Introduction to Heaps

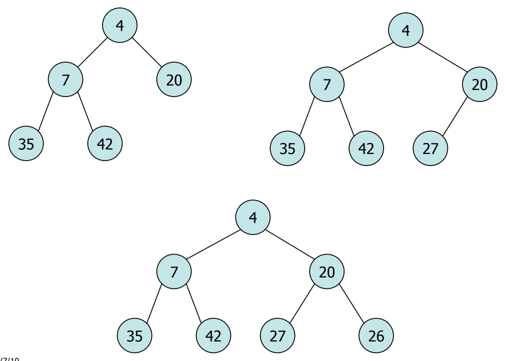
7/7/10 1

Definition of heap properties

- Previously, we saw one special kind of binary tree: a BST
- A heap is another special type of tree, with two requirements
  1. The data stored in any node is less than or equal to the value of all of that node's descendents
     - There are no restrictions on ordering of values between left and right sub-trees; just between the parent and children
     - This means that the value at the root is always the smallest value in the tree/heap
     - This is referred to as the **order property** of heaps
  2. A heap must be a *complete* binary tree.
     - This means that heaps are frequently implemented using arrays.
     - This is referred to as **structure property** of heaps

7/7/10 2

Examples of heaps



7/7/10 3

Important heap operations

- There are two critical operations that can be performed on heaps
  - Inserting a new value into the heap
  - Removing the smallest value stored in the heap
- These operations must be sure to maintain both the order and structure properties of the heap
- Everything else is pretty simple (e.g., "isEmpty", etc.)

Sample Heap interface

```
public interface Heap<T implements Comparable> {
   /**
    * Adds the specified value to the heap.
    * @param newValue  the (non-null) value being added
    */
   public void insertHeapNode( T newVal );

   /**
    * Removes and returns the smallest value in the heap.
    */
   public T getSmallest();
}
```

Heap insertions

- Adding data to the heap takes place in three basic steps:
  1. Check to make sure that there's enough room in the array for another element.
     - If there isn't, the array is resized to fit.
  2. Put the new value in the next (unused) spot in the array.
     - This maintains the **structure property**, since the heap will be complete.
     - *Question #1: what is the expected complexity of this operation?*
  3. While the new value is less than the value stored in its parent node, swap the value with its parent
     - This makes the new value "climb up" the tree (while its original ancestors are pushed down a level) until it finds its correct location.
     - This maintains the **order property**.
     - *Question #2: what is the expected complexity of this operation?*
- Final question: what is the **total** expected complexity for this operation?

Department of Computer Science

Heap removals

- Removing data also takes three steps:
  1. Save the value currently in the root of the tree.
  2. Move the last value into the root (and decrement the heap size by 1).
     – This preserves the **structure property**.
     – *Question #1: what is the expected complexity of this operation?*
  3. While the relocated value has one or more children that are less than its value, swap it with its smallest child.
     – This preserves the **order property**.
     – *Question #2: what is the expected complexity of this operation?*
- Final question: what is the **total** expected complexity for this operation?

Department of Computer Science

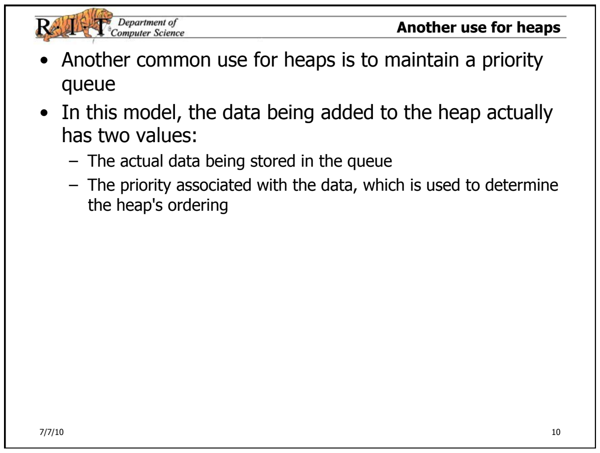
Heapsort

- Performing a heap sort is simple:
  – First, build a heap with your (presumably unordered) data
  – Then, keep removing the smallest value from the heap until it is empty.
- Complexity calculations:
  – Building the heap:
    - We're doing an O(log n) operation n times
    - Total complexity is therefore O(n log n)
  – Emptying the heap:
    - We're doing an O(log n) operation n times
    - Total complexity is therefore O(n log n)
  – Total complexity: O(2 * n log n), or O(n log n)

Department of Computer Science

How does heap sort compare?

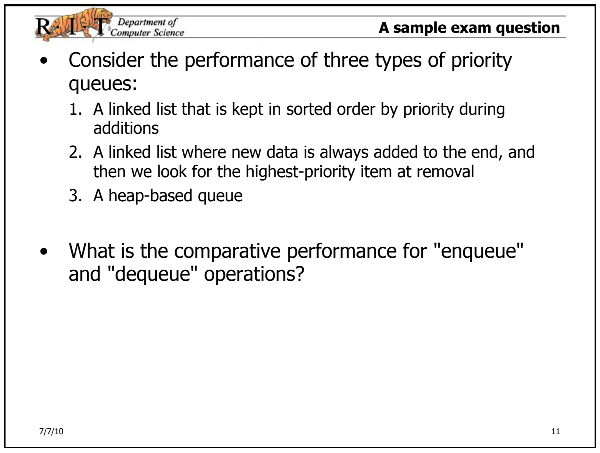
- Heap sort falls into the same performance range as merge and quick sort
  – In general, quick sort has a lower constant associated with it than heap sort
    - this is due in part to the need to build up the array as we add data
  – However, quick sort carries the risk of O(n^2) performance in worst-case scenarios
  – Heap sort is actually Θ(n log n), just like merge sort; as a result, it's a very nice sort to have on hand


Department of Computer Science

Another use for heaps

- Another common use for heaps is to maintain a priority queue
- In this model, the data being added to the heap actually has two values:
  - The actual data being stored in the queue
  - The priority associated with the data, which is used to determine the heap's ordering

7/7/10 10


Department of Computer Science

A sample exam question

- Consider the performance of three types of priority queues:
  1. A linked list that is kept in sorted order by priority during additions
  2. A linked list where new data is always added to the end, and then we look for the highest-priority item at removal
  3. A heap-based queue

- What is the comparative performance for "enqueue" and "dequeue" operations?

7/7/10 11