

Priority Queues

Instructor: Meng-Fen Chiang

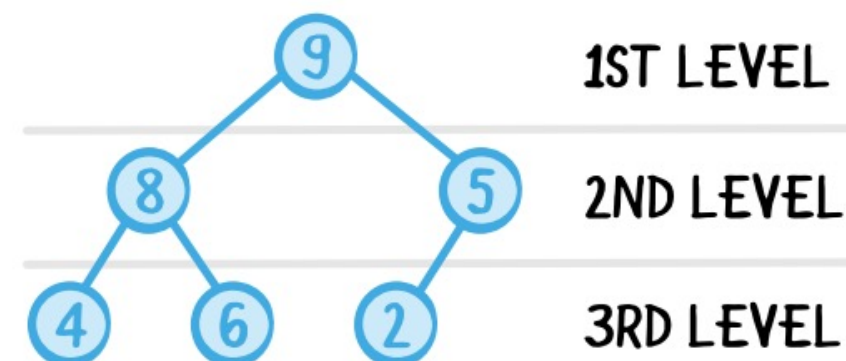
COMPCSI220: WEEK 9



Slides adapted from Mark Wilson, Georgy Gimel'farb, Simone Linz and Tanya Gvozdeva

OUTLINE

- Priority Queue
- Heaps
 - Illustrating examples
 - Basic Operations
 - Implementation
 - Complexity Analysis



Definition of Priority Queue

- A priority queue is a container Abstract Data Type (ADT), where each element has a **key** (from a totally ordered set, as with sorting) called its priority.
 - E.g., Queuing in a bank, VIPs have higher priority
 - E.g., A TODO list highlighting the importance of the items
- Applications: Sorting, Graph algorithms
- Three key operations: insert an element, and to find and delete the element of highest priority
- Implementations: unsorted array, sorted array and binary heap.

Sorted and Unsorted Arrays

- The three key operations: **Insert**, **FindMax** and **DeleteMax**



	FindMax	DeleteMax	Insert
Unsorted Array	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$

Unsorted array:

- Find maximum element:** needs $\Theta(n)$ to scan the array
- Delete maximum element:** needs $\Theta(n)$ to find the max element, and $\Theta(n)$ to move all elements on its right hand side.
- Insert an element:** Insert at the end in $\Theta(1)$

Sorted and Unsorted Arrays (Contd.)

- The three key operations: **Insert**, **FindMax** and **DeleteMax**



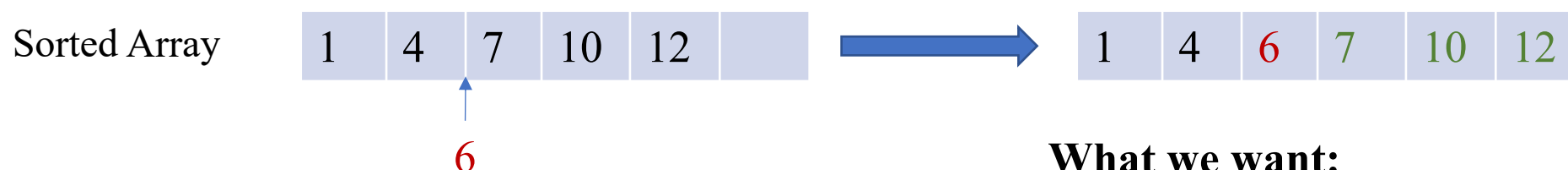
	FindMax	DeleteMax	Insert
Unsorted Array	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
Sorted Array	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$

Sorted array:

- Find maximum element:** needs $\Theta(1)$ to retrieve the last element
- Delete maximum element:** needs $\Theta(1)$ to remove the last element
- Insert an element:** Find the location to insert in $\Theta(n)$, move at most n elements in $\Theta(n)$

Sorted and Unsorted Arrays (Contd.)

- The three key operations: **Insert**, **FindMax** and **DeleteMax**



What we want:

A data structure that can support dynamically organizing the items efficiently:

1. **Inserting** new items
2. **Finding** the most important one
3. **Deleting** the most important one and reorganizing the structure

	FindMax	DeleteMax	Insert
Unsorted Array	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$
Sorted Array	$\Theta(1)$	$\Theta(1)$	$\Theta(n)$
Heap (Binary)	$\Theta(1)$	$\Theta(\log n)$	$\Theta(\log n)$

Property of Heaps

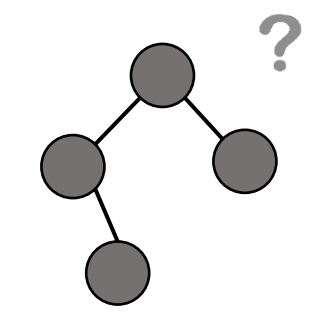
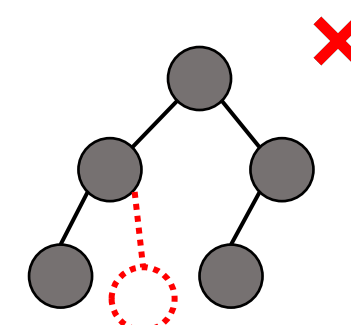
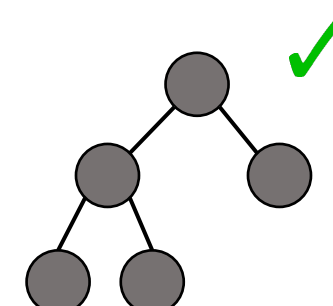
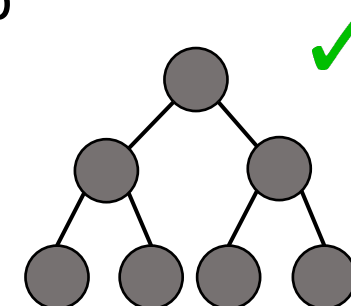
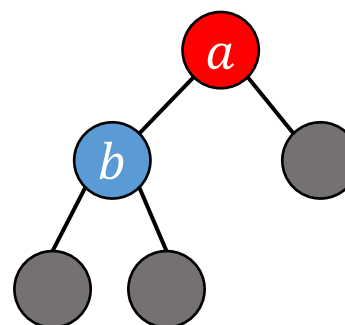
- A binary heap: A complete binary tree that satisfies heap ordering property.

- Complete Binary Tree

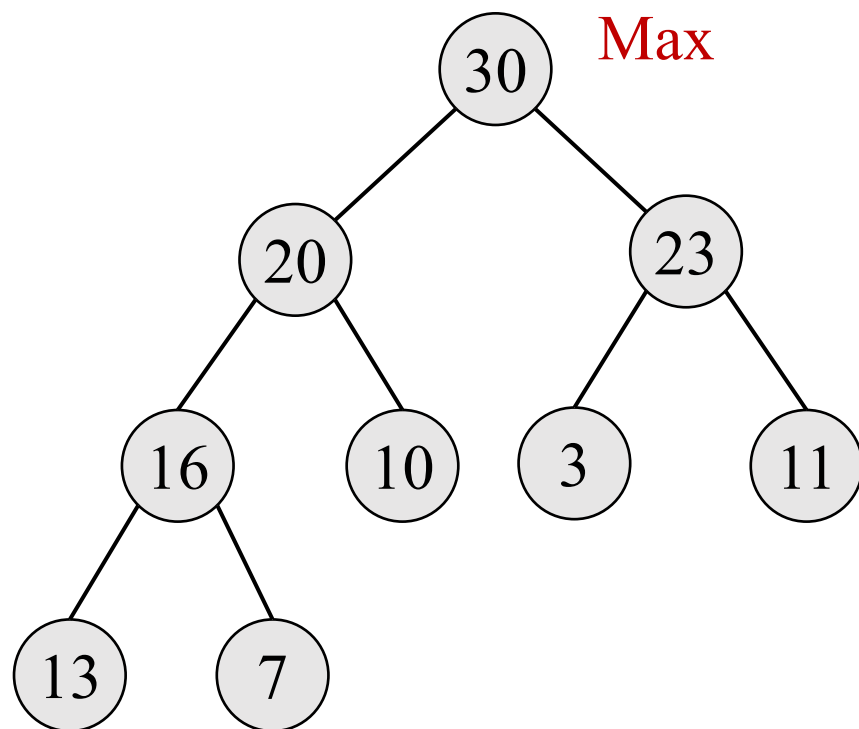
- All levels except the last level are full
- Nodes in last level are placed left to right

- Heap ordering property: Suppose a node a has child node b

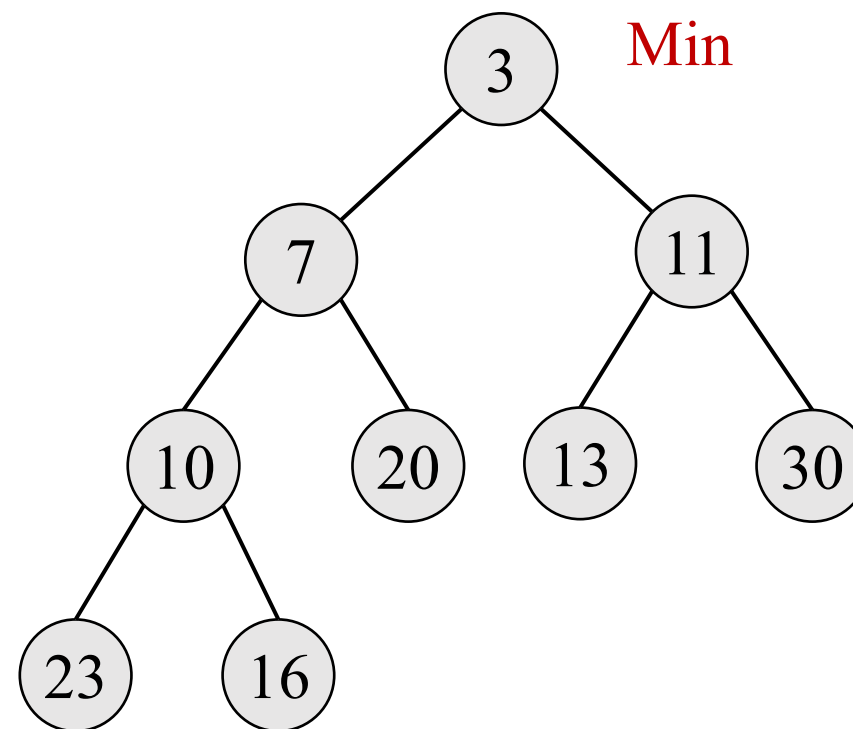
- max-heap property: $val(a) \geq val(b)$
- min-heap property: $val(a) \leq val(b)$



Example: [23, 13, 11, 20, 10, 3, 30, 16, 7] → Heaps



Max Heap



Min Heap

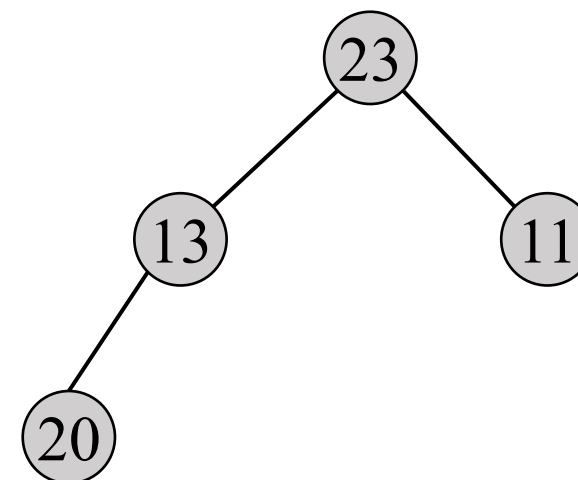
Heap Operations

- FindMax() / FindMin()
- Insert()
- DeleteMax() / DeleteMin()

Insertion

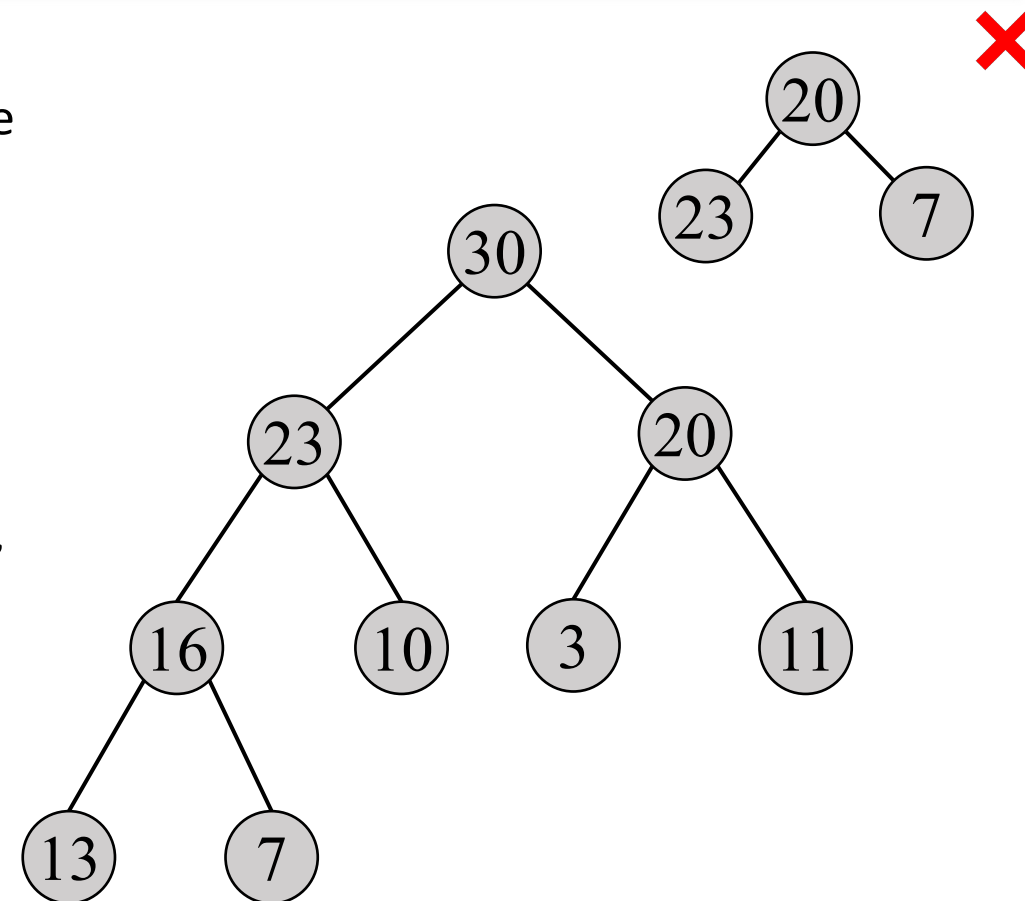
- Step 1 – Insert a new node N at the end of the tree.
- Step 2 – Compare the value of the node n with its parent.
- Step 3 – If the parent is smaller than node N , swap them.
- Step 4 – Repeat step 2 & 3 until heap ordering property holds.

Input → 23, 13, 11, 20, 10, 3, 30, 16, 7



Deletion

- Step 1 – Delete the root node and move the last node N to the root.
- Step 2 – Compare the value of node N with its children.
- Step 3 – If node N has smaller value than its children, swap N with the larger child.
- Step 4 – Repeat step 2 & 3 until heap ordering property holds.

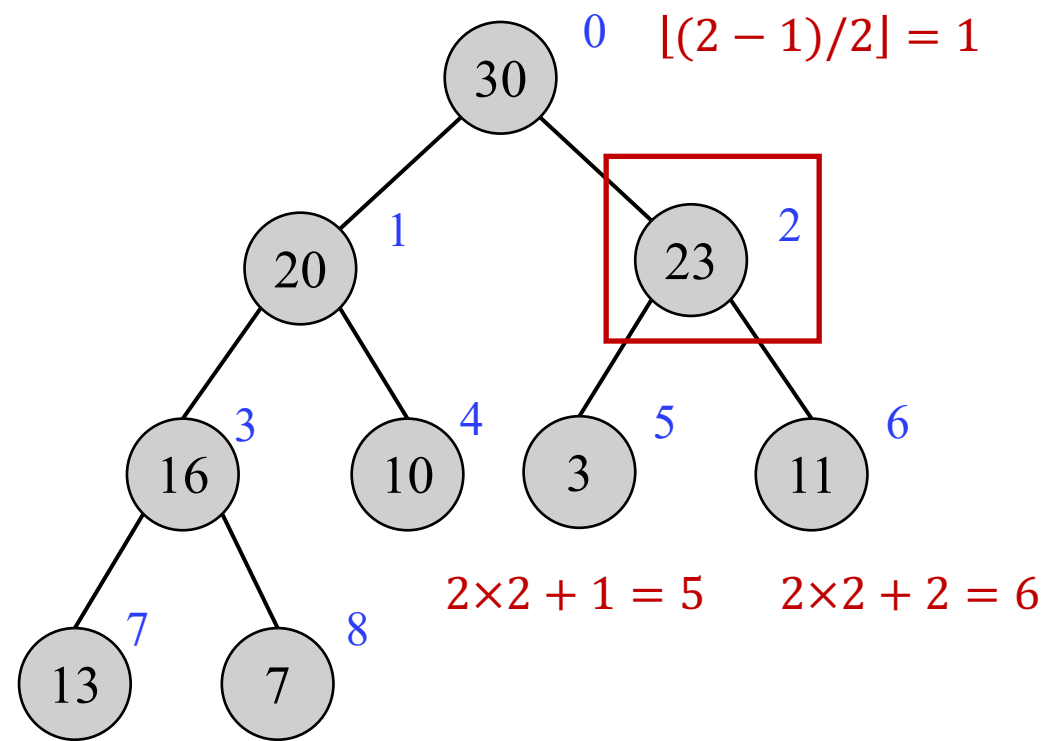


Array Implementation

- Array Implementation for heap
 - Compact representation
 - Easy to swap
- Find parent and children quickly?

0	1	2	3	4	5	6	7	8
30	20	23	16	10	2	11	13	7

- For the k -th element in the array
 - Left child $\rightarrow 2k+1$
 - Right child $\rightarrow 2k+2$
 - Parent $\rightarrow \lfloor (k-1)/2 \rfloor$



Implementation - Insertion

- Heap operation on arrays?

Algorithm 1 Insert an element to a heap

1: **function** Insert(array $a[0..n-1]$, key x)

2: $a \leftarrow \text{append}(a[0..n-1], x)$

3: $k \leftarrow n$

4: **while** $k > 0$ **do**

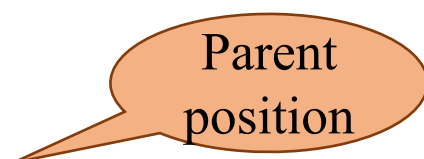
5: **if** $a[k] > a[\lfloor k/2 \rfloor]$ **then**

6: $\text{swap}(a, k, \lfloor k/2 \rfloor)$

7: $k \leftarrow \lfloor k/2 \rfloor$

7: **else**

8: **return**



Parent
position

For the k -th element in the array

- Left child $\rightarrow 2k$
- Right child $\rightarrow 2k + 1$
- Parent $\rightarrow \lfloor k/2 \rfloor$

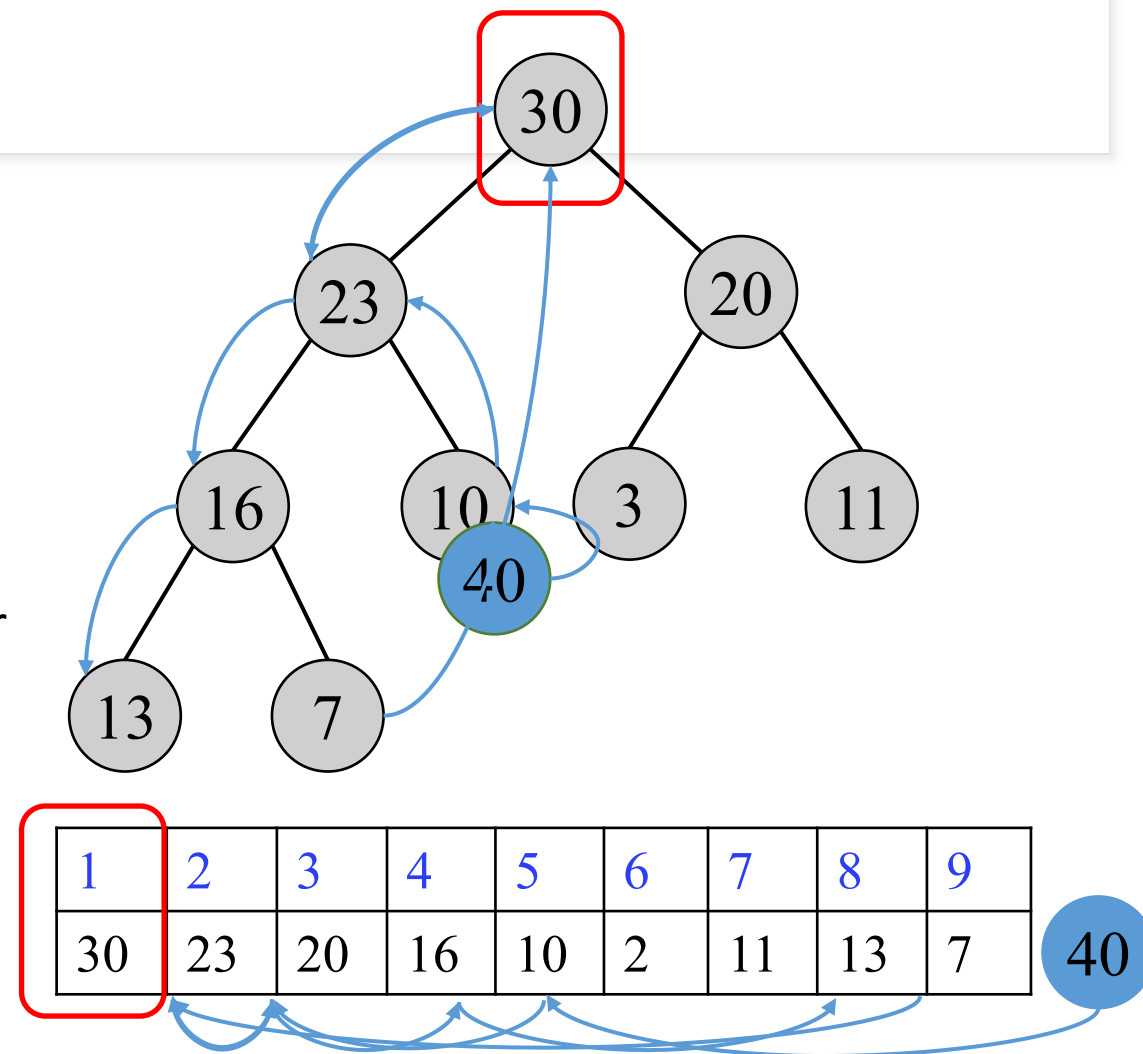
Complexity Analysis

- Worst-case analysis
 - FindMax() / FindMin() $\Theta(1)$
 - Insert() h
 - DeleteMax() / DeleteMin() $2h$
- Let h be the height of the tree. The number of nodes at level $l < h$ is 2^l , then the total number of nodes up to level l is

$$2^0 + 2^1 + \dots + 2^l = 2^{l+1} - 1$$
- The number of nodes up to level $h-1$ is then $2^h - 1$, therefore, the number of nodes in a heap of height h always satisfies

$$2^h - 1 + 1 \leq n \leq 2^{h+1} - 1$$

$$h = \lfloor \log_2 n \rfloor$$



SUMMARY

- Priority Queue
- Heaps
 - Illustrating examples
 - Basic Operations
 - Implementation
 - Complexity Analysis

