Heap Sort

Heap Sort

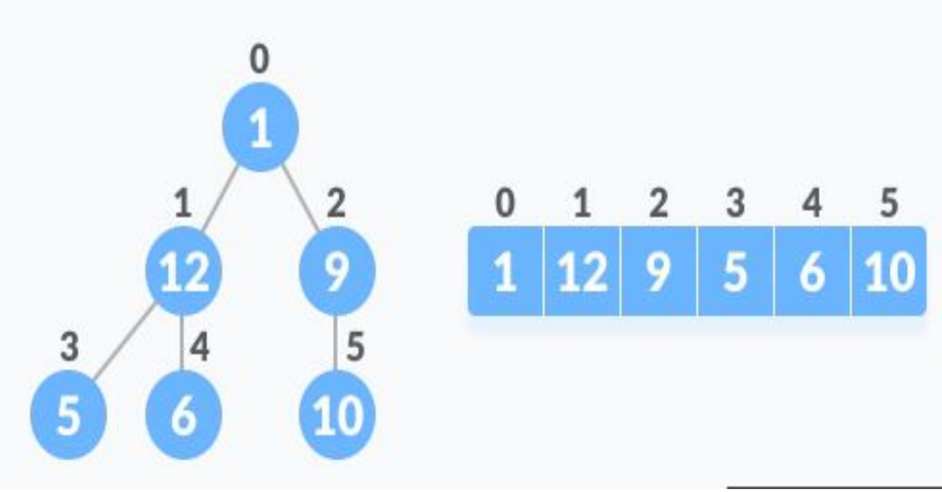
- Heap Sort is a popular and efficient sorting algorithm in computer programming.
- The initial set of numbers that we want to sort is stored in an array
- e.g. [10, 3, 76, 34, 23, 32] and after sorting, we get a sorted array [3,10,23,32,34,76].
- Heap sort works by visualizing the elements of the array as a special kind of complete binary tree called a heap.

https://www.programiz.com/dsa/heap-sort

Complete Binary Tree

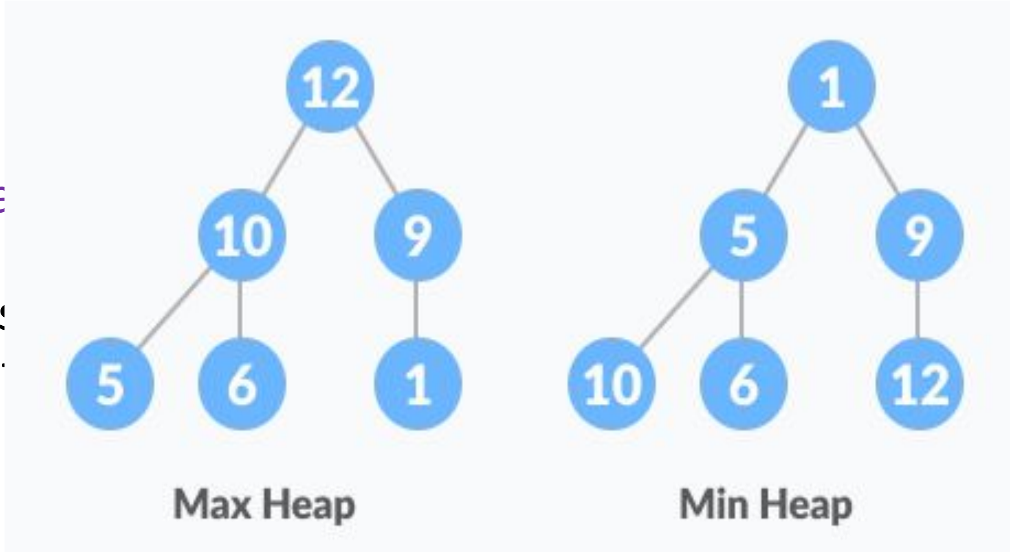
- A complete binary tree is a binary tree where all levels are completely filled except for the last level, which is filled from left to right
- A complete binary tree has an interesting property that we can use to find the children and parents of any node.

- If the index of any element in the array is i,
  - the element in the index 2i+1 will become the left child
  - The element in 2i+2 index will become the right child.
  - Also, the parent of any element at index i is given by the lower bound of (i-1)/2.



What is Heap Data Structure?

- Heap is a special tree-based data structure.
- A binary tree is said to follow a heap data structure if
  - it is [a complete binary tree]
  - All nodes in the tree follow the property that they are greater tha their children i.e. the largest element is at the root and both its children and smaller than the roo and so on.
  - Such a heap is called a max-heap.
  - If instead, all nodes are smaller than their children, it is called a min-heap
- The following example diagram shows Max-Heap and Min-Heap.



How to "heapify" a tree

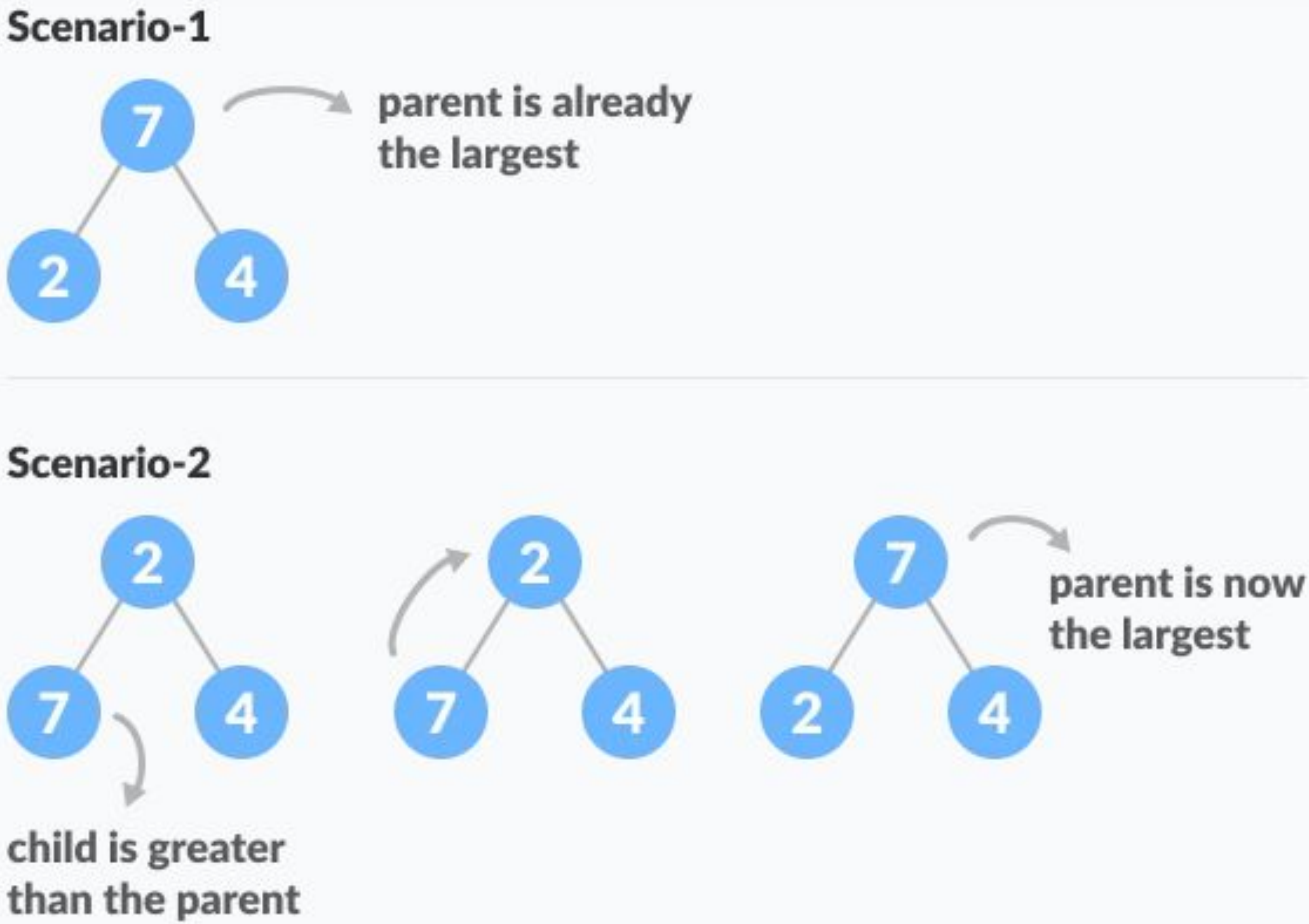
- Starting from a complete binary tree, we can modify it to become a Max-Heap by running a function called heapify on all the non-leaf elements
- heapify(array)

  Root = array[0]

  Largest = largest(array[0] , array [2*0 + 1]. array[2*0+2])

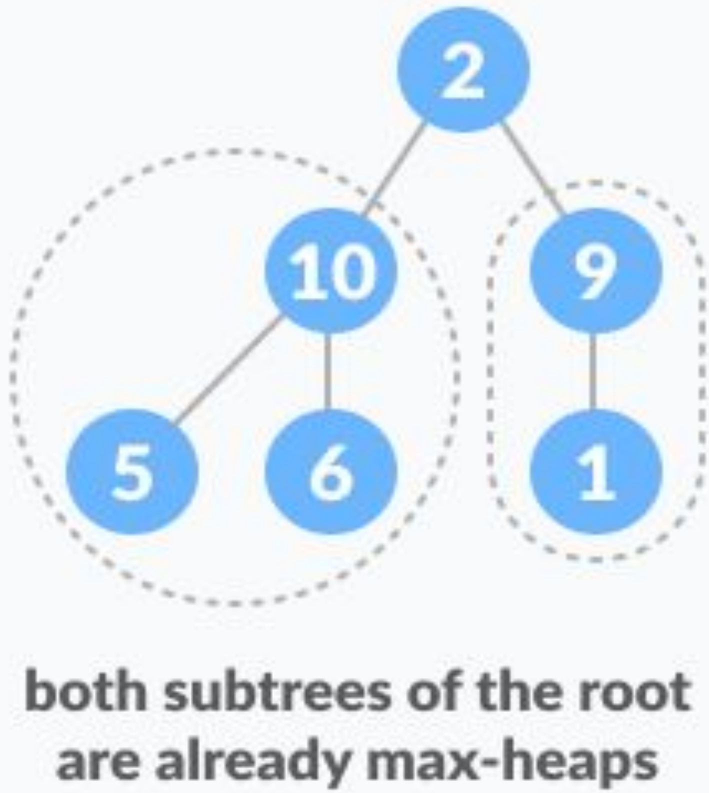
  if(Root != Largest)

  Swap(Root, Largest)of the heap.

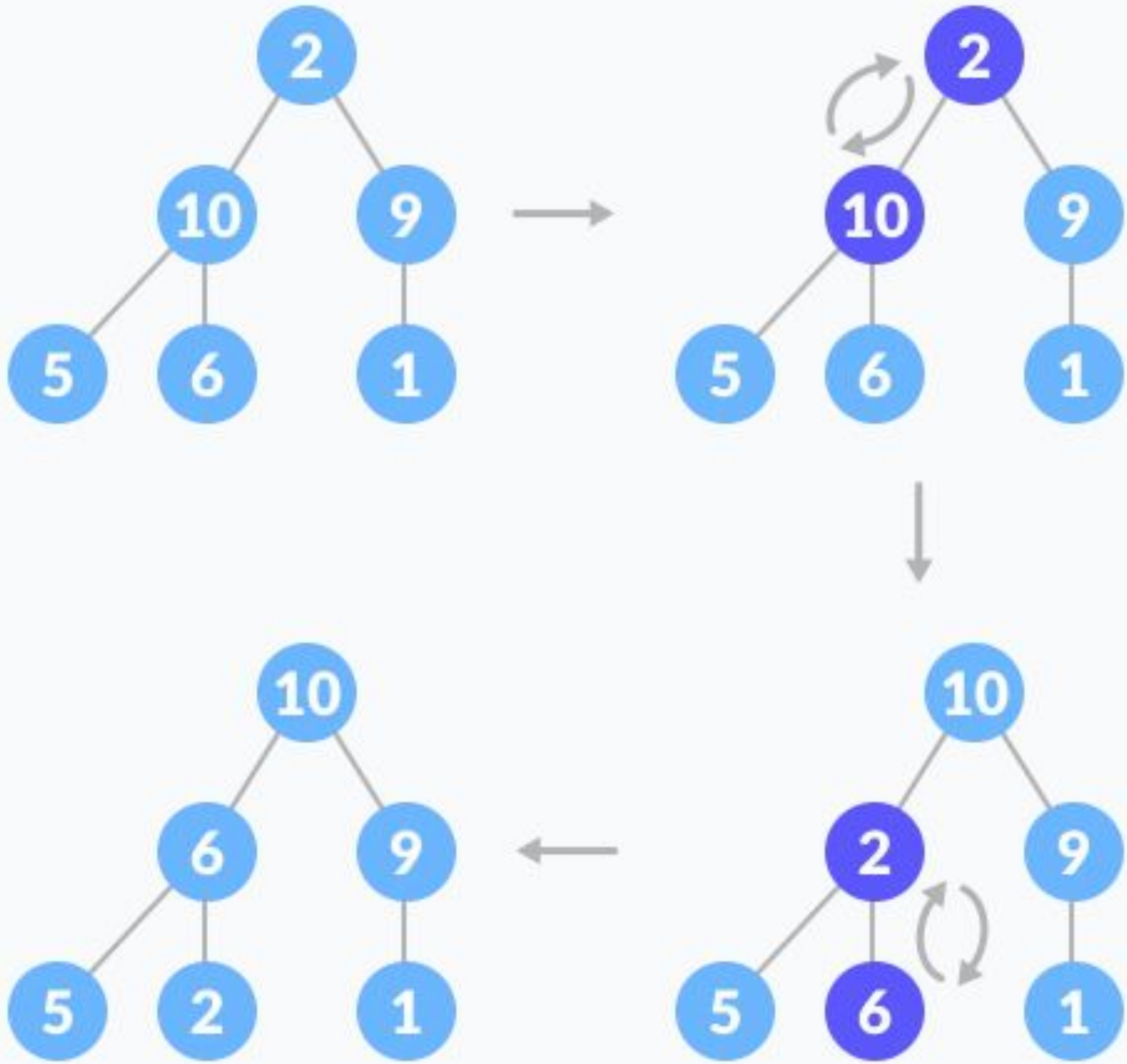


Heapify base cases

How to heapify root element when its subtrees are already max heaps



How to heapify root element when its subtrees are already max heaps



How to heapify root element when its subtrees are max-heaps

- Thus, to maintain the max-heap property in a tree where both sub-trees are max-heaps, we need to run heapify on the root element repeatedly until it is larger than its children or it becomes a leaf node.

Build max-heap

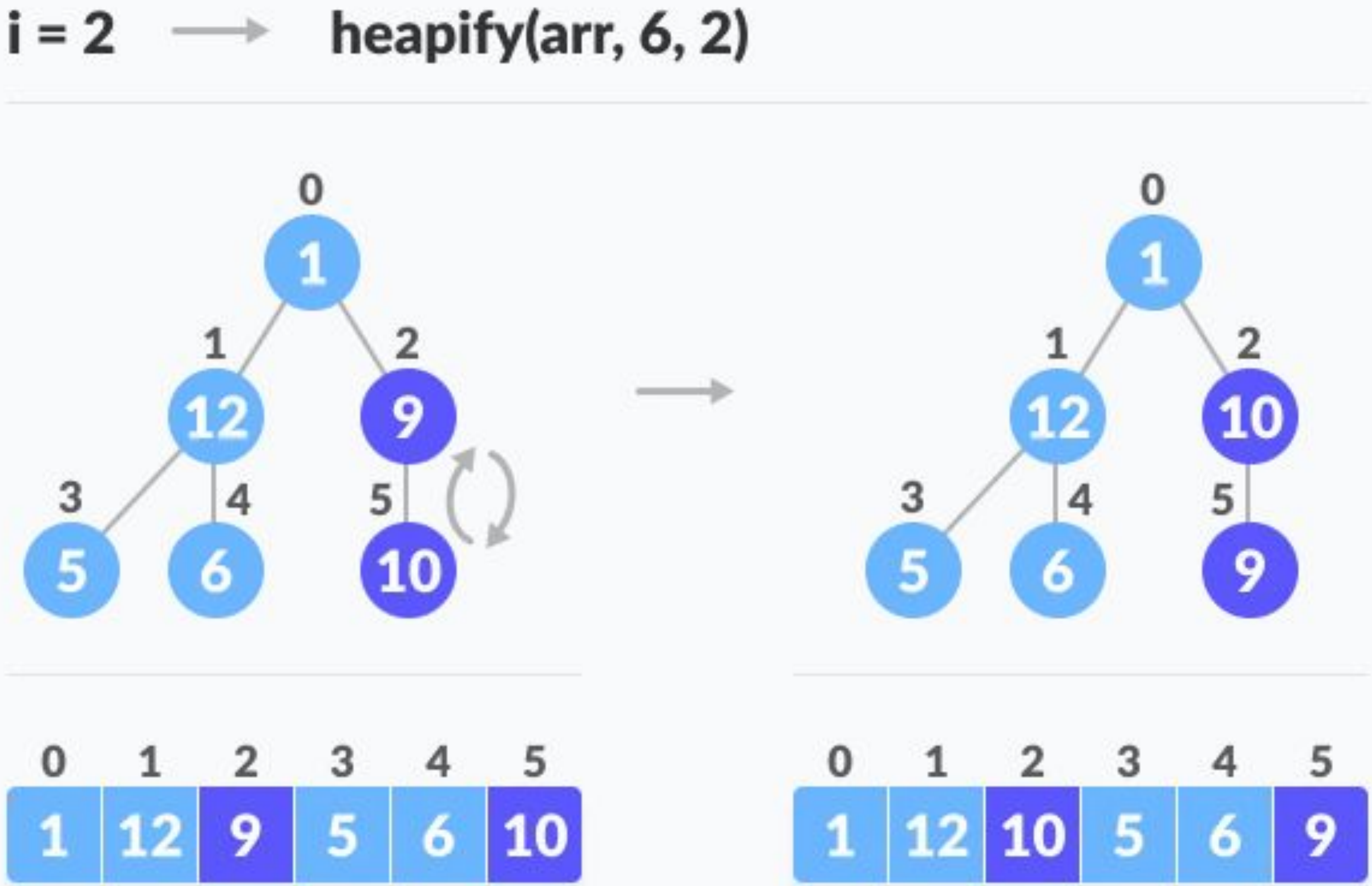
- To build a max-heap from any tree, we can thus start heapifying each sub-tree from the bottom up and end up with a max-heap after the function is applied to all the elements including the root element.
- In the case of a complete tree, the first index of a non-leaf node is given by n/2 - 1.
- All other nodes after that are leaf-nodes and thus don't need to be heapified.

	0	1	2	3	4	5
arr	1	12	9	5	6	10

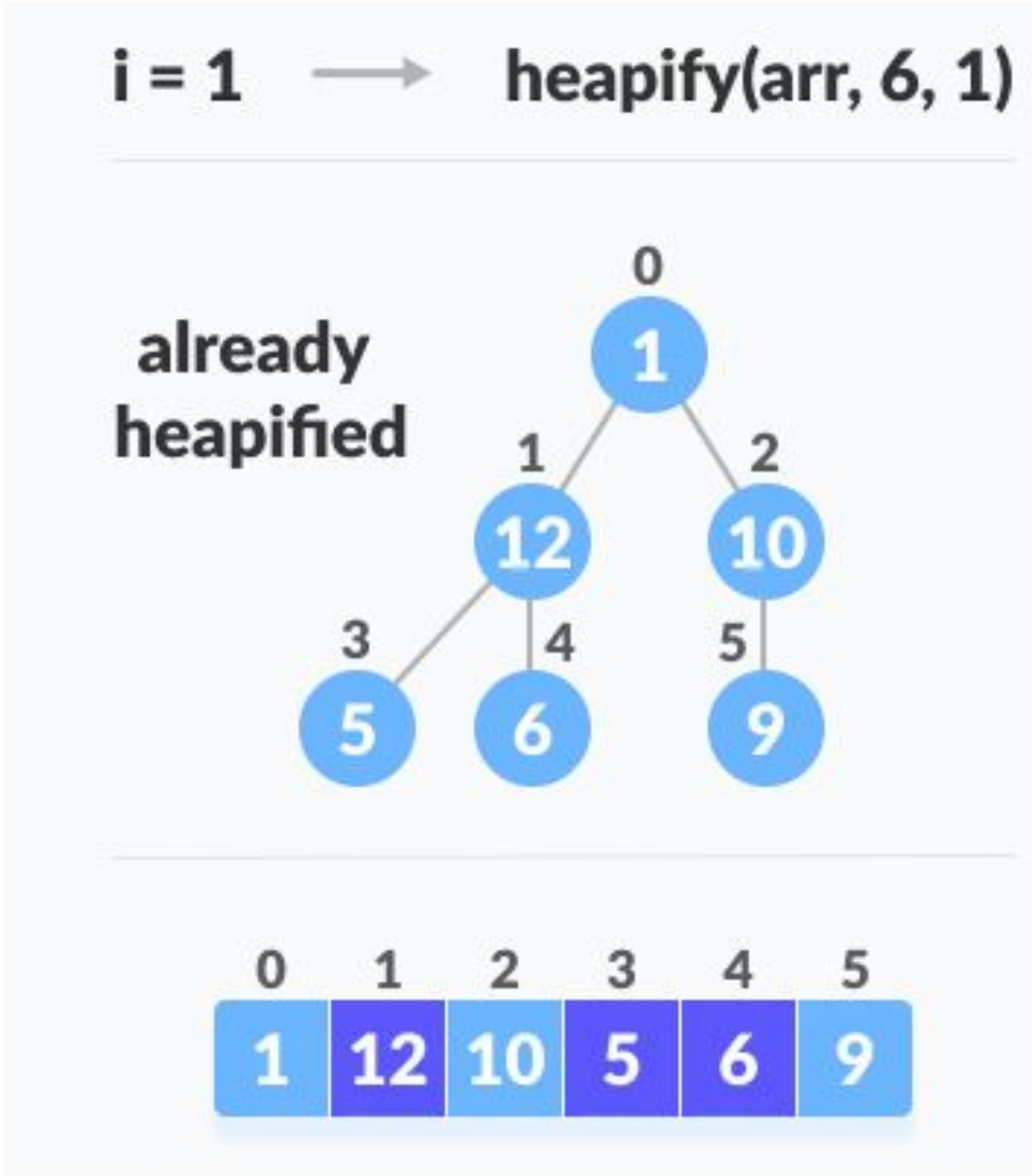
n = 6

i = 6/2 - 1 = 2 # loop runs from 2 to 0

Create array and calculate i

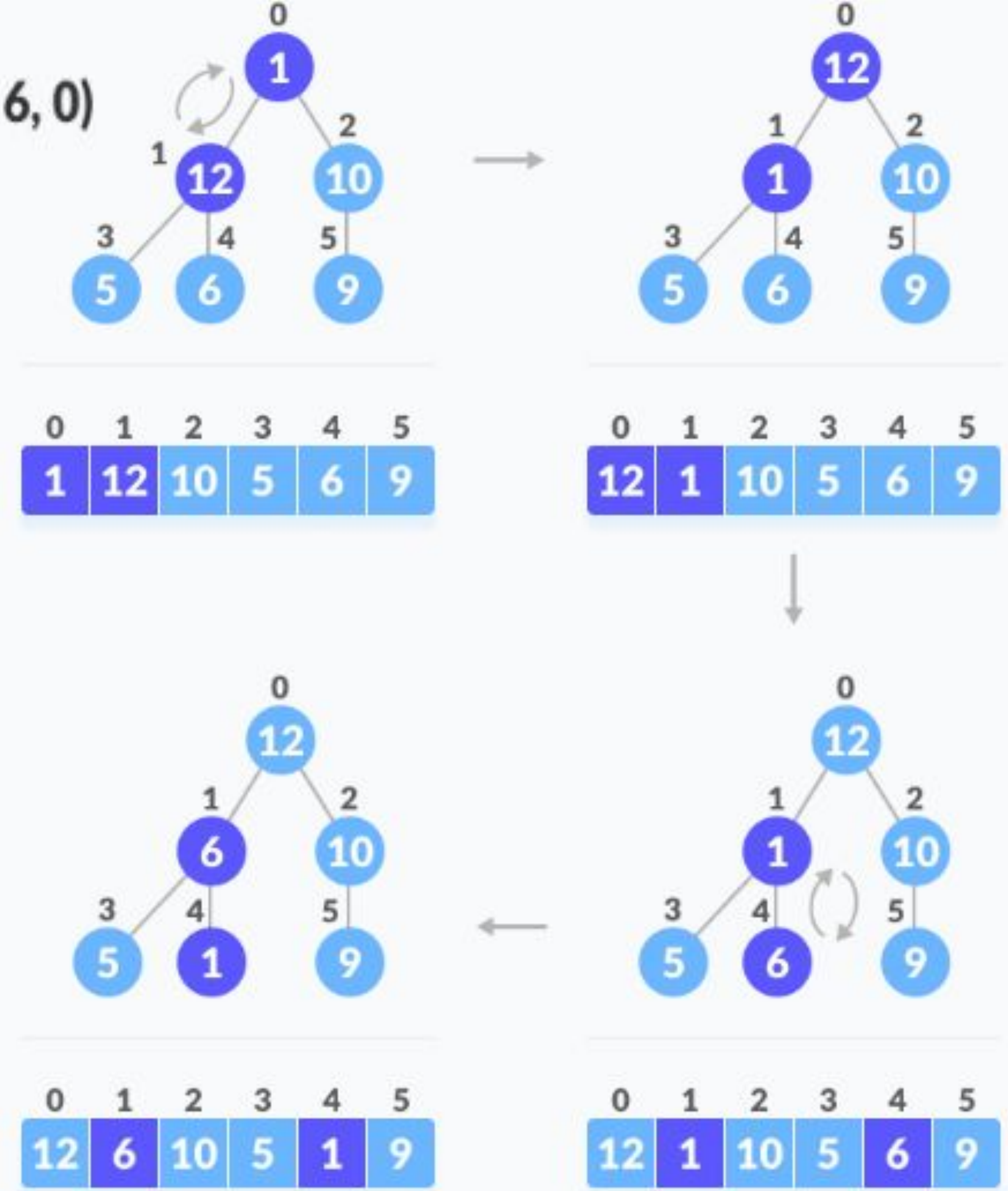


Steps to build max heap for heap sort



Steps to build max heap for heap sort

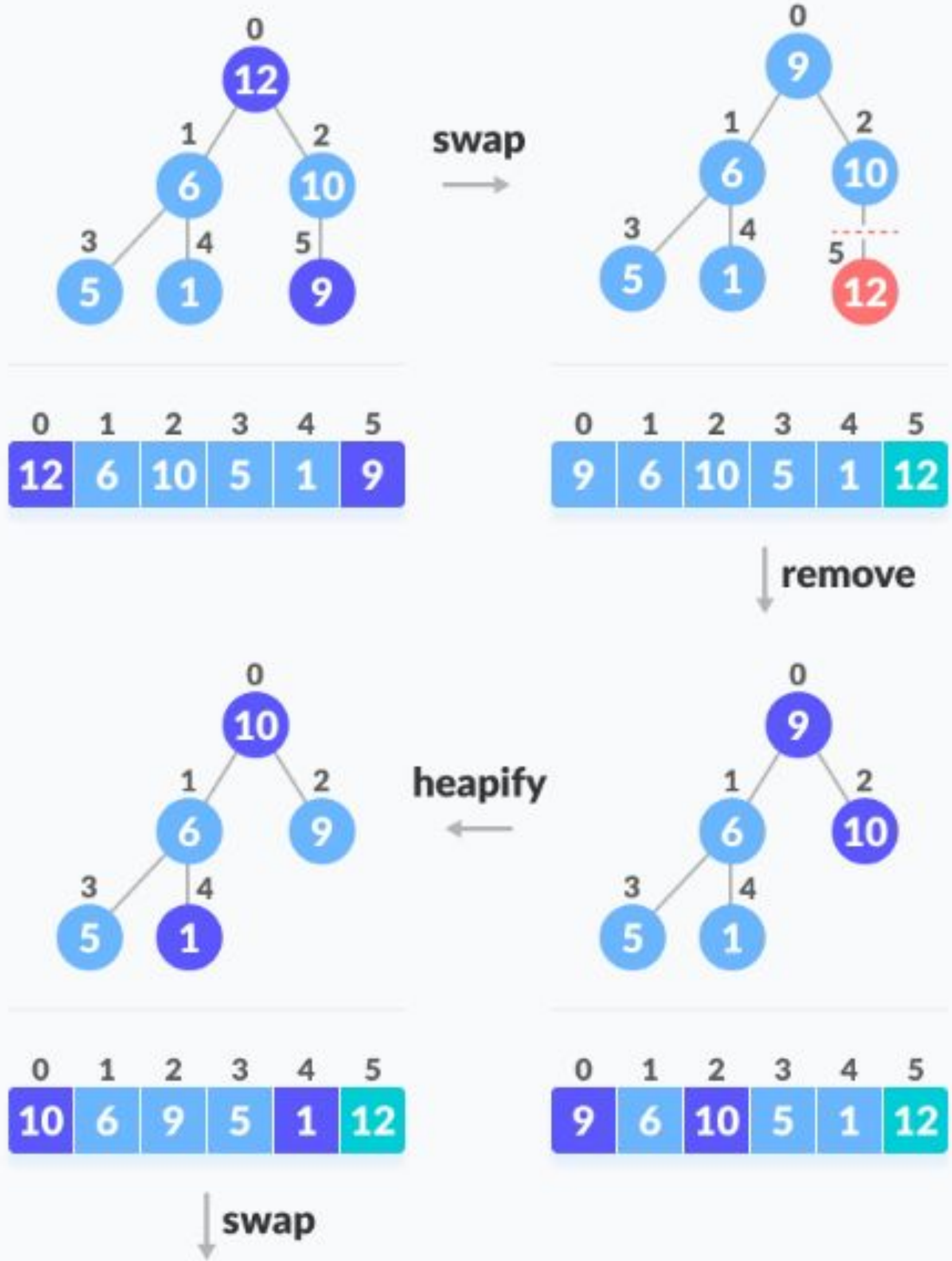
i = 0 → heapify(arr, 6, 0)

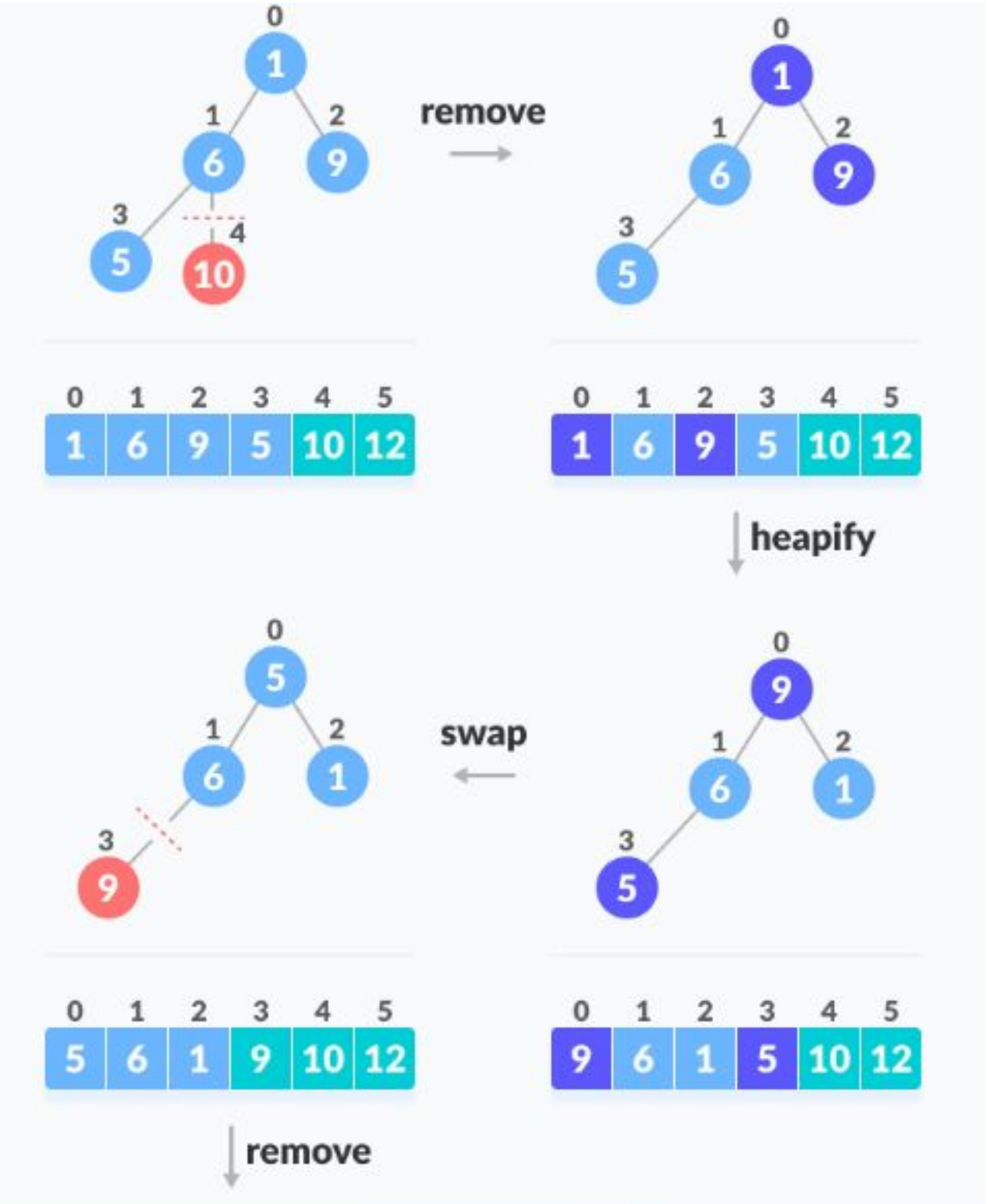


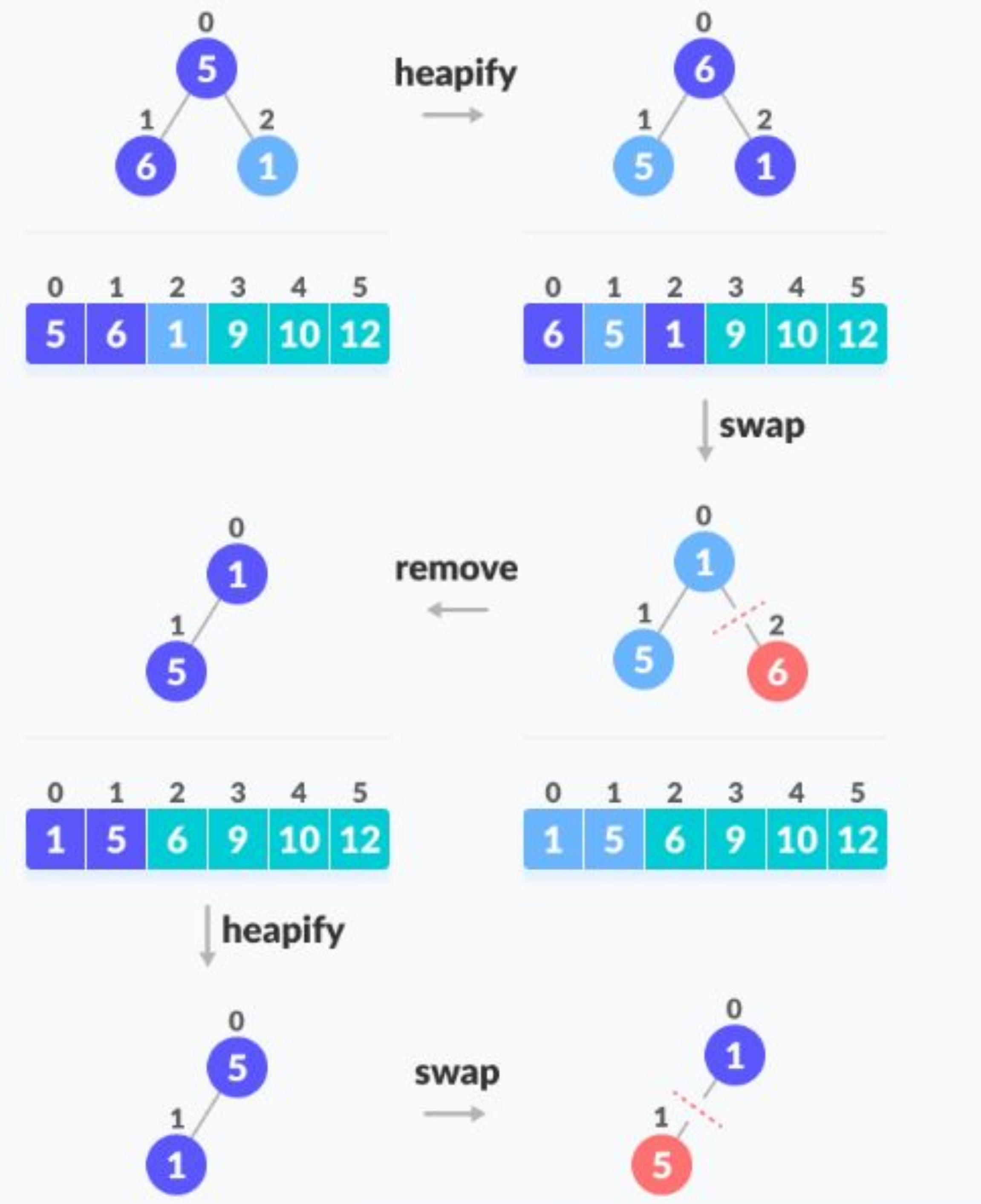
As shown in the above diagram, we start by heapifying the lowest smallest trees and gradually move up until we reach the root element.

Working of Heap Sort

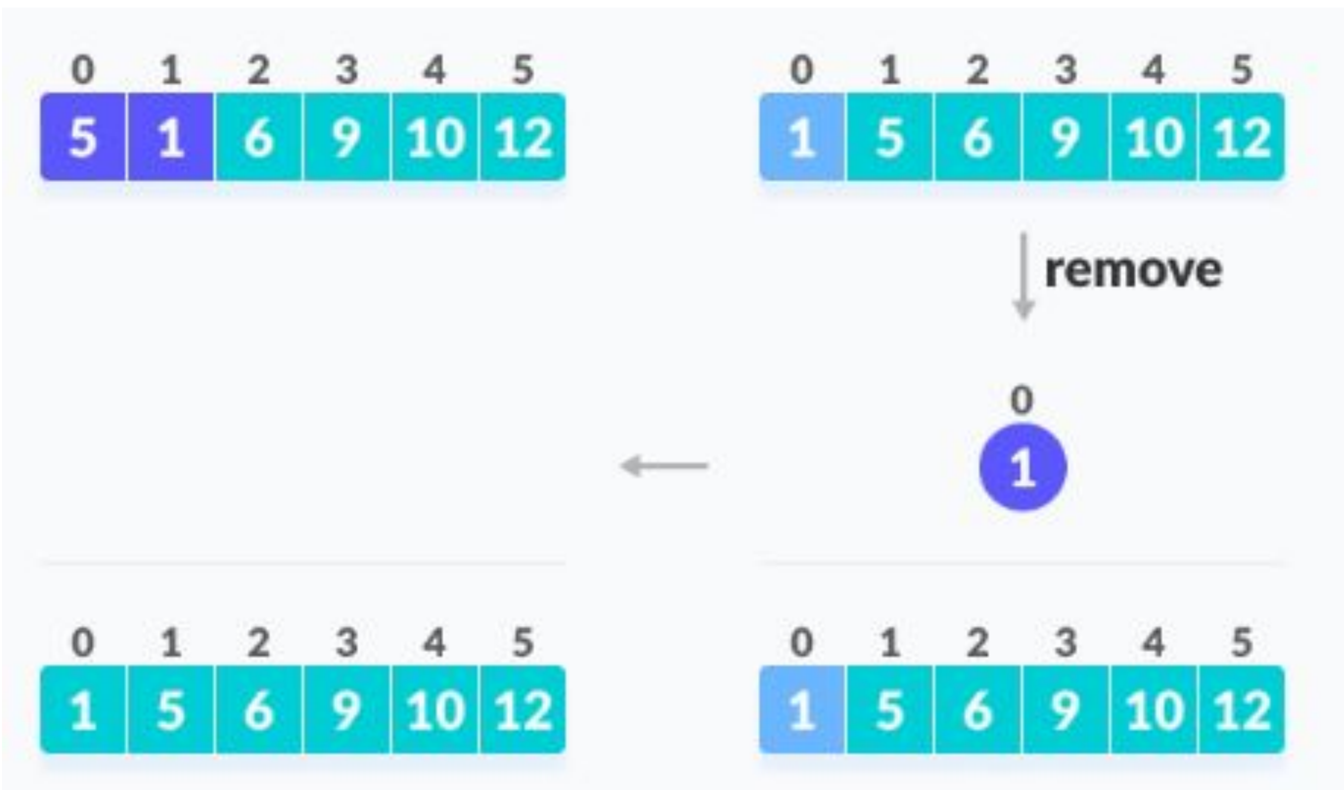
- Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
- **Swap:** Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
- **Remove:** Reduce the size of the heap by 1.
- **Heapify:** Heapify the root element again so that we have the highest element at root.
- The process is repeated until all the items of the list are sorted.


swap
remove
heapify
swap


remove
heapify
swap
remove



heapify
swap
remove
heapify
swap

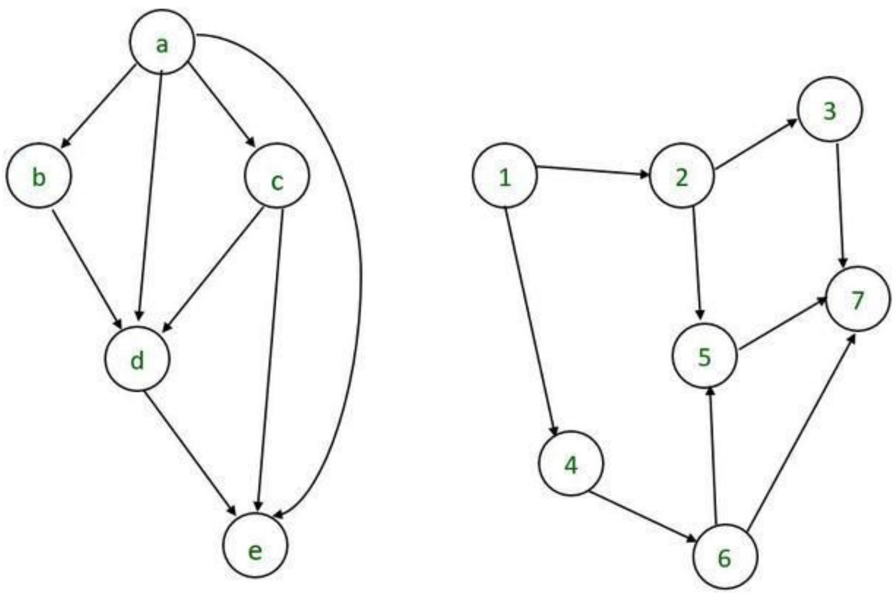

0 1 2 3 4 5
5 1 6 9 10 12
0 1 2 3 4 5
1 5 6 9 10 12
remove
0
1
0 1 2 3 4 5
1 5 6 9 10 12
0 1 2 3 4 5
1 5 6 9 10 12

- Heap Sort has O(nlog n) time complexities for all the cases (best case, average case, and worst case).

Topological Sort:

- A topological sort of a dag G=(V, E) is a linear ordering of all its vertices such that if G contains an edge (u,v) , then u appears before v in the ordering.
- Topological Sorting is possible if and only if the graph is a **Directed Acyclic Graph**.
- There may exist multiple different topological orderings for a given directed acyclic graph.

DAG examples

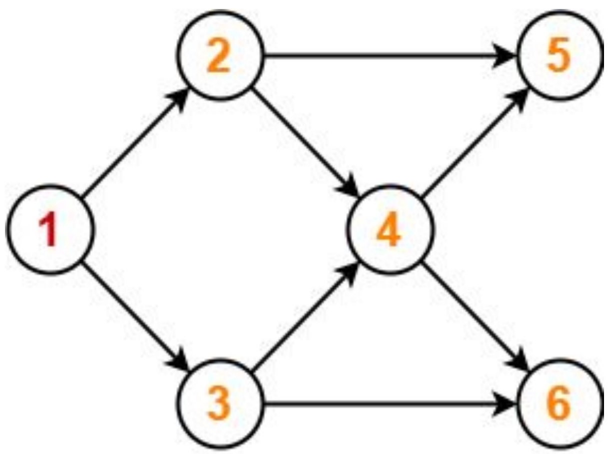


(A) (B)

- *A **Directed Acyclic Graph (DAG)** is a directed graph that does not contain any cycles.*

Topological Sort Example

- Consider the following directed acyclic graph-



Topological Sort Example

For this graph, following 4 different topological orderings are possible

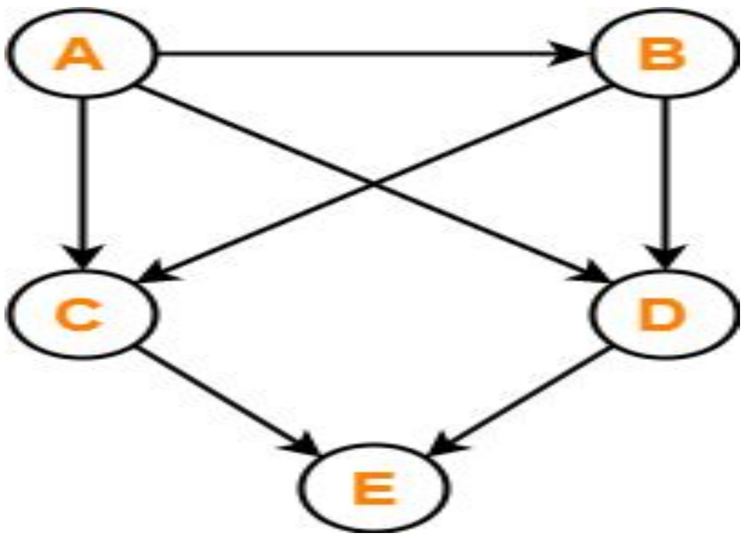
- 1 2 3 4 5 6
- 1 2 3 4 6 5
- 1 3 2 4 5 6
- 1 3 2 4 6 5

Applications of Topological Sort

- Few important applications of topological sort are-
  - Scheduling jobs from the given dependencies among jobs
  - Instruction Scheduling
  - Determining the order of compilation tasks to perform in makefiles
  - Data Serialization

PRACTICE PROBLEMS BASED ON TOPOLOGICAL SORT-

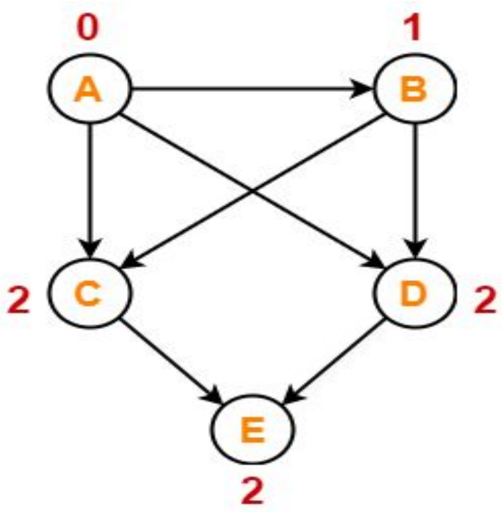
- Find the number of different topological orderings possible for the given graph-



Solution

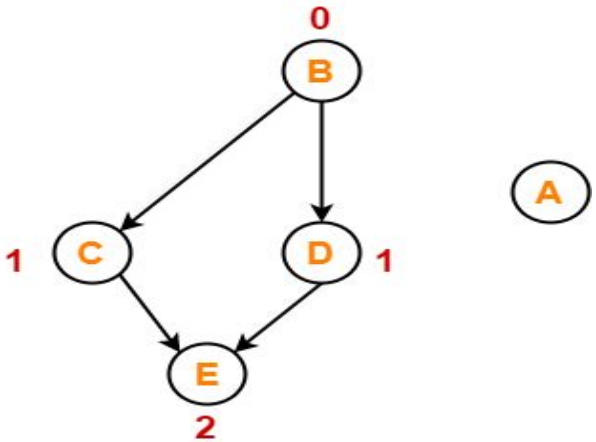
<u>Step-01:</u>

- Write in-degree of each vertex-



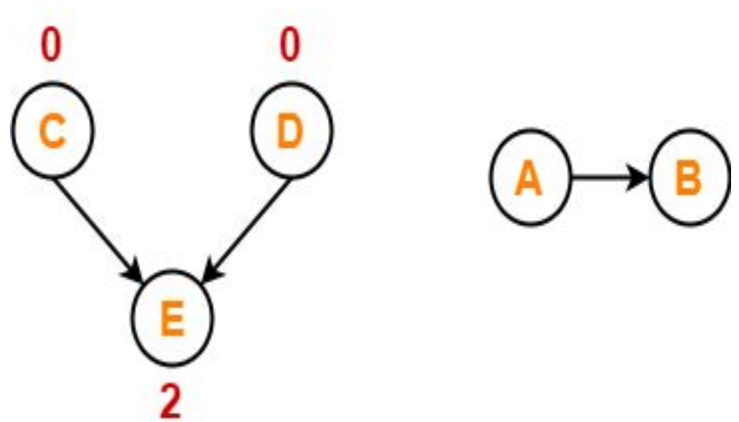
<u>Step-02:</u>

- Vertex-A has the least in-degree.
- So, remove vertex-A and its associated edges.
- Now, update the in-degree of other vertices.



Step-03:

- Vertex-B has the least in-degree.
- So, remove vertex-B and its associated edges.
- Now, update the in-degree of other vertices.



Step-04:

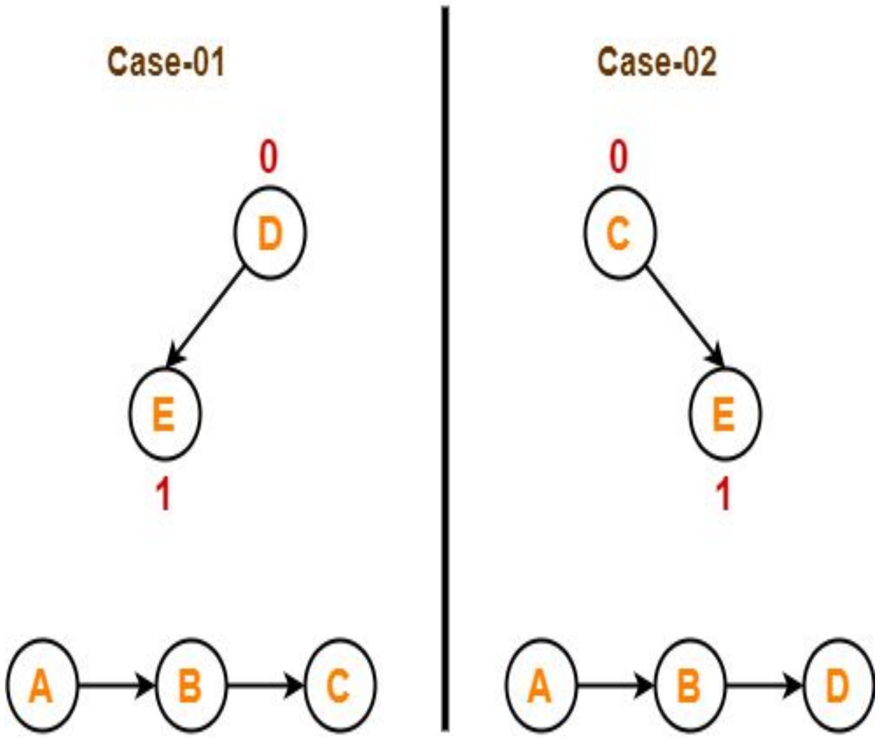
- There are two vertices with the least in-degree. So, following 2 cases are possible-

In case-01,

- Remove vertex-C and its associated edges.
- Then, update the in-degree of other vertices.

In case-02,

- Remove vertex-D and its associated edges.
- Then, update the in-degree of other vertices.



Step-05:

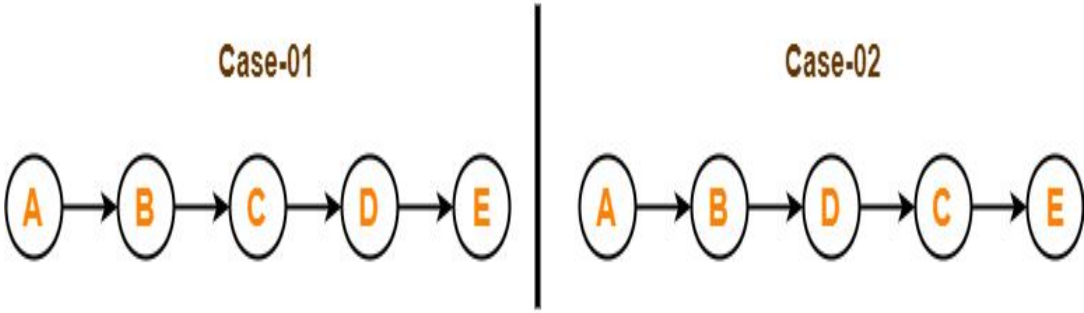
- Now, the above two cases are continued separately in the similar manner.

In case-01,

- Remove vertex-D since it has
- the least in-degree.
- Then, remove the remaining vertex-E.

In case-02,

- Remove vertex-C since it has the least in-degree.
- Then, remove the remaining vertex-E.



Conclusion-

- For the given graph, following **2** different topological orderings are possible-
  - **A B C D E**
  - **A B D C E**

Example 2

- Find the number of different topological orderings possible for the given graph-