Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Contents:

Topics to be covered:

- Introduction to Heap
- Max heap and Min heap
- Operation on Heap
- Heap sort
- Priority Queue using Binary Heap

Heap Introduction

- Heap data structure is a complete binary tree that satisfy heap property. It is also called as binary heap.
- A complete binary tree is a tree in which
  - every level, except possibly the last, is filled
  - all the nodes are as far left as possible

- the *min-heap property*: the value of each node is greater than or equal to the value of its parent, with the minimum-value element at the root.

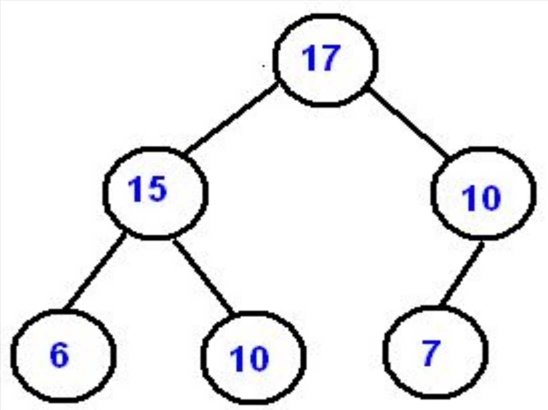
the *max-heap property*: the value of each node is less than or equal to the value of its parent, with the maximum-value element at the root.

Types of Heap

Max-heaps

- (largest element at root), have the *max-heap property:*
  - for all nodes i, excluding the root:

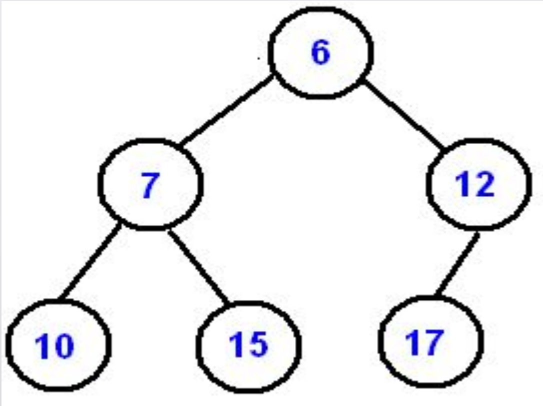
A[PARENT(i)] ≥ A[i]



Min-heaps

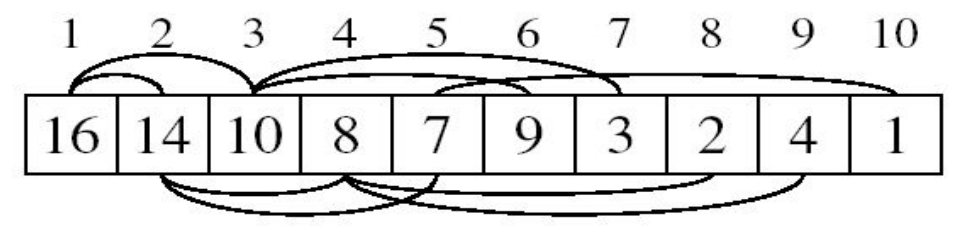
- (smallest element at root), have the *min-heap property:*
  - for all nodes i, excluding the root:

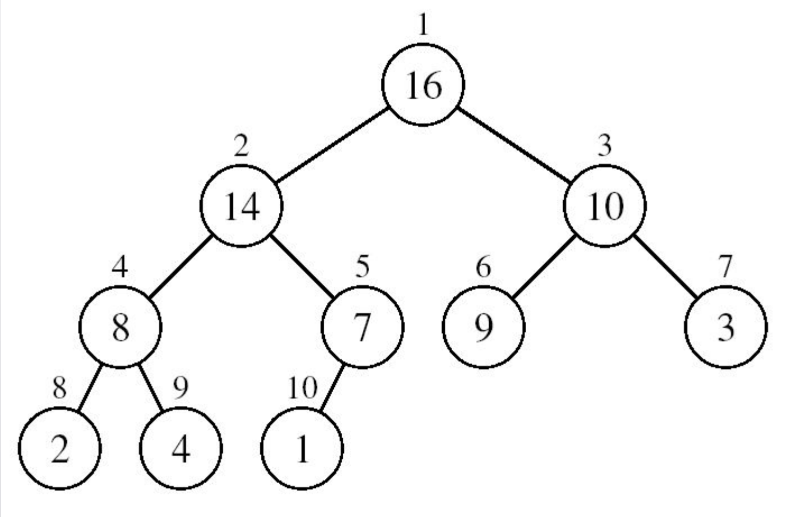
A[PARENT(i)] ≤ A[i]



Array Representation of Heap

- A heap can be stored as an array *A*.
  - Root of tree is A[1]
  - Left child of A[i] = A[2i]
  - Right child of A[i] = A[2i + 1]
  - Parent of A[i] = A[$\lfloor i/2 \rfloor$]





Operation on Heap

- Restoring or maintaining max-heap property
  - MAX-HEAPIFY
- Create a max heap from an unordered array
  - BUILD-MAX-HEAP
- Sort an array
  - Heap Sort
- Priority Queue

Restoring the heap property

- Consider the situation in which a particular node does not follow heap property means A[parent[i]]< A[i] for max heap
- How to resolve the problem so that integrity of heap will be maintained?
    - Exchange with larger child
    - Move down the tree
    - Continue until node is not smaller than children

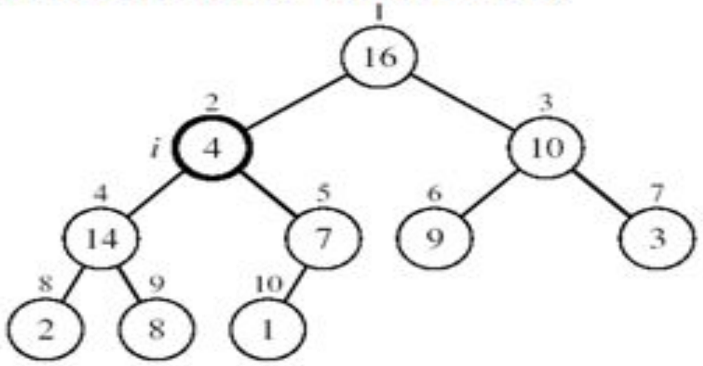
Maintaining heap property using MAX_HEAPIFY

- Assumptions:
  - Left and Right sub trees of i are max-heaps
  - A[i] may be smaller than its children

MAX-HEAPIFY(A, i, n)

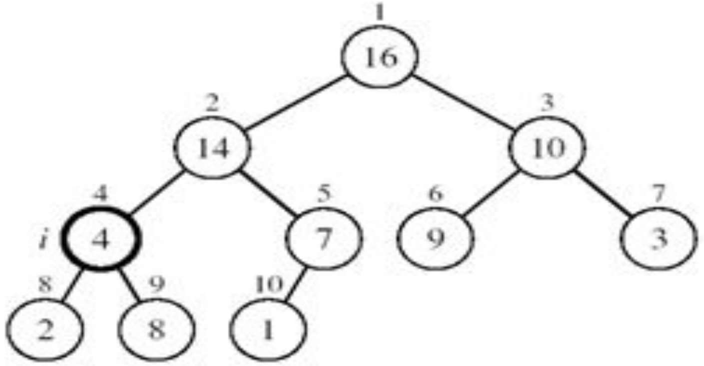
```
l ← LEFT(i)
r ← RIGHT(i)
if l ≤ n and A[l] > A[i]
   then largest ←l
   else largest ←i
if r ≤ n and A[r] > A[largest]
   then largest ←r
if largest ≠ i
   then exchange A[i] ↔ A[largest]
        MAX-HEAPIFY(A, largest, n)
```

MAX-HEAPIFY(A, 2, 10)



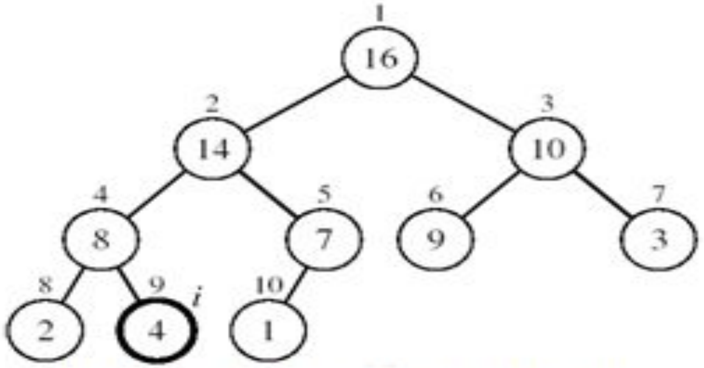
A[2] violates the heap property

A[2] $\leftrightarrow$ A[4]



A[4] violates the heap property

A[4] $\leftrightarrow$ A[9]



Heap property restored

Create a max heap from unordered array(BUILD-MAX-HEAP)

- BUILD-MAX-HEAP will convert an array A[1……n] into max heap.
- Apply MAX-HEAPIFY between 1 and $\lfloor n/2 \rfloor$

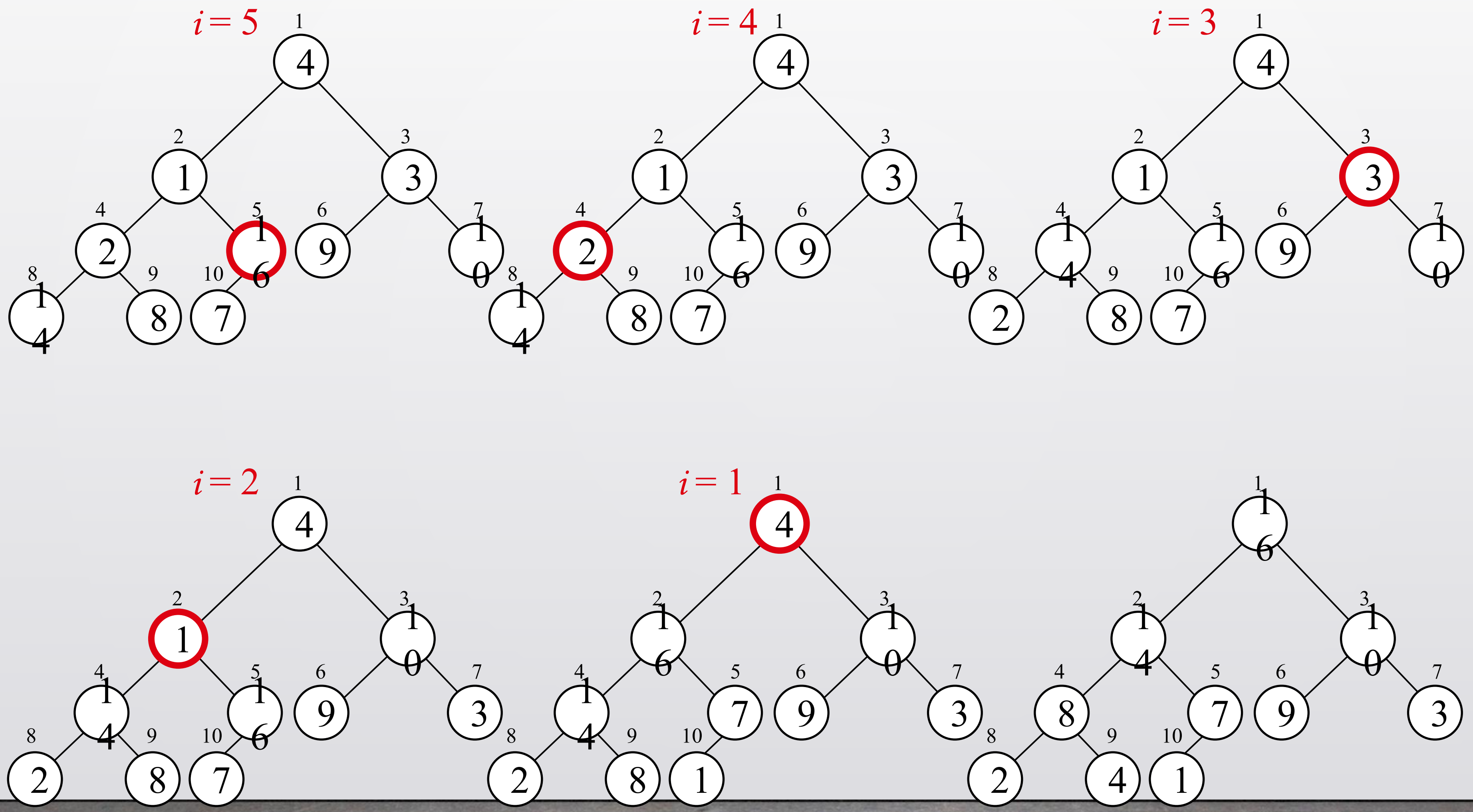
Alg: BUILD-MAX-HEAP(A)

1. n = length[A]
2. **for** i ← $\lfloor n/2 \rfloor$ **downto** 1
3. **do** MAX-HEAPIFY(A, i, n)

Example

A

4	1	3	2	16	9	10	14	8	7



4 1 3 2 16 9 10 14 8 7

Running time

- MAX-HEAPIFY-- $O(\log n)$
- in terms of the height of the heap, as being $O(h)$
- BUILD-MAX-HEAP--

BUILD-MAX-HEAP(A)

```
n = length[A]
for i ← ⌊n/2⌋ downto 1
      do MAX-HEAPIFY(A, i, n)        O(lgn)
```

$\left.\begin{array}{l}\text{lines 2–3}\end{array}\right\} O(n)$

$\Rightarrow$ Running time: $O(n\lg n)$

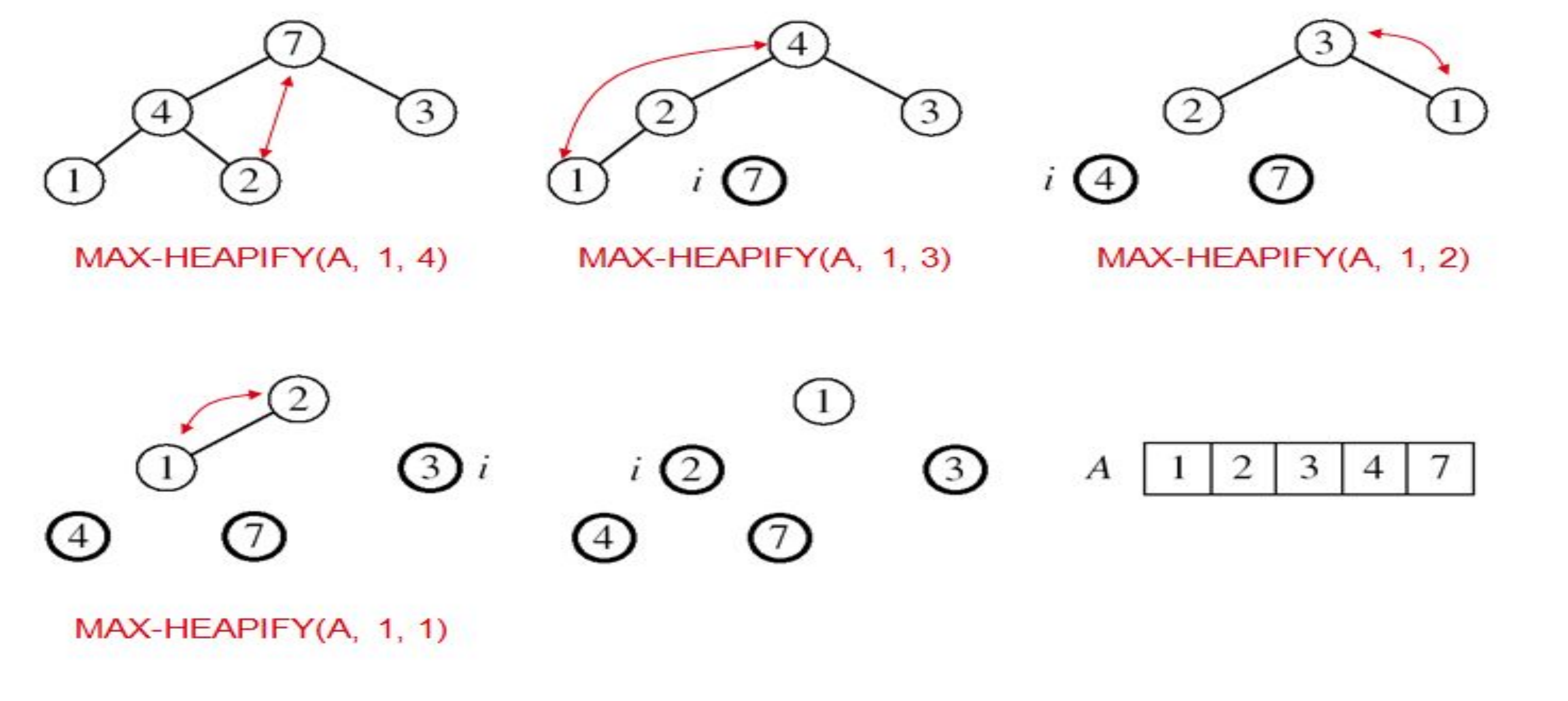
HEAP-SORT

- Steps to be followed for Heap Sort—
- Build the max heap using unordered array
- Exchange the root element with the last element
- Discard the last element by decreasing the heap size
- Call MAX-HEAPIFY on the new root node
- Repeat this process until only one node remains

HEAP SORT ALGORITHM

```
HEAP-SORT(A)
    // Input: A: an (unsorted) array
    // Output: A modified to be sorted from smallest to largest
    // Running Time: O(n log n) where n = length[A]
    BUILD-MAX-HEAP(A)
    for i = length[A] downto 2
        exchange A[1] and A[i]
        heap-size[A] ← heap-size[A] − 1
        MAX-HEAPIFY(A, 1)
```

EXAMPLE A=[7, 4, 3, 1, 2]



Priority Queue using binary heap

- Each element is associated with a value
- The key with highest priority will be extracted first
- There are two types of priority queue:-
  - MAX-PRIORITY QUEUE- max element is extracted
  - MIN-PRIORITY QUEUE – min element is extracted

Operation on priority queue

- MAX- priority queue support following operation:-
  - INSERT(S, x): <u>inserts</u> element x into set S
  - EXTRACT-MAX(S): <u>removes and returns</u> element of S with largest key
  - MAXIMUM(S): <u>returns</u> element of S with largest key
  - INCREASE-KEY(S, x, k): <u>increases</u> value of element x's key to k (Assume $k \geq$ x's current key value)

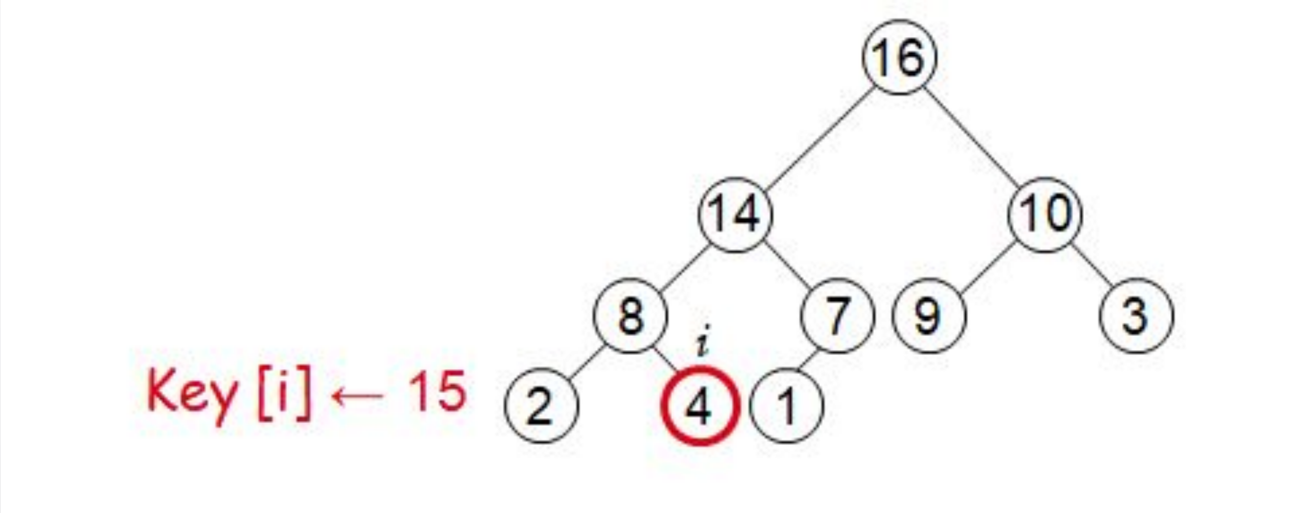
MAXIMUM(S)

- Return the largest element of the heap
- **return** $A[1]$
- Running time – $O(1)$

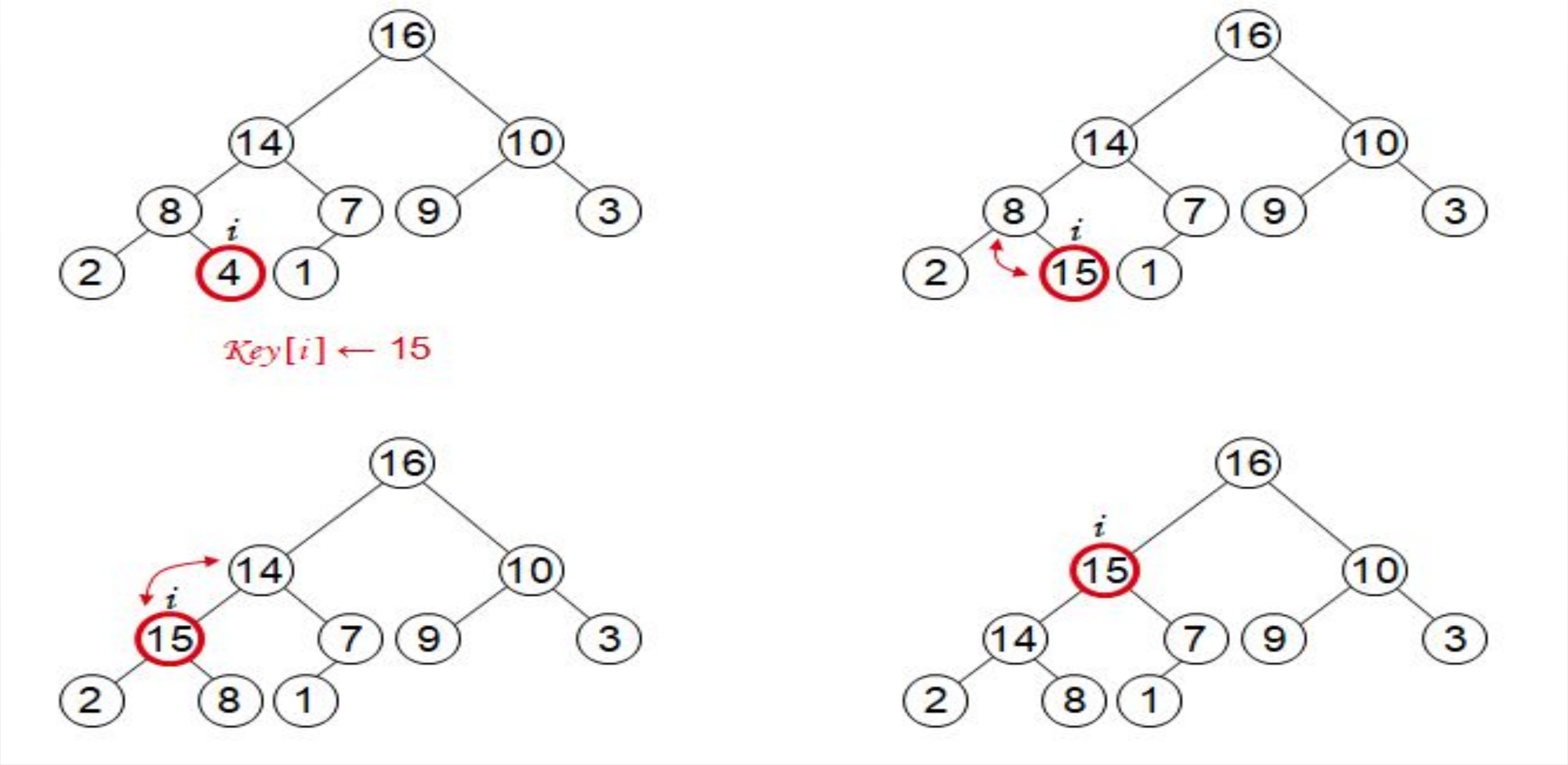
- EXTRACT-MAX(S):
  - Exchange the root element with the last
  - Decrease the size of the heap by 1 element
  - Call MAX-HEAPIFY on the new root, on a heap of size n-1

INCREASE-KEY(S, x, k):

- Increment the key of A[i] to its new value
- If the max-heap property does not hold anymore: traverse a path toward the root to find the proper place for the newly increased key



Example


16
14
10
8
7
9
3
2
4
1
i
Key[i] ← 15
16
14
10
8
7
9
3
2
15
1
i
16
14
10
15
7
9
3
2
8
1
i
16
15
10
14
7
9
3
2
8
1
i

References

- *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009) [1990]*
- *Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03384-4. 1320 pp.*
- Adam Drozdek, Data Structures and Algorithms in C++ (2nd Edition), 2001
- Data Structures A Pseudocode Approach with C, Second Edition by Richard F. Gilberg Behrouz A. Forouzan
- https://www.geeksforgeeks.org/iterative-postorder-traversal/
- https://www.tutorialspoint.com/cplusplus-program-to-perform-inorder-non-recursive-traversal-of-a-given-binary-tree
- https://www.tutorialspoint.com/cplusplus-program-to-perform-preorder-non-recursive-traversal-of-a-given-binary-tree
- **[1]https://www.cs.cmu.edu/~adamchik/15-121/lectures/Trees/trees.html**
- **[2]https://www.javatpoint.com/binary-search-tree**
- **[3] http://web.eecs.umich.edu/~akamil/teaching/su02/080802.ppt**
- **https://courses.csail.mit.edu/**