

Data Structures

Chapter 6: Priority Queue

Instructor Maher Hadiji
hdiji.maher@gmail.com

2015-2016

Queues and Priority Queues

- A queue is a first-in/first-out data structure.
- Elements are appended to the end of the queue and are removed from the beginning of the queue.
- In a priority queue, elements are assigned priorities.
- When accessing elements, the element with the highest priority is removed first.



Priority Queues

- A priority queue is a data structure for temporary storage that delivers the stored items in order of their priority: an item with higher priority is delivered first.
- The objects in a priority queue are Comparable (or a comparator is provided).
- According to a convention, the smaller item has higher priority.



Priority Queues

- Works like an ordinary queue
- However, the order of how things are removed depend on a priority key
- Ascending Priority Queue – The item with the smallest key has the highest priority
- Descending Priority Queue – The item with the largest key has the highest priority

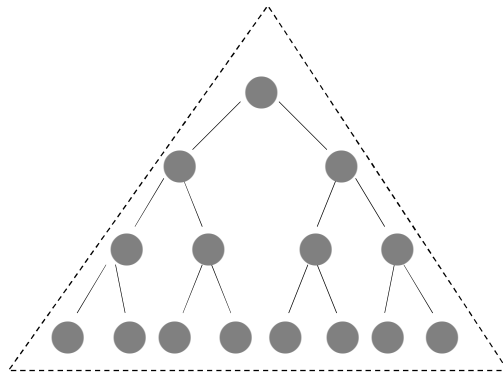


Heaps

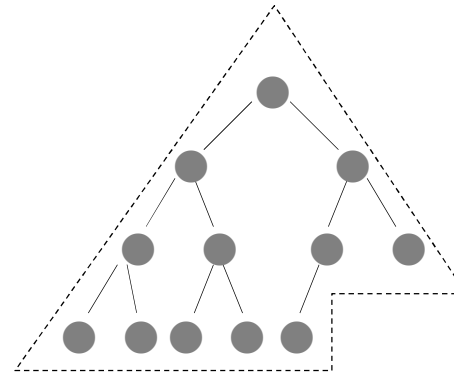
- A *heap* is a particular kind of a binary tree.
- Heaps provide a way to implement priority queues in such a way that both **add** and **remove** take $O(\log n)$ time.
- A heap can be stored in an array (or in an **ArrayList**).

Full and Complete Binary Trees

Full tree:
all levels are filled; a full tree with h levels holds $2^h - 1$ nodes.

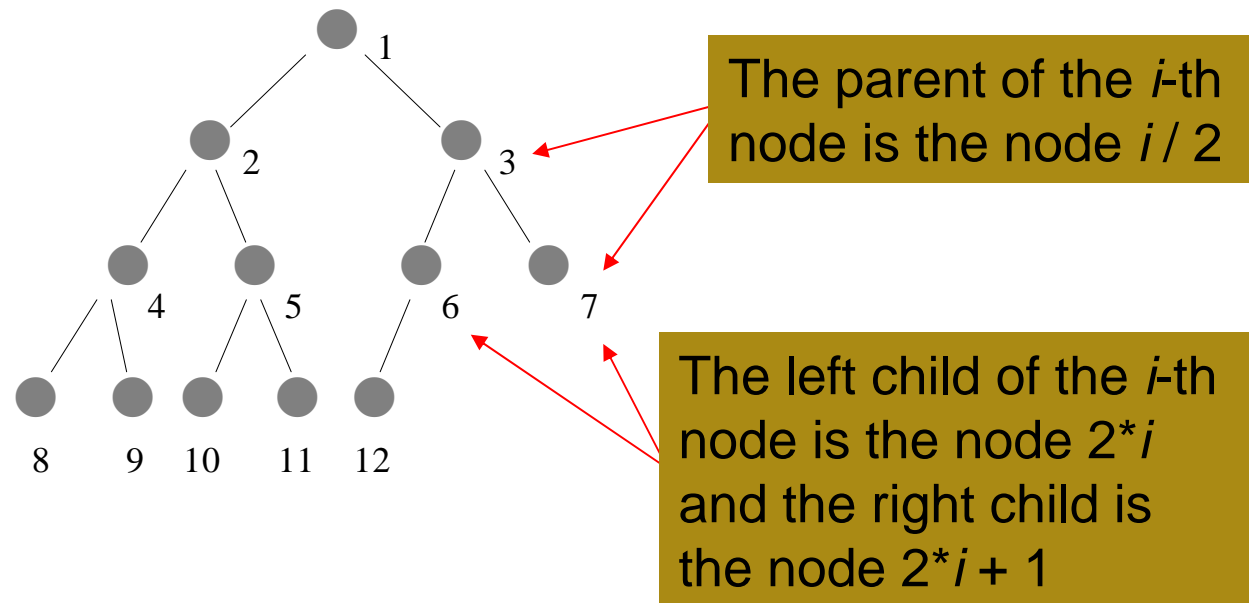


Complete tree:
all levels are filled, except perhaps the bottom one



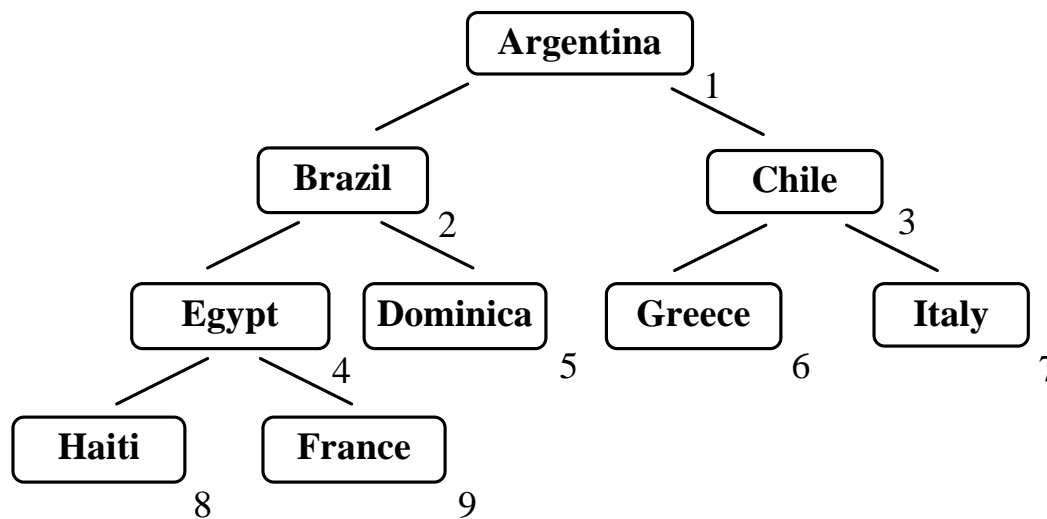
Complete Trees

- Nodes can be numbered in level-by-level order:




Complete Trees (cont'd)

- It is convenient to store a complete binary tree in an array in the order of nodes, starting at index 1:

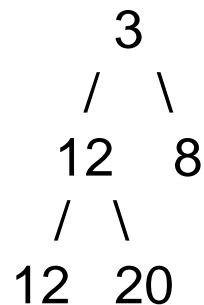


items[0]:	<null>
items[1]:	Argentina
items[2]:	Brazil
items[3]:	Chile
items[4]:	Egypt
items[5]:	Dominica
items[6]:	Greece
items[7]:	Italy
items[8]:	Haiti
items[9]:	France

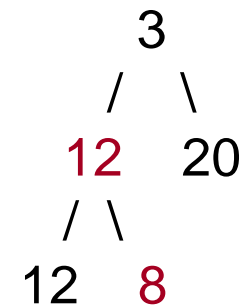
Heaps (cont'd)

- A (min) heap is a *complete* binary tree 
- The value in each node does not exceed any of the values in that node's left and right subtrees.
- In a heap, the root holds the smallest value.

A heap:



Not a heap:



Heaps (cont'd)

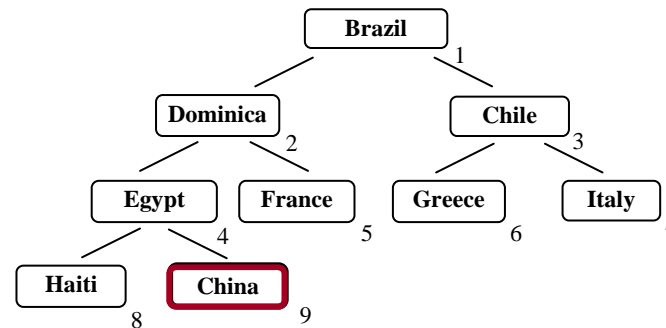
- Either adding or removing an item takes $O(\log n)$ time.
- The algorithm for **add** uses the *reheap-up* procedure.
- The algorithm for **remove** uses the *reheap-down* procedure.

Add a leaf. Starting at the last leaf, swap the node with its parent as many times as needed to repair the heap.

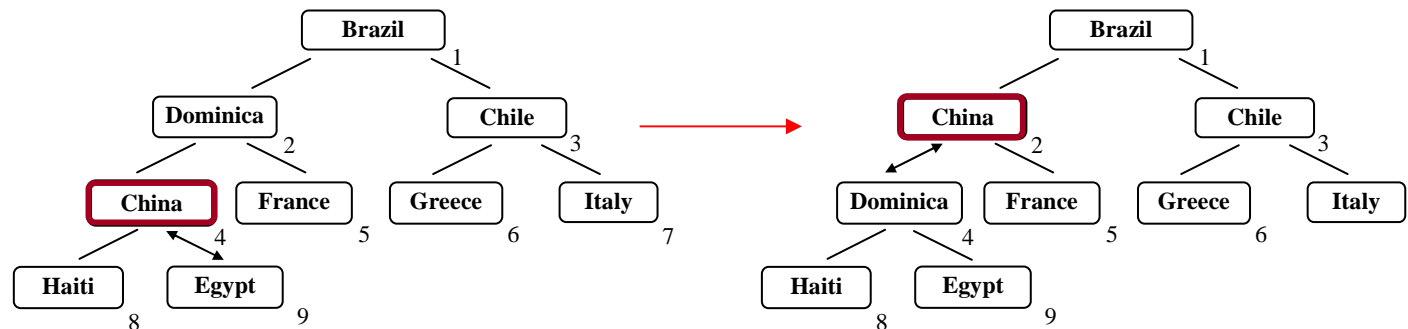
Remove the root and place the last leaf at the root. Starting at the root, swap the node with its smaller child, as many times as needed to repair the heap.

The Algorithm for **add**

Step 1: the new value is added as the rightmost leaf at the bottom level, keeping the tree complete

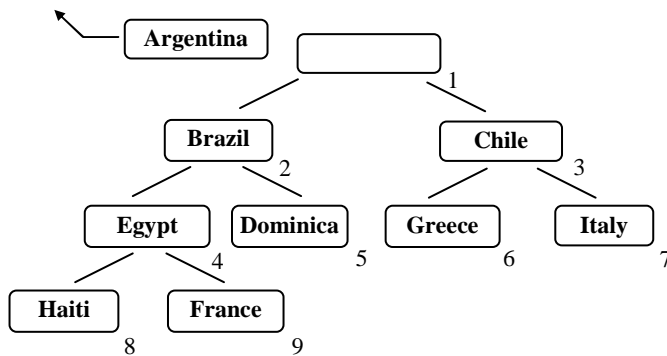


Step 2: “reheap up”: the new value keeps swapping places with its parent until it falls into place

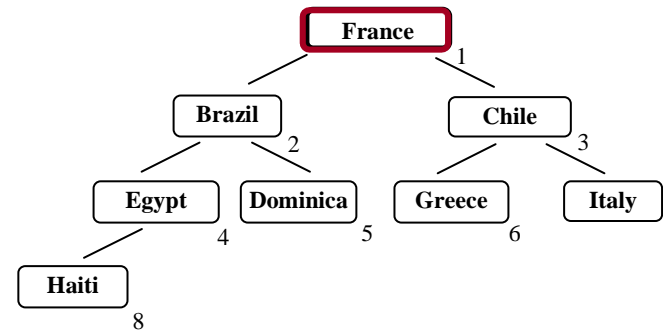


The Algorithm for **remove**

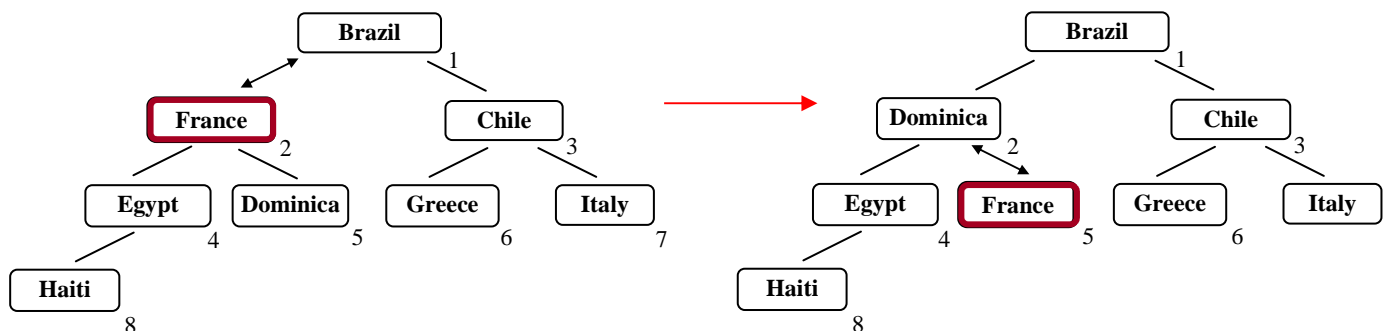
Step 1: the root is removed



Step 2: the rightmost leaf from the bottom level replaces the root



Step 3: “reheap down”: the new root value keeps swapping places with its smaller child until it falls into place



java.util.PriorityQueue<E>

- Implements java.util.Queue<E> with methods:

```
boolean isEmpty ();  
void add (E obj);  
E remove ();  
E peek ();
```

- The implementation is a *heap*.
- **add** and **remove** are $O(\log n)$; **peek** is $O(1)$.