Data Structures & Algorithms

15: Heaps & HeapSort

(Chapter 6 – CLRS)

Dr Ram Prasad Krishnamoorthy

Associate Professor
School of Computing and Data Science

ram.krish@saiuniversity.edu.in

SAI UNIVERSITY

Heaps

HEAPS

Heap data structure is a **complete binary tree** implemented as an **array** object.

Each **node** of the tree corresponds to an **element** in the array.

*(Not to be confused with **heaps in memory architectures** of programming languages)*

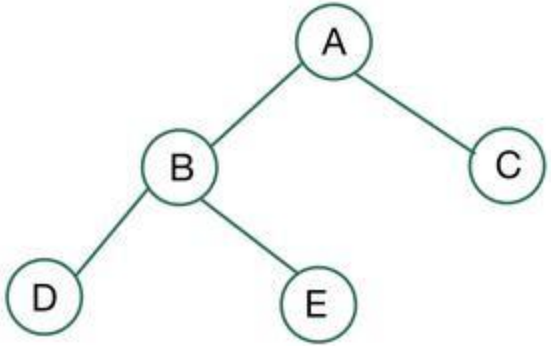
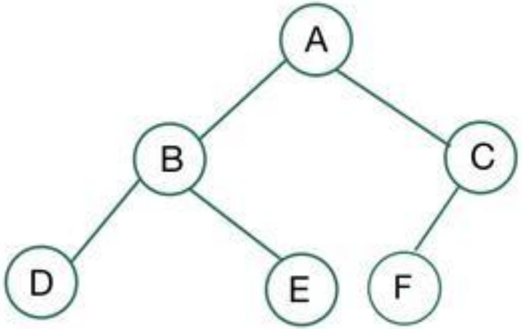
Full Binary Tree:

- Every node must have either **zero** or **two** children (exactly two or none).
- Nodes can be arranged in any order as long as the above rule is followed.
- Not all levels need to be completely filled.

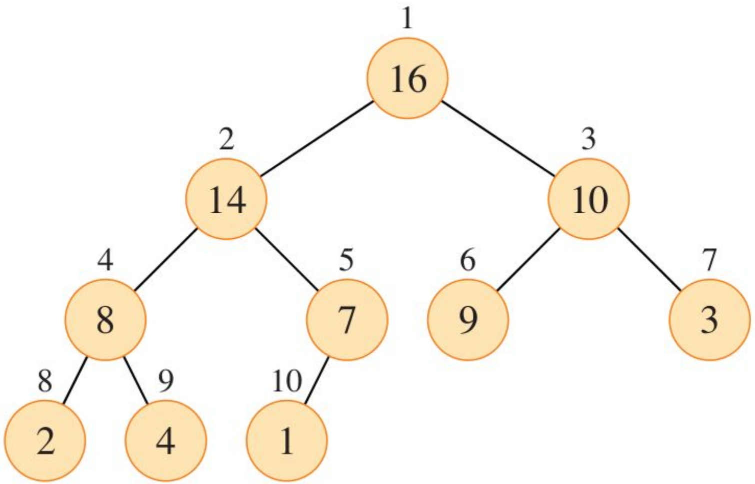
Complete Binary Tree:

- All levels except possibly the last level must be completely filled.
- The last level must be filled with nodes **as far left as possible** (left-filled).
- Nodes in the last level can have **zero** or **one** child.

HEAPS

Feature	Full Binary Tree	Complete Binary Tree
Children per Node	0 or 2	Exactly 2 (except possibly last level)
Last Level Filling	Not necessarily filled completely	Filled from left, may miss nodes on right
Node Ordering	No specific order	Left-filled
Example Use Case	Not widely used itself	Heaps (priority queues)
		

Heaps

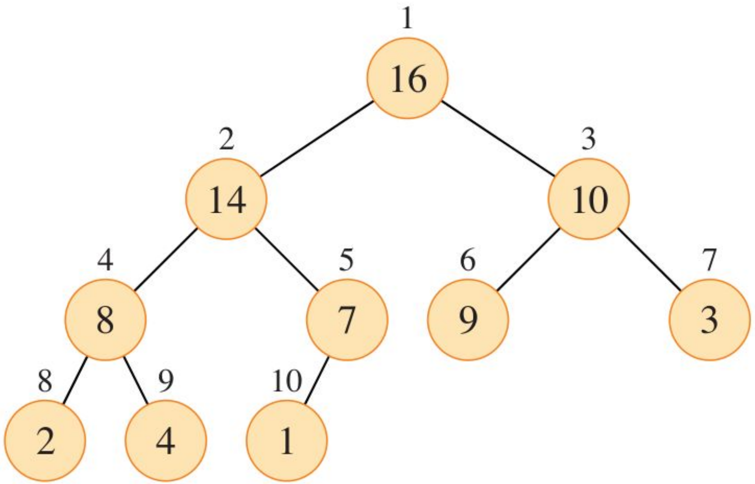


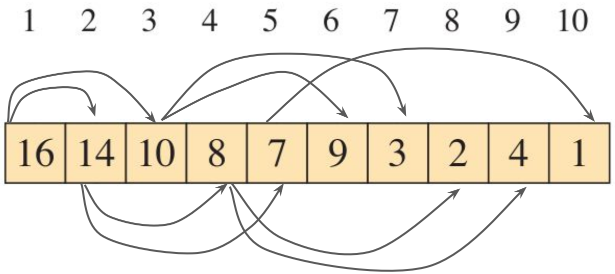
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

A heap can be stored as an array A.

- Root of tree is $A[1]$.
- Parent of $A[i] = A[\lfloor i/2 \rfloor]$.
- Left child of $A[i] = A[2i]$.
- Right child of $A[i] = A[2i + 1]$.

HEAPS

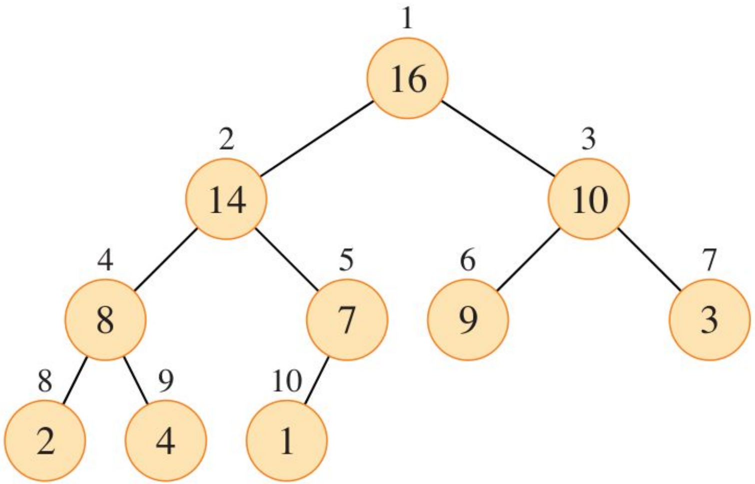


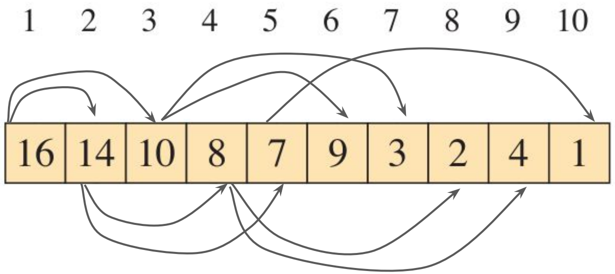


A heap can be stored as an array A.

- Root of tree is $A[1]$.
- Parent of $A[i] = A[\lfloor i/2 \rfloor]$.
- Left child of $A[i] = A[2i]$.
- Right child of $A[i] = A[2i + 1]$.

HEAPS





A heap can be stored as an array A.

- Root of tree is $A[1]$.
- Parent of $A[i] = A[\lfloor i/2 \rfloor]$.
- Left child of $A[i] = A[2i]$.
- Right child of $A[i] = A[2i + 1]$.

PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

HEAPS

Types of binary heaps:

1. **Max-Heap** (largest element at root)

   Property - For every node other than the root:

   $A[\text{PARENT}(i)] \geq A[i]$

   Usually used in sorting.

HEAPS

Types of binary heaps:

1. **Max-Heap** (largest element at root)

   Property - For every node other than the root:
   A[PARENT(i)] ≥ A[i]

   Usually used in sorting.

2. **Min-Heap** (smallest element at root)

   Property - For every node other than the root:
   A[PARENT(i)] ≤ A[i]

   Usually used in priority queue.

HEAPS

Basic procedures in heap:

1. **MAX-HEAPIFY** → maintains max heap property.

2. **BUILD-MAX-HEAP** → produces max heap from unsorted input array.

3. **HEAPSORT** → sorts an array in place.

Heaps

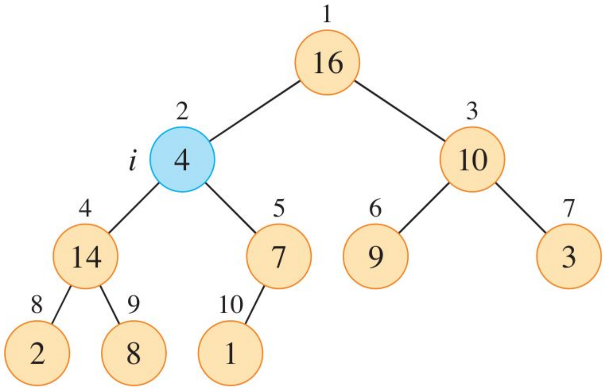
Basic procedures in heap:

1. **Max-Heapify** → maintains max heap property.
2. **Build-Max-Heap** → produces max heap from unsorted input array.
3. **Heapsort** → sorts an array in place.
4. **Max-Heap-Insert**
   Max-Heap-Extract-Max
   Max-Heap-Increase-Key
   Max-Heap-Maximum

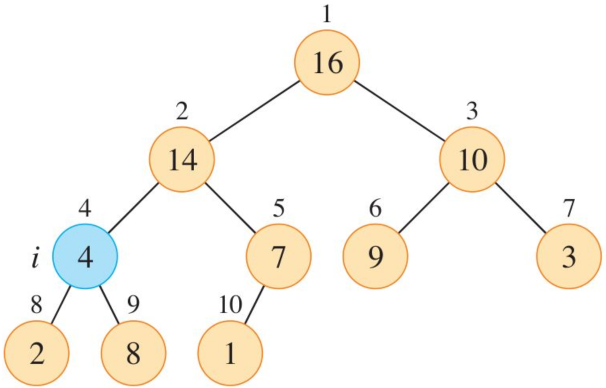
Implements Priority Queue

Max-Heap Conditions

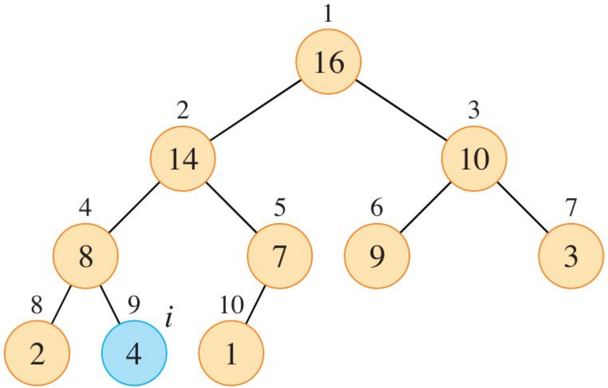
Heaps



1
16
2
3
i
4
10
4
5
6
7
14
7
9
3
8
9
10
2
8
1



1
16
2
3
14
10
4
5
6
7
i
4
7
9
3
8
9
10
2
8
1



1
16
2
3
14
10
4
5
6
7
8
7
9
3
8
9
i
10
2
4
1

Max-Heapify

Heaps

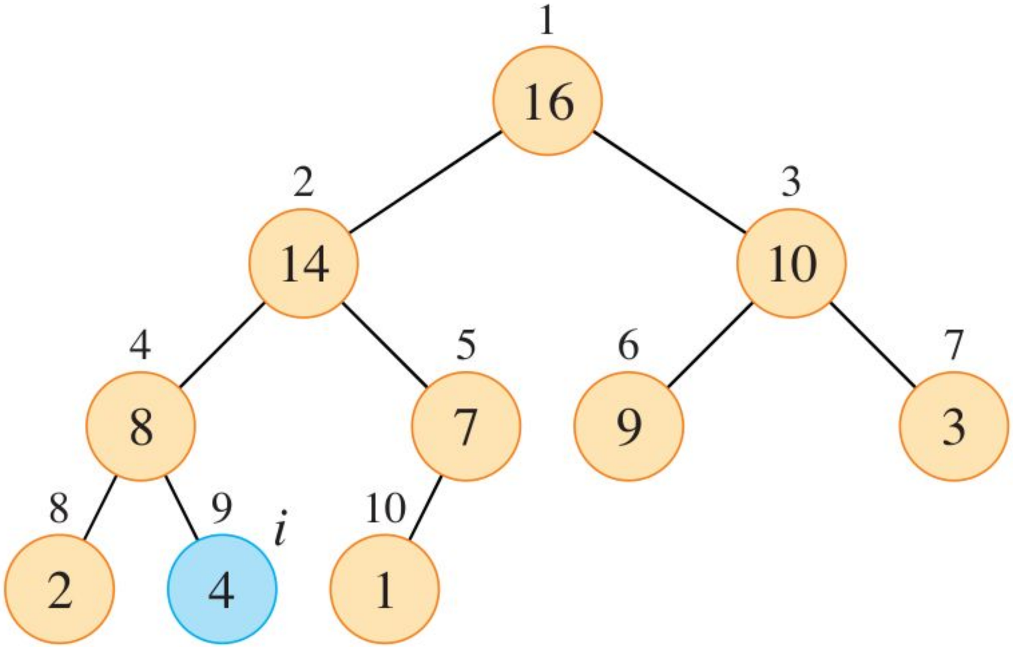
```
MAX-HEAPIFY(A, i)
  l = LEFT(i)
  r = RIGHT(i)
  if l ≤ A.heap-size and A[l] > A[i]
      largest = l
  else largest = i
  if r ≤ A.heap-size and A[r] > A[largest]
      largest = r
  if largest ≠ i
      exchange A[i] with A[largest]
      MAX-HEAPIFY(A, largest)
```

The way MAX-HEAPIFY works:

- Compare $A[i]$, $A[\text{LEFT}(i)]$, and $A[\text{RIGHT}(i)]$.
- If necessary, swap $A[i]$ with the larger of the two children to preserve heap property.
- Continue this process of comparing and swapping down the heap, until subtree rooted at i is max-heap. If we hit a leaf, then the subtree rooted at the leaf is trivially a max-heap.

Build-Max-Heap

Heaps



Note that in a binary heap, the subarray $A[\lfloor n/2 \rfloor + 1 : n]$ are all leaves of the tree.

BUILD-MAX-HEAP(A, n)

  $A.heap\text{-}size = n$

  for $i = \lfloor n/2 \rfloor$ **downto** 1

    MAX-HEAPIFY(A, i)

A	4	1	3	2	16	9	10	14	8	7

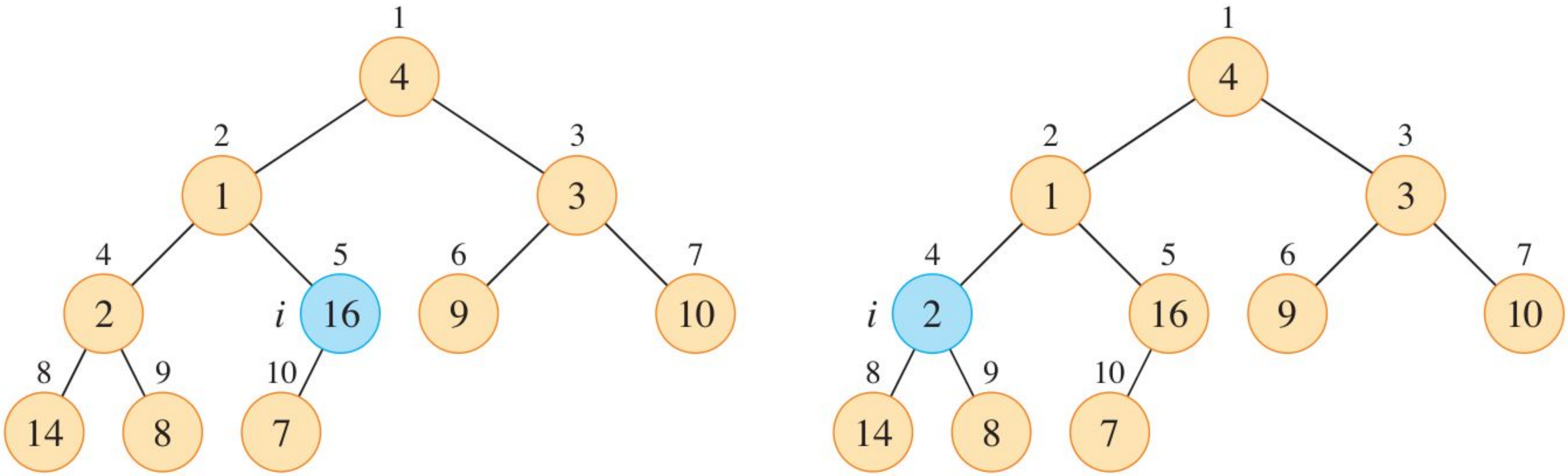
Example

Building a max-heap by calling BUILD-MAX-HEAP$(A, 10)$ on the following unsorted array $A[1:10]$ results in the first heap example.

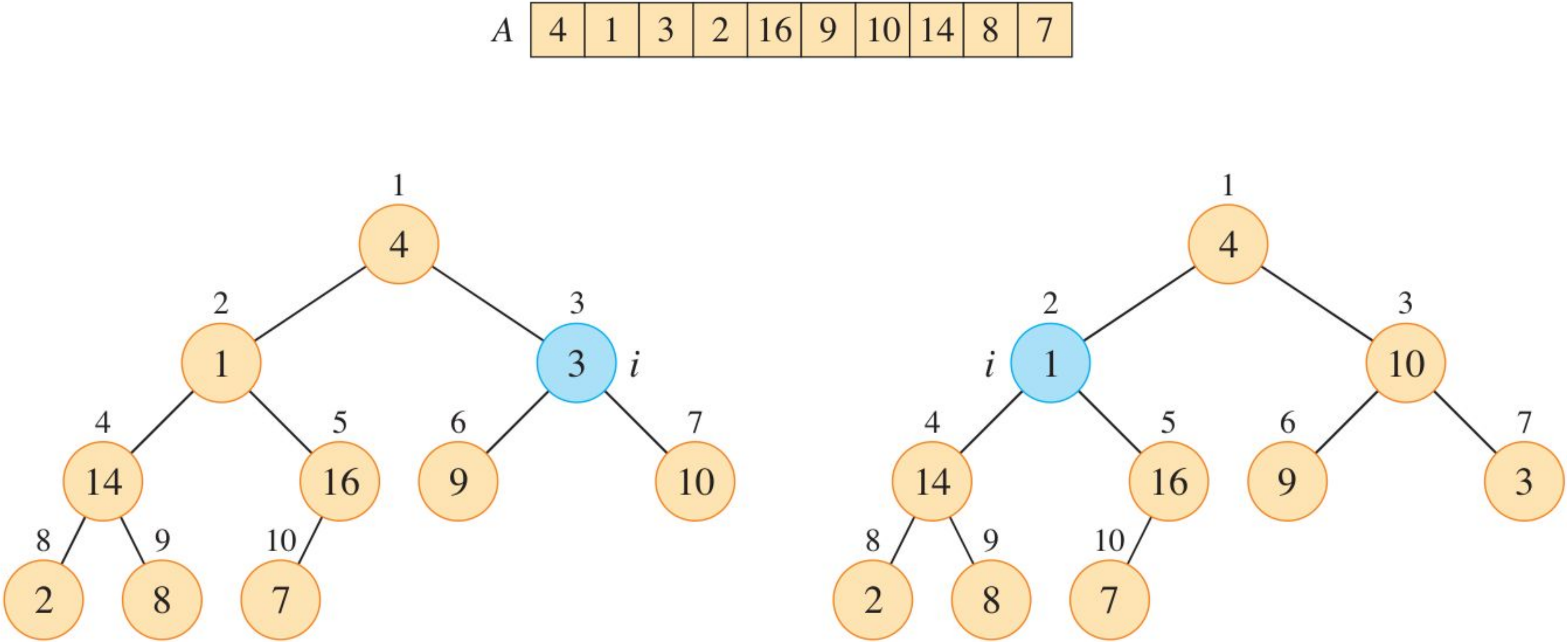
- $A.heap\text{-}size$ is set to 10.
- i starts off as 5.
- MAX-HEAPIFY is applied to subtrees rooted at nodes (in order): $A[5]$, $A[4]$, $A[3]$, $A[2]$, $A[1]$.

Heaps

A									
4	1	3	2	16	9	10	14	8	7

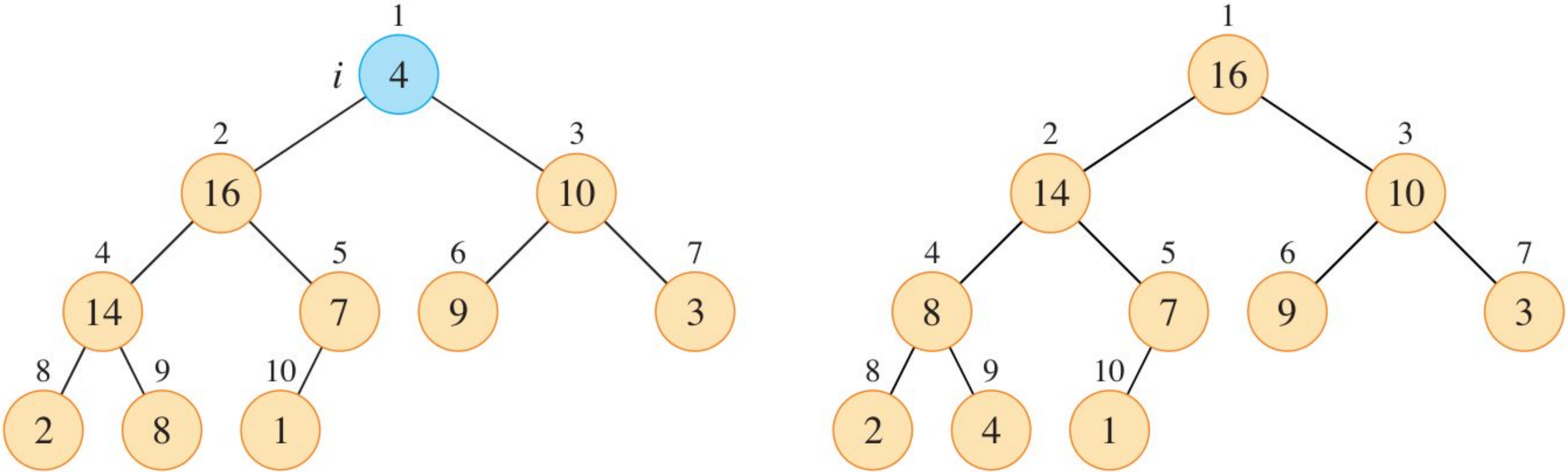


Heaps



HEAPS

A	4	1	3	2	16	9	10	14	8	7



Applications

- Heap Sort
- Priority Queue

Heap Sort

HEAPS

```
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)
```

HEAPS