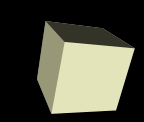
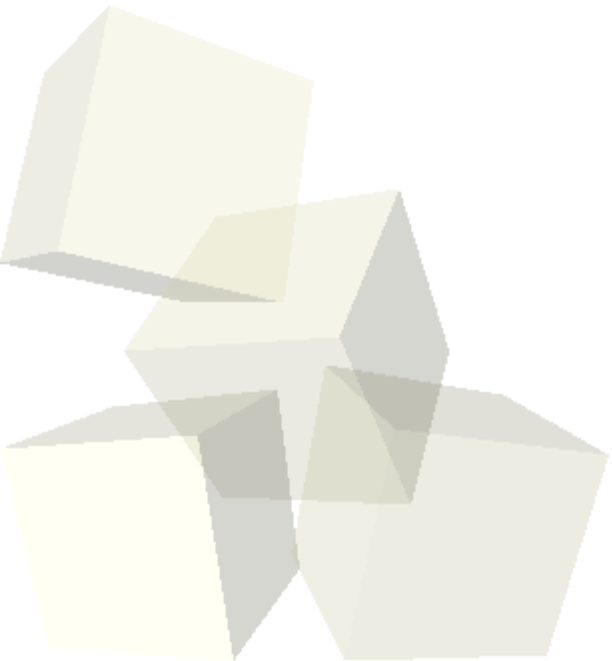


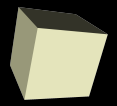
Heaps (Binary Heaps)

1-31-2006



Opening Discussion

- Do you have any questions about the reading?
- Do you have any questions about the test?
- Do you have any questions about the assignment?



Heaps

- We are now switching our attention over to the data structure of a heap. Heaps are efficient data structures for repeatedly finding the min or max of a set of elements.
- All heap implementations we will discuss have at least O(log n) performance for adding, peeking, and removing. They also have similar performance for adjusting an element or removing and element if you know where the element is.

Binary Heaps

- The heap that you should all be familiar with it the standard binary heap. This type of heap can be viewed as a complete binary tree with proper heap ordering.
- Unlike a BST, heap ordering saying nothing about relative values of children, all it says is that parents have a higher priority than children so the root is always the element that will be taken off next.
- For simplicity, we represent these in arrays instead of as linked trees. Being complete means there are no holes and if the root is at index 1, children are always at 2*n and 2*n+1.

Adding to a Binary Heap

- When you add to a binary heap you put the new element at the end of the array and let it “bubble up”. It keeps moving up until it either gets to the root or it hits a parent with a higher priority.
- CLR and most other books do swaps for this process. You can get better performance by keeping the new element in a temporary and only putting it in the array once you have found the place it belongs. This matters less in Java than it might in C++ because you are only copying a reference in Java.

Removing from a Binary Heap

- To take off the highest priority element of a heap we simply pull off the root, then take the last element off the end and sink it down from the root to where ever it stops. Each time the higher priority child moves up assuming that child is higher priority than the last element.

Binomial Trees

- Next class we will talk about binomial heaps. These are a neat alternative to the binary heap in which we can merge to heaps in O(log n) time.
- Binomial heaps are made from binomial trees so lets discuss those for a bit today.
- Binomial trees are NOT binary. A level 0 binomial tree has only a single element in it. Recursively we define a level n binomial tree as two level n-1 trees where the root of one is added as the first child of the other.
- This leads to trees with a size equal to 2^{level}.

Reminders

- The second test is due on Thursday.
- The second assignment is due a week from today. I'll have a larger input and sample outputs up soon. At that time I'll also tell you the timing results for my trees with each of the heap types.