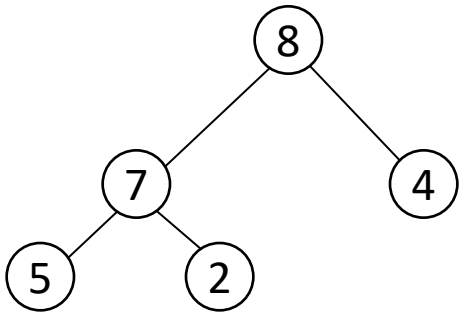
CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

The Heap Data Structure

- *Def:* A **heap** is a nearly complete binary tree with the following two properties:
  - **Structural property:** all levels are full, except possibly the last one, which is filled from left to right
  - **Order (heap) property:** for any node x

$$Parent(x) \geq x$$



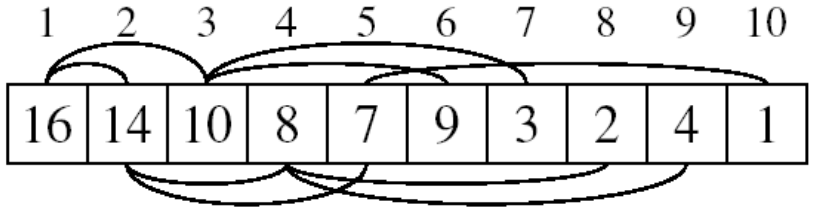
Heap

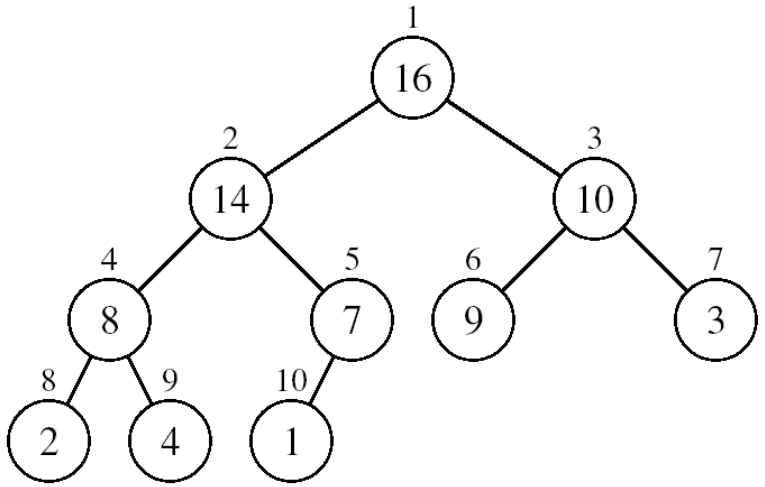
From the heap property, it follows that:
"The root is the maximum element of the heap!"

A heap is a binary tree that is filled in order

Array Representation of Heaps

- 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 $A[i] = A[\lfloor i/2 \rfloor]$
  - Heapsize[A] ≤ length[A]
- The elements in the subarray $A[(\lfloor n/2 \rfloor+1) .. n]$ are leaves



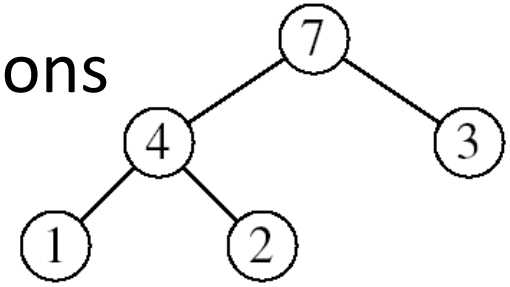


Operations on Heaps

- Maintain/Restore the max-heap property
  - MAX-HEAPIFY - O(log n)
- Create a max-heap from an unordered array
  - BUILD-MAX-HEAP – O(n)
- Sort an array in place
  - HEAPSORT
- Priority queues

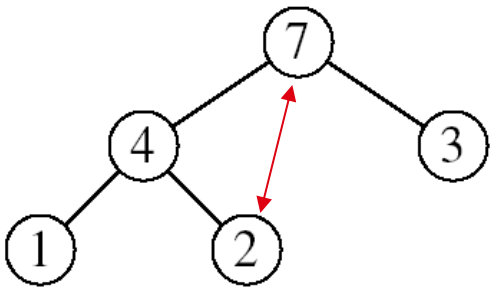
Heapsort

- Goal:
  - Sort an array using heap representations

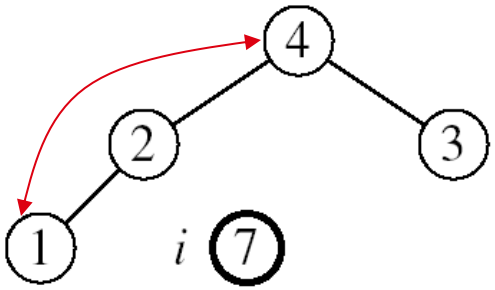


- Idea:
  - Build a **max-heap** from the array
  - Swap the root (the maximum element) with the last element in the array
  - “Discard” this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root

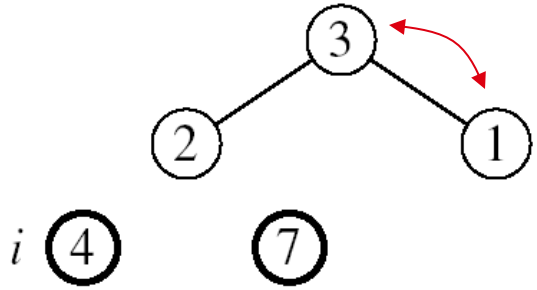
Example: A=[7, 4, 3, 1, 2]



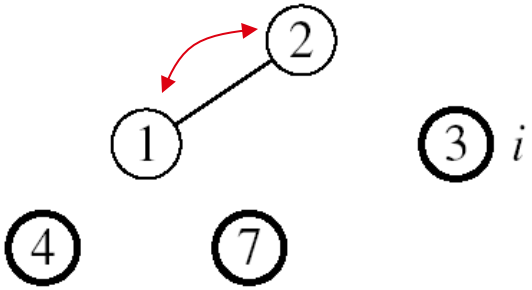
MAX-HEAPIFY(A, 1, 4)

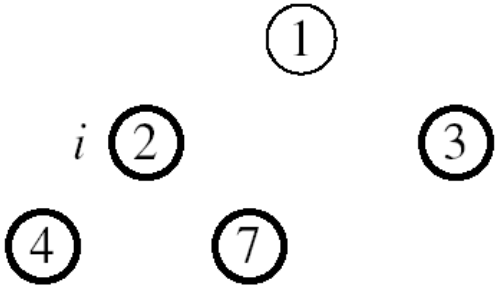


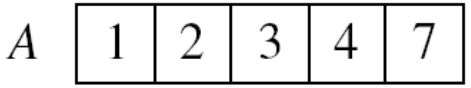
MAX-HEAPIFY(A, 1, 3)



MAX-HEAPIFY(A, 1, 2)







MAX-HEAPIFY(A, 1, 1)

Alg: HEAPSORT*(A)*

1. BUILD-MAX-HEAP(*A*) — *O*(n)
2. **for** i ← length[*A*] **downto** 2
3. **do** exchange *A*[1] ↔ *A*[i]
4. MAX-HEAPIFY(*A*, 1, i - 1) — *O*(lgn)

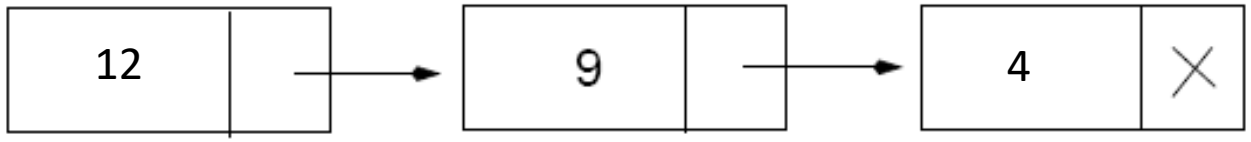
n-1 times

- Running time: *O*(nlgn) --- Can be shown to be *Θ*(nlgn)

Priority Queues

Properties

- Each element is associated with a value (priority)
- The key with the highest (or lowest) priority is extracted first



Operations on Priority Queues

- Max-priority queues support the following operations:
  - INSERT(S, x): inserts element x into set S
  - EXTRACT-MAX(S): removes and returns element of S with largest key
  - MAXIMUM(S): returns element of S with largest key
  - INCREASE-KEY(S, x, k): increases value of

HEAP-MAXIMUM

Goal:

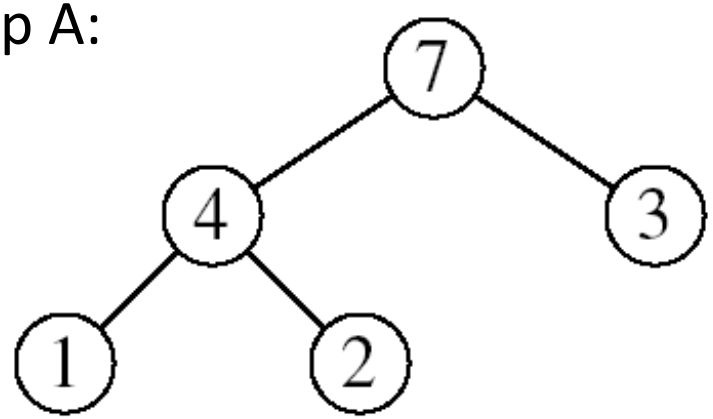
- Return the largest element of the heap

Running time: $O(1)$

Alg: HEAP-MAXIMUM(*A*)

1. **return** *A*[1]

Heap A:



Heap-Maximum(A) returns 7

HEAP-EXTRACT-MAX

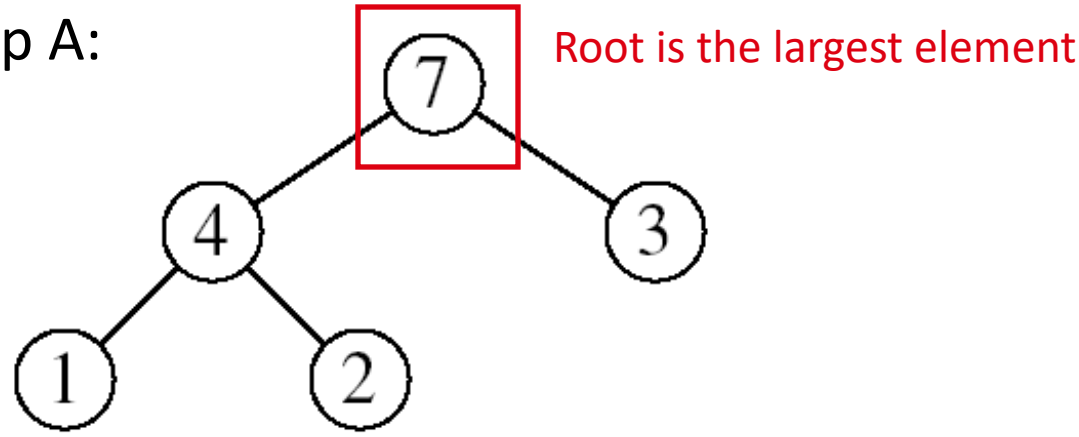
Goal:

- Extract the largest element of the heap (i.e., return the max value and also remove that element from the heap

Idea:

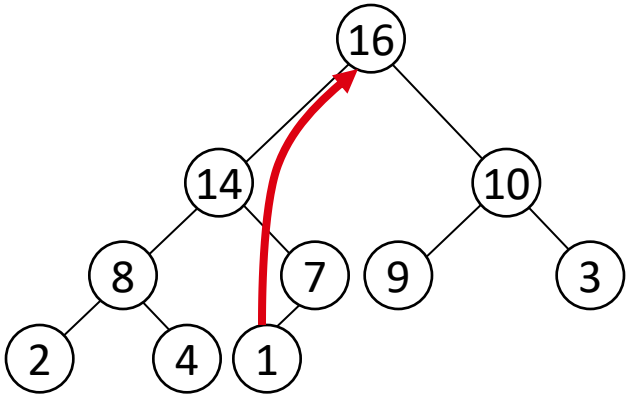
- Exchange the root element with the last
- Decrease the size of the heap by 1 element
- Call MAX-HEAPIFY on the new root, on a heap of size n-1

Heap A:

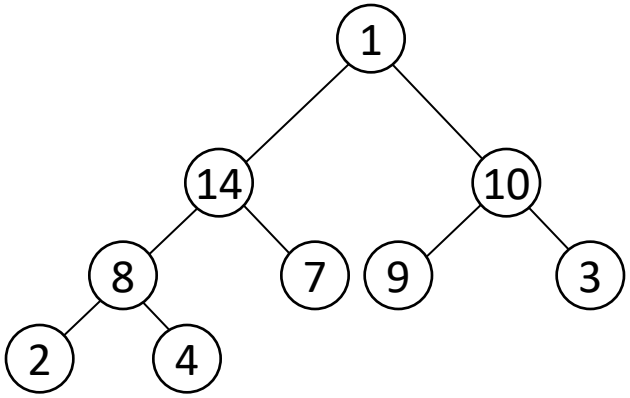


Root is the largest element

Example: HEAP-EXTRACT-MAX

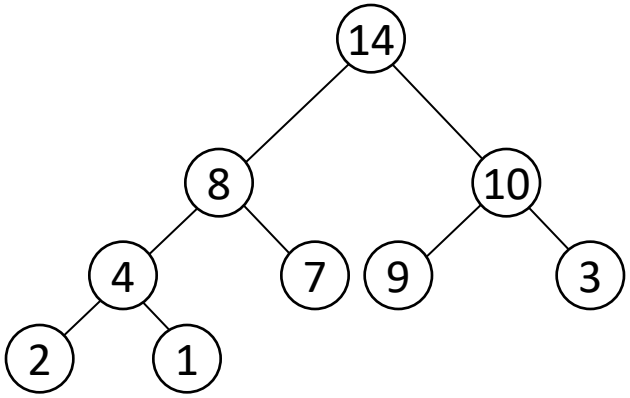


max = 16



Heap size decreased with 1

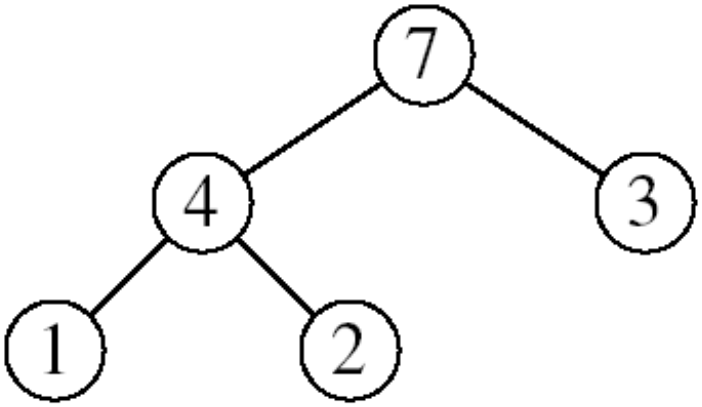
Call MAX-HEAPIFY(*A*, 1, n-1)



HEAP-EXTRACT-MAX

Alg: HEAP-EXTRACT-MAX(A, n)

1. **if** $n < 1$
2. **then error** "heap underflow"
3. $max \leftarrow A[1]$
4. $A[1] \leftarrow A[n]$
5. MAX-HEAPIFY(A, 1, n-1)
6. **return** max

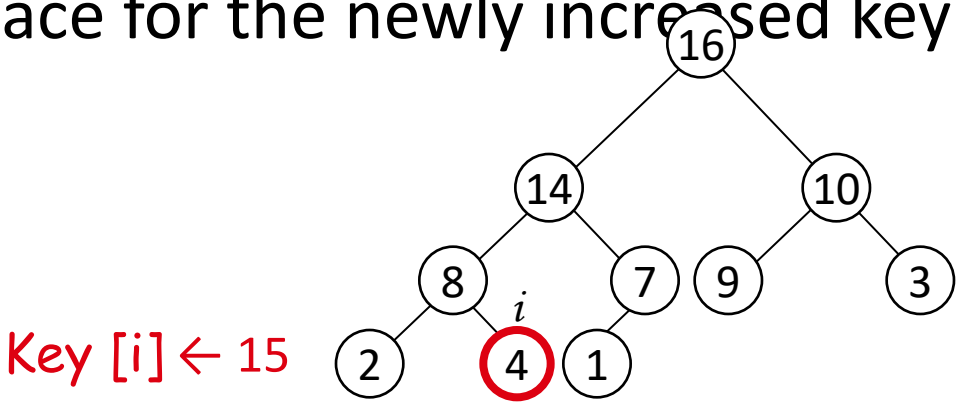


remakes heap

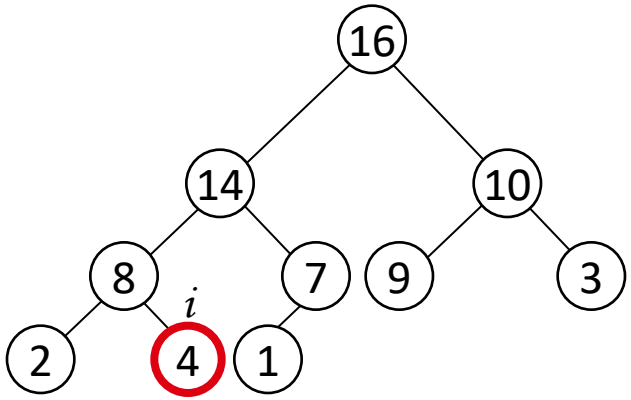
Running time: $O(lgn)$

HEAP-INCREASE-KEY

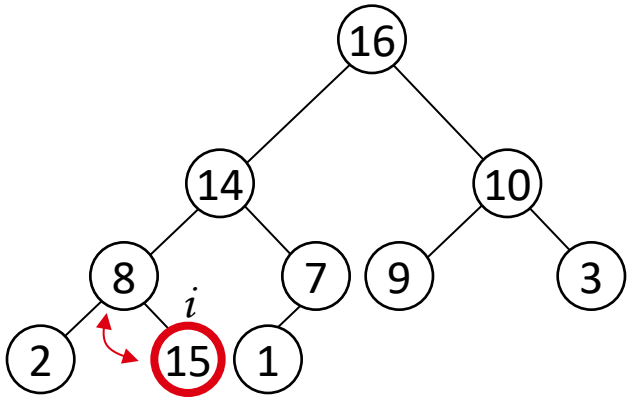
- Goal:
  - Increases the key of an element i in the heap
- Idea:
  - Increment the key of $A[i]$ to its new value
  - If the max-heap property does not hold anymore: traverse a path toward the root to find the proper place for the newly increased key

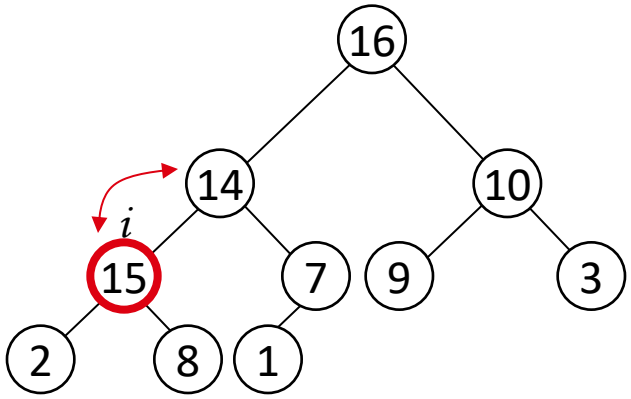


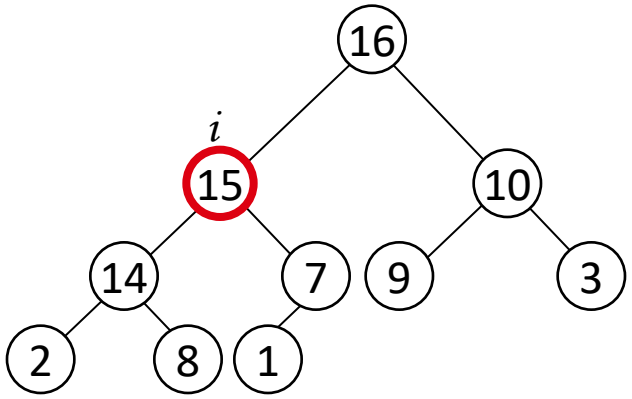
Example: HEAP-INCREASE-KEY



$Key[i] \leftarrow 15$





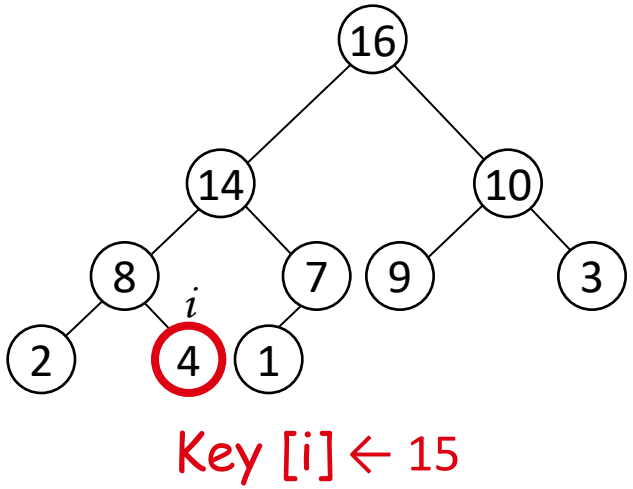


HEAP-INCREASE-KEY

Alg: HEAP-INCREASE-KEY(A, i, key)

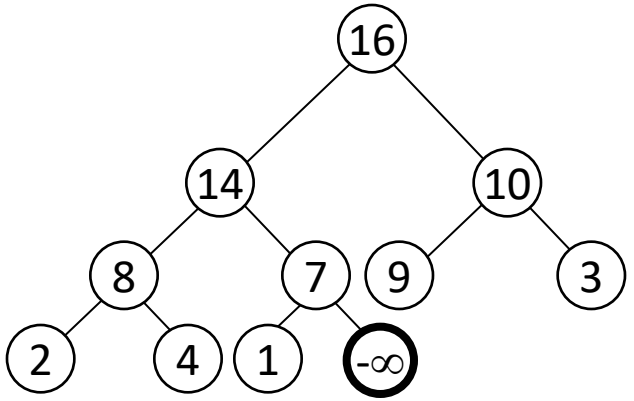
1. **if** key < $A[i]$
2. **then error** "new key is smaller than current key"
3. $A[i] \leftarrow$ key
4. **while** i > 1 and $A[PARENT(i)] < A[i]$
5. **do** exchange $A[i] \leftrightarrow A[PARENT(i)]$
6. $i \leftarrow PARENT(i)$

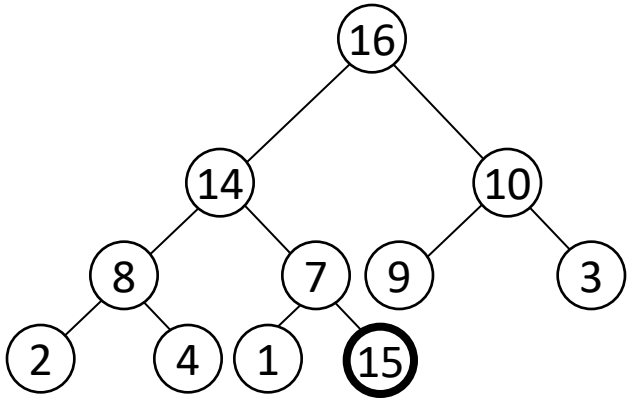
- Running time: $O(\lg n)$



MAX-HEAP-INSERT

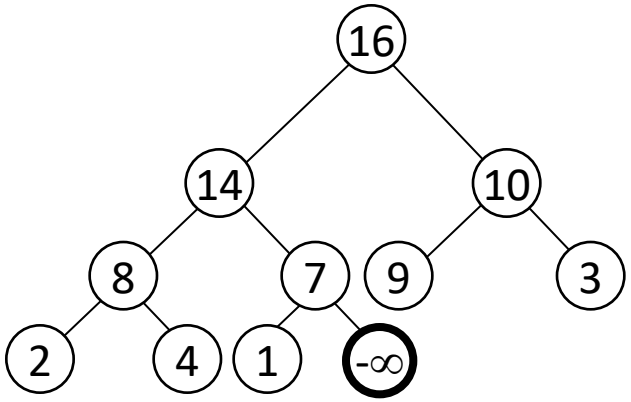
- Goal:
  - Inserts a new element into a max-heap
- Idea:
  - Expand the max-heap with a new element whose key is -∞
  - Calls HEAP-INCREASE-KEY to set the key of the new node to its correct value and maintain the max-heap property



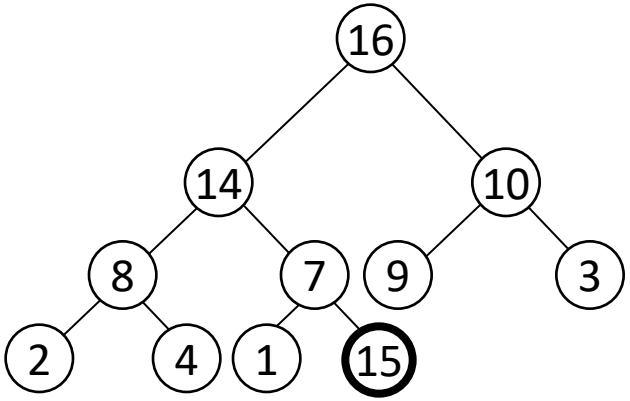


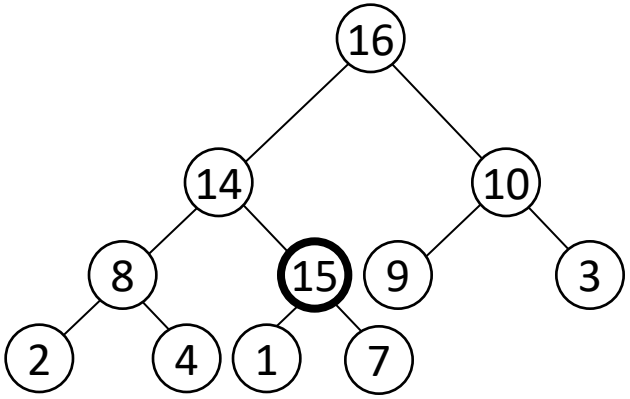
Example: MAX-HEAP-INSERT

Insert value 15:
- Start by inserting -∞

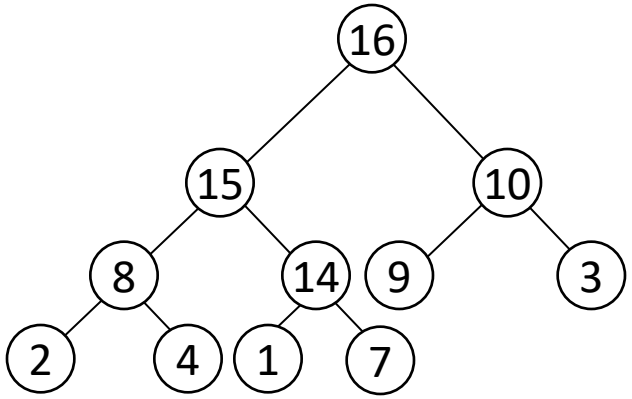


Increase the key to 15
Call HEAP-INCREASE-KEY on A[11] = 15





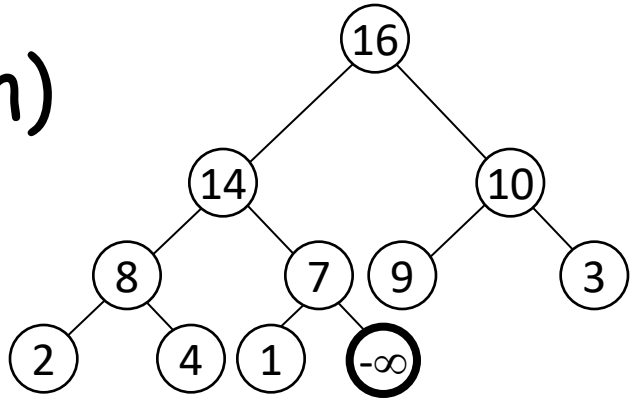
The restored heap containing the newly added element



MAX-HEAP-INSERT

Alg: MAX-HEAP-INSERT(*A*, key, n)

1. heap-size[*A*] ← n + 1
2. *A*[n + 1] ← -∞
3. HEAP-INCREASE-KEY(*A*, n + 1, key)



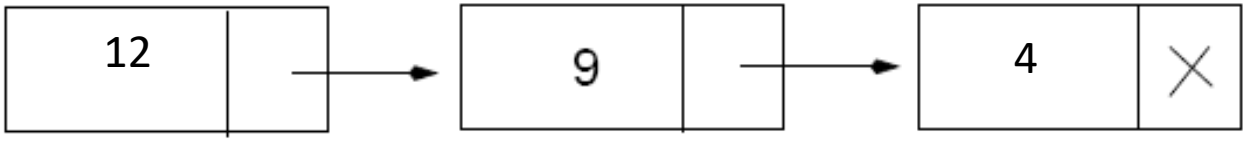
Running time: *O*(lgn)

Summary

- We can perform the following operations on heaps:
  - MAX-HEAPIFY $O(\lg n)$
  - BUILD-MAX-HEAP $O(n)$
  - HEAP-SORT $O(n \lg n)$
  - MAX-HEAP-INSERT $O(\lg n)$
  - HEAP-EXTRACT-MAX $O(\lg n)$
  - HEAP-INCREASE-KEY $O(\lg n)$
  - HEAP-MAXIMUM $O(1)$

} Average $O(\lg n)$

Priority Queue Using Linked List



Remove a key: O(1)

Insert a key: O(n)

Increase key: O(n)

Extract max key: O(1)

Average: O(n)