Priority Queues
Heaps and Heapsort

Reading Assignment
Chapter 7

Priority Queues

- A priority queue stores a collection of prioritized elements
- Applications
  - 911 event queues
  - Airport landing patterns
  - Triage in a hospital
- Operations
  - **insert(), deleteMin()**or **deleteMax()** but not both
  - functions like **member(), search()** or **find()** are not supported - why?
- Implementation strategies
  - Linear lists or sequences
  - Heaps

Priority Queue Interface

```
public interface PriorityQueue {
   void insert(Comparable x);
   Comparable deleteMin(); // or deleteMax() instead
   Comparable getMin(); // gets min, does not delete it
   int size();
   boolean isEmpty();
}
```

- We use an interface to insert any comparable objects into our PQ.

Priority Queue Sort

- The priority queue operations allow for a simple sorting algorithm by definition

```
void pqSort(Integer a[]) {
    SomePQ pq = new SomePQ();
    for (int k=0; k<a.length; k++) { // first loop
       pq.insert(a[k]);
    }
    k = 0;
    while ( !pq.empty() ){ // second loop
       a[k] = pq.deleteMin();
       k++;
    }
}
```

Time Complexity of list PQ Operations

- Can we implement the Priority Queue operations efficiently in a list?
- Running time analysis of **pqSort()** assuming n input values
- **First loop**

$$T_{fl}(n) = n * T(insert)$$

- **Second loop**

$$T_{sl}(n) = n * T(deleteMin)$$

- **Total**

$$T_{pq}(n) = T_{fl}(n) + T_{sl}(n) = n * T(insert) + n * T(deleteMin)$$

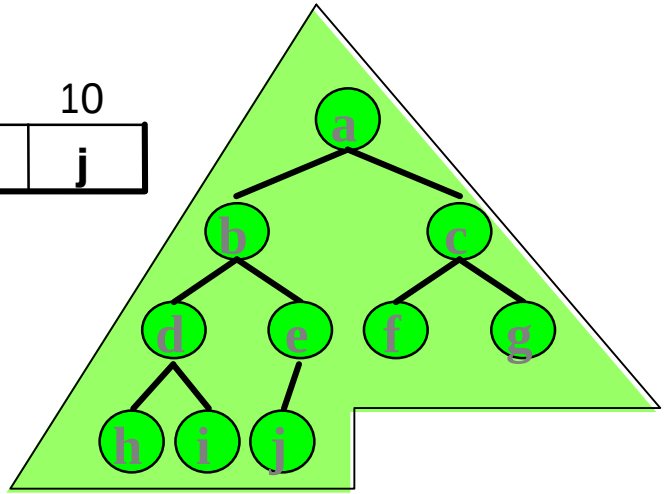
$$T_{pq}(n) = n * \{T(insert) + T(deleteMin)\}$$

- **Linked list implementation**
  - Linked list is sorted at insert time
  - T(insert) = ? O(n)
  - T(delete) = ? O(1)
  - $T_{pq}(n)$? $O(n^2)$ + O(n) ? $O(n^2)$ ✍ ✍

Heap Encoding

- Array representation
- Assume **complete binary tree**
  - All levels are full except possibly the last level
  - No holes
  - *called the Heap shape property*
- Heap encoding
  - Process the binary tree in level order and enter the elements in an array starting with array index 1 (zero is not used)

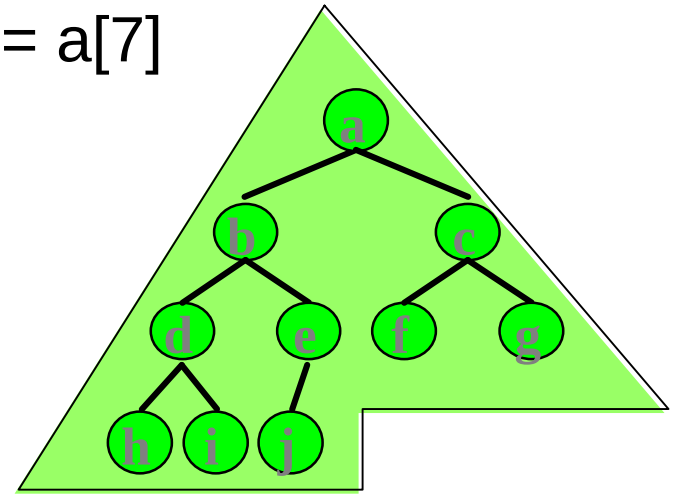
0	1	2	3	4	5	6	7	8	9	10
	a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**



  - Parent of a[k] is at a[k/2]
  - Left child of a[k] is at a[2k]
  - Right child of a[k] is at a[2k+1]

Heap Encoding: Example

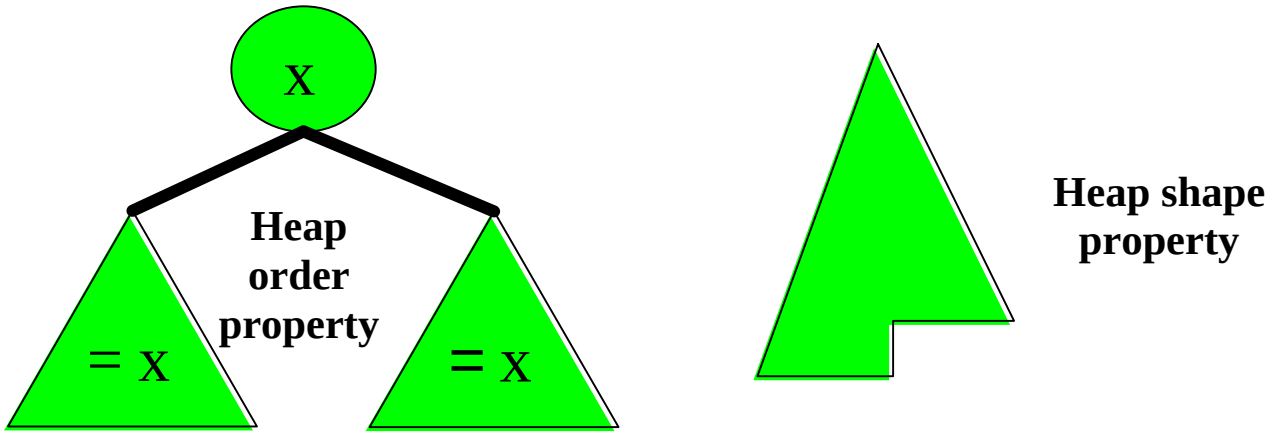
- Parent of a[5] is at a[5/2] = a[2]
  - Parent of “e” is “b”
- Left child of a[3] is at a[2*3] = a[6]
  - Left child of “c” is “f”
- Right child of a[3] is at a[2*3+1] = a[7]
  - Right child of “c” is “g”



0	1	2	3	4	5	6	7	8	9	10
	a	b	c	d	e	f	g	h	i	j

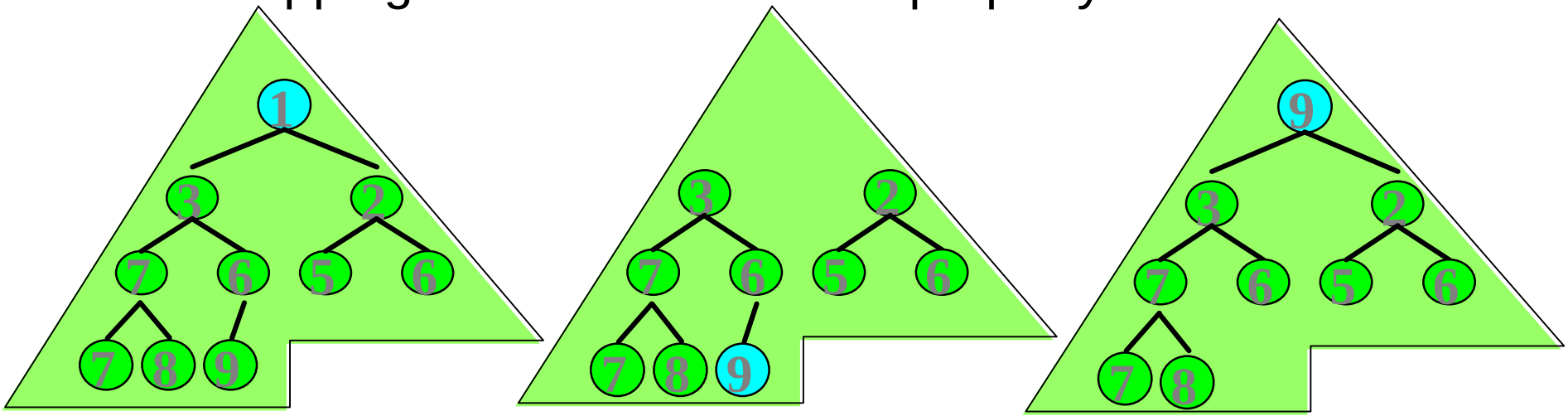
Heap Properties

- Shape property
  - All levels in a heap are complete except possibly the last level.
- Order property
  - A heap is a binary tree in which the nodes are labelled with elements of a set such that all elements in the left and right subtrees of a node labelled x are greater than or equal to x.
- A Heap is a partially ordered tree



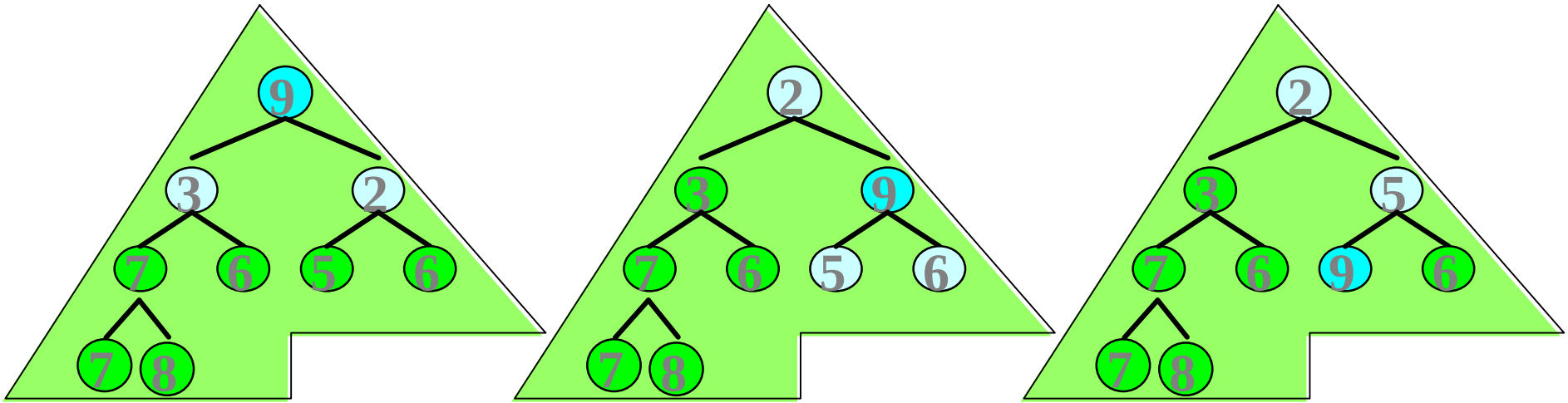
DeleteMin operation

- The smallest element is the root node
- Remove and return root node which is constant time O(1)
- Re-establish shape property
  - Move last element in the tree to the root
  - Except for the root node, order is fine too
- Re-establish order property
  - Push the root element, which is out of order, down by swapping elements until order property is established



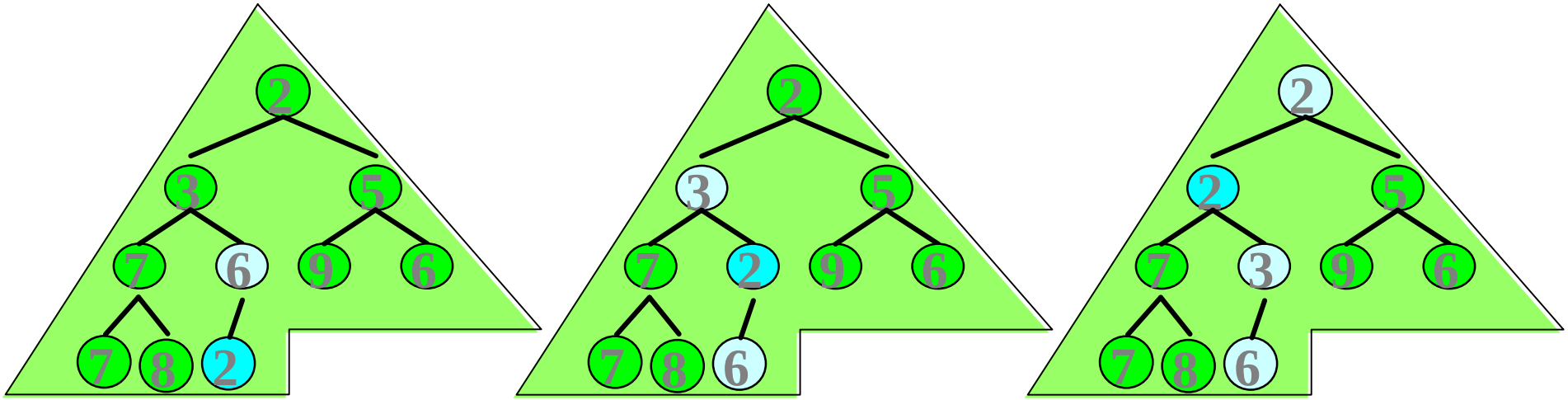
DeleteMin() - percolateDown()

- ***Push element 9 down until Heap order property is re-established***
- ***Keep on swapping with the smallest child***
- ***At most log n swap operations (i.e., # of levels)***
- ***Thus, the time complexity of percolateDown() is O(log n)***
- ***The time complexity of deleteMin() is also of O(log n)***



Insert ? percolateUp()

- Insert the element at the first open array position
- Shape property is trivially established
- Push the element up the tree until the order property is re-established by swapping with the parent
- At most log n swap operations (i.e., # of levels)
- Thus, the time complexity of **insert()** is also O(log n)



Code walkthrough

•

Heapsort

```
void heapSort(int a[]) {
    IntHeap heap = new IntHeap();
    for (int k=0; k<a.length; k++)
        heap.insert(a[k]);
    k = 0;
    while (!heap.empty()) {
        a[k] = heap.deleteMin();
        k++;
    }
}
```

HeapSort – Analysis

- ***insert() and deleteMin() each take O(log n) time***
- ***The running time of Heapsort is therefore***

$$T_{hs}(n) = n \log n + n \log n = 2 n \log n$$

- ***Hence the time complexity of Heapsort is of O(n log n)***
- ***Fundamental result of Computer Science***
  - ***Sorting takes O(n log n) time***

Summary

- PriorityQueue
  - **insert(), deleteMin() (or deleteMax())**
  - Applications
  - Implementation strategies: list or heap
- PriorityQueue Sort
  - Using linear list data structure $O(n^2)$
- Heap
  - Encoding of a binary tree in an array
  - Shape and order property
- **deleteMin()**
  - Remove min (root); bubble down by swapping
- **insert()**
  - Insert at the end of array; bubble up by swapping
- Heapsort
  - Using heap data structure O(n log n)