

# Algorithms and Programming I

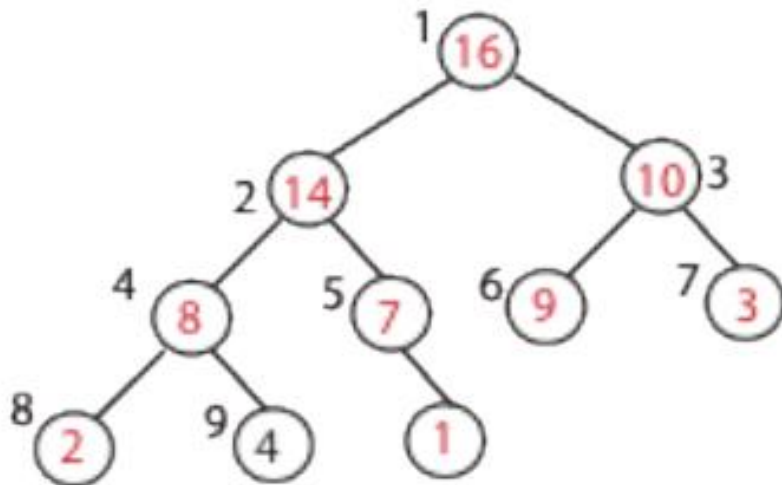
Lecture 7

Heap Sorting

# Review of last lecture

- Heap : is a nearly complete binary tree. Of height  $\Theta(\lg n)$

**Max-Heap Property:** The key of a node is  $\geq$  than the keys of its children.



# Review of last lecture...

## Visualizing an Array as a Tree

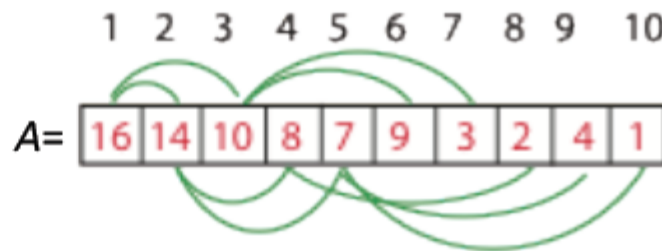
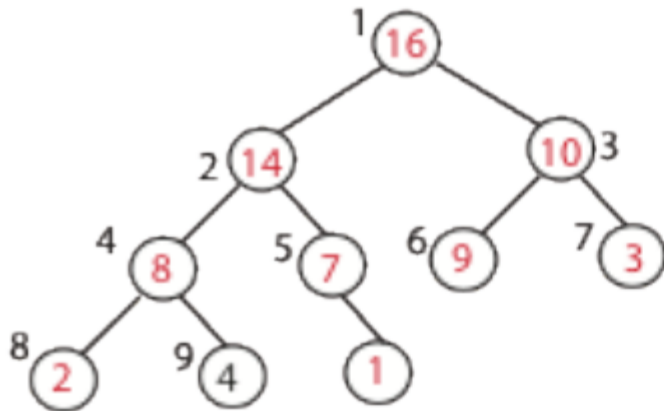
*Root of tree:* first element in the array, corresponding to index = 1

*If a node's index is  $i$  then:*

$\text{parent}(i) = \left\lfloor \frac{i}{2} \right\rfloor$ ; returns index of node's parent, e.g.  $\text{parent}(5)=2$

$\text{left}(i) = 2i$ ; returns index of node's left child, e.g.  $\text{left}(4)=8$

$\text{right}(i) = 2i + 1$ ; returns index of node's right child, e.g.  $\text{right}(4)=9$



# Operations with Heaps

## - *Max\_Heapify* ( $A, i$ )

Correct a single violation of the heap property occurring at the root  $i$  of an otherwise perfect subtree...

**Setting:** Assume that the trees rooted at  $\text{left}(i)$  and  $\text{right}(i)$  are max-heaps, but element  $A[i]$  violates the max-heap property;

i.e.  $A[i]$  is smaller than at least one of  $A[\text{left}(i)]$  or  $A[\text{right}(i)]$ .

**Goal:** fix the subtree rooted at  $i$ .

# Operations with Heaps

## Max\_heapify (A, $i$ )

*Find the index of the largest element among  $A[i]$ ,  $A[\text{left}(i)]$  and  $A[\text{right}(i)]$*

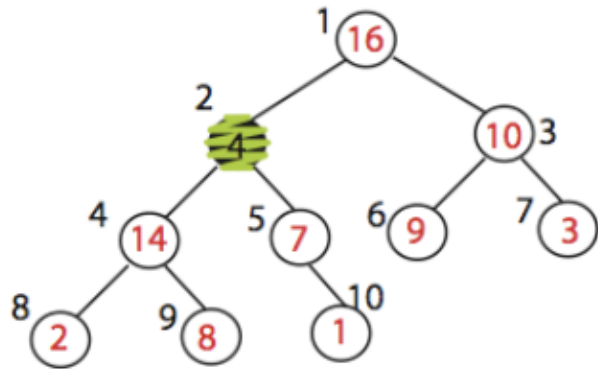
```
 $l \leftarrow \text{left}(i)$   
 $r \leftarrow \text{right}(i)$   
if  $l \leq \text{heap-size}(A)$  and  $A[l] > A[i]$   
  then largest  $\leftarrow l$   
  else largest  $\leftarrow i$   
if  $r \leq \text{heap-size}(A)$  and  $A[r] > A[\text{largest}]$   
  then largest  $\leftarrow r$ 
```

*If this index is different than  $i$ , exchange  $A[i]$  with largest element; then recurse on subtree*

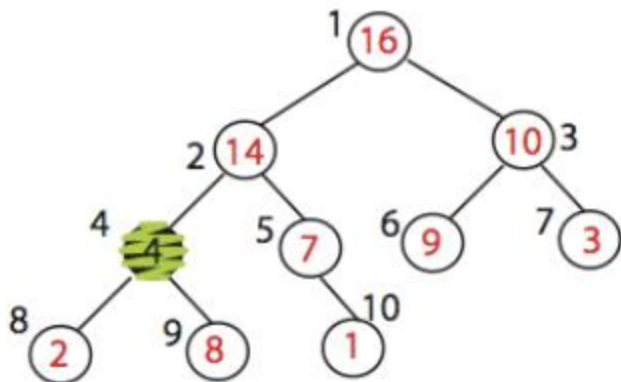
```
if largest  $\neq i$   
  then exchange  $A[i]$  and  $A[\text{largest}]$   
  MAX_HEAPIFY( $A$ , largest)
```

If  $A[i]$  is smaller than both  $A[\text{left}(i)]$  and  $A[\text{right}(i)]$  why do I insist on swapping with largest and not with any one of them, arbitrarily?

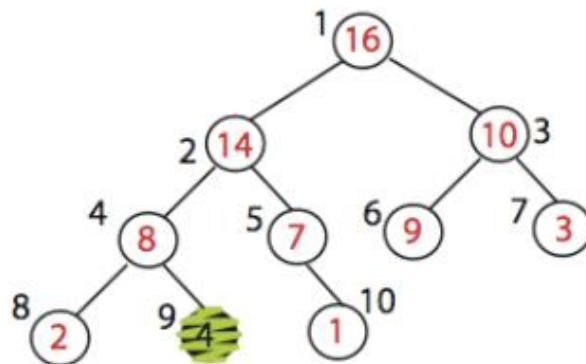
# Max\_Heapify (Example)



MAX\_HEAPIFY (A,2)  
heap\_size[A] = 10



Exchange A[2] with A[4]  
Call MAX\_HEAPIFY(A,4)  
because max\_heap property  
is violated



Exchange A[4] with A[9]  
No more calls

# Operations with Heaps

## - *Max\_Heapify* ( $A, i$ )

Correct a single violation of the heap property occurring at the root  $i$  of an otherwise perfect subtree.

Time  $O(\log n)$ .

## - *Build\_Max\_Heap* ( $A$ )

Produce a max-heap from an unordered array  $A$ .

# Operations with Heaps

Build\_Max\_Heap( $A$ ):

$\text{heap\_size}(A) = \text{length}(A)$

    for  $i \leftarrow \lfloor \text{length}[A]/2 \rfloor$  downto 1

        do Max\_Heapify( $A, i$ )



# Operation with Heaps

## - *Max\_Heapify* ( $A, i$ )

- Correct a single violation of the heap property occurring at the root  $i$  of an otherwise perfect subtree.
- Time  $O(\log n)$ .

## - *Build\_Max\_Heap* ( $A$ )

- Produce a max-heap from an unordered array  $A$ .

## - *Heapsort* ( $A$ )

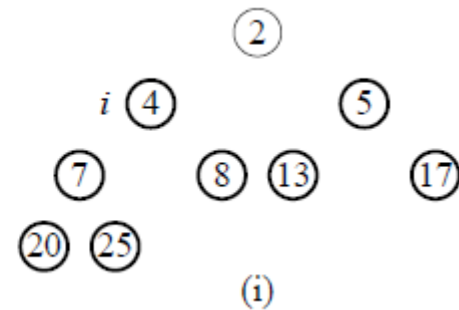
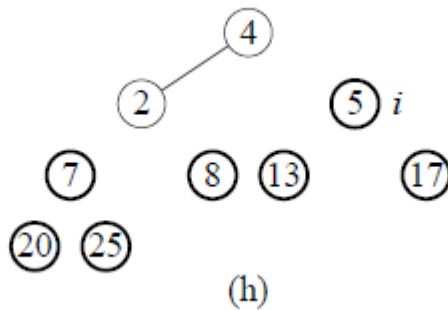
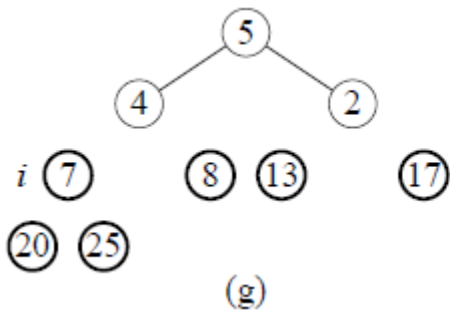
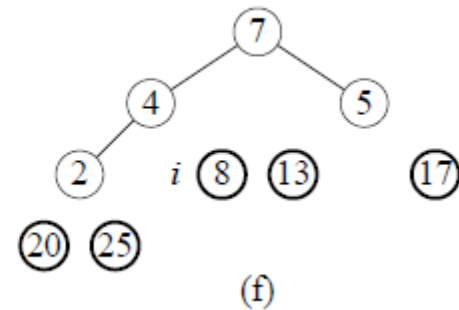
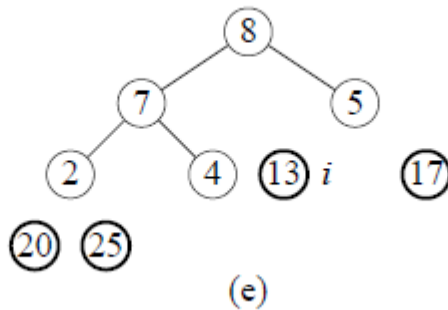
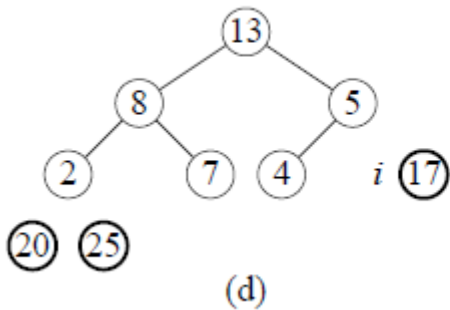
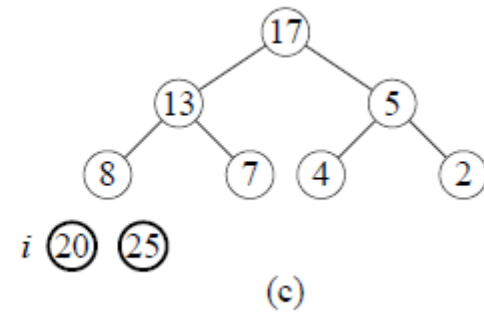
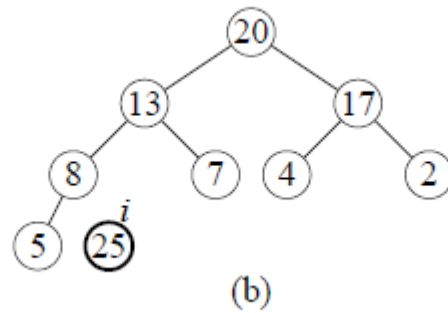
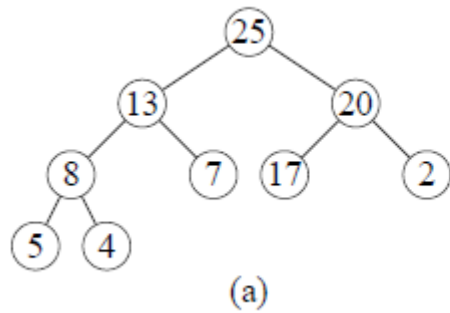
- Sort an array  $A$  using heaps.

# Operation with Heaps

## HeapSort

1. Build Max Heap from unordered array;
2. Find maximum element  $A[1]$ ;
3. Swap elements  $A[n]$  and  $A[1]$ :  
now max element is at the end of the array!
4. Discard node  $n$  from heap  
(by decrementing heap-size variable)
5. New root may violate max heap property, but its children are max heaps. Run `max_heapify` to fix this.
6. Go to step 2.

# Illustrate the operation of Heapsort on the array $A[5,13,2,25,7,17,20,8,4]$



# Heap implementation of priority queue

- Heaps efficiently implement priority queues.
- Max-priority queues implemented with max-heaps. Min-priority queues are implemented with min-heaps similarly.

# Priority queue

- is a data structure for maintaining a dynamic set  $S$  of elements, each with an associated value called a *key*.

# MAX-Priority Queue Operations

Max-priority queue supports 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.

# Finding the Max element

- Getting the maximum element is easy: it's the ROOT

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

# Extracting Max Element

Given the array A:

- Make sure heap is not empty.
- Make a copy of the maximum element (the root).
- Make the last node in the tree the new root.
- Re-heapify the heap, with one fewer node.
- Return the copy of the maximum element.



# Extracting Max Element...

HEAP-EXTRACT-MAX( $A, n$ )

**if**  $n < 1$

**error** “heap underflow”

$max = A[1]$

$A[1] = A[n]$

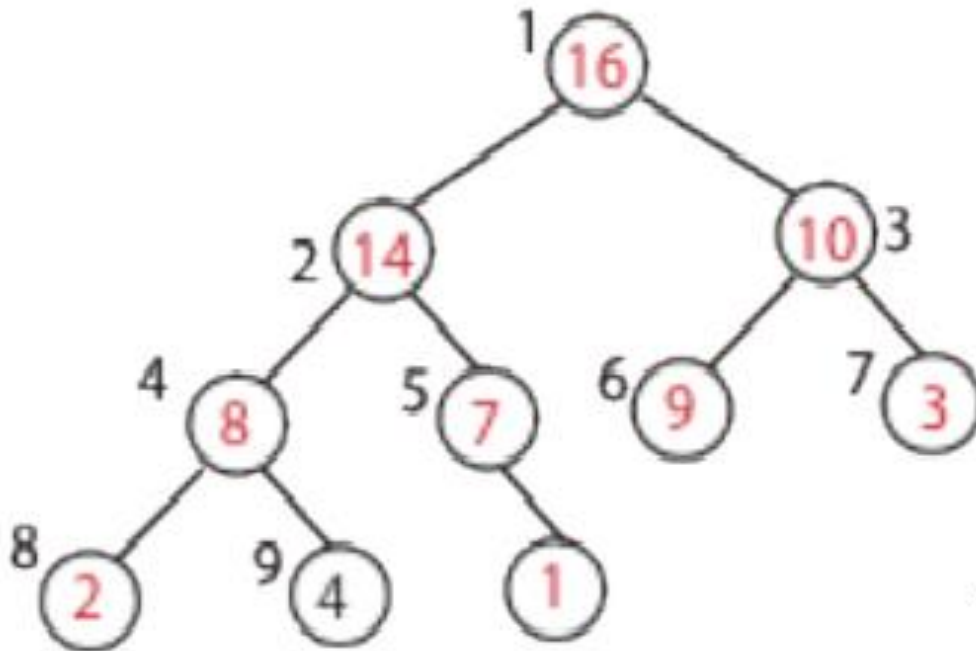
$n = n - 1$

    MAX-HEAPIFY( $A, 1, n$ )      // remakes heap

**return**  $max$

# EXAMPLE

Run HEAP-EXTRACT-MAX on the following heap



# Increasing Key value

Given set  $S$ , element  $x$ , and new key value  $k$ :

- Make sure  $k \geq x$ 's current key.
- Update  $x$ 's key value to  $k$ .
- Traverse the tree upward comparing  $x$  to its parent and swapping keys if necessary , until  $x$ 's key is smaller than its parent's key.

# Increasing Key value...

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

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

**error** “new key is smaller than current key”

$A[i] = key$

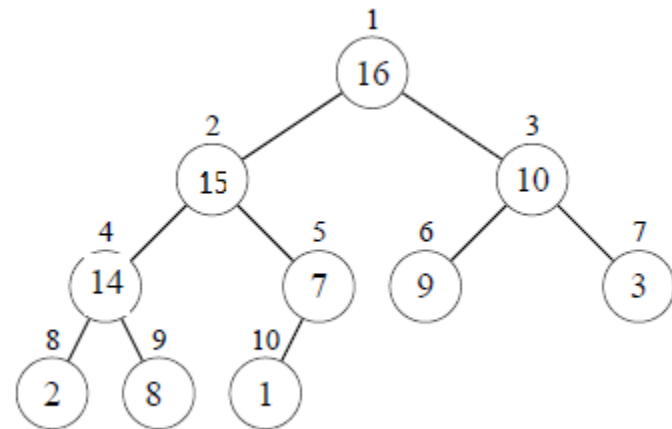
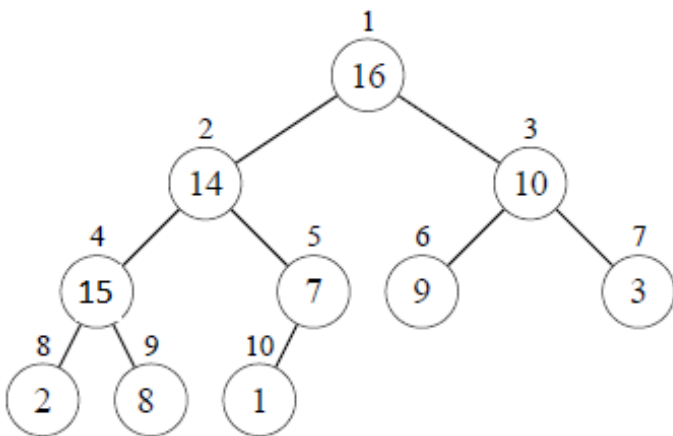
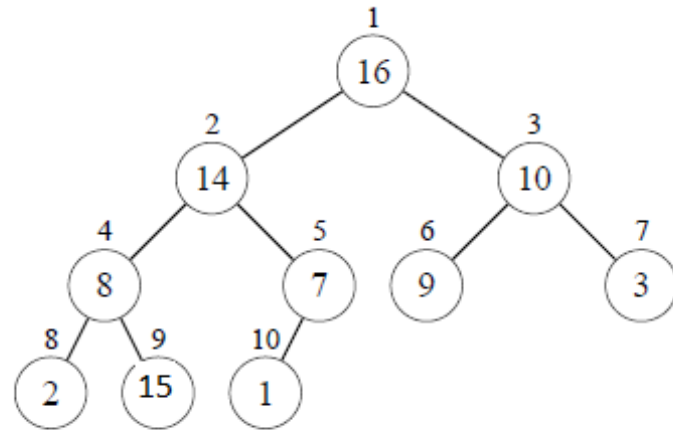
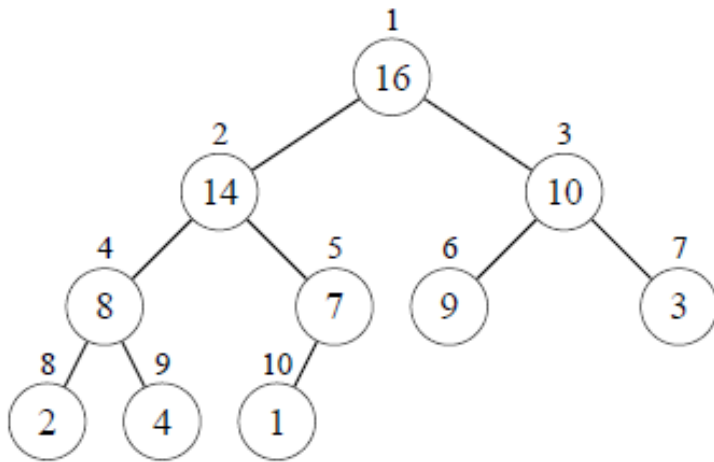
**while**  $i > 1$  and  $A[\text{PARENT}(i)] < A[i]$

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

$i = \text{PARENT}(i)$

# EXAMPLE

Increase key of node 9 in the following heap to have a value of 15.



# Inserting into the heap

Given a key  $k$  to insert into the heap:

- Increment the heap size.
- Insert a new node in the last position in the heap, with key  $-\infty$ .
- Increase the  $-\infty$  key to  $k$  using the HEAP-INCREASE-KEY procedure defined above.

# Inserting into the heap...

MAX-HEAP-INSERT( $A, key, n$ )

$$n = n + 1$$

$$A[n] = -\infty$$

HEAP-INCREASE-KEY( $A, n, key$ )

# Example

