M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of CSE (AIML) and CSE (Cyber Security)

Course Name: Design and Analysis of Algorithms
Course Code: CY43
Credits: 3:0:0
UNIT 3

Reference:
Jon Kleinberg and Eva Tardos
Algorithm Design, Pearson (1st Edition),2013
Anany Levitin, Introduction to the Design and Analysis of Algorithms, Pearson (2017)

Unit III

Transform and Conquer

Greedy Algorithms

RAMAIAH
Institute of Technology

Unit III

UNIT - 3

Transform and Conquer: Heaps and Heapsort.

Greedy Algorithms: Interval Scheduling: The Greedy Algorithm Stays Ahead: Designing a Greedy Algorithm, Analyzing the Algorithm, Scheduling to Minimize Lateness: An Exchange Argument: The Problem, Designing the Algorithm, Designing and Analyzing the Algorithm.

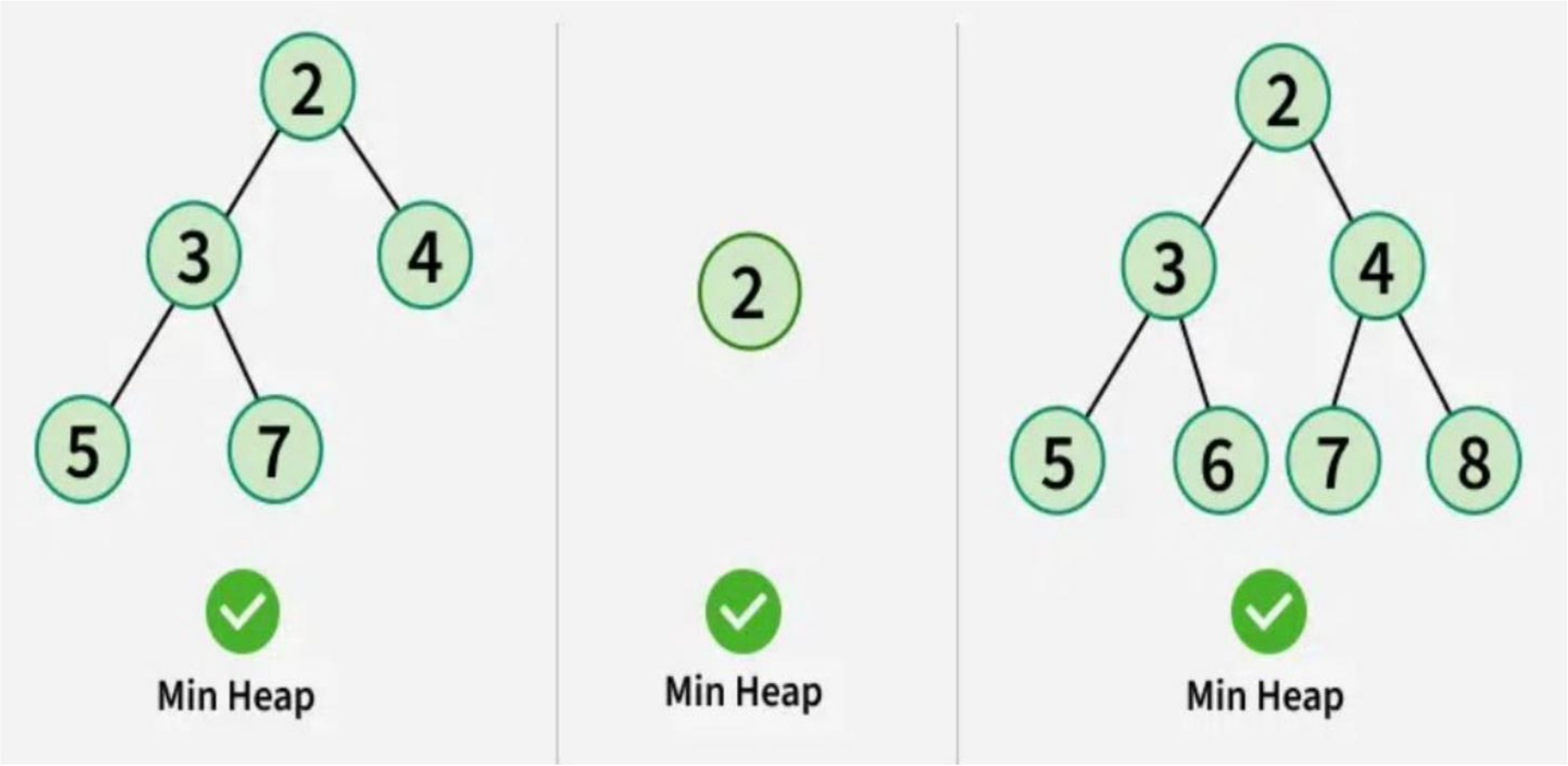
Prim's Algorithm, Kruskal's Algorithm, Dijkstra's Algorithm, Huffman Trees and Codes.

Unit III

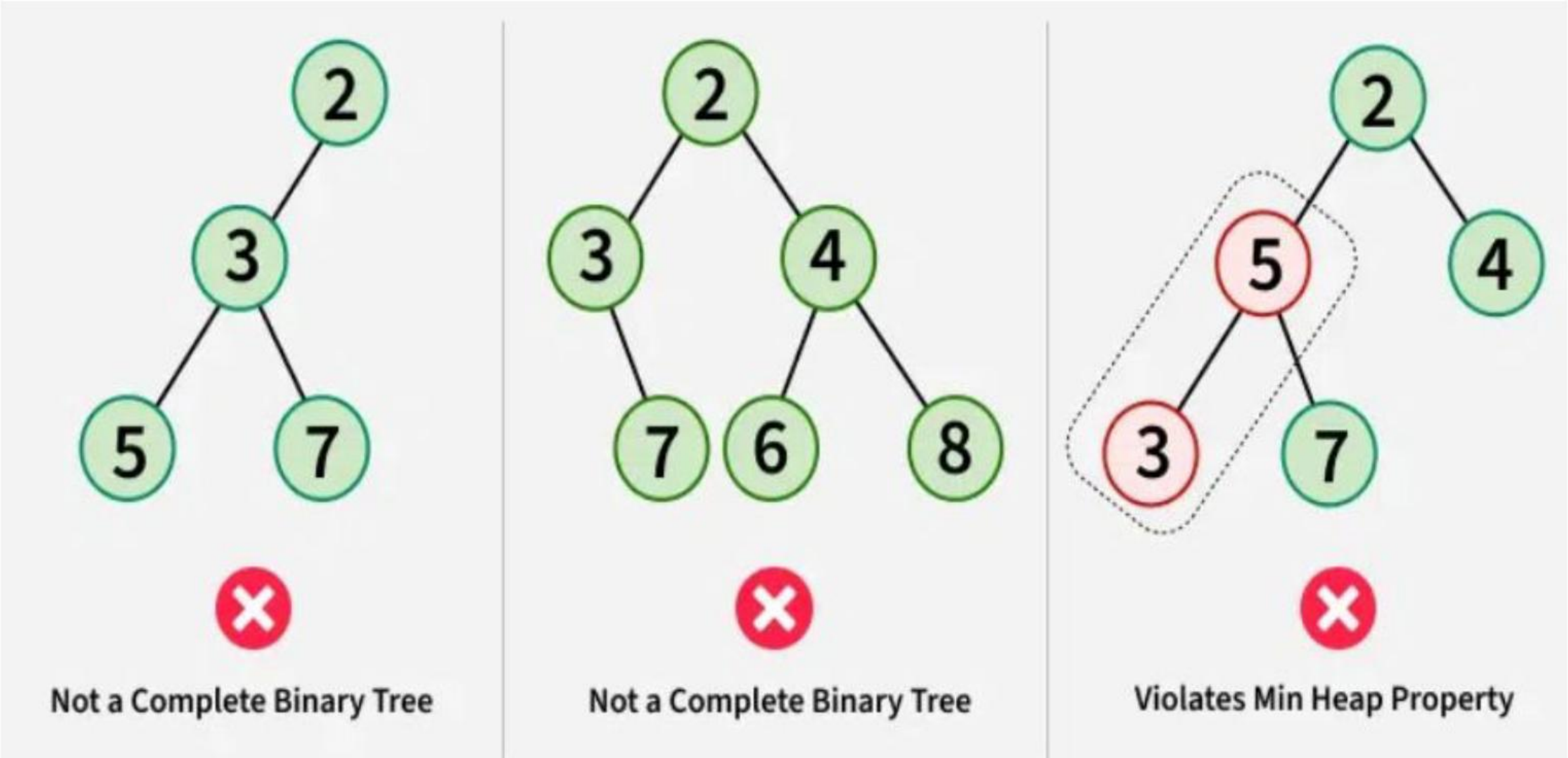
Heaps and Heapsort

- A **Heap** is a complete binary tree data structure that satisfies the heap property
- A **Binary Heap** is a complete binary tree that stores data efficiently, allowing quick access to the maximum or minimum element, depending on the type of heap. It can either be a Min Heap or a Max Heap.

Unit III

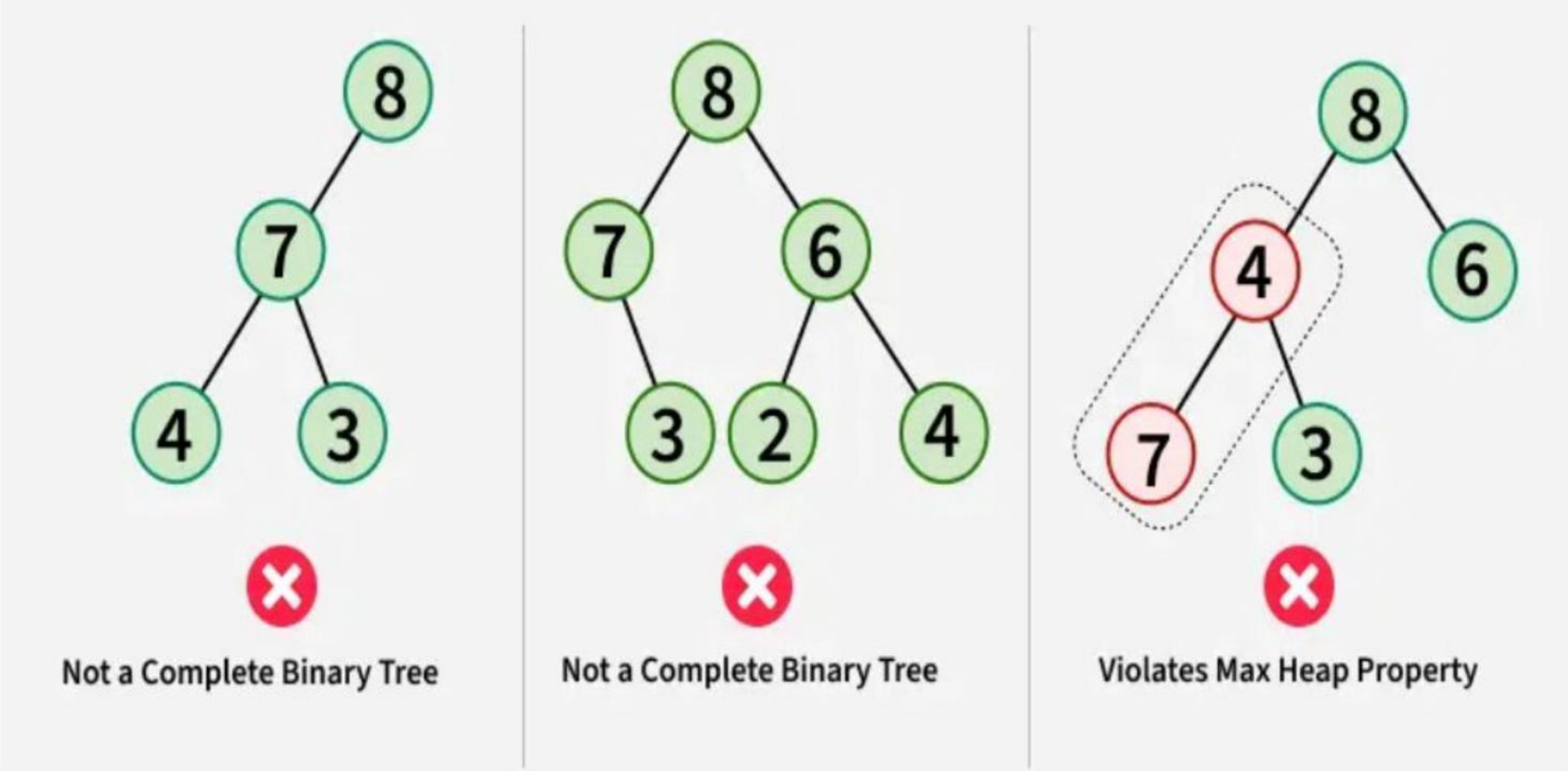


Unit III


2
3
5
7
Not a Complete Binary Tree
2
3
4
7
6
8
Not a Complete Binary Tree
2
5
4
3
7
Violates Min Heap Property

Unit III

Unit III


8
7
4
3
Not a Complete Binary Tree
8
7
6
3
2
4
Not a Complete Binary Tree
8
4
6
7
3
Violates Max Heap Property

Unit III

Notion of the Heap

A heap can be defined as a binary tree with keys assigned to its nodes, one key per node, provided the following two conditions are met:

1. The shape property: the binary tree is essentially complete (or simply complete), i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing.

2. The parental dominance: the key in each node is greater than or equal to the keys in its children.

Unit III

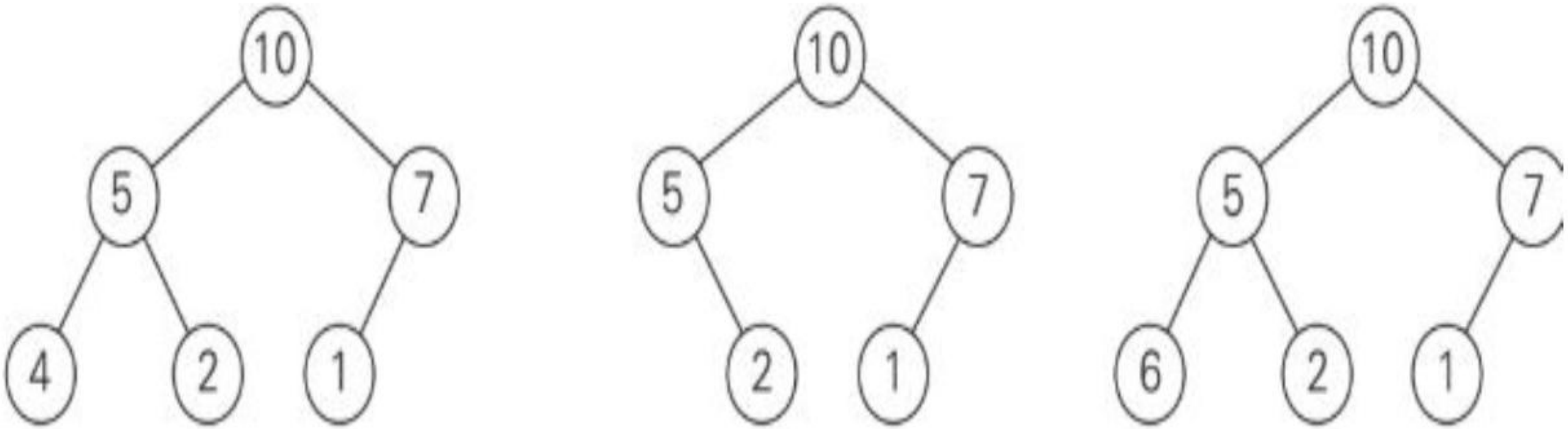


FIGURE 6.9 Illustration of the definition of heap: only the leftmost tree is a heap.

Unit III

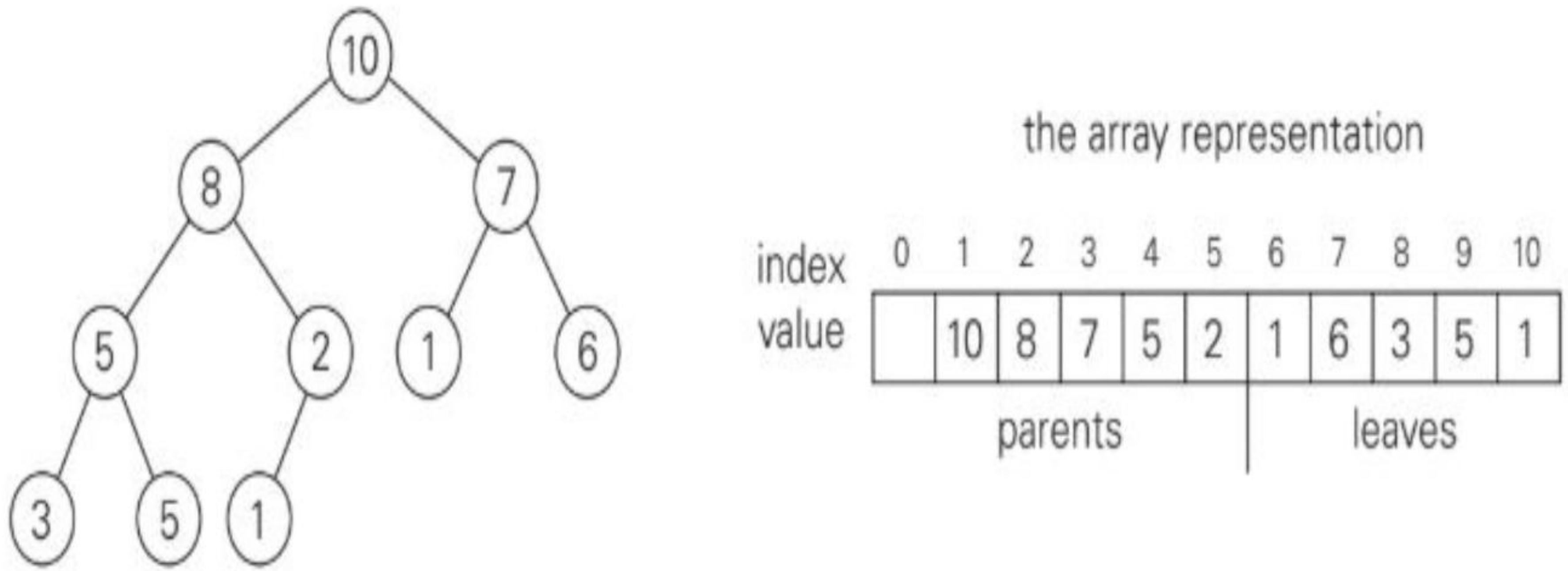


FIGURE 6.10 Heap and its array representation.

Unit III

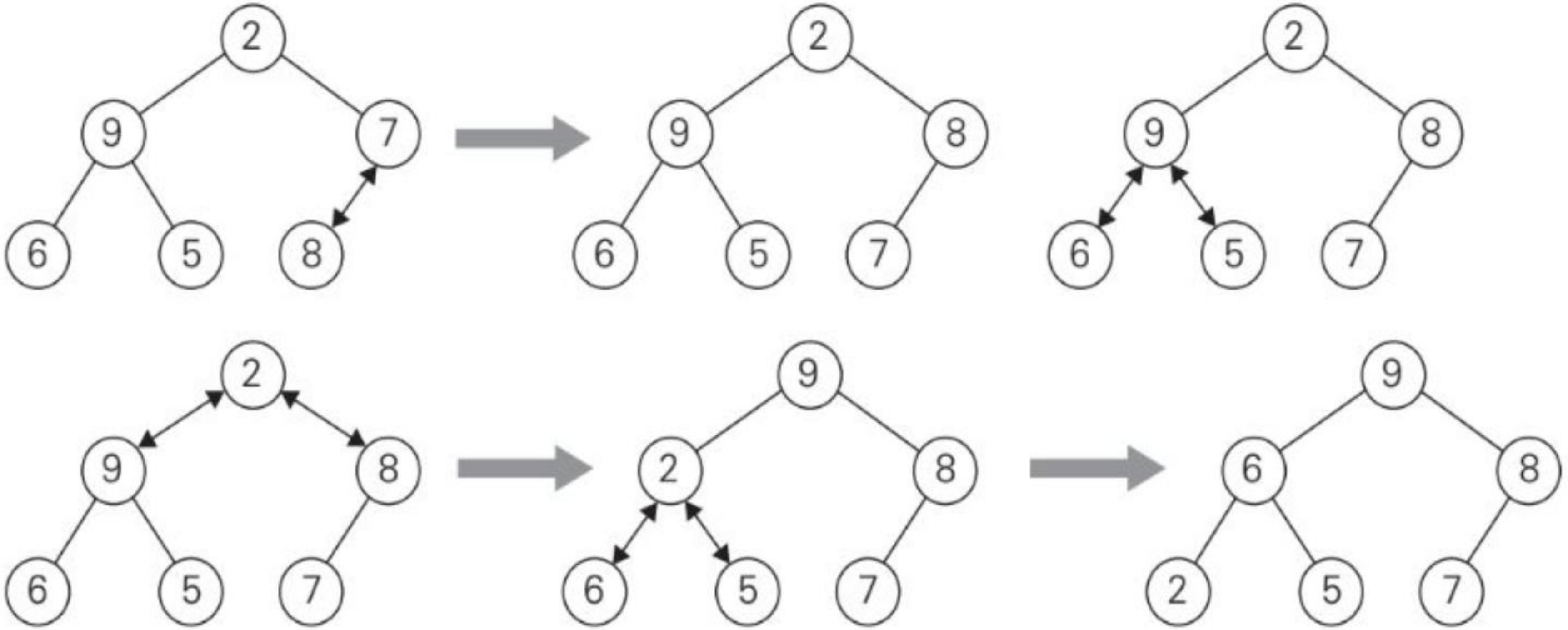


FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8. The double-headed arrows show key comparisons verifying the parental dominance.

Unit III

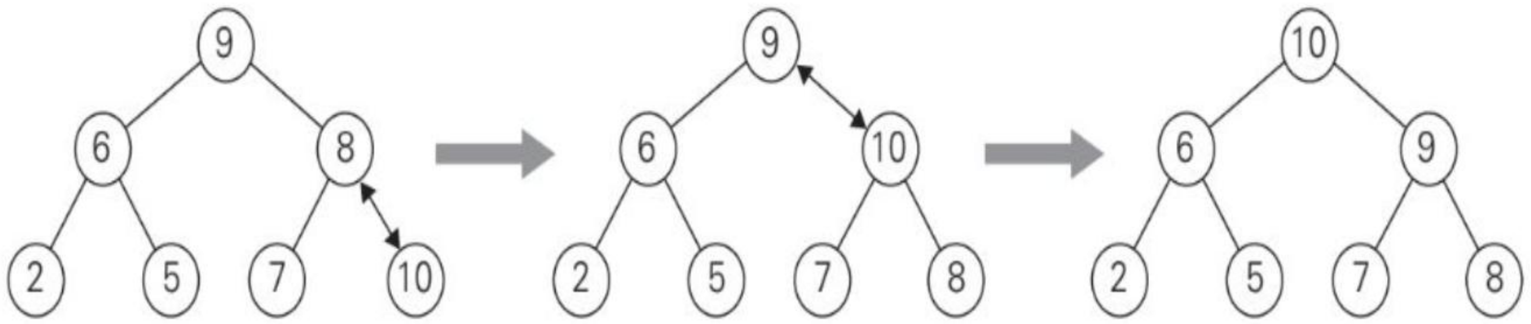


FIGURE 6.12 Inserting a key (10) into the heap constructed in Figure 6.11. The new key is sifted up via a swap with its parent until it is not larger than its parent (or is in the root).

Note: Insertion operation cannot require more key comparisons than the heap's height. The time efficiency of insertion is in O(log n).

Unit III

Maximum Key Deletion from a heap

Step 1: Exchange the root's key with the last key K of the heap.

Step 2: Decrease the heap's size by 1.

Step 3: "Heapify" the smaller tree by sifting K down the tree exactly in the same way we did it in the bottom-up heap construction algorithm.

Unit III

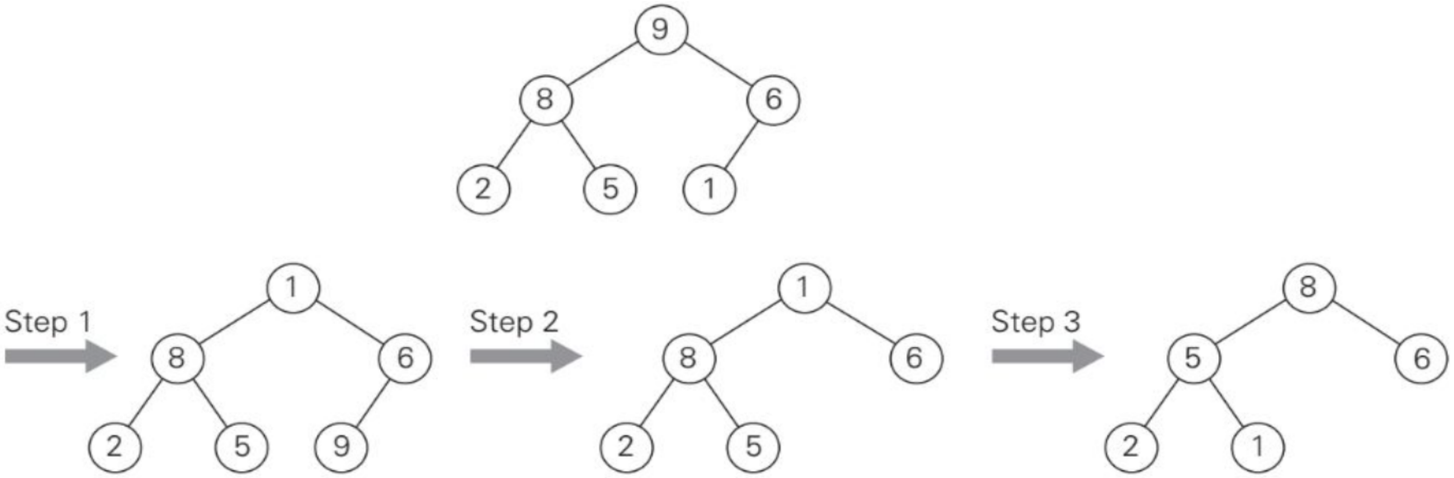


FIGURE 6.13 Deleting the root's key from a heap. The key to be deleted is swapped with the last key after which the smaller tree is "heapified" by exchanging the new key in its root with the larger key in its children until the parental dominance requirement is satisfied.

Note: Since this cannot require more key comparisons than twice the heap's height, the time efficiency of deletion is in O(log n) as well.

Unit III

Heapsort

Stage 1 (heap construction): Construct a heap for a given array.

Stage 2 (maximum deletions): Apply the root-deletion operation n − 1 times to the remaining heap.

As a result, the array elements are eliminated in decreasing order. But since under the array implementation of heaps an element being deleted is placed last, the resulting array will be exactly the original array sorted in increasing order.

Unit III

Stage 1 (heap construction)

2 9 **7** 6 5 8

2 **9** 8 6 5 7

2 9 8 6 5 7

9 **2** 8 6 5 7

9 6 8 2 5 7

Stage 2 (maximum deletions)

9 6 8 2 5 7

7 6 8 2 5 | **9**

8 6 7 2 5

5 6 7 2 | **8**

7 6 5 2

2 6 5 | **7**

6 2 5

5 2 | **6**

5 2

2 | **5**

2

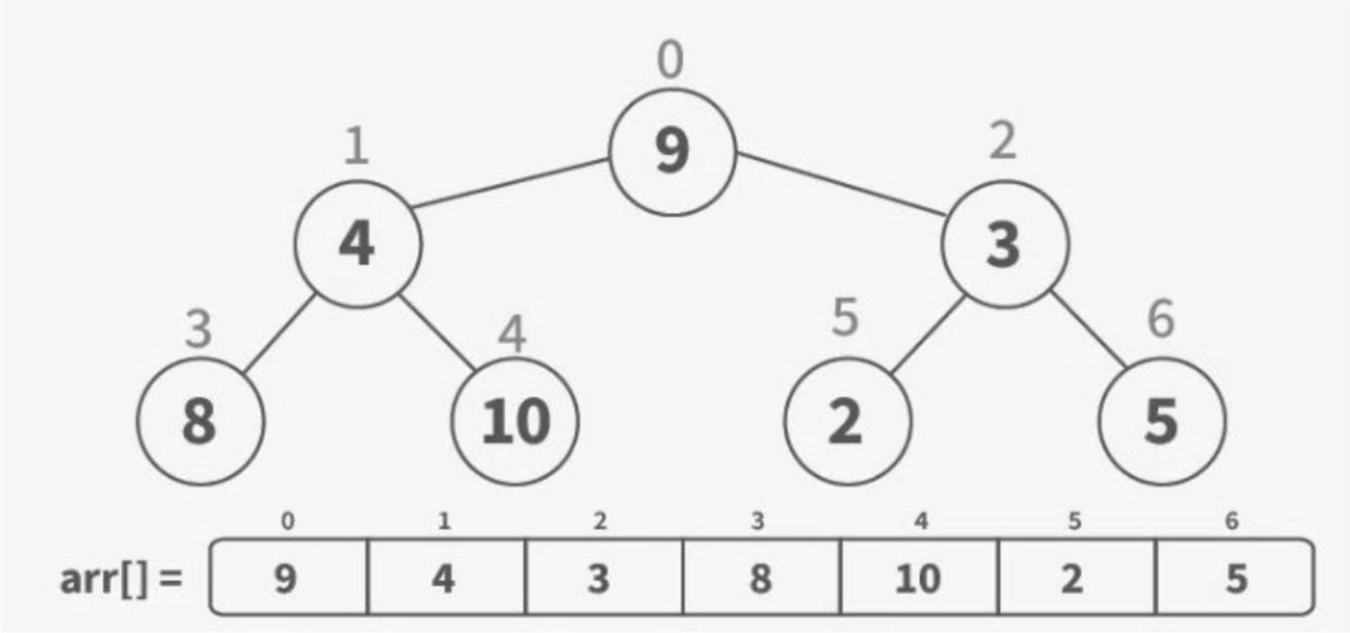
FIGURE 6.14 Sorting the array 2, 9, 7, 6, 5, 8 by heapsort.

RAMAIAH
Institute of Technology

Unit III

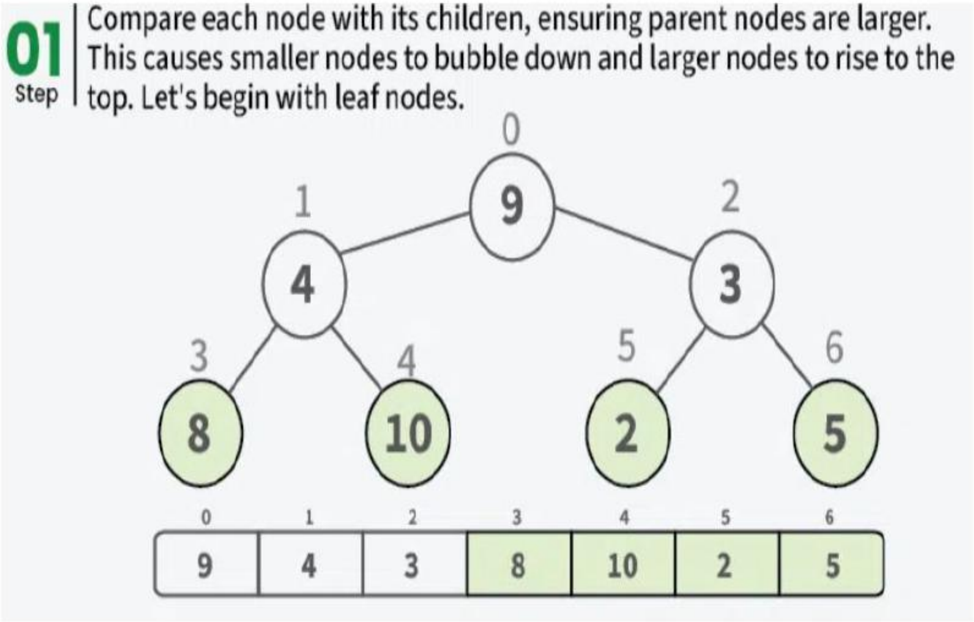
1. Treat the Array as a Complete Binary Tree

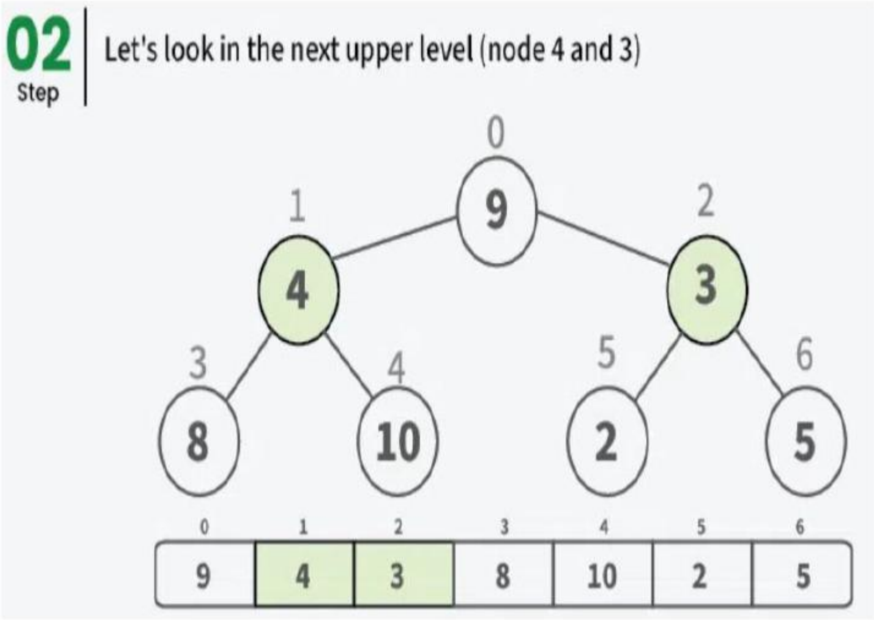
We first need to visualize the array as a **complete binary tree**



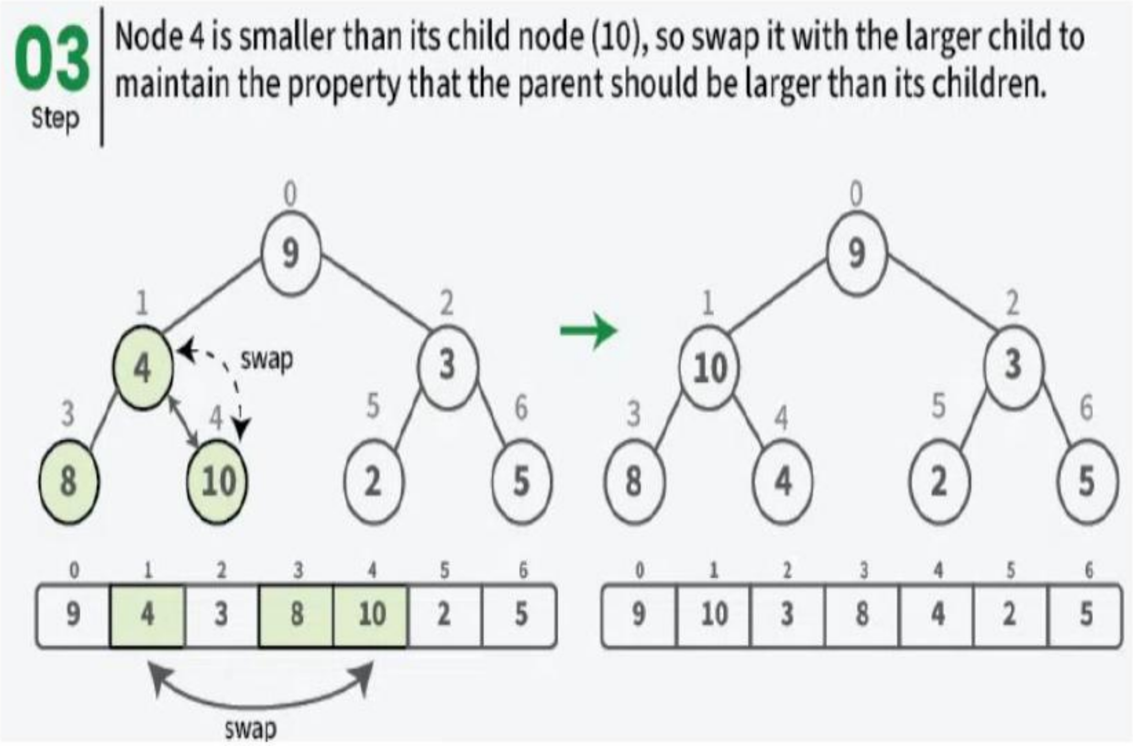
Unit III

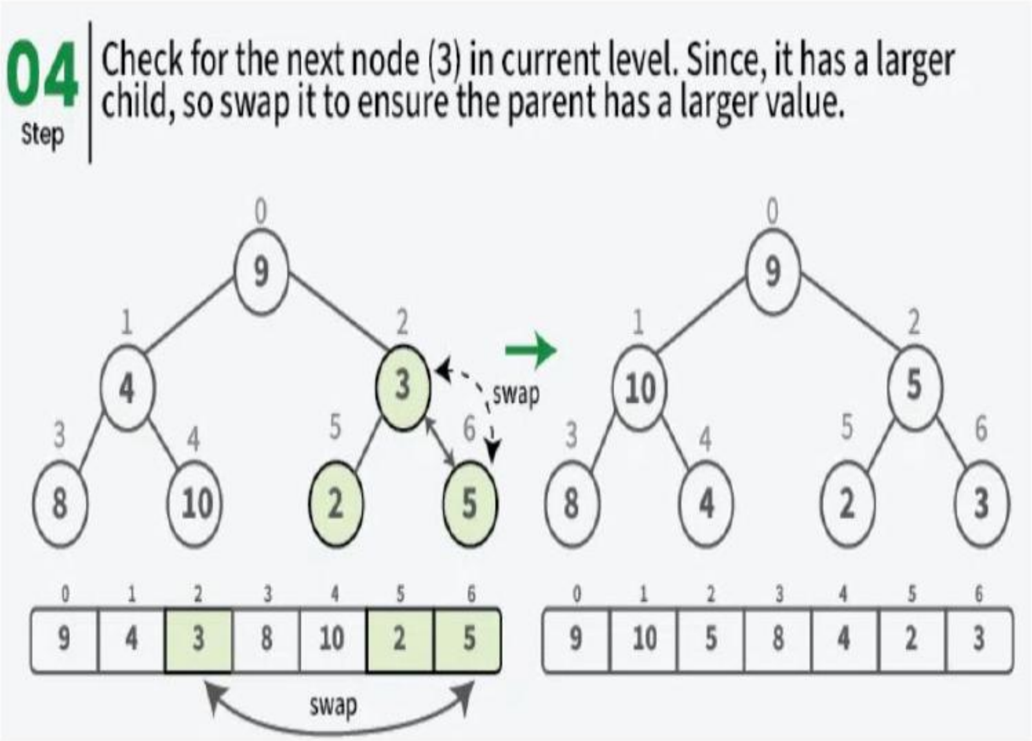
2.Build Heap Method (Max/Min)



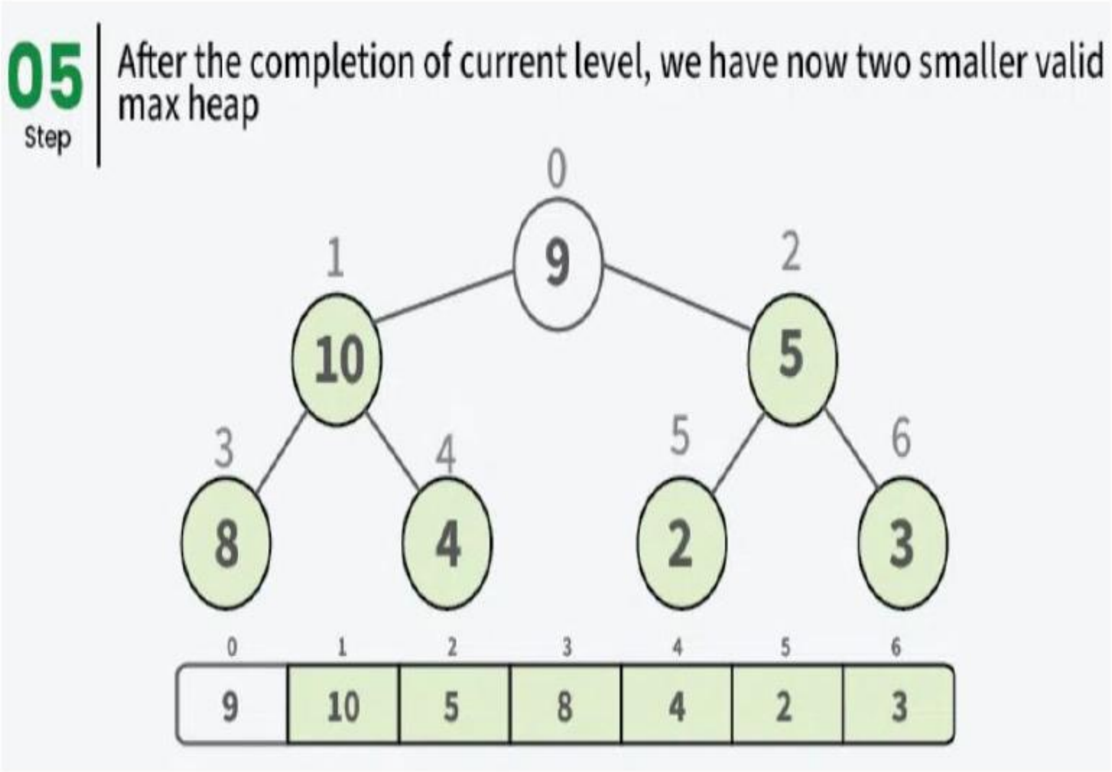


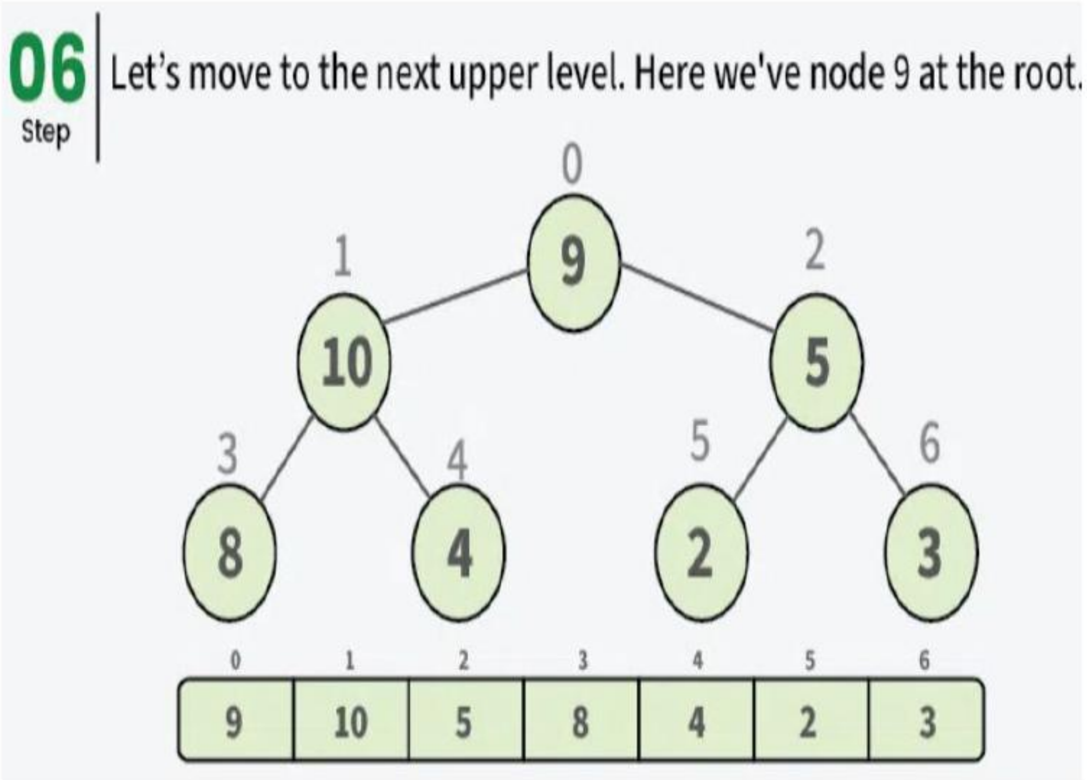
Unit III


03
Step
Node 4 is smaller than its child node (10), so swap it with the larger child to maintain the property that the parent should be larger than its children.
swap
swap

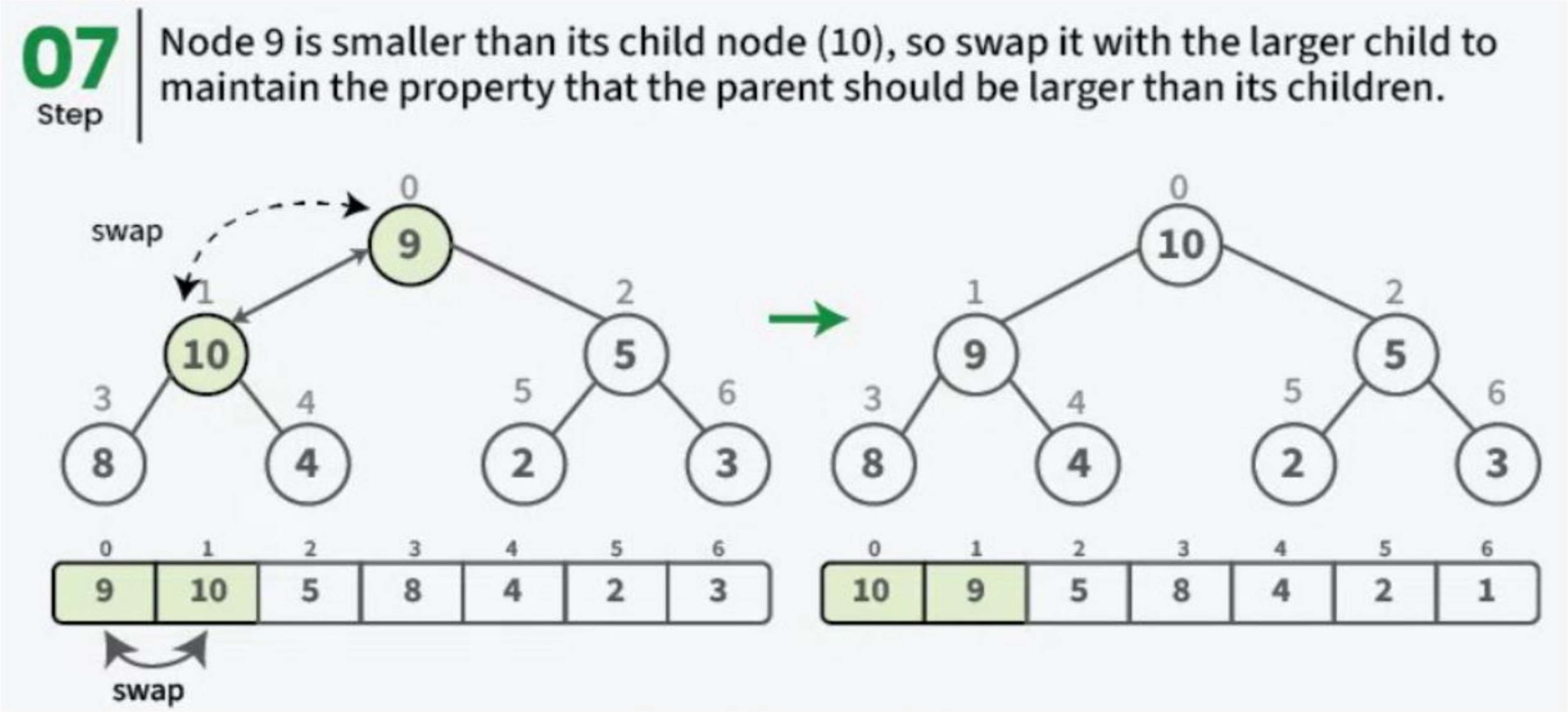

04
Step
Check for the next node (3) in current level. Since, it has a larger child, so swap it to ensure the parent has a larger value.
swap
swap

Unit III


05
Step
After the completion of current level, we have now two smaller valid max heap
0
9
1
10
2
5
3
8
4
4
5
2
6
3
0 1 2 3 4 5 6
9 10 5 8 4 2 3

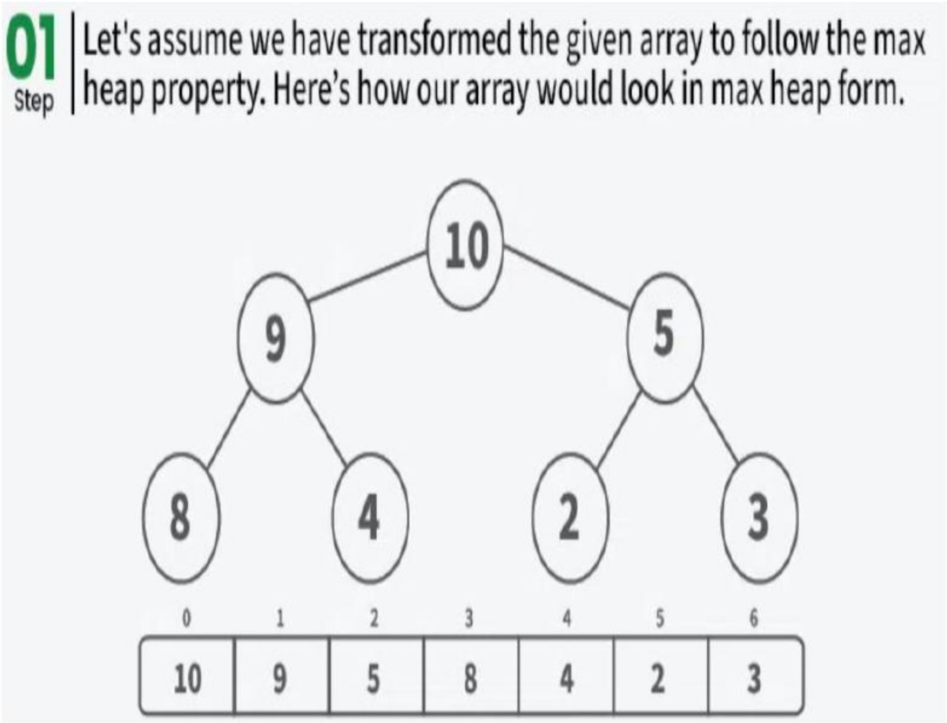

06
Step
Let's move to the next upper level. Here we've node 9 at the root.
0
9
1
10
2
5
3
8
4
4
5
2
6
3
0 1 2 3 4 5 6
9 10 5 8 4 2 3

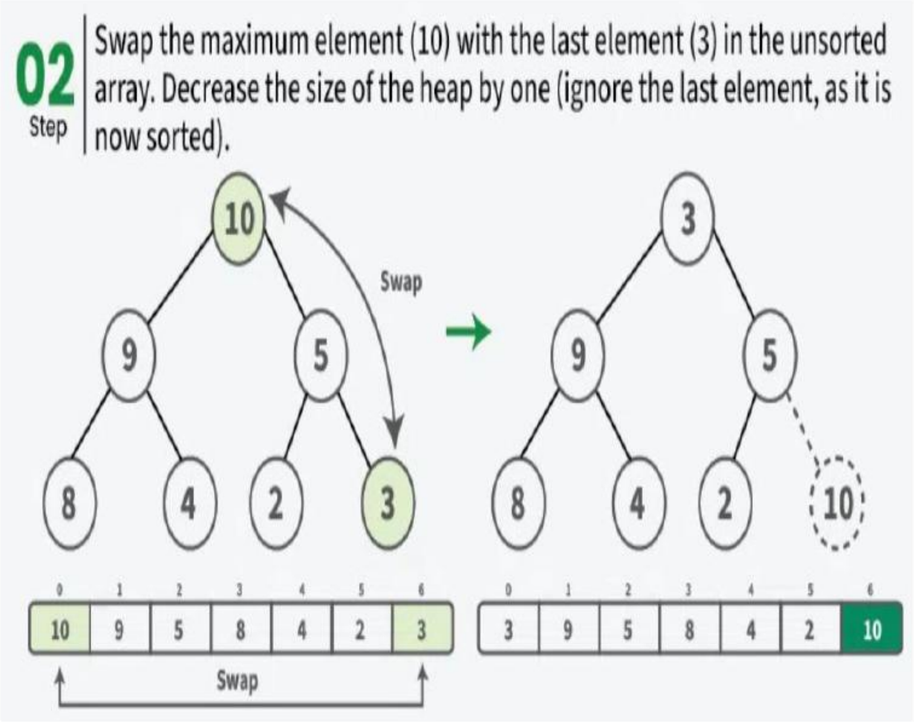
Unit III


07
Step
Node 9 is smaller than its child node (10), so swap it with the larger child to maintain the property that the parent should be larger than its children.
swap
0
9
1
10
2
5
3
8
4
4
5
2
6
3
0 1 2 3 4 5 6
9 10 5 8 4 2 3
swap
0
10
1
9
2
5
3
8
4
4
5
2
6
3
0 1 2 3 4 5 6
10 9 5 8 4 2 1

Unit III

3.Sort array by placing largest element at end of unsorted array



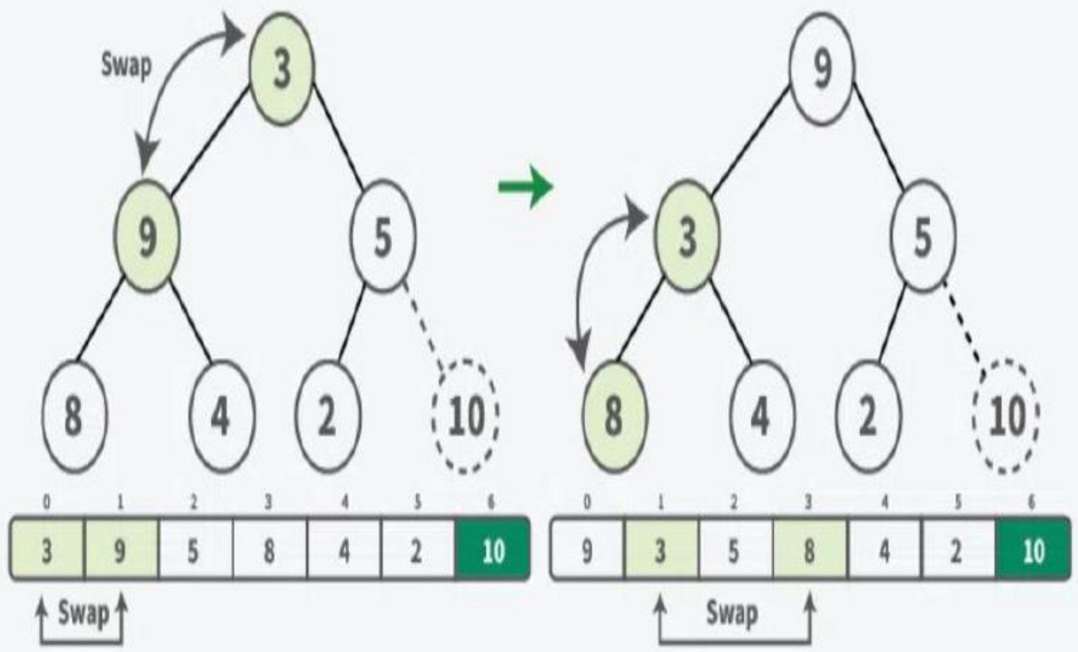


RAMAIAH Institute of Technology

Unit III

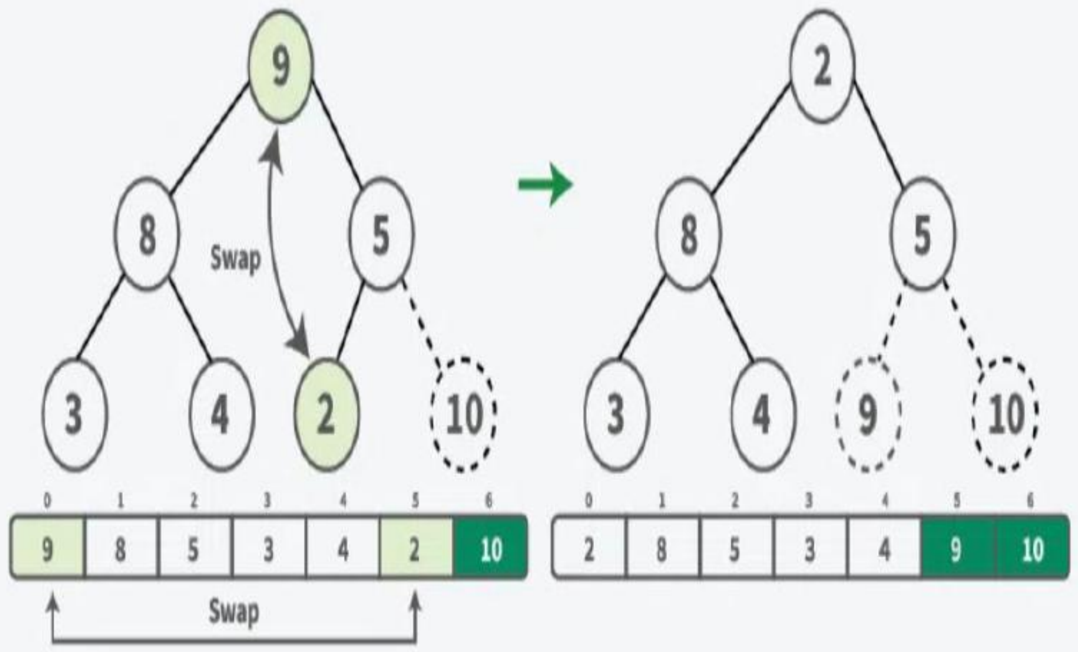
03 Step

Now, root violate the max-heap property, So, heapify it.
Swap node 3 with its largest child (9).
Still node 3 have larger children, so swap it with largest one (node 8).

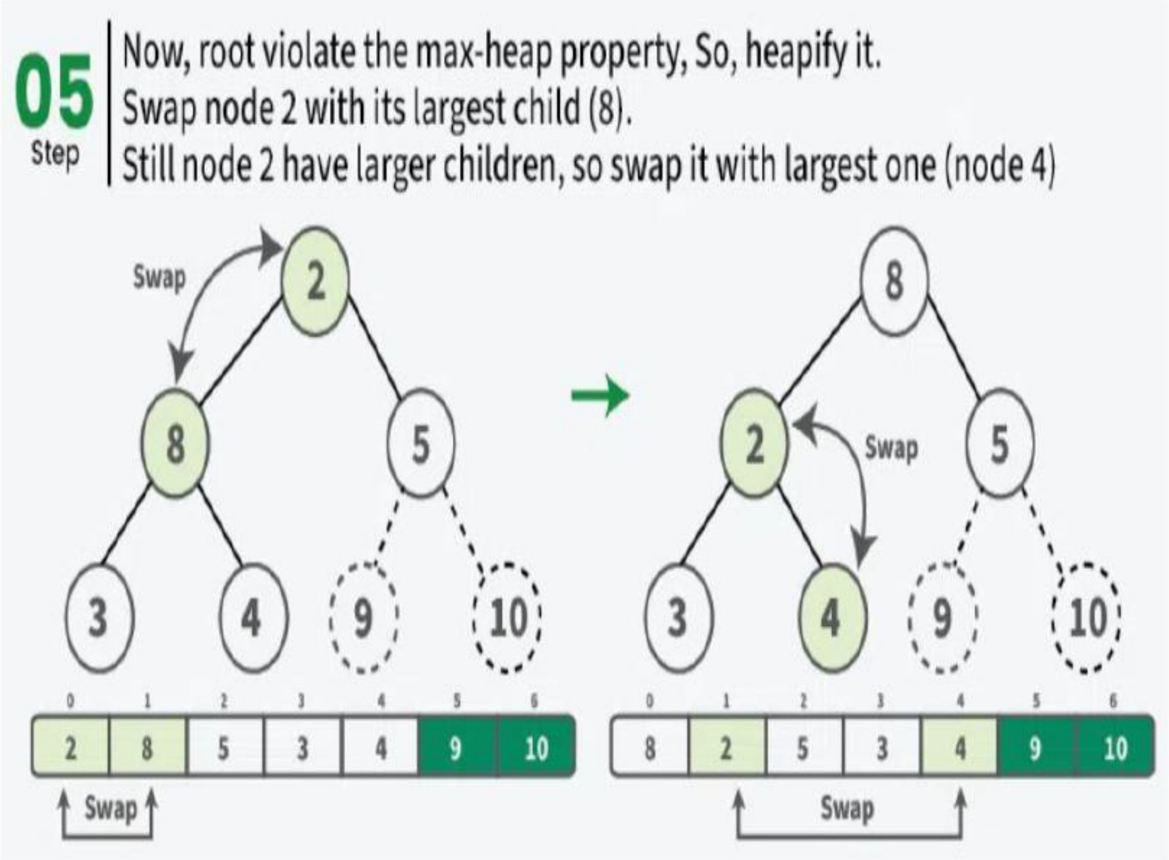


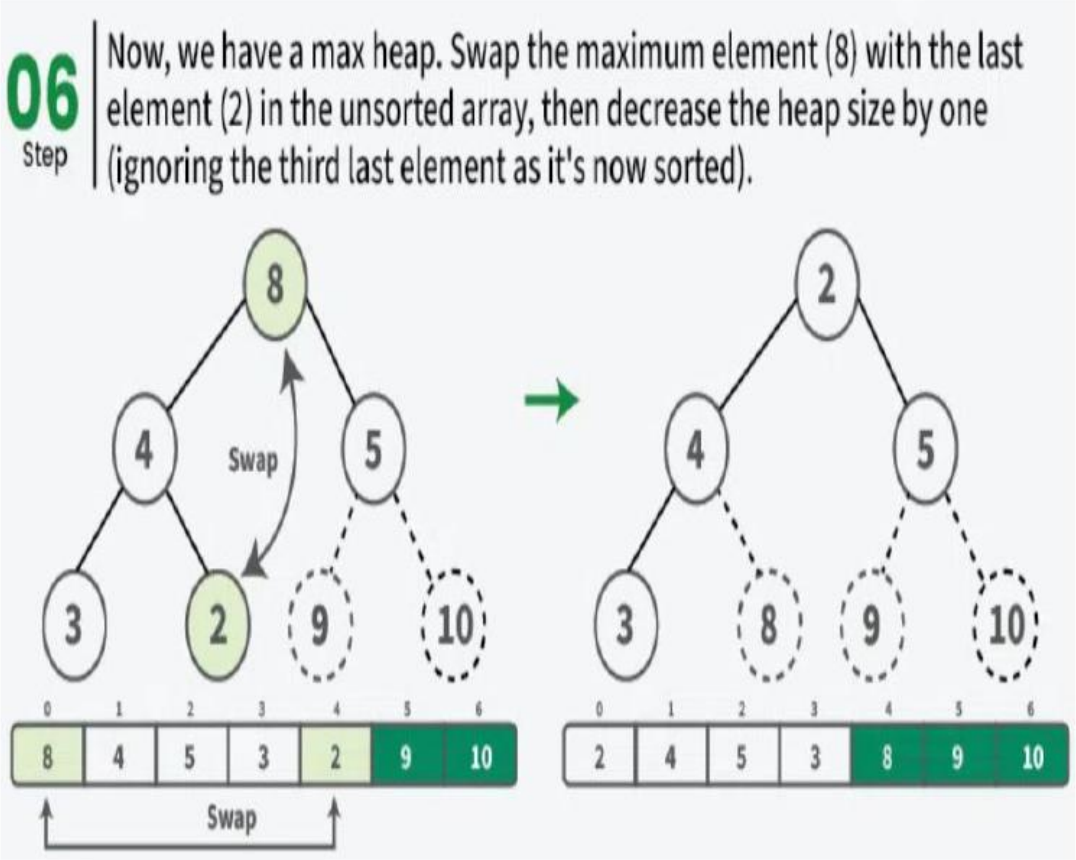
04 Step

Now, we have a max heap. Swap the maximum element (9) with the last element (2) in the unsorted array, then decrease the heap size by one (ignoring the second last element as it's now sorted).



Unit III


05
Step
Now, root violate the max-heap property, So, heapify it.
Swap node 2 with its largest child (8).
Still node 2 have larger children, so swap it with largest one (node 4)
Swap
Swap
Swap
Swap


06
Step
Now, we have a max heap. Swap the maximum element (8) with the last element (2) in the unsorted array, then decrease the heap size by one (ignoring the third last element as it's now sorted).
Swap
Swap

Unit III

Time Complexity Analysis of Heap Sort

Time Complexity: O(n log n)

Auxiliary Space: O(log n), due to the recursive call stack. However, auxiliary space can be O(1) for iterative implementation.

Unit III

Applications of Heaps

- **Heap Sort:** Heap Sort uses Binary Heap to sort an array in O(nLogn) time.
- **Priority Queue**: Priority queues can be efficiently implemented using Binary Heap because it supports insert(), delete() operations in O(log N) time.
- **Graph Algorithms:** The priority queues are especially used in Graph Algorithms like Dijkstra's Shortest Path and Prim's Minimum Spanning Tree.