HEAP & HEAPSORT

INSTRUCTOR: KASHFIA SAILUNAZ

SLIDES ADAPTED FROM THE TEXTBOOK (CHAPTER 6) & ENSF 593/594 LECTURE BY MOHAMMAD MOSHIRPOUR

OUTLINE

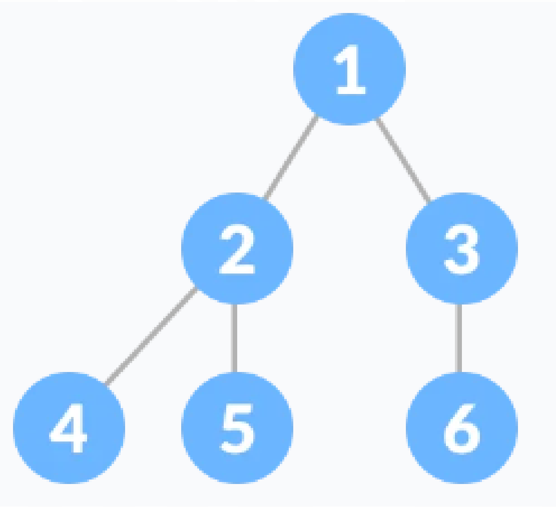
- Complete Binary Tree
- Heap
- Types of Heap
- Heap Operations
- Priority Queue with Heap
- Heapsort

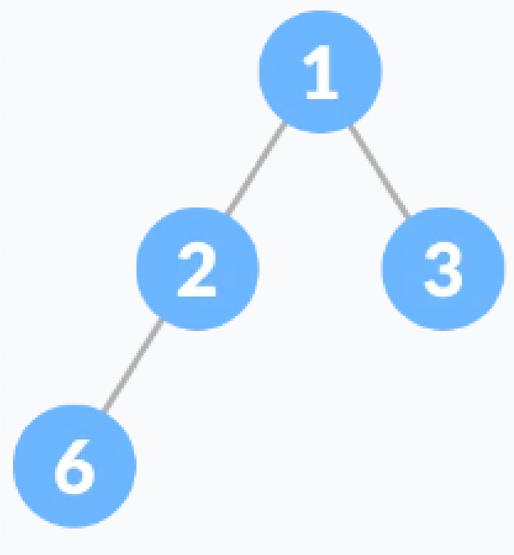
LEARNING OUTCOME

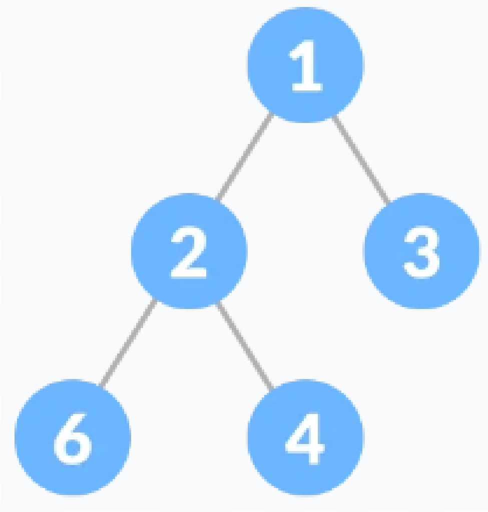
- At the end of this lecture, we will be able to-
  - Understand heap data structure, types and operations,
  - Explain heapsort and priority queue with heap logic.

COMPLETE BINARY TREE

- A binary tree
- Nodes are always inserted from the left
- All levels are completely filled from the left until possibly the lowest one



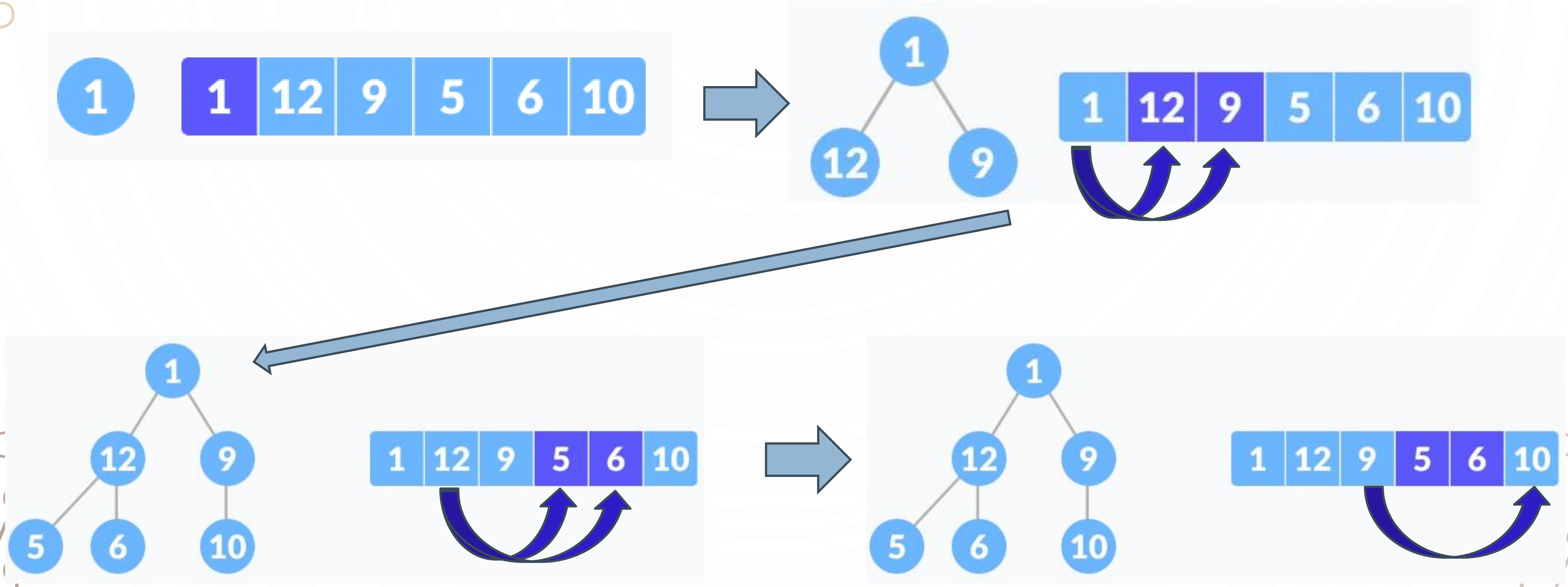




Source: https://www.programiz.com/dsa/complete-binary-tree

COMPLETE BINARY TREE

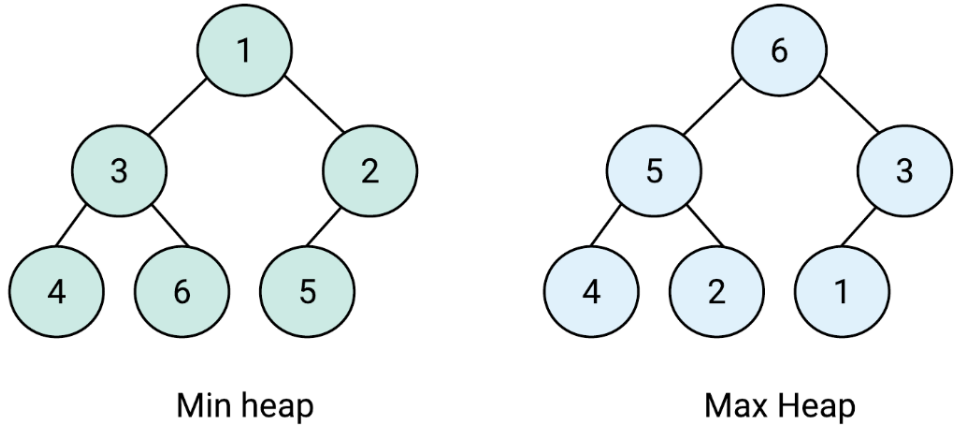
- Complete binary tree in an array



Source: https://www.programiz.com/dsa/complete-binary-tree

HEAP

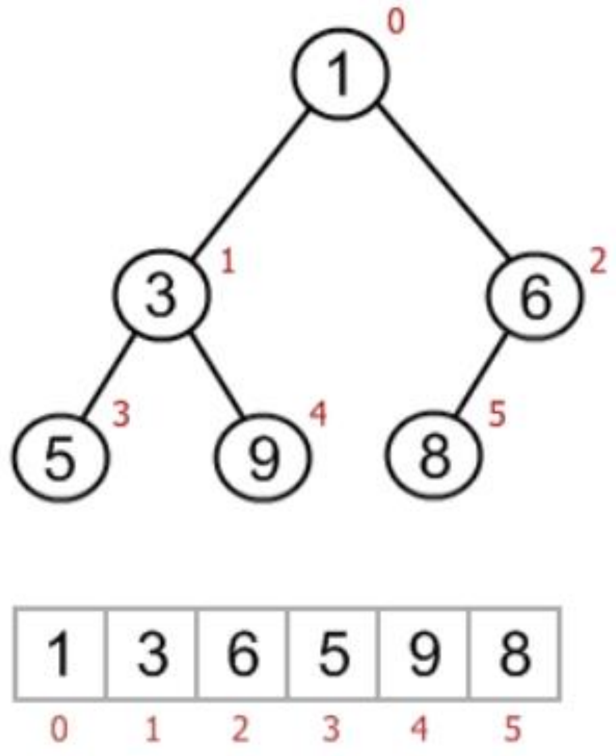
- A complete binary tree
- A balanced tree
- Value of each node is
  - ≥ value of its children nodes (Max Heap) OR
  - ≤ value of its children nodes (Min Heap)
  - And only one condition is applicable for the whole tree
- Not a perfectly ordered tree
  - The order of the elements are limited to each node and its children
- Tree height - lg n
- Complexity – O(lg n)

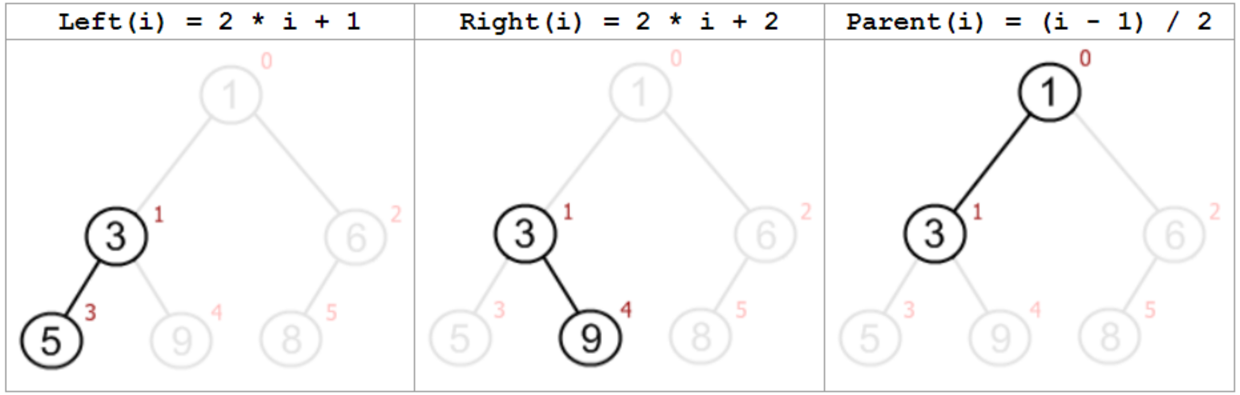


Source: https://guides.codepath.com/compsci/Heaps

HEAP

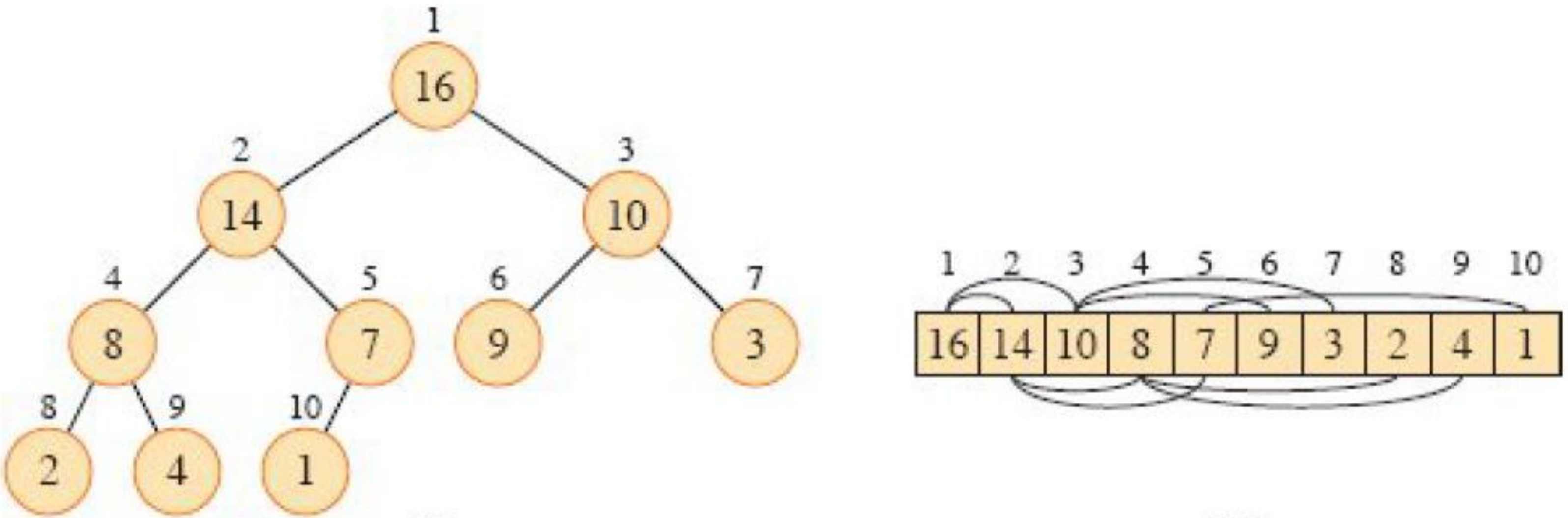
- Heaps are generally implemented with arrays
- Array value sequence
  - Top to bottom
  - Left to right
- Root node is at Array[0]
- All node positions should be from 0 to n-1
- For any node at position i-
  - Left child of Array[i] is at Array [(2 * i) + 1]
  - Right child of Array[i] is at Array [(2 * i) + 2]
  - Parent of Array[i] is at Array[(i – 1)/2]
    - Integer division





Source: https://www.algolist.net/Data_structures/Binary_heap/Array-based_int_repr

HEAP



(a) (b)

Figure 6.1 A max-heap viewed as **(a)** a binary tree and **(b)** an array. The number within the circle at each node in the tree is the value stored at that node. The number above a node is the corresponding index in the array. Above and below the array are lines showing parent-child relationships, with parents always to the left of their children. The tree has height 3, and the node at index 4 (with value 8) has height 1.

PARENT(i)

1 **return** $\lfloor i/2 \rfloor$

LEFT(i)

1 **return** $2i$

RIGHT(i)

1 **return** $2i + 1$

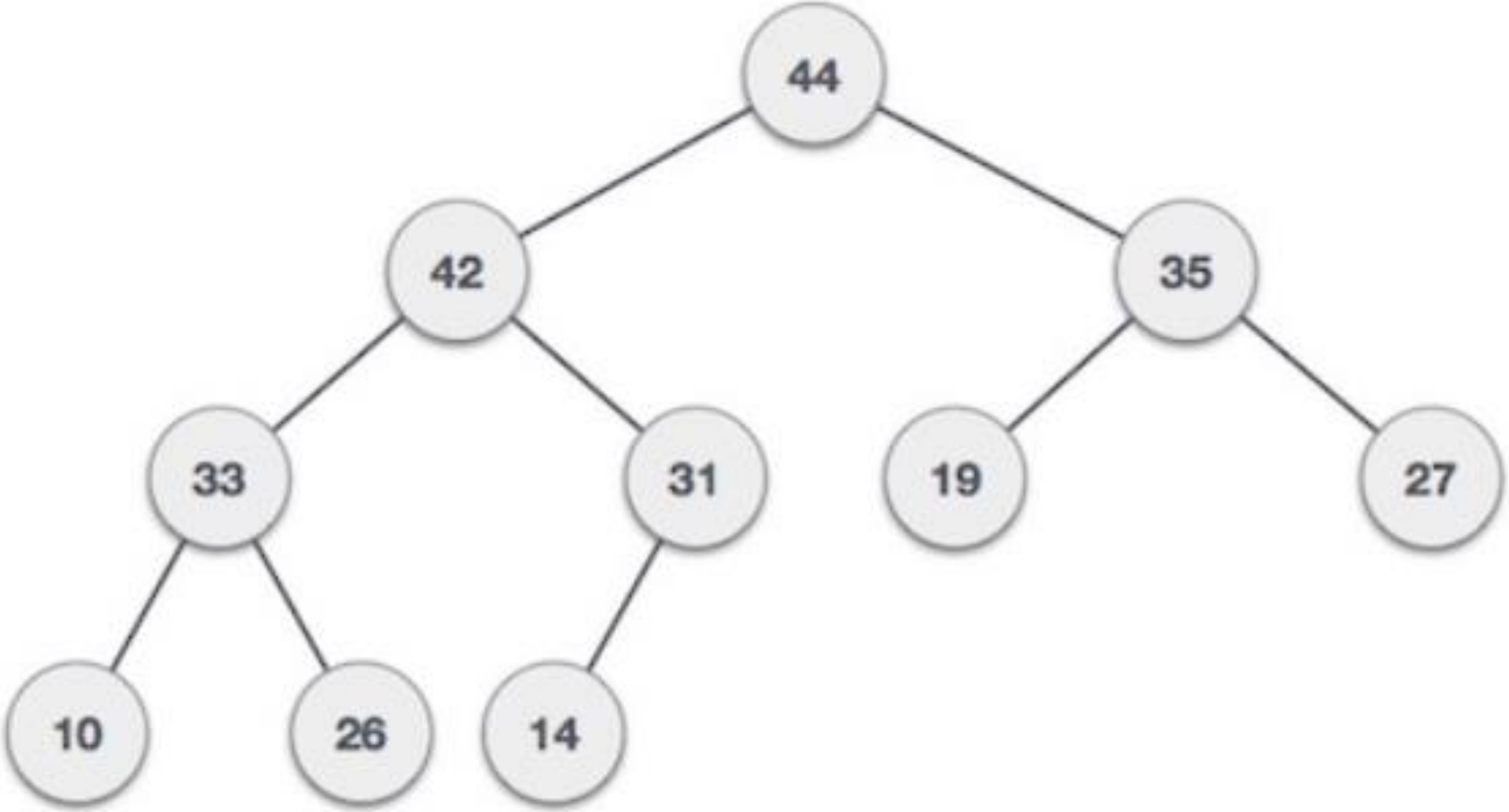
*** In this example, the array start from 1 instead of 0, so the parent and children node positions is increased by 1

TYPES OF HEAP

- Max Heap
  - A complete binary tree where value of each node ≥ value of its children nodes
  - Root contains the largest element
- Min Heap
  - A complete binary tree where value of each node ≤ value of its children nodes
  - Root contains the smallest element

TYPES OF HEAP

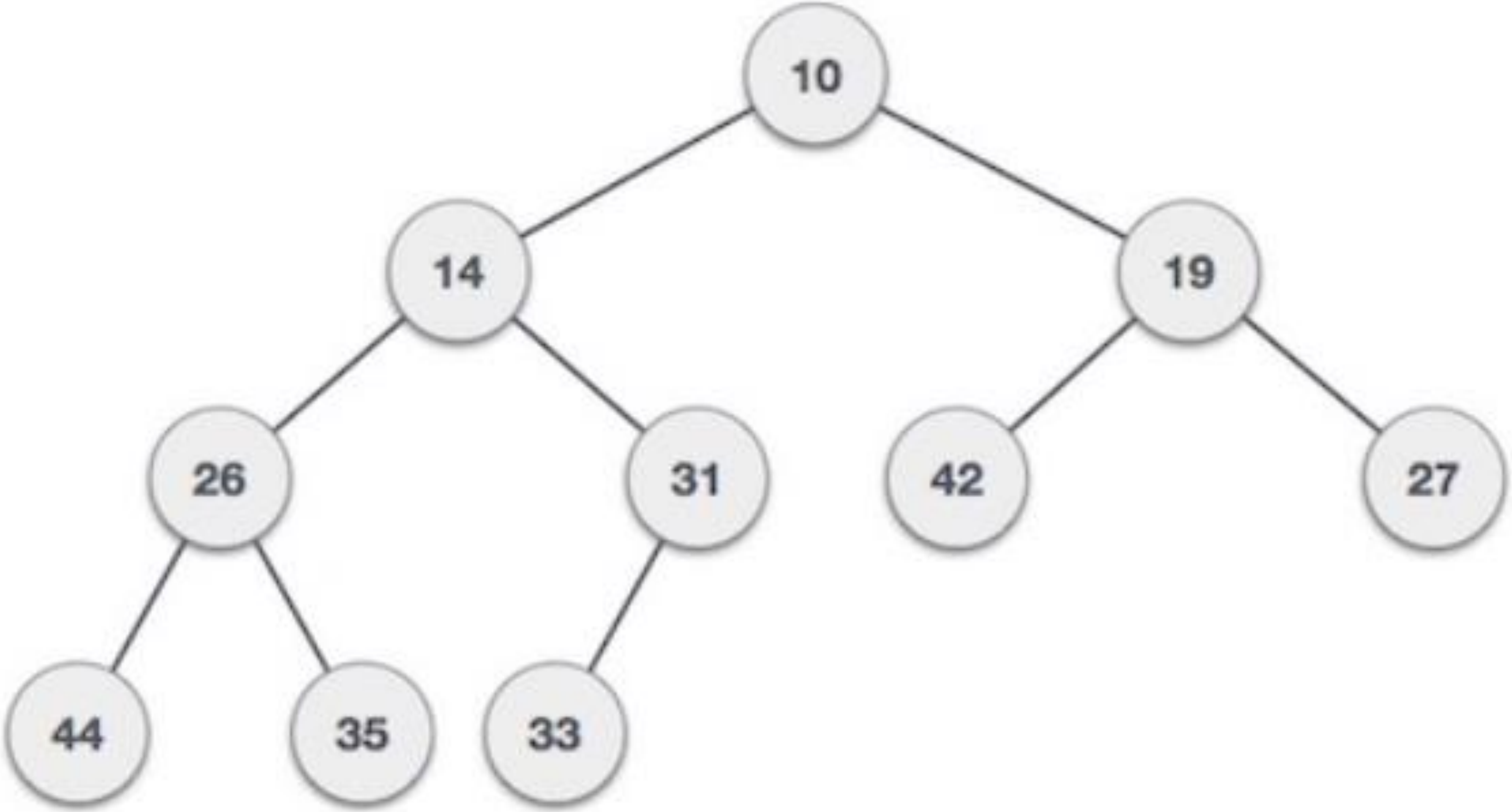
- Max Heap
  - Input nodes – 35, 33, 42, 10, 14, 19, 27, 44, 26, 31



Source: https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

TYPES OF HEAP

- Min Heap
  - Input nodes – 35, 33, 42, 10, 14, 19, 27, 44, 26, 31



Source: https://www.tutorialspoint.com/data_structures_algorithms/heap_data_structure.htm

HEAP OPERATIONS

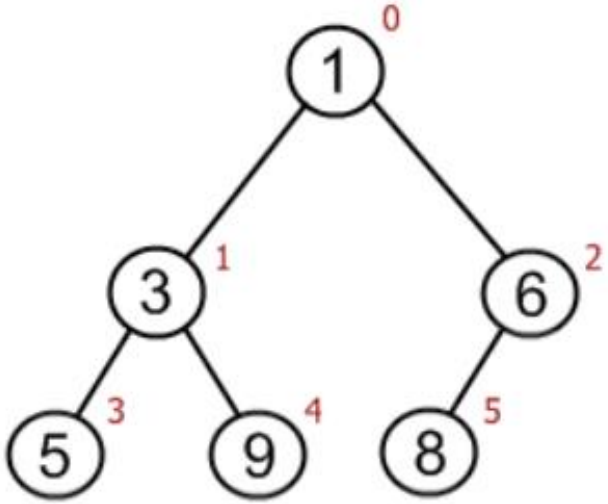
- Heapify
  - Process of creating a heap from array or binary tree
- Insertion
  - Inserting an element in the heap and maintain heap property
- Deletion
  - Deleting an element from the heap and maintain heap property
- Peek
  - Check the top element in a heap
- Display
  - Traverse the heap and show the elements

HEAP OPERATIONS

- Heapify
  - Max Heapify
    - Process of creating a Max Heap from a binary tree or array
  - Min Heapify
    - Process of creating a Min Heap from a binary tree or array

HEAP OPERATIONS

- Max Heapify
  - Create a binary tree
  - Check if the tree nodes maintain the Max Heap condition
    - i.e., value of each node ≥ value of its children nodes
  - Start from the first non-leaf node from bottom
    - i.e., node at position (n/2) -1
  - Check if current node i and its children (2i+1), (2i+2) maintain heap
    - Set 'current element' at node i as 'largest' element
      - largest = i (i.e., parent)
    - Check if left child (2i+1) is larger than 'largest'
      - If so, then update 'largest' with left child
        - largest = 2i+1 (i.e., left child)
    - Then check if right child (2i+2) is larger than 'largest'
      - If so, then update 'largest' with right child
        - largest = 2i+2 (i.e., right child)
  - If largest ≠ i, then exchange/swap i value with largest value
  - Continue this process from node (n/2) -1 to node 0

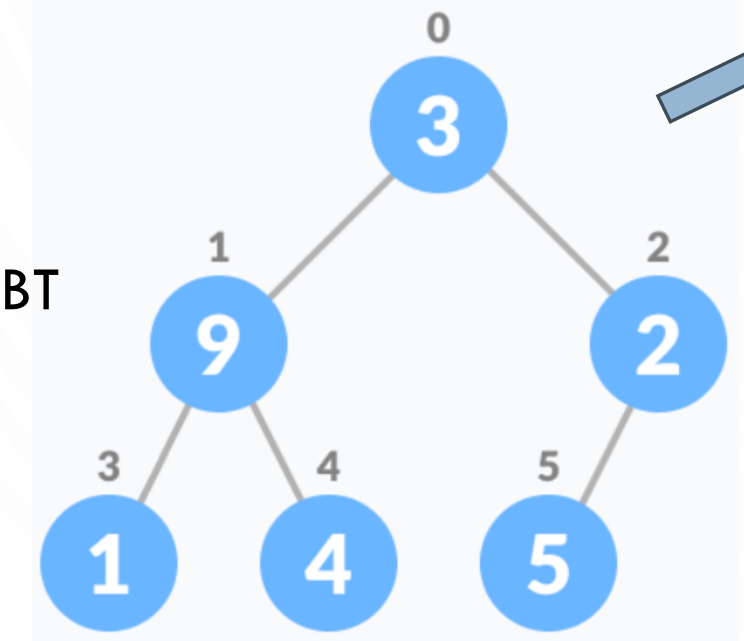


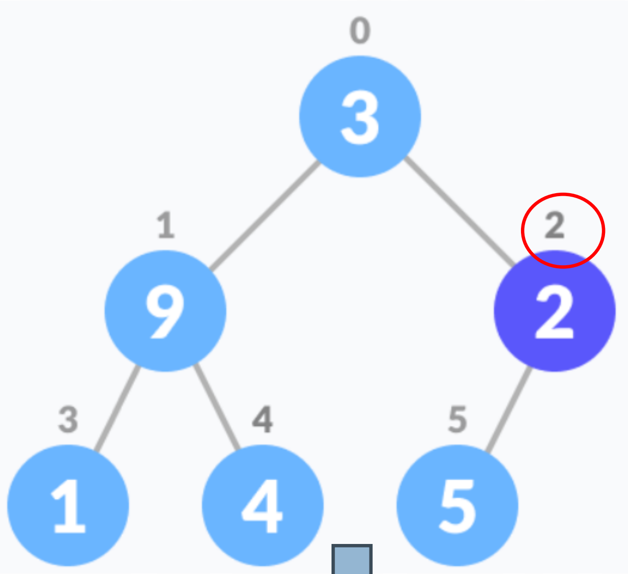
Source: https://www.algolist.net/Data_structures/Binary_heap/Array-based_int_repr

HEAP OPERATIONS

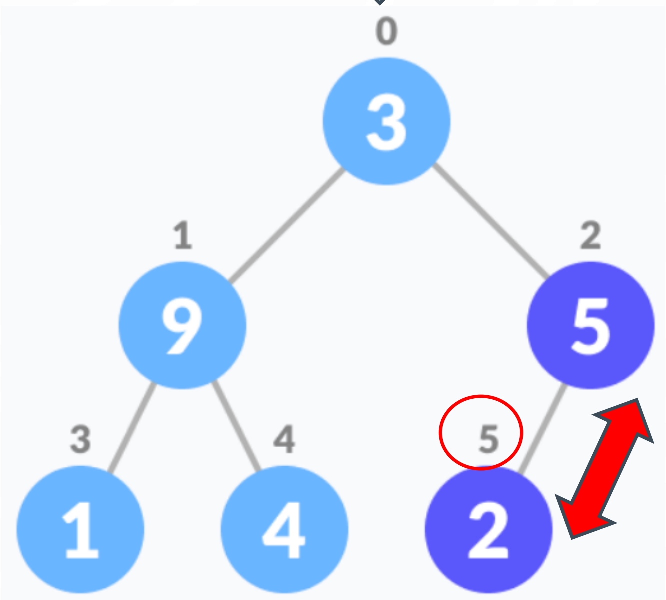
- Max Heapify

BT





(n/2) -1
= (6/2)-1
= 3 – 1
= 2



2i + 1
= (2*2) + 1
= 5

Source: https://www.programiz.com/dsa/heap-data-structure#:~:text=Heapify%20is%20the%20process%20of,Heap%20or%20a%20Max%2DHeap.&text=Start%20from%20the%20first%20index,by%20n%2F2%20%2D%201%20.&text=Set%20current%20element%20i%20as%20largest%20.

HEAP OPERATIONS

- MAX-HEAPIFY(A, i)

```
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

- BUILD-MAX-HEAP(A, n)

```
A.heap-size = n
for i = ⌊n/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

HEAP OPERATIONS

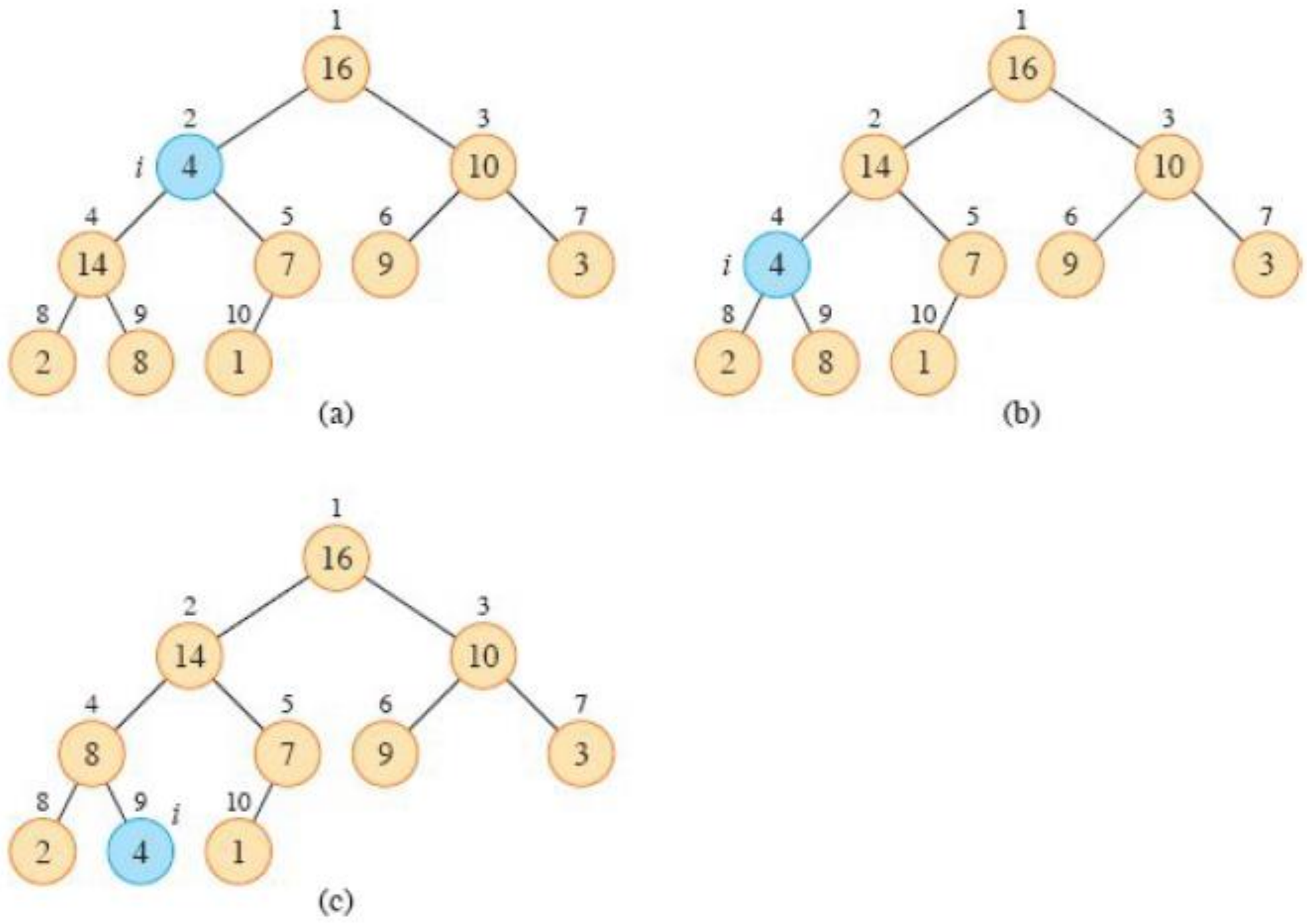


Figure 6.2 The action of MAX-HEAPIFY(A, 2), where $A.heap\text{-}size = 10$. The node that potentially violates the max-heap property is shown in blue. **(a)** The initial configuration, with $A[2]$ at node $i = 2$ violating the max-heap property since it is not larger than both children. The max-heap property is restored for node 2 in **(b)** by exchanging $A[2]$ with $A[4]$, which destroys the max-heap property for node 4. The recursive call MAX-HEAPIFY(A, 4) now has $i = 4$. After $A[4]$ and $A[9]$ are swapped, as shown in **(c)**, node 4 is fixed up, and the recursive call MAX-HEAPIFY(A, 9) yields no further change to the data structure.

HEAP OPERATIONS

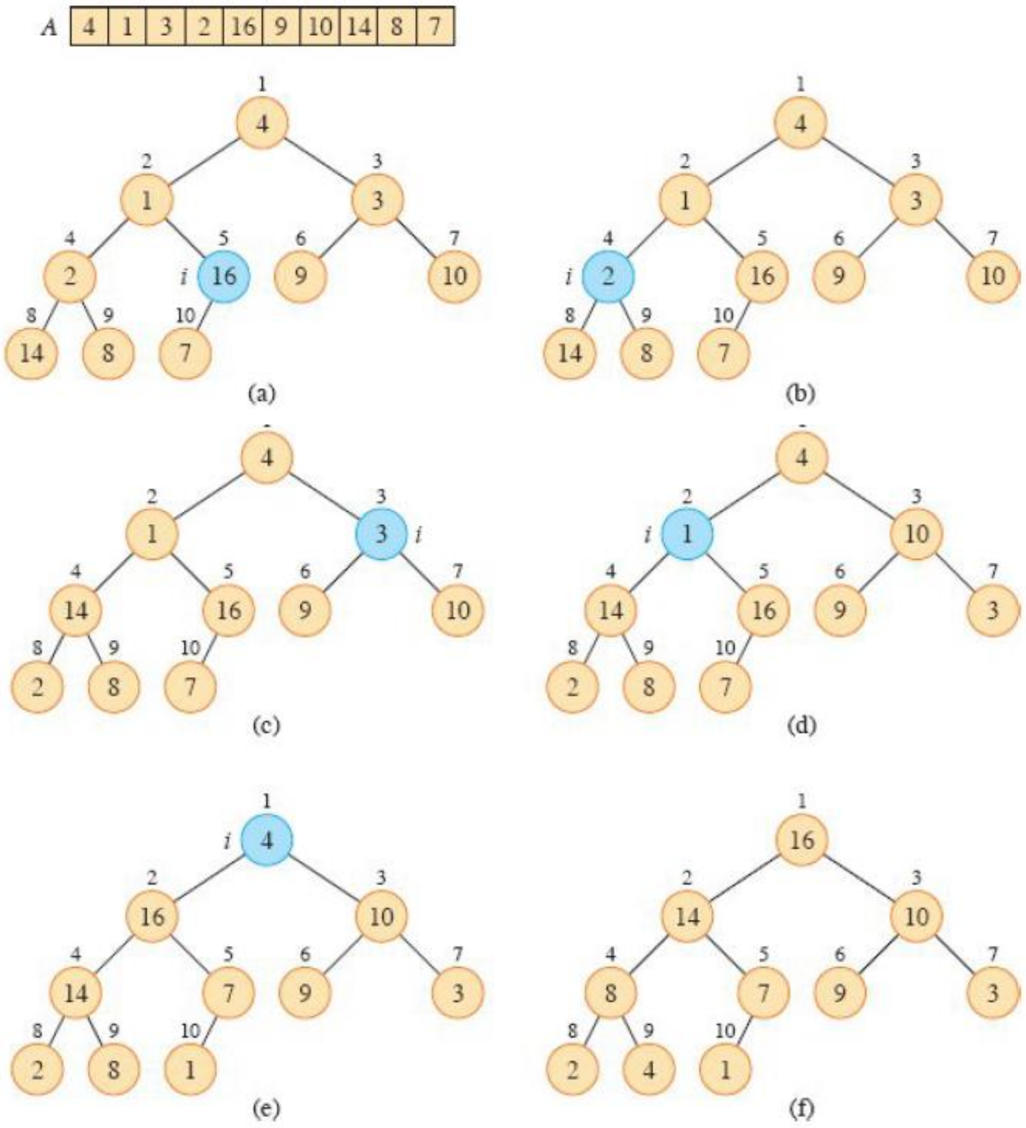


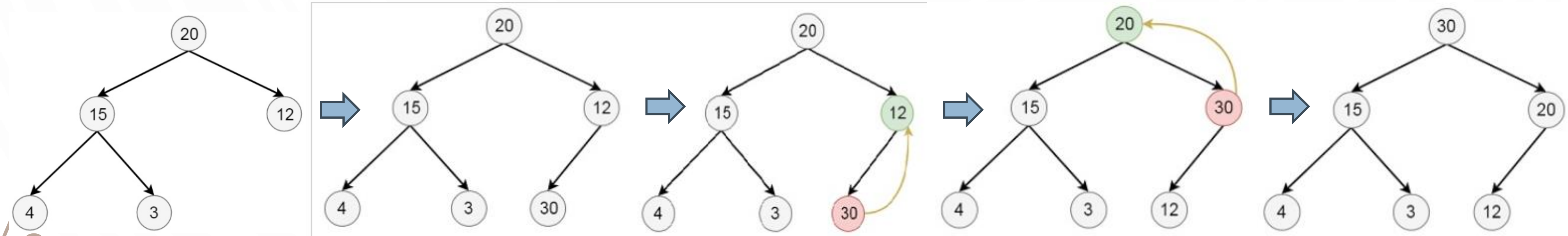
Figure 6.3 The operation of BUILD-MAX-HEAP, showing the data structure before the call to MAX-HEAPIFY in line 3 of BUILD-MAX-HEAP. The node indexed by i in each iteration is shown in blue. **(a)** A 10-element input array A and the binary tree it represents. The loop index i refers to node 5 before the call MAX-HEAPIFY(A, i). **(b)** The data structure that results. The loop index i for the next iteration refers to node 4. **(c)–(e)** Subsequent iterations of the **for** loop in BUILD-MAX-HEAP. Observe that whenever MAX-HEAPIFY is called on a node, the two subtrees of that node are both max-heaps. **(f)** The max-heap after BUILD-MAX-HEAP finishes.

HEAP OPERATIONS

- Min Heapify and Build Min Heap are similar with the $\leq$ condition
- Try it yourself

HEAP OPERATIONS

- Insertion
  - Assuming there is a max/min heap already
  - Add the new node as the last leaf node at the first available position (checking from left)
  - Check the heap property (max/min) of the current heap
  - If condition is violated, do a heapify (max/min)

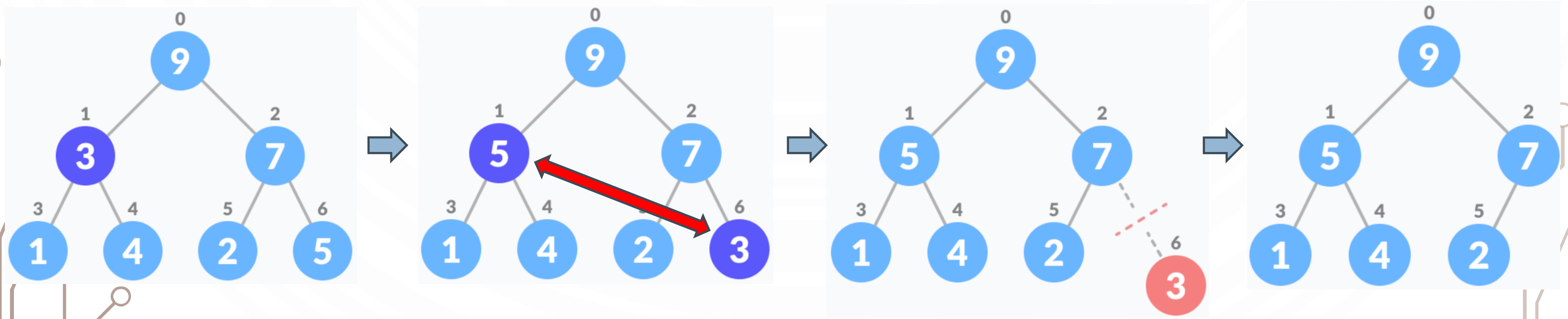


Inserting a new node: 30

Source: https://www.tutorialspoint.com/insertion-and-deletion-in-heaps-in-data-sturcture

HEAP OPERATIONS

- Deletion
  - Assuming there is a max/min heap already search for the node to be deleted (i.e., key node)
  - If key is already the last leaf, then just delete it
  - If key is not the last leaf, then
    - Swap key with the last leaf node at the first available position (checking from left) and delete key
    - Check the heap property (max/min) of the current heap
    - If condition is violated, do a heapify (max/min)



Deleting node: 3

Source: https://www.programiz.com/dsa/heap-data-structure

PRIORITY QUEUE WITH HEAP

- Priority Queue
  - An extension of queue data structure
  - Every item is associated with a priority score
  - The highest priority element is dequeued first
  - In case of multiple elements with the same priority score, they are dequeued in the order they were enqueued
  - Priority queue operation complexity (with linear data structure) – O(n)
- Priority queue implementation with binary heap
  - Heap data structure can be used to implement priority queue
  - Minimizes the complexity to O (lg n)
  - Uses the heap structure to store the priority of elements

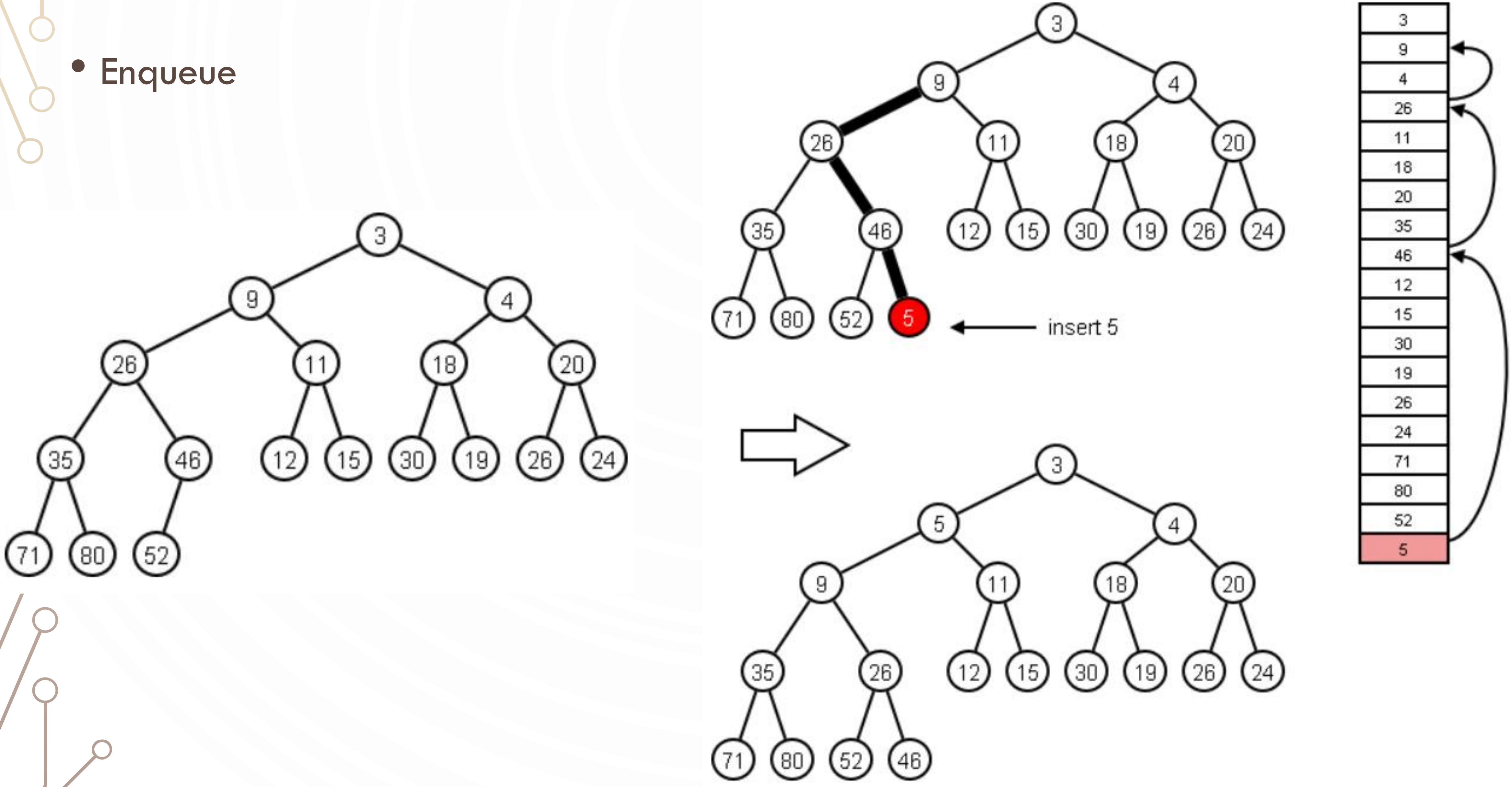
PRIORITY QUEUE WITH HEAP

- Enqueue
  - Add the new element at the end of the heap (i.e., as the last leaf in a heap)
  - If needed, apply Heapify on the current heap to maintain heap property

- Dequeue
  - Remove root (i.e., highest priority) element
  - Replace the roof with the last leaf node
  - If needed, apply Heapify on the current heap to maintain heap property

PRIORITY QUEUE WITH HEAP

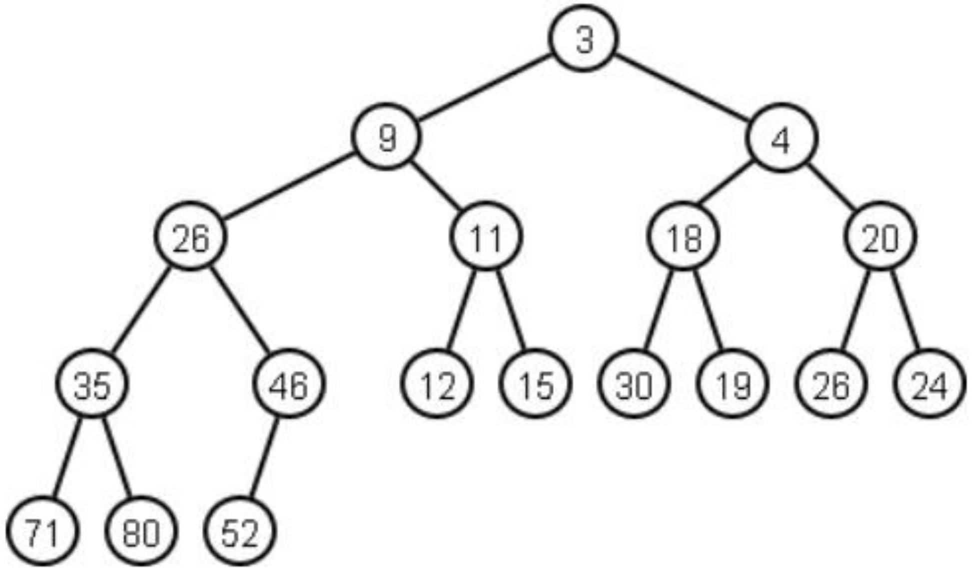
- Enqueue

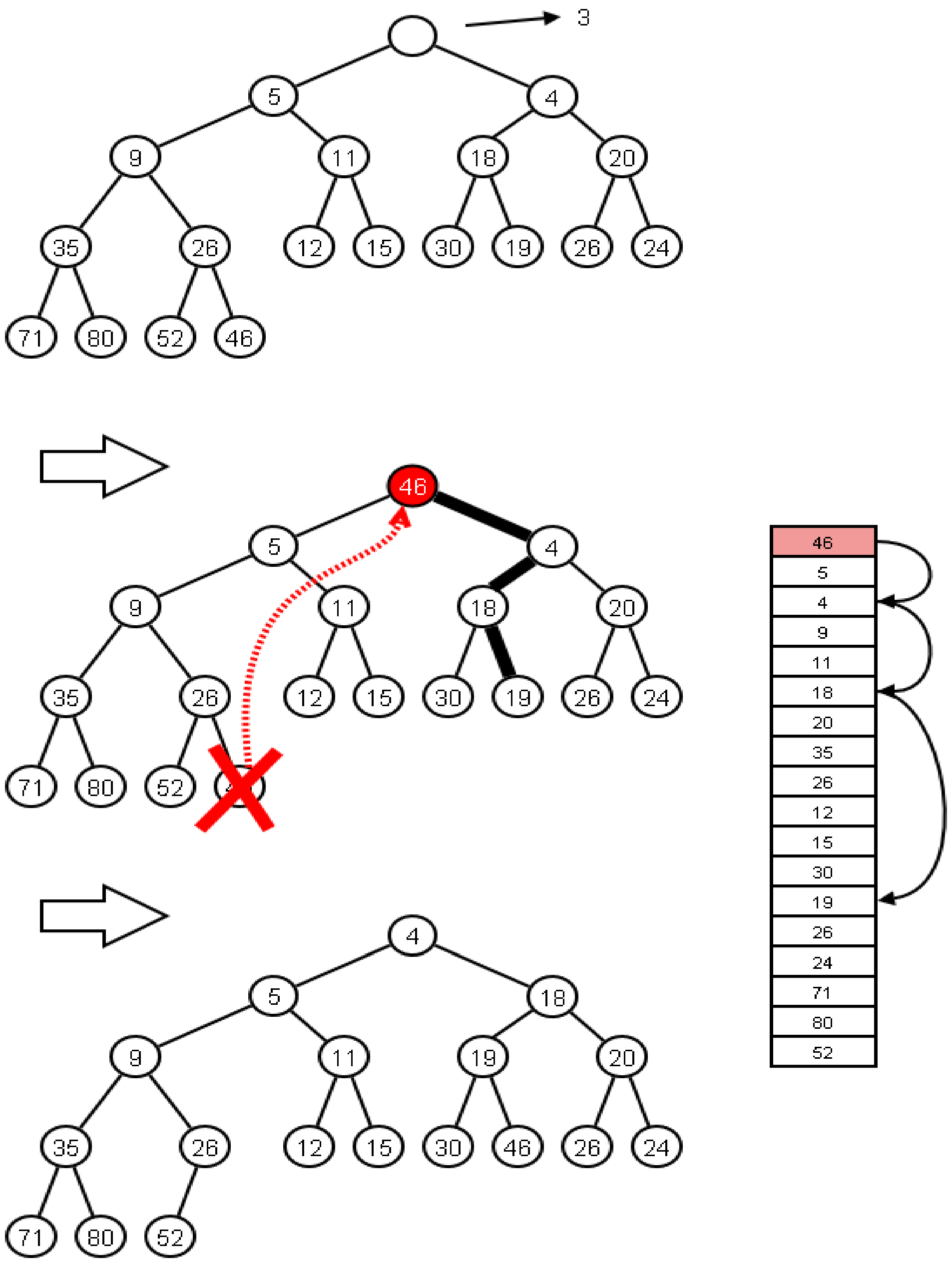


Source: https://cs.lmu.edu/~ray/notes/pqueues/

PRIORITY QUEUE WITH HEAP

- Dequeue



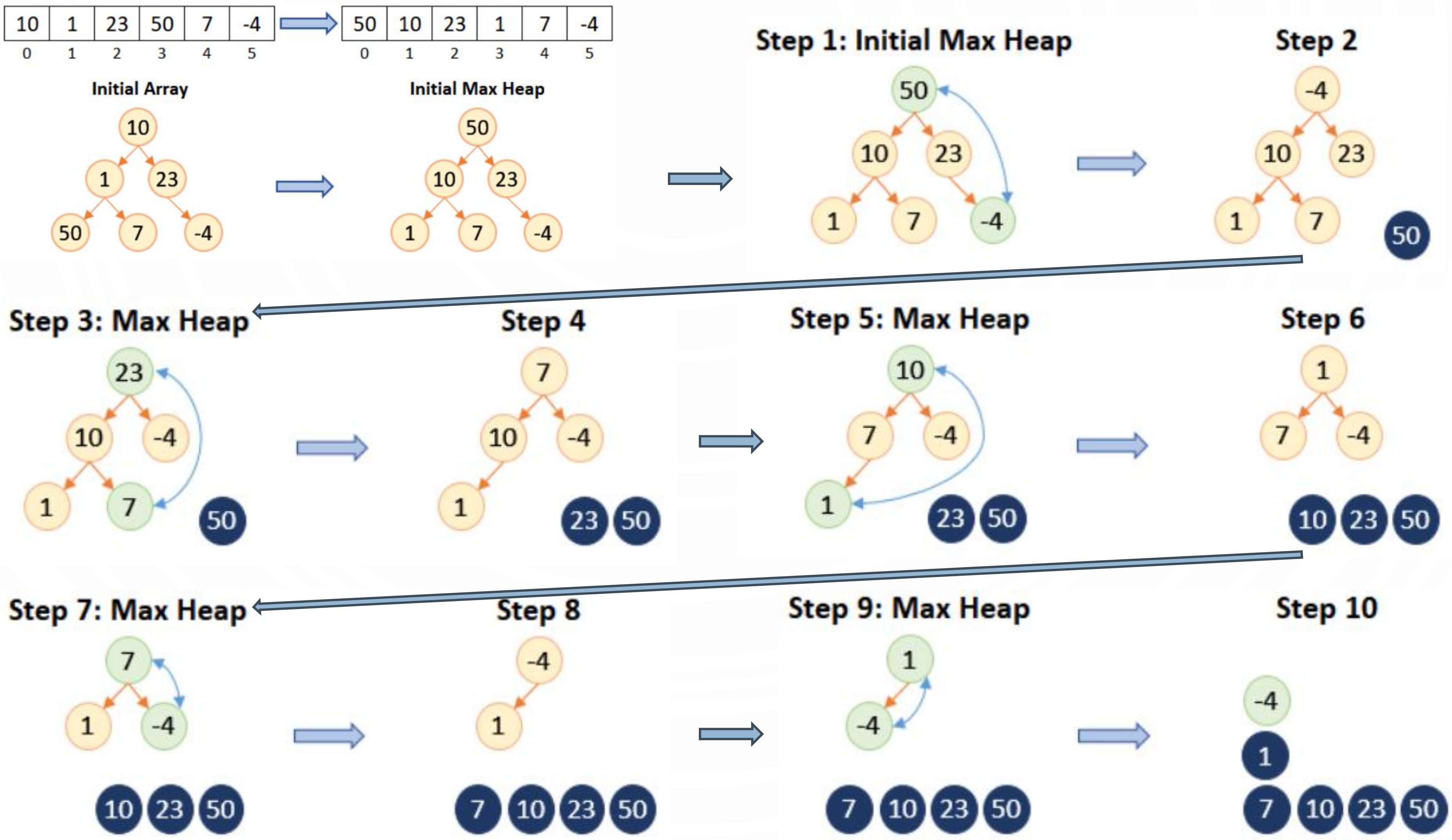


Source: https://cs.lmu.edu/~ray/notes/pqueues/

HEAPSORT

- A sorting algorithm based on binary heap data structure
- Uses comparison-based sorting technique
- Algorithm
  - Convert the input array into heap (max/min)
  - Repeat the steps until no node is left in the heap-
    - Swap the root with the last leaf
    - Delete the last leaf (i.e., the node with max value in max heap OR min value in min heap)
      - Place it into its correct position in the sorted array (from n-1 or from 0 based on the required sorting order)
    - Apply heapify on the current nodes

HEAPSORT



Source: https://www.alphacodingskills.com/algo/heap-sort.php

HEAPSORT

- HEAPSORT(A, n)

```
BUILD-MAX-HEAP(A, n)
for i = n downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size – 1
    MAX-HEAPIFY(A, 1)
```

HEAPSORT

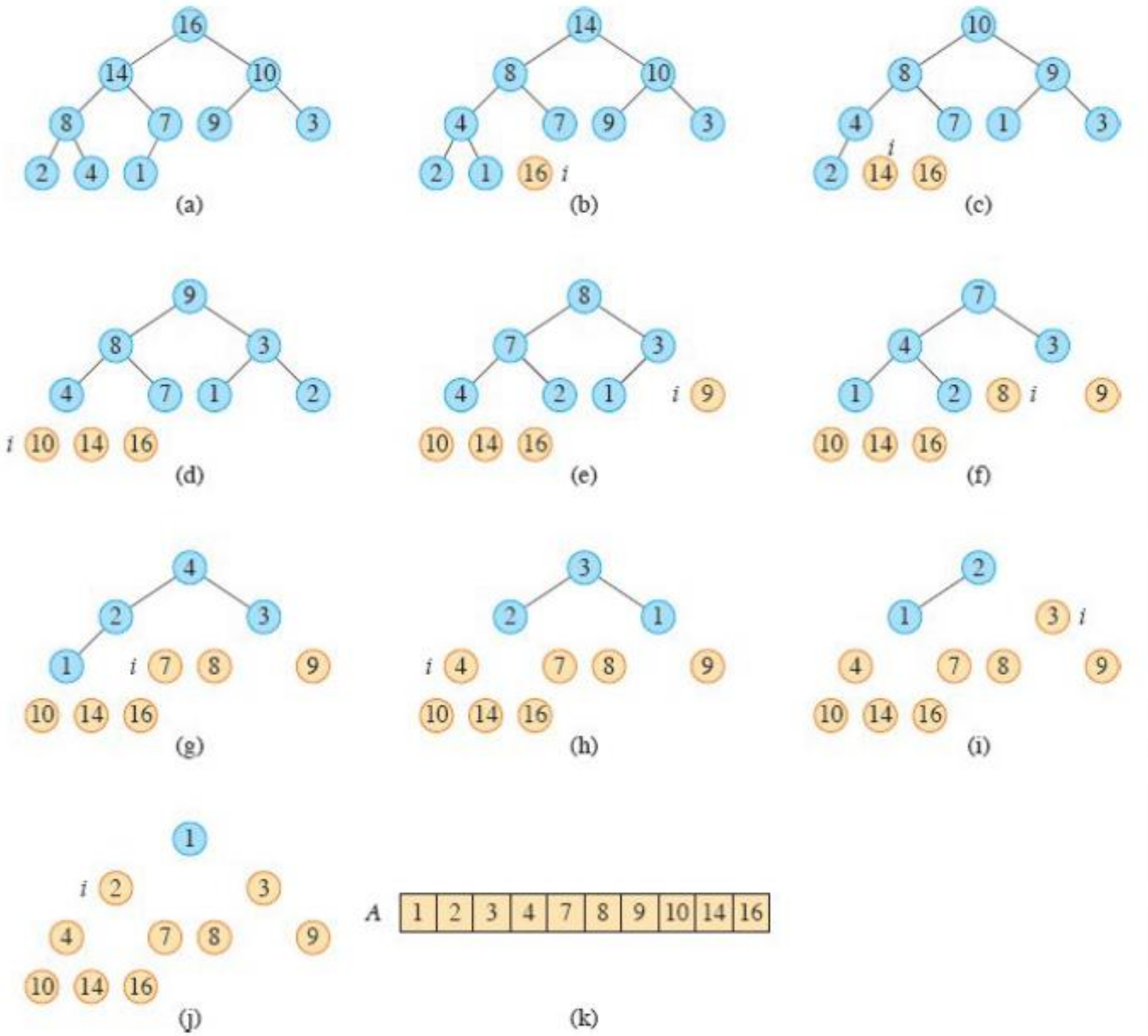


Figure 6.4 The operation of HEAPSORT. **(a)** The max-heap data structure just after BUILD-MAX-HEAP has built it in line 1. **(b)–(j)** The max-heap just after each call of MAX-HEAPIFY in line 5, showing the value of i at that time. Only blue nodes remain in the heap. Tan nodes contain the largest values in the array, in sorted order. **(k)** The resulting sorted array A.

HEAPSORT

- Heapsort with Max Heap
  - Ascending order : Use original max heapsort algorithm
  - Descending order : Assign extracted max nodes in the opposite order
- Heapsort with Min Heap
  - Descending order : Use original min heapsort algorithm
  - Ascending order : Assign extracted min nodes in the opposite order

HEAPSORT

- Complexity
  - Best case: O(n) – if all nodes are identical
  - Average case: O(n lg n)
  - Worst case: O(n lg n) - array sorted in reverse order

SUMMARY

- Heap is a complete binary tree data structure.
- Value of each node is ≥ or ≤ of value of its children nodes.
- Build heap with Heapify operations are applied to maintain the heap structure.
- Heap can be used to implement priority queues.
- Heapsort is a sorting algorithm that uses max heap or min heap structure to sort arrays into ascending or descending orders.

THANK YOU