Priority Queue

Dr. Jeevani Goonetillake

Priority Queue

- A priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it.
- In a priority queue, an element with high priority is served before an element with low priority.

Binary Heap

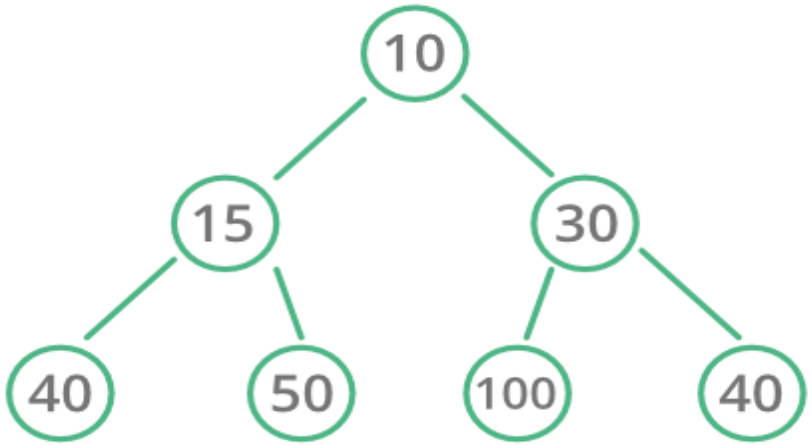
Binary heaps are a common way of implementing priority queues. A binary heap is defined as a binary tree with two additional constraints:

- **Shape property**: a binary heap is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
- **Heap property**: the key stored in each node is either greater than or equal to (≥) or less than or equal to (≤) the keys in the node's children, according to some total order.

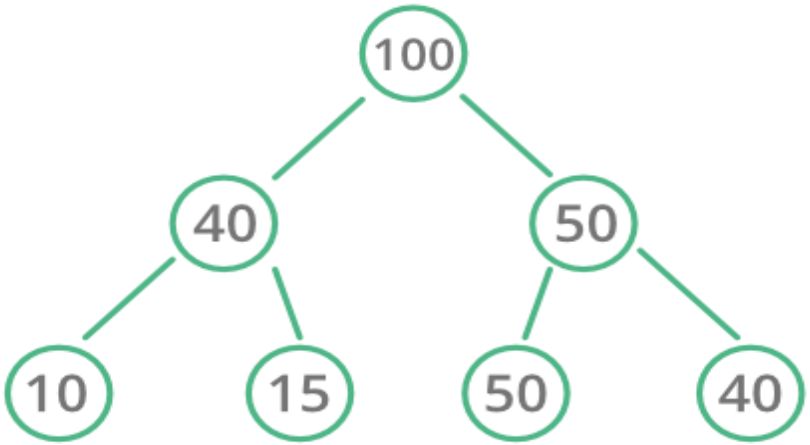
Binary Heap

- Heaps where the parent key is greater than or equal to (≥) the child keys are called max-heaps.
- Those where it is less than or equal to (≤) are called min-heaps.
- Efficient algorithms are known for the two operations needed to implement a priority queue on a binary heap: inserting an element, and removing the smallest or largest element from a min-heap or max-heap, respectively.

Binary Heap



Min Heap



Max Heap

Max-Priority Queue

A max-priority queue supports the following operations.

- INSERT(S, x) inserts the element x into the set S. This operation could be written as $S \leftarrow S \cup \{x\}$.
- MAXIMUM(S) returns the element of S with the largest key.
- EXTRACT-MAX(S) removes and returns the element of S with the largest key.
- INCREASE-KEY(S, x, k) increases the value of element x's key to the new value k, which is assumed to be at least as large as x's current key value.

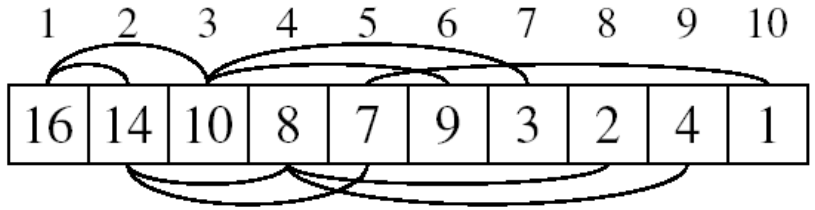
Min-Priority Queue

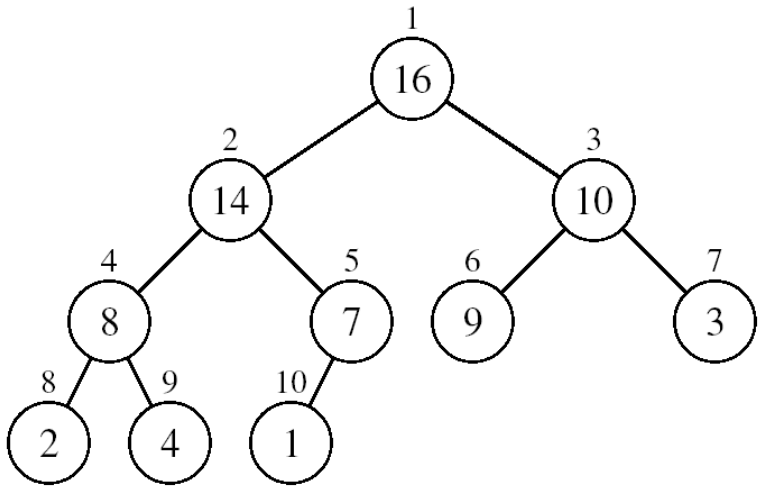
In addition to INSERT(S,x) min-priority queue supports the following operations.

- MINIMUM(S) returns the element of S with the smallest key.
- EXTRACT-MIN(S) removes and returns the element of S with the smallest key.
- DECREASE-KEY(S, x, k) decreases the value of element x's key to the new value k. If the decreases key value of a node is smaller than the parent of the node, it is necessary to traverse up to fix the violated heap property.

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
- The root is the maximum element of the heap





A heap is a binary tree that is filled in order

Max-Priority Queue

- The procedure HEAP-MAXIMUM implements the MAXIMUM operation in O(1) time.

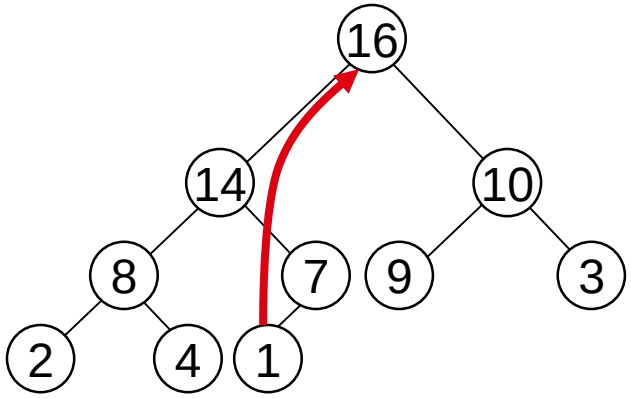
```
HEAP-MAXIMUM(A)
    return A[1]
```

EXTRACT-MAX

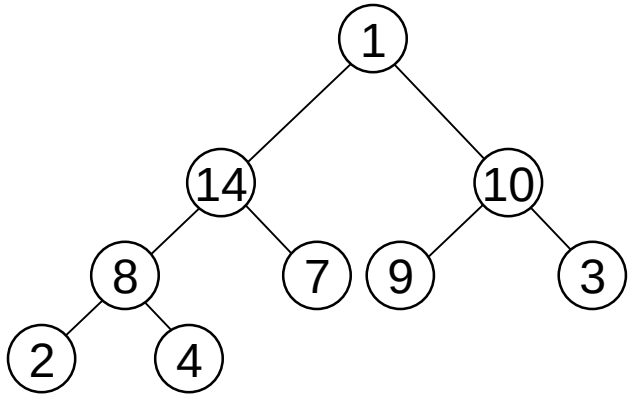
```
HEAP-EXTRACT-MAX(A)
  if heap-size[A] < 1
    then error "heap underflow"
  max ← A[1]
  A[1] ← A[heap-size[A]]
  heap-size[A] ← heap-size[A] − 1
  MAX-HEAPIFY(A, 1)
  return max
```

- The running time of HEAP-EXTRACT-MAX is $O(\lg n)$, since it performs only a constant amount of work on top of the $O(\lg n)$ time for MAX-HEAPIFY.

HEAP-EXTRACT-MAX

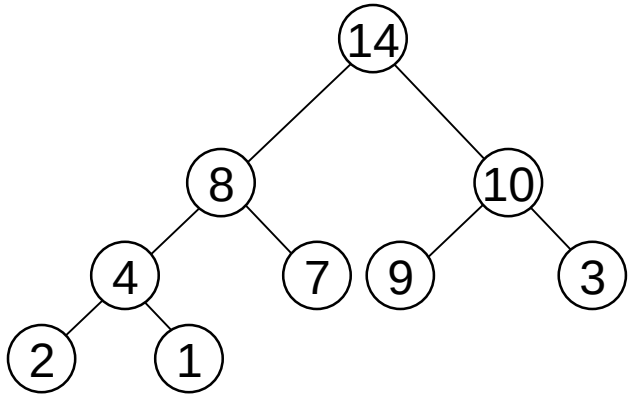


max = 16



Heap size decreased with 1

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



INCREASE-KEY

HEAP-INCREASE-KEY*(A, i, key)*

if *key* < *A*[*i*]

then error “new key is smaller than current key”

A[*i*] ← *key*

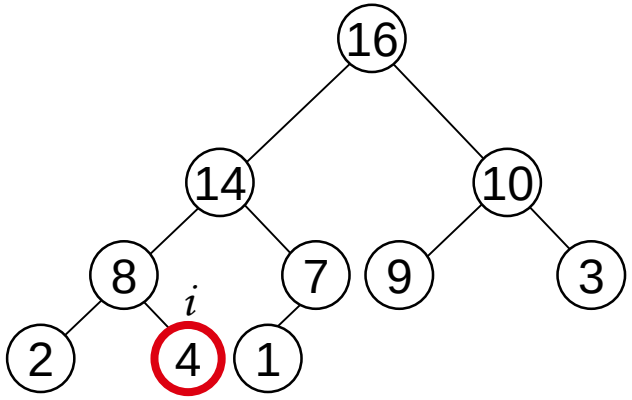
while *i* > 1 and *A*[PARENT*(i)*] < *A*[*i*]

do exchange *A*[*i*] ↔ *A*[PARENT*(i)*]

i ← PARENT*(i)*

- The running time of HEAP-INCREASE-KEY on an *n*-element heap is *O(*lg *n)*, since the path traced from the node updated in line 3 to the root has length *O(*lg *n)*.

HEAP-INCREASE-KEY



$Key[i] \leftarrow 15$