Cont…



Cont...

```
void heapSort(int a[], int n)
{
  for (int i = n/2-1; i>=0; i--)
    heapify(a, n, i);
    // One by one extract an element from heap

  for (int i = n-1; i>=0; i--)
  {
     /* Move current root element to end*/
     // swap a[0] with a[i]
     int temp = a[0];
     a[0] = a[i];
     a[i] = temp;

     heapify(a, i, 0);
  }
}
```

Cont…

- Time Complexity
  - Best Case Complexity - O(n log n)
  - Average Case Complexity - O(n log n)
  - Worst Case Complexity - O(n log n)