ADVANCED DATA STRUCTURES AND ALGORITHMS

Heaps

Dr G.Kalyani

Department of Information Technology

VR Siddhartha Engineering College

Binary Heaps

- A **Heap is a Tree data structure that forms a complete binary tree**, and **satisfies the heap property**.

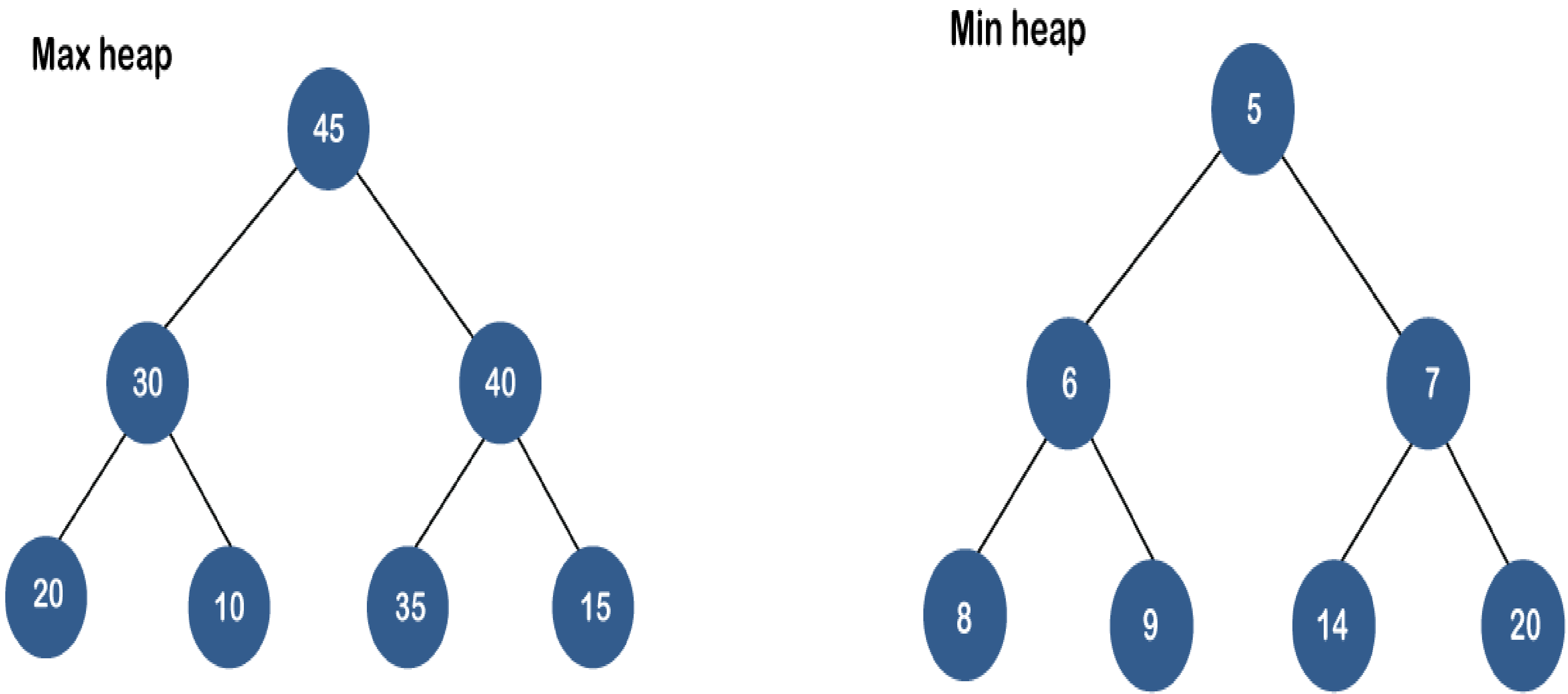
- **Heap Property:**
  - the value of the **parent node could be more than or equal to the value of the child node, or**
  - the value of the **parent node could be less than or equal to the value of the child node.**

- Therefore, we can say that there are **two types of heaps**:

  •**Max Heap:** The max heap is a heap in which the value of the parent node is greater than the value of the child nodes.

  •**Min Heap:** The min heap is a heap in which the value of the parent node is less than the value of the child nodes.

Binary Heaps



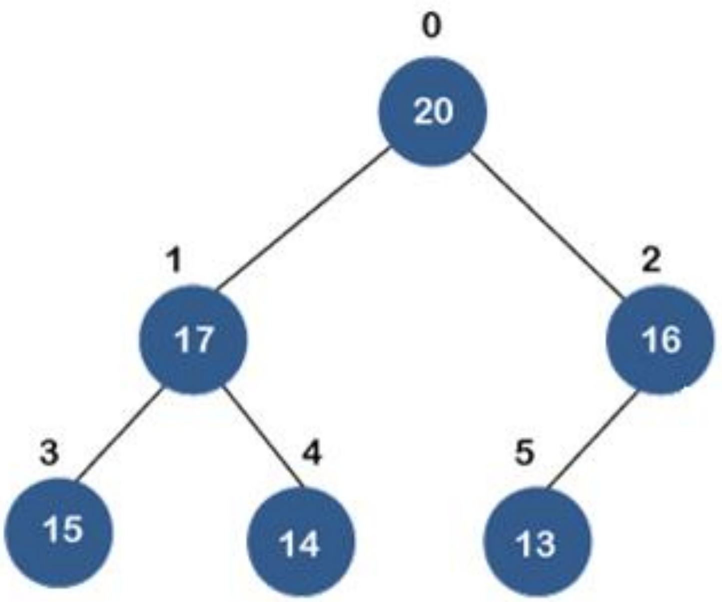
Operations on Max Heaps

- The common operations that we can perform on a Heap are
  - Insertion and
  - Deletion.

Inserting the element in a Max Heap

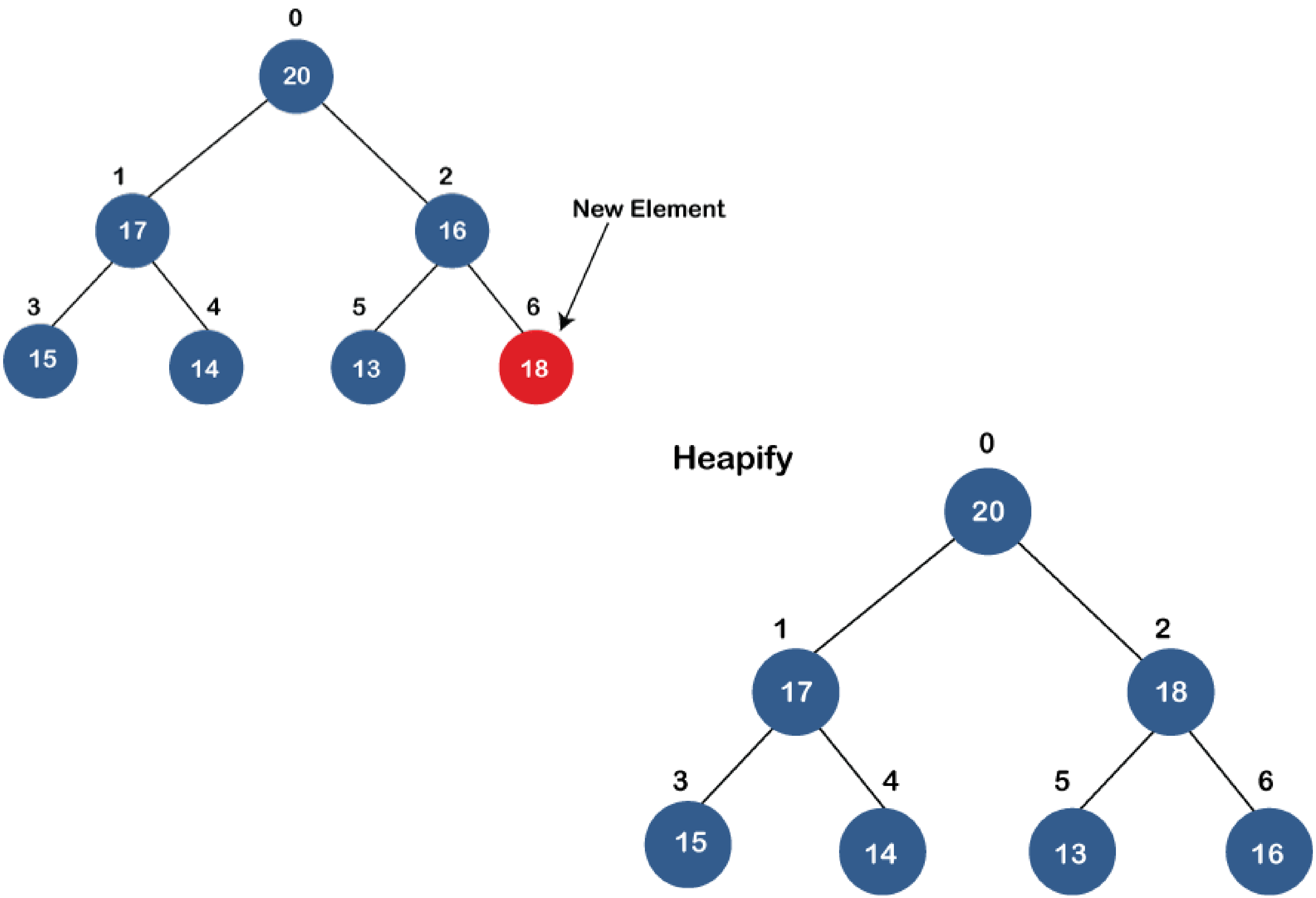
- If we insert an element in a Max Heap, it will **move to the empty slot by looking from top to bottom and left to right.**

- After the insertion, then **it is compared with the parent node; if it is found out of order, elements are swapped.** This process continues until the element is placed in a correct position.



Insert 18

Inserting the element in a Max Heap



MAX Heap Construction/Insertion

- **Step 1** – Create a new node at the end of heap.
- **Step 2** – Assign new value to the node.
- **Step 3** – Compare the value of this child node with its parent.
- **Step 4** – If value of parent is less than child, then swap them.
- **Step 5** – Repeat step 3 & 4 until Heap property holds up to the root.

Delete Element from Heap

- **Deletion in Max (or Min) Heap always happens at the root** to remove the Maximum (or minimum) value.

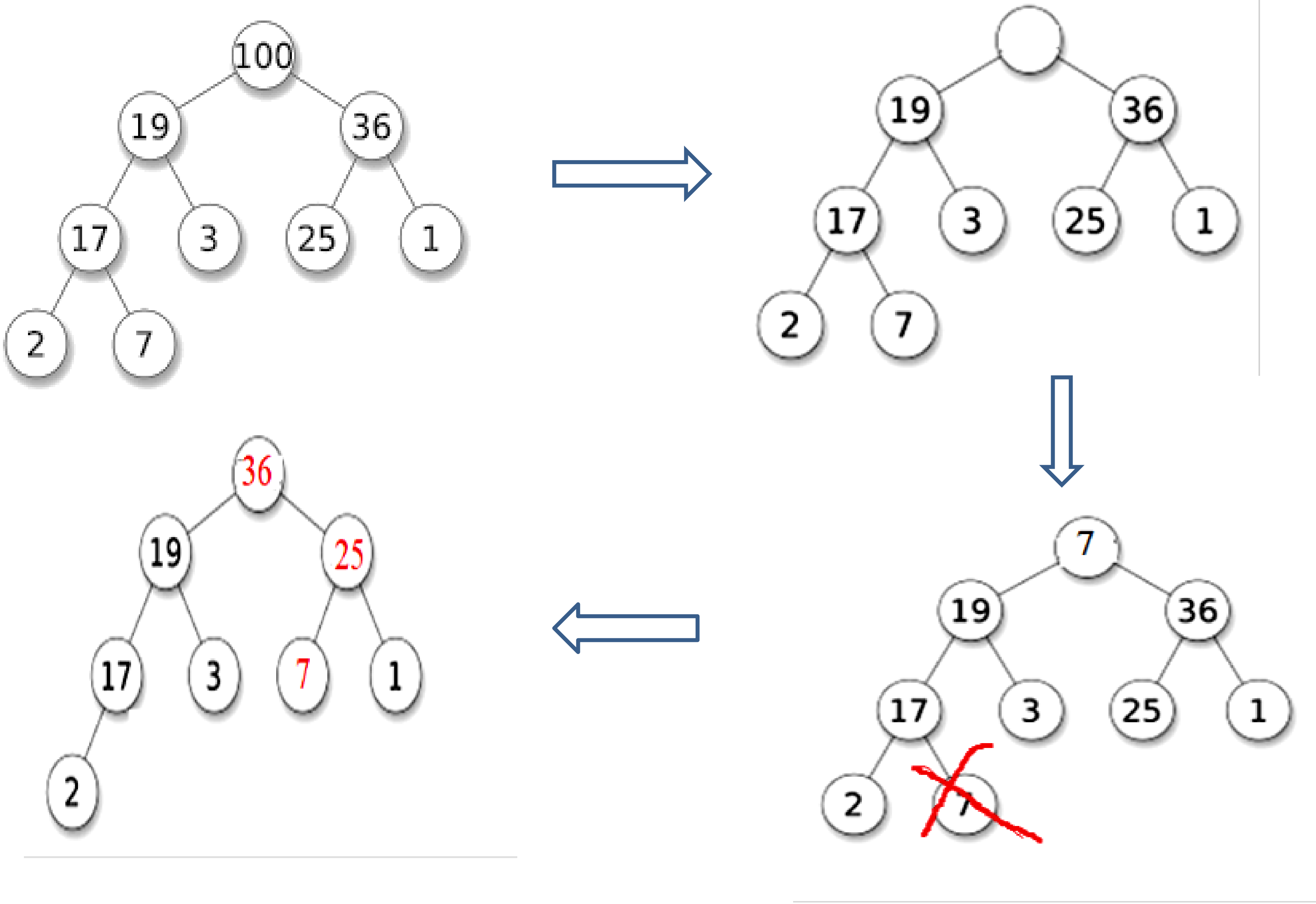
- As we know that in a max heap, the maximum element is the root node. When we **remove the root node, it creates an empty slot.**

- The **last element in the heap will be added in this empty slot**.

- Then, this **element is compared with the child nodes**, i.e., left-child and right child, and **swap with the Larger of the two**.

- It keeps **moving down the tree until the heap property is restored.**

Delete Element from Heap



MAX Heap Deletion

- **Step 1** – Remove root node.
- **Step 2** – Move the last element of last level to root.
- **Step 3** – Compare the value of this parent node with its child nodes.
- **Step 4** – If value of parent is less than child, then swap with larger child.
- **Step 5** – Repeat step 3 & 4 until Max Heap property holds.

Example on Heap

- Create a **MAX HEAP** for Input
  - 35 33 42 10 14 19 27 44 26 31
  - Apply 4 delete operations on the constructed heap.

- Create a **MIN HEAP** for Input
  - 35 33 42 10 14 19 27 44 26 31
  - Apply 4 delete operations on the constructed heap.

Application: Heap Sort

Heap Sort Algorithm

- For sorting in increasing order:

- **Step 1:** Build a max heap from the input data.
- **Step 2:** At this point, the largest item is stored at the root of the heap. Replace it with the last item of the heap followed by reducing the size of heap by 1.
- **Step 3:** heapify from the root of the tree.
- **Step 4:** Repeat step 2 & 3 while size of heap is greater than 1.

Heap Sort Example

- Sort the following elements in ascending order:

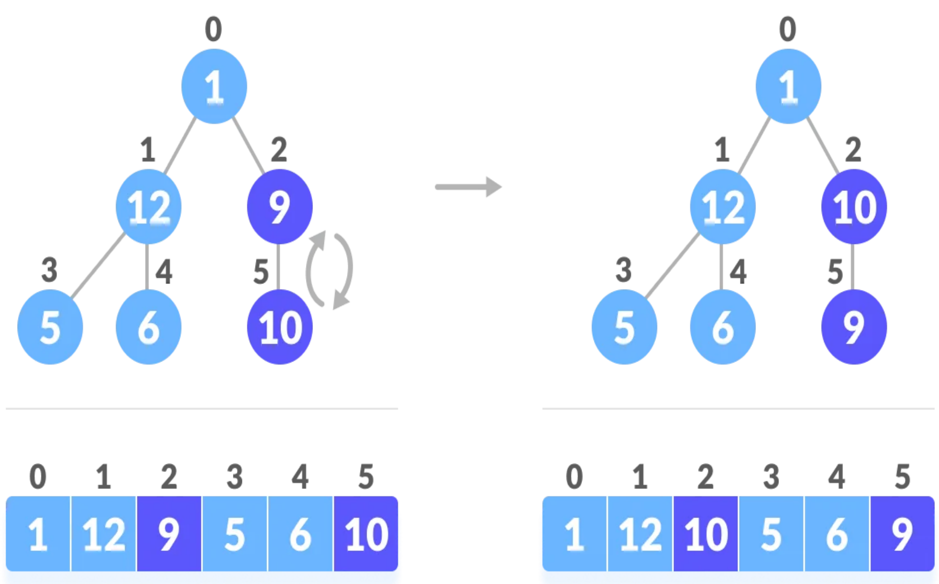
  1, 12, 9, 5, 6, 10

- Step 1: Build Max Heap

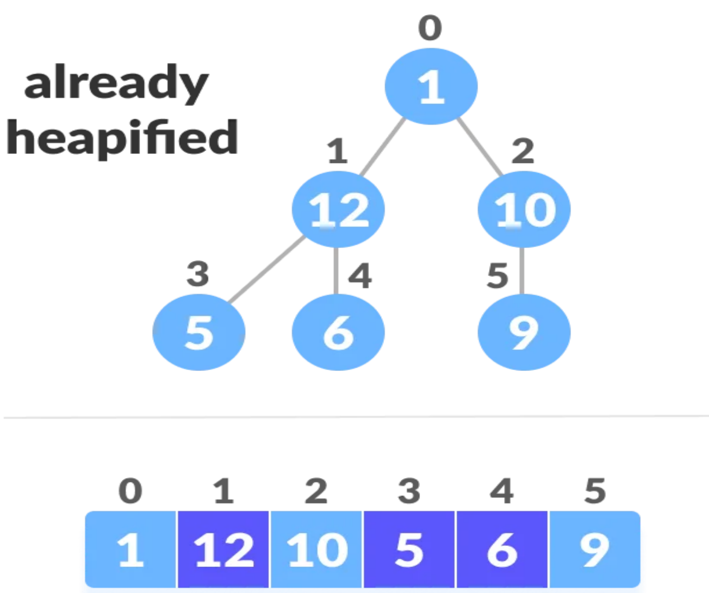
Heap Sort Example

1, 12, 9, 5, 6, 10

i = 2 → heapify(arr, 6, 2)

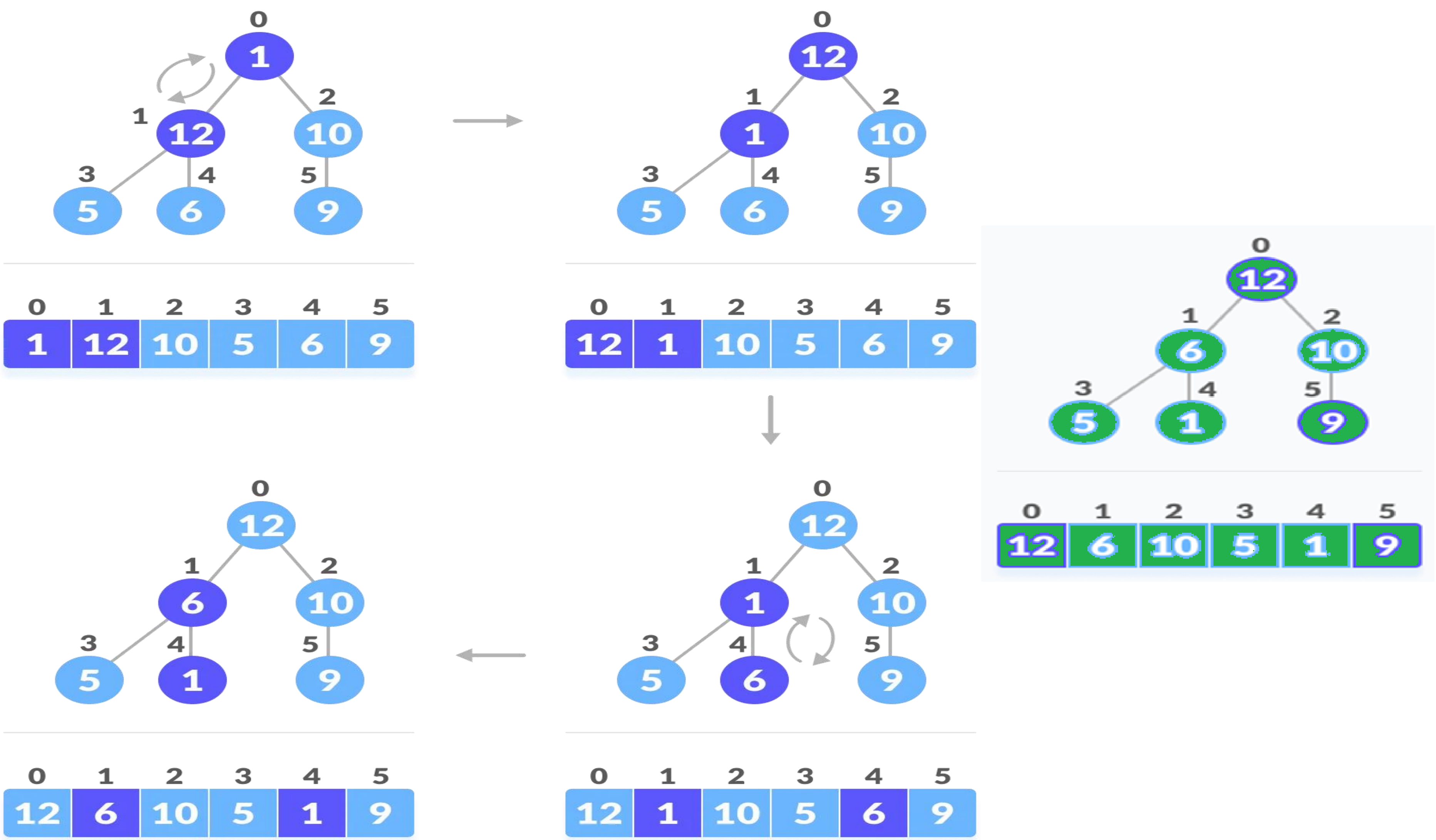


i = 1 → heapify(arr, 6, 1)

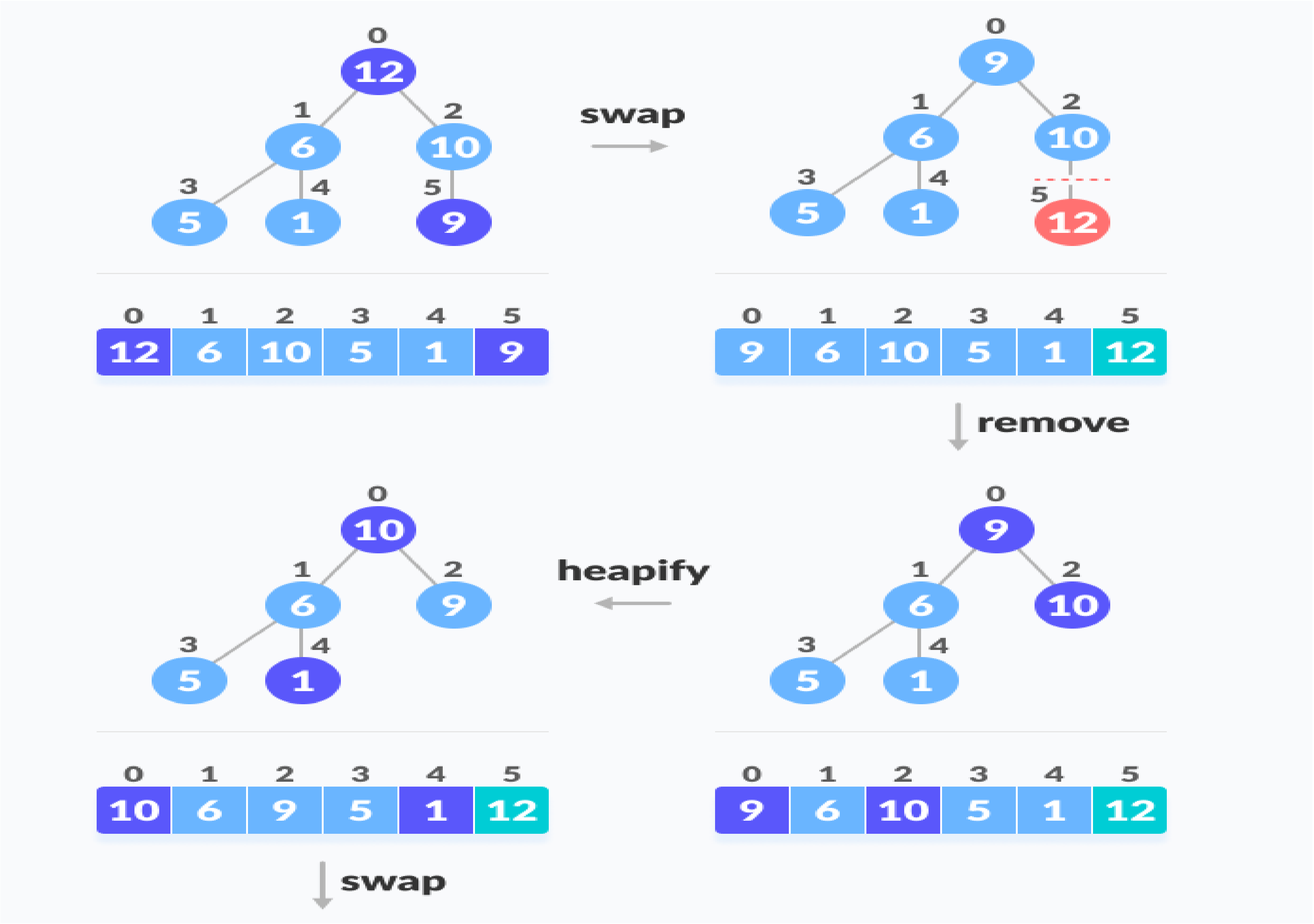


Heap Sort Example

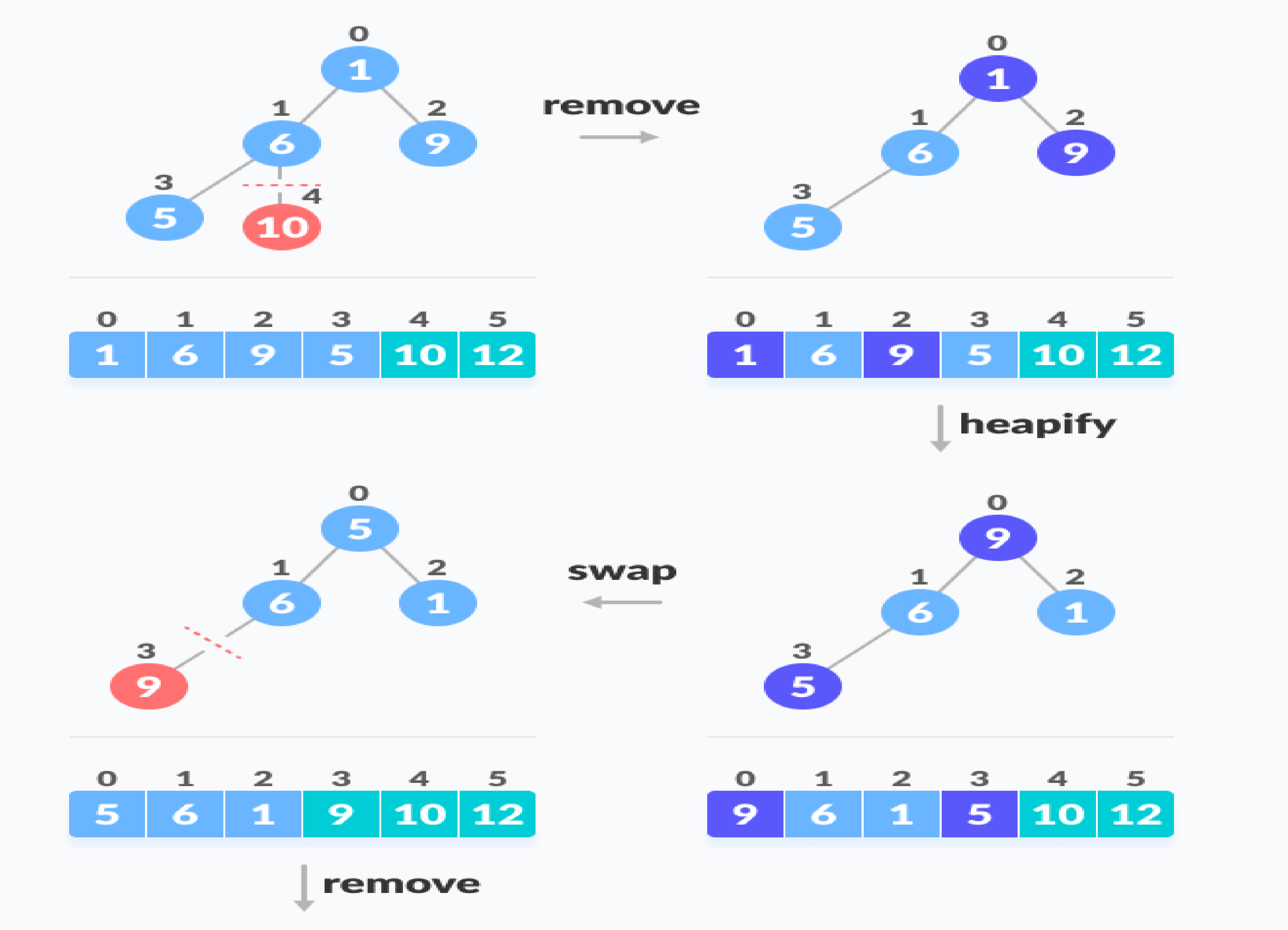
i = 0 → heapify(arr, 6, 0)



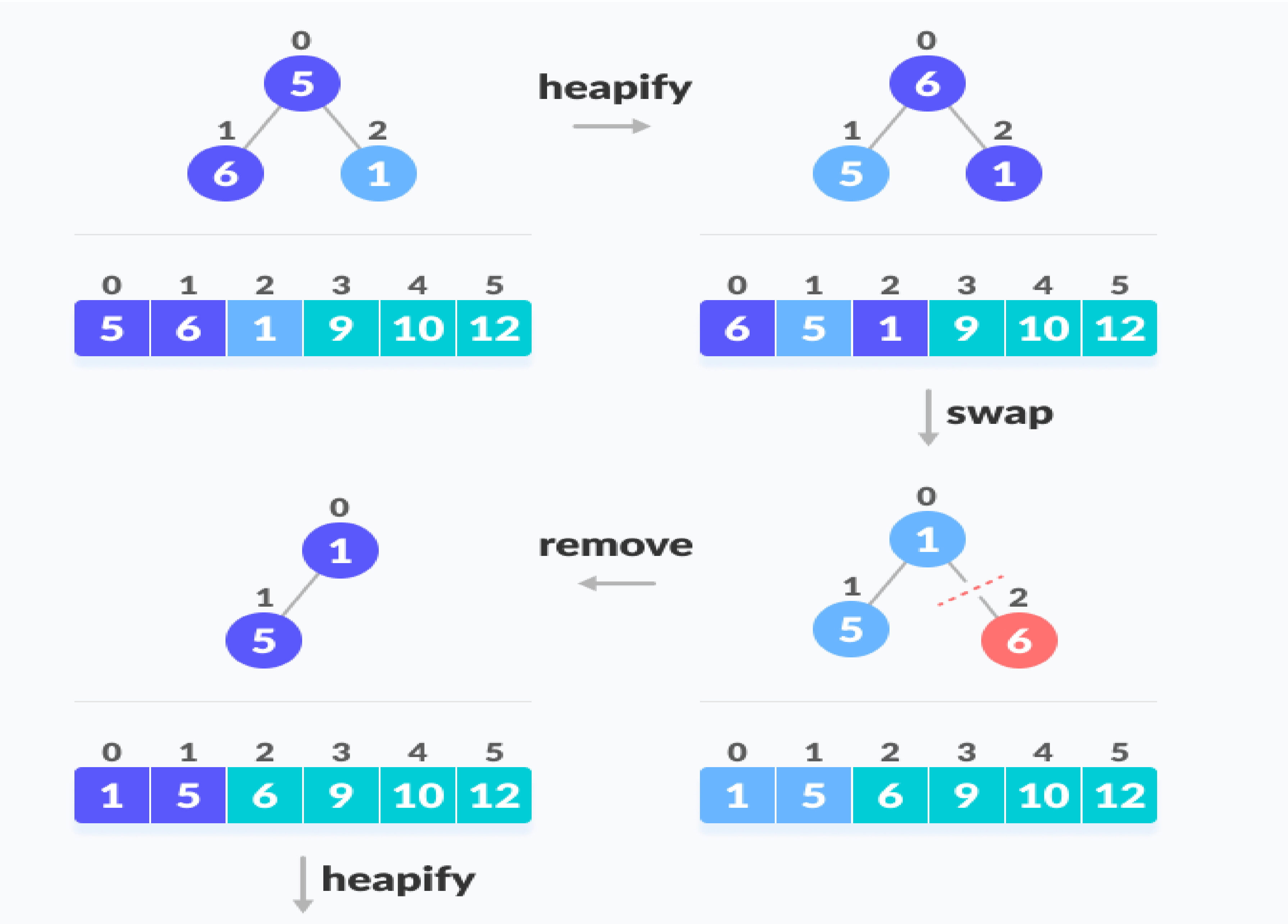
Heap Sort Example



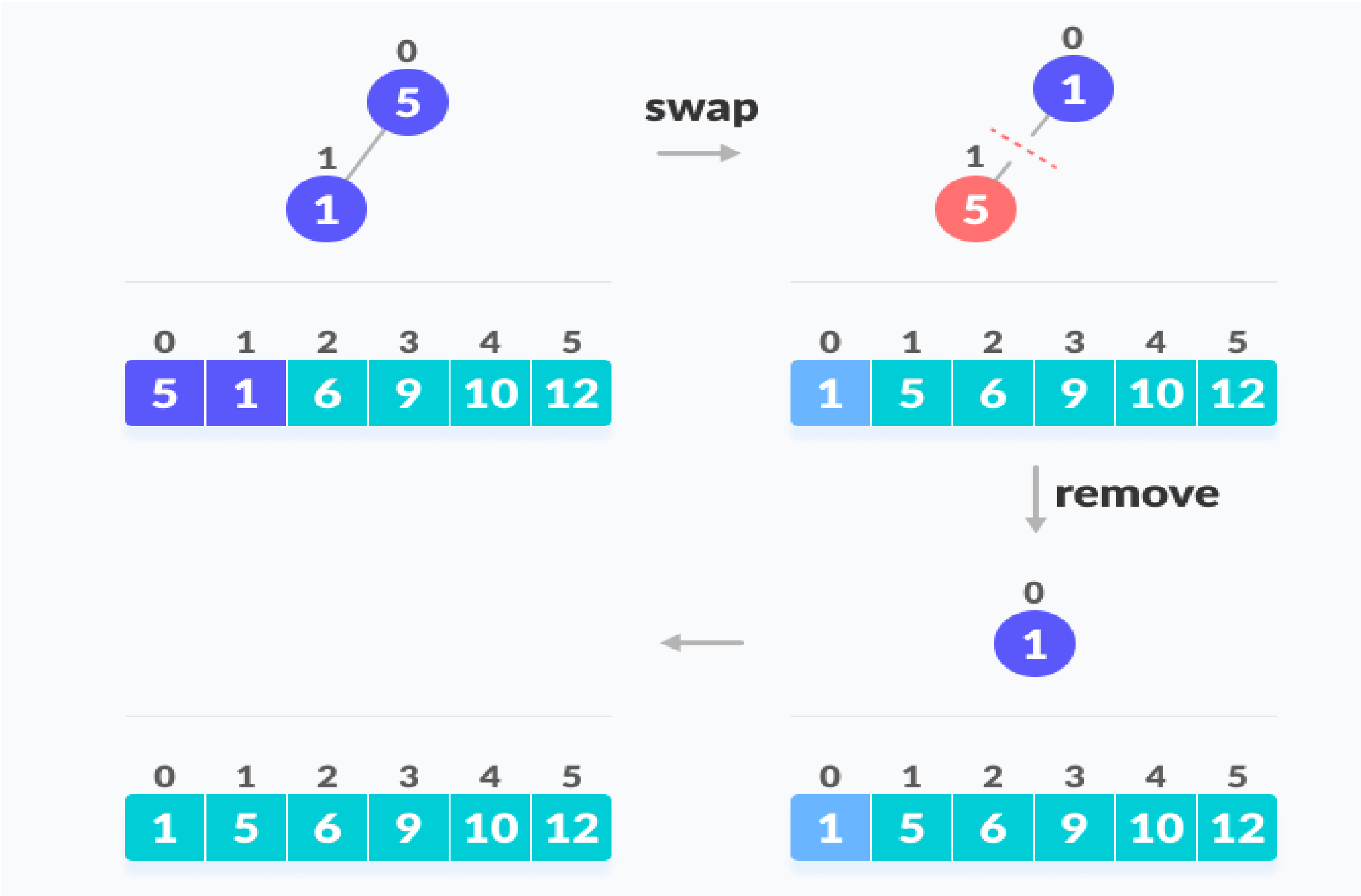
Heap Sort Example



Heap Sort Example



Heap Sort Example



Algorithm for Heap Sort

```
Algorithm heapSort(arr[], n)
{
                                                // Build heap (rearrange array)
   for i = (n/2 – 1) to 0 step -1
          heapify(arr, n, i);

                                                // to sort the elements
   for i = n – 1 to 1 step -1
   {
          swap(arr[0], arr[i]);                 // Move current root to end
          heapify(arr, i, 0);                   // call max heapify on the reduced heap
   }
}
```

```
Algorithm heapify(arr[],size, i)
{
   largest = i;                                   // Initialize largest as root
   left = 2 * i + 1;                                // left = 2*i + 1
   right = 2 * i + 2;                               // right = 2*i + 2

  if (left < size && arr[left] > arr[largest])         // If left child is larger than root
        largest = left;

  if (right < size && arr[right] > arr[largest])  // If right child is larger than largest so far
        largest = right;

  if (largest != i)                              // If largest is not root
     {
        swap (arr[i], arr[largest]);
        heapify (arr, size, largest);     // Recursively heapify the affected sub-tree
     }
}
```

Time complexity of Heap Sort

Heap Sort Complexity

Time Complexity	
Best	O(nlog n)
Worst	O(nlog n)
Average	O(nlog n)
Space Complexity	O(1)

Exercise on Heap Sort

- Sort the following elements using Heap sort:

  1, 6, 3, 2, 5, 8, 9, 12, 4, 10

8. Write the heap sort algorithm to sort a set of integers. **15M**

8. a. Illustrate deletion of any two elements in a max heap with atleast 10 elements. **9M**

   b. Compare the performance of the following sorting methods:
   i) Quick ii) Merge iii) Heap **6M**

   b. Discuss about max heap and min heap, with an example. **9M**

   b. How do we delete an element from a Max Heap? **7M**

   a. Write a procedure for Heap sort with an example. **8M**

   b. How do you insert an element into maxheap? **8M**

   b. Write a C program to insertion an element into Max Heap. **5M**

   b. What is Priority Queue? Explain with a suitable example implementation of maximum priority queue using max heap. **8M**

   b. Define max heap. Show the step-by-step construction of a max heap resulting from the insertion of the following sequence of keys: 6, 5, 3, 1, 8, 7, 2 and 4. **7M**