Heapsort & Priority Queue

Instructor: Ashok Singh Sairam

Lecture Plan

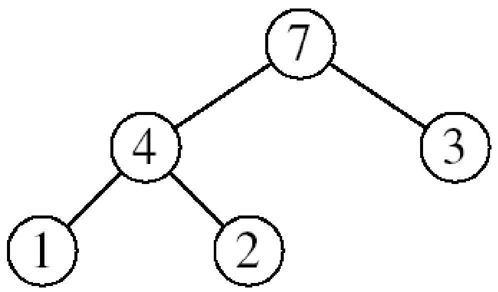
- Heapsort
  - Algorithm, example, running time
- Priority Queue
  - Motivation and definition
  - Operations

Review

- Demonstrate, step by step, the operation of Build-Heap on the array

  A=[5, 3, 17, 10, 84, 19, 6, 22, 9]

Heapsort Idea

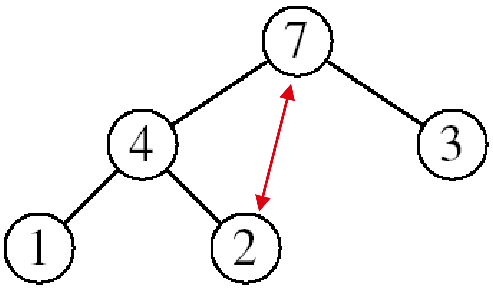


- Goal:
  - Sort an array using heap representations
- Idea:
  - Build a **max-heap** from the array
  - Swap the root (the maximum element) with the last element in the array
  - "Discard" this last node by decreasing the heap size
  - Call MAX-HEAPIFY on the new root
  - Repeat this process until only one node remains

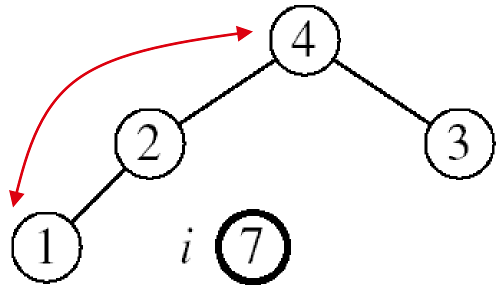
Heapsort Algorithm

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

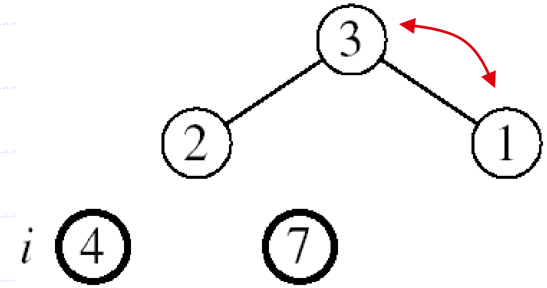
Example: A=[7, 4, 3, 1, 2]



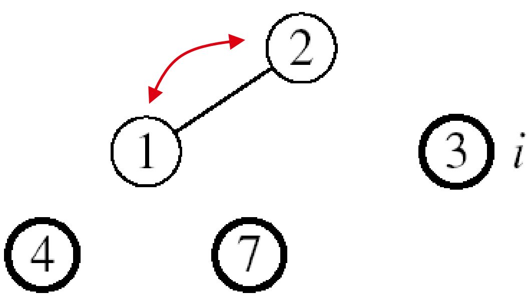
i=5; A[1] ↔ A[i]
MAX-HEAPIFY(A, 1)



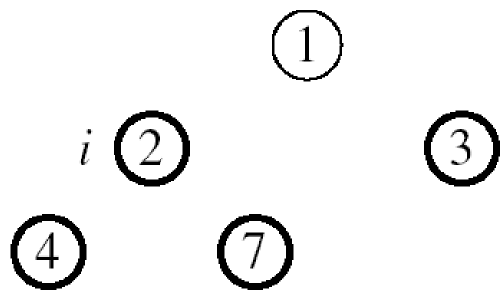
i=4; A[1] ↔ A[i]
MAX-HEAPIFY(A, 1,)



i=3; A[1] ↔ A[i]
MAX-HEAPIFY(A, 1)



i=2; A[1] ↔ A[i]
MAX-HEAPIFY(A, 1, 1)



A	1	2	3	4	7

Running Time of Heapsort

- Build-Max-Heap takes time O(n) and each of the n-1 calls to Max-heapify takes time O(lg n)

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)                          O(n)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1                  n-1 times
    MAX-HEAPIFY(A, 1)                      O(lgn)
```

- Thus Heapsort takes O(n lg n) time

Priority Queue

- A popular application of heaps
- Definition: A priority queue is a data structure for maintaining a set of **S** elements, each with an associated value called key
- The key with the highest (or lowest) priority is extracted first

Priority Queue vs. Queue

- Queue: implements FIFO policy
- Priority Queue: Similar to queue, but each element has a priority
  - Element with highest priority is removed first
- Queue can be thought of as a priority queue where oldest element has highest priority

Motivation

- Standing in a grocery store/movie hall
  - Queue: FIFO
- Boarding a plane
  - First person to arrive is not the first person served
  - Business class passengers, passengers with wheelchair get higher priority
- Hospital
  - Emergency patients get higher priority
- Need to express these priorities in the representation

Operations on Priority Queues

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

HEAP-MAXIMUM

Goal:

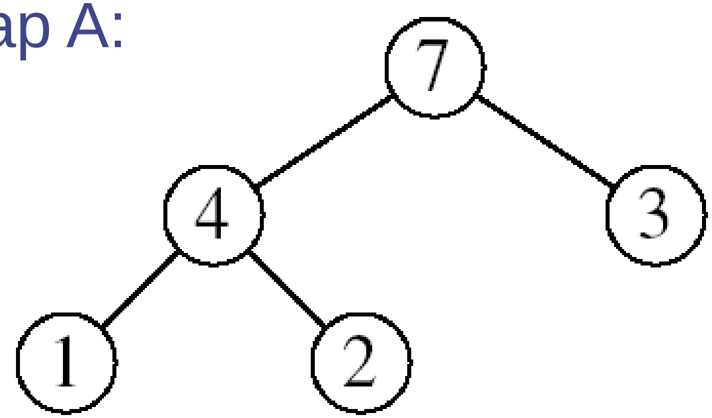
- Return the largest element of the heap

Running time: $O(1)$

HEAP-MAXIMUM(A)

return $A[1]$

Heap A:



Heap-Maximum(A) returns 7

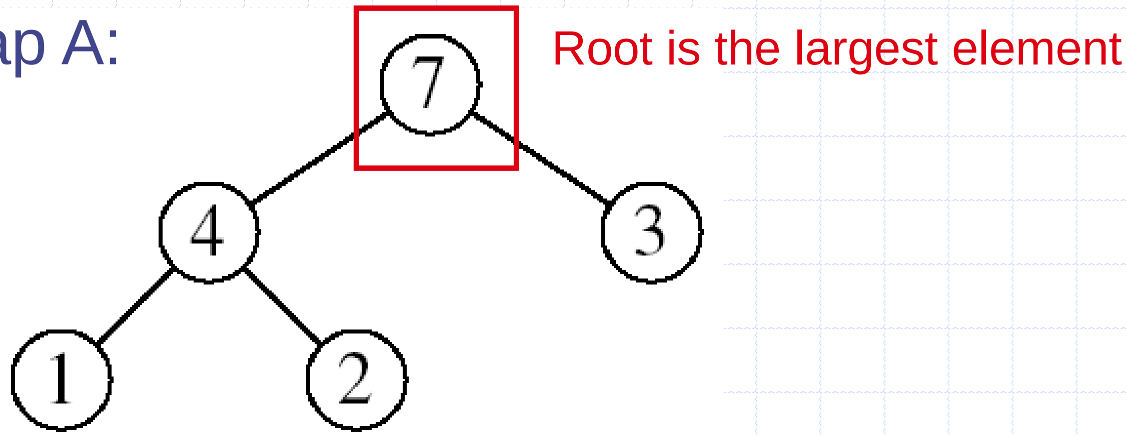
HEAP-EXTRACT-MAX

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:



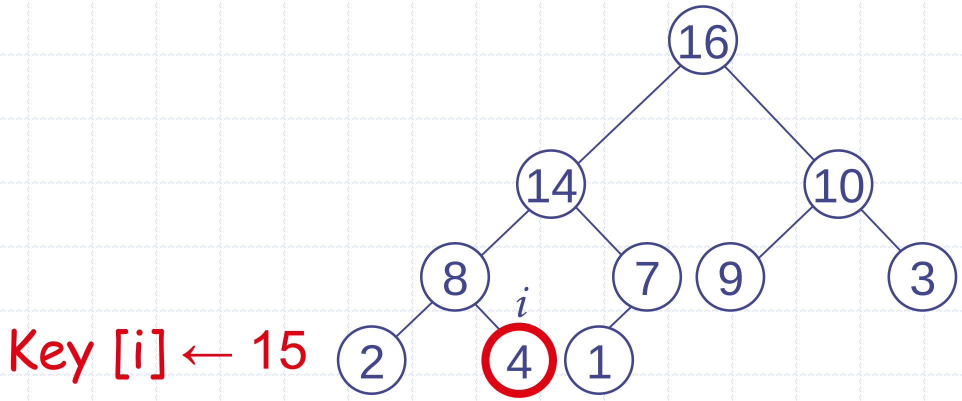
Algorithm: Heap-Extract-Max

```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```

Running time: $O(\lg n)$

HEAP-INCREASE-KEY

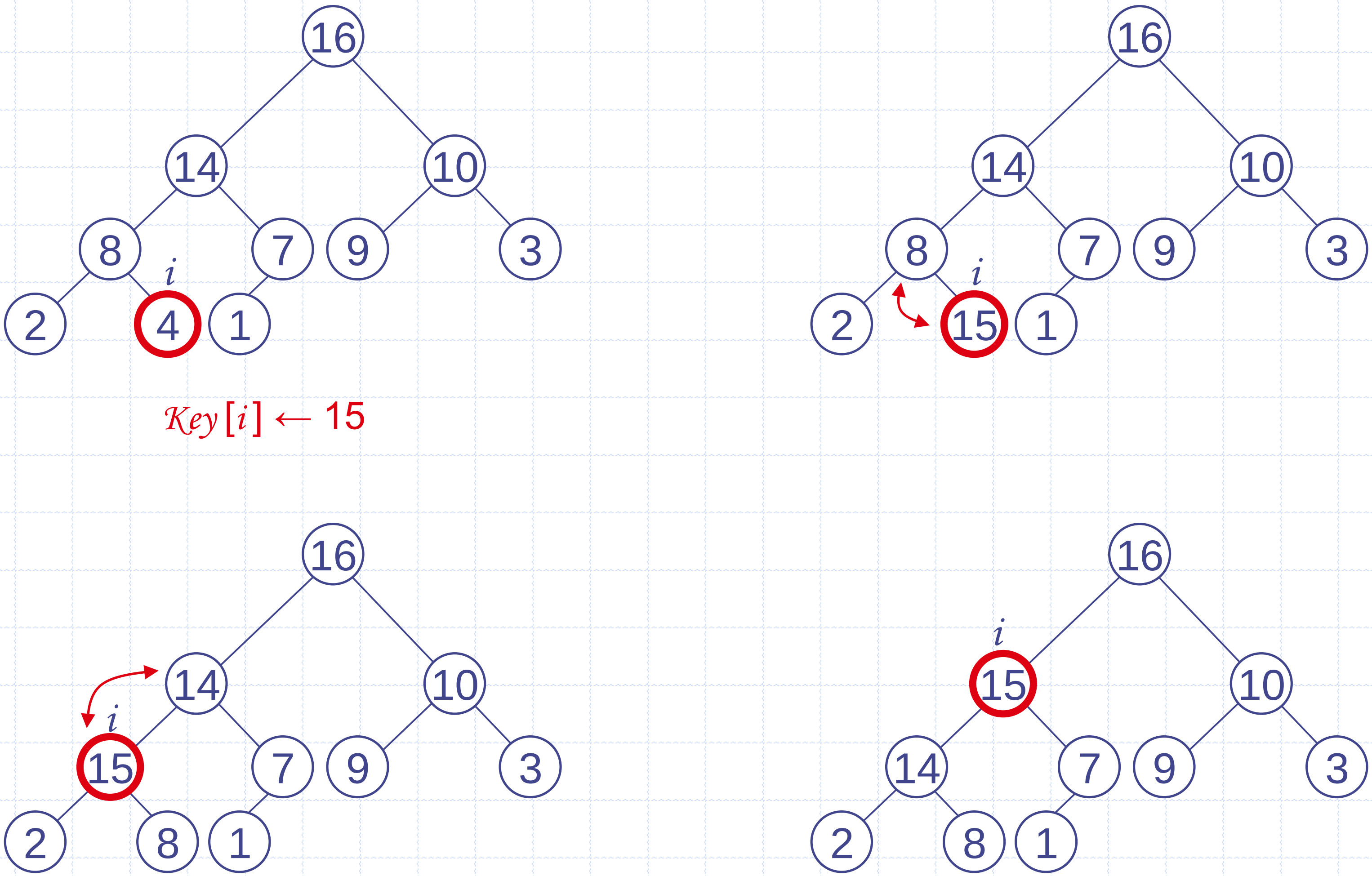
- Goal: Increases the key of an element i in the heap
- Idea:
  - Increment the key of $A[i]$ to its new value
  - If the max-heap property does not hold anymore: traverse a path toward the root to find the proper place for the newly increased key



Algorithm: HEAP-INCREASE-KEY

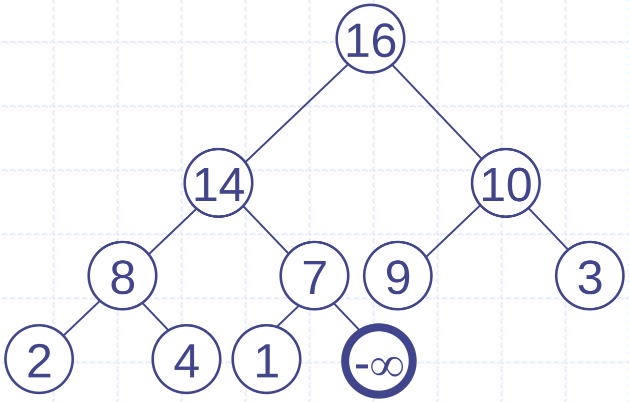
```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```

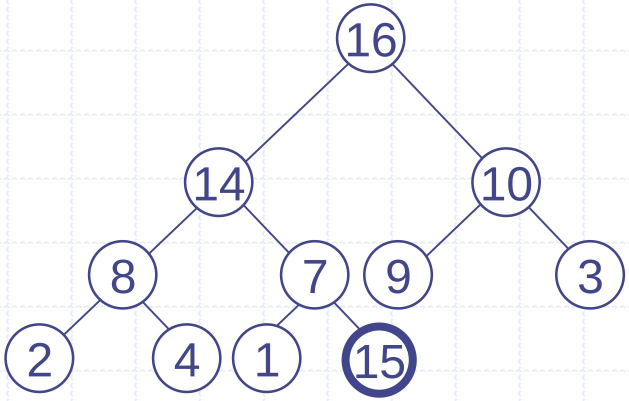
Example: HEAP-INCREASE-KEY



MAX-HEAP-INSERT

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





Algorithm: MAX-HEAP-INSERT

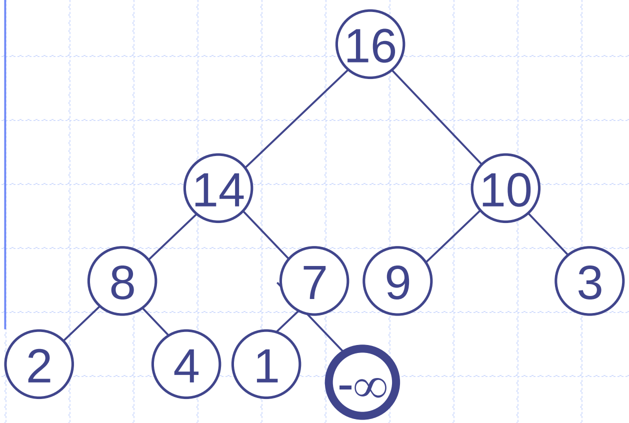
MAX-HEAP-INSERT(A, key)

1 $A.heap\text{-}size = A.heap\text{-}size + 1$
2 $A[A.heap\text{-}size] = -\infty$
3 HEAP-INCREASE-KEY$(A, A.heap\text{-}size, key)$

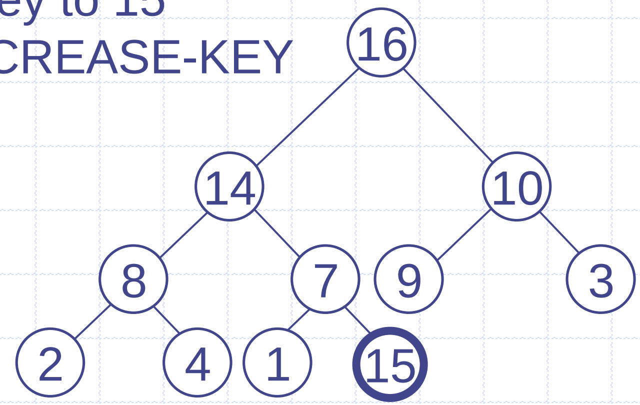
Running time: $O(\lg n)$

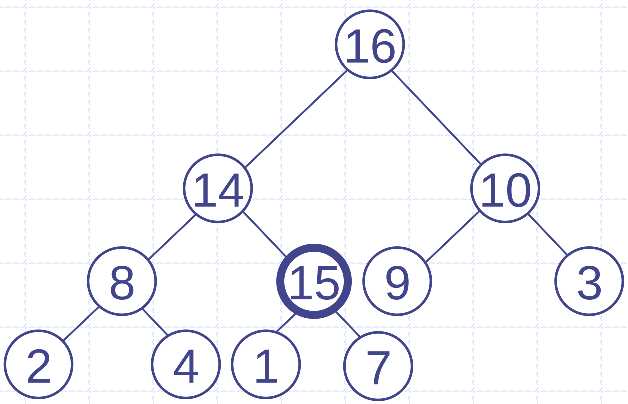
Example: MAX-HEAP-INSERT

Insert value 15:
- Start by inserting -∞

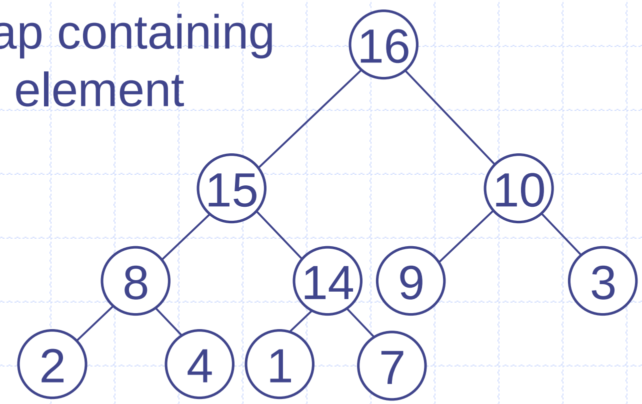


Increase the key to 15
Call HEAP-INCREASE-KEY
on A[11] = 15





The restored heap containing
the newly added element



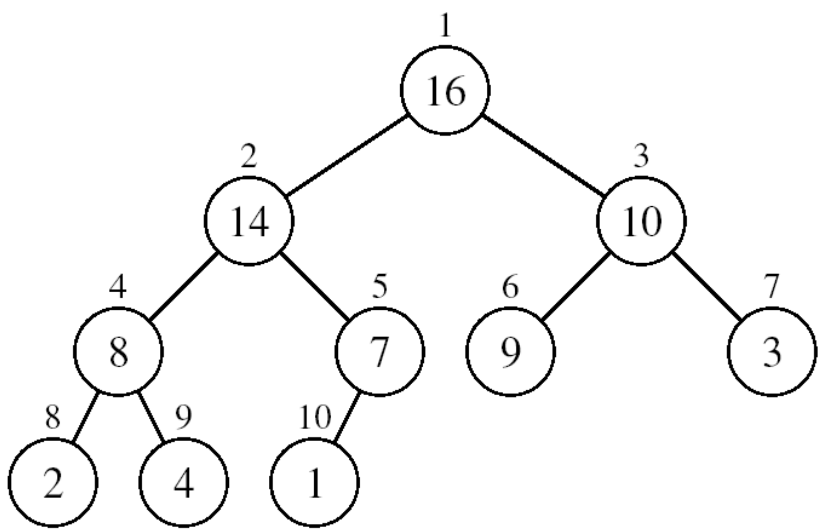
Summary

- 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	$O(1)$

Exercise

- Assuming the data in a max-heap are distinct, what are the possible locations of the second-largest element?



Exercise

(a) What is the maximum number of nodes in a max heap of height h?

(b) What is the maximum number of leaves?

(c) What is the maximum number of internal nodes?

Acknowledgement

- Dr George Bebis, Foundation Professor, Dept of Computer Science and Engineering, University of Nevada Reno