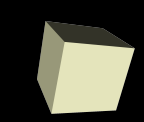
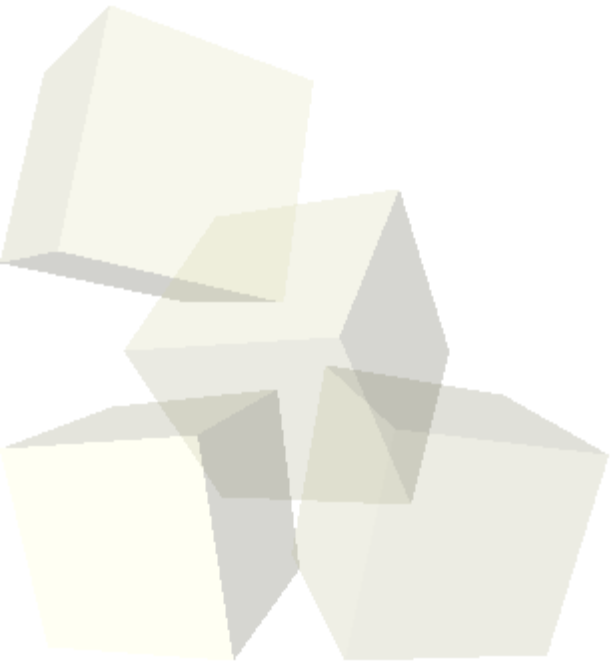


Searching Solution Space/D&C

1-24-2006



Opening Discussion

- Do you have any questions about the assignment?
- Do you have any questions about the reading? (Ch. 2 from the other book.)
- What do you know about searching through solution space for an answer?
- What do you know about divide and conquer algorithms?
- In recent reading I found that HashMap is preferred over Hashtable in newer versions of Java.
- Let's make sure everyone can get into the grading application.

Solution Spaces

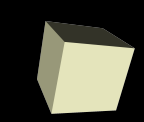
- There are lots of problems where our goal is to find a particular solution to some problem from a large set of potential solutions.
- With these types of problems the typical approach is to search through all of the solutions until you find one that is suitable. There are numerous ways to do this and some are more optimized than others.
- Some nice sample problems for this are the shortest path through a maze and the 0/1 knapsack problem.

Recursion/Depth First

- The most straightforward method of searching a solution space is with a recursive function. This produces a depth first search. Sometimes this method is referred to as backtracking. To see why one need only look at the maze problem.
- Depth first searching is ideal if all solutions are at the same depth or if valid solutions are all deep in the solution tree. There is also the advantage that it is easy to write.
- Depth first searches typically don't require that much memory either because the solution trees are typically much broader than they are deep.
- Sort things beginning with smallest branching factor for speed.

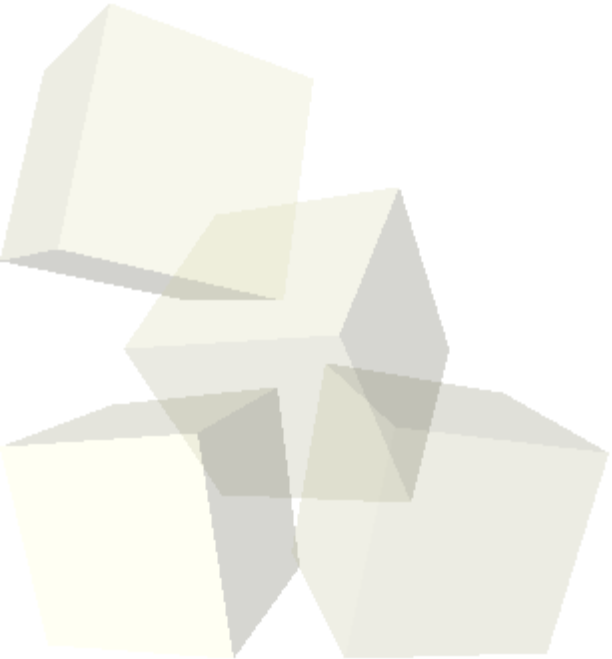
Breadth First

- Recursion implicitly uses a stack. If we use a queue instead then we get a breadth first search. This runs across each level of the solution tree in turn.
- When trees can be very deep and solutions can be higher in them, breadth first can be beneficial. It is harder to write because you have to put in the queue yourself.
- Memory can be a major issue. Solution trees can get very wide and breadth first must store an entire row at a time.



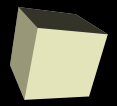
Iteration

- In some situations, you might be able to figure out a way to iterate through the possible final solutions. Not all problems led themselves to this.
- An example where this can work well is 0/1 knapsack where we can solve the problem by counting in binary.



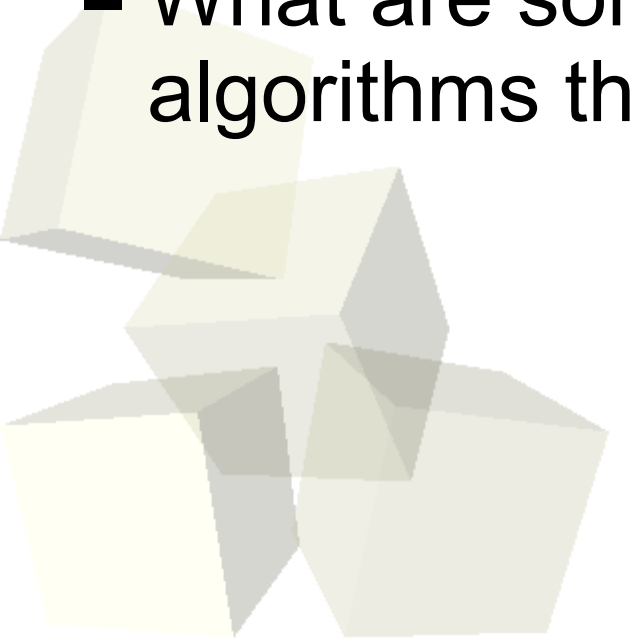
Pruning Trees

- Solution trees are very often exponential in size. As a result, searching the entire tree is not acceptable. We want to find ways to prune off paths.
- With a depth first approach we can often terminate certain recursive paths early by keeping extra information. One approach to this is memoization. Other approaches can be made that are more specific to the problem.



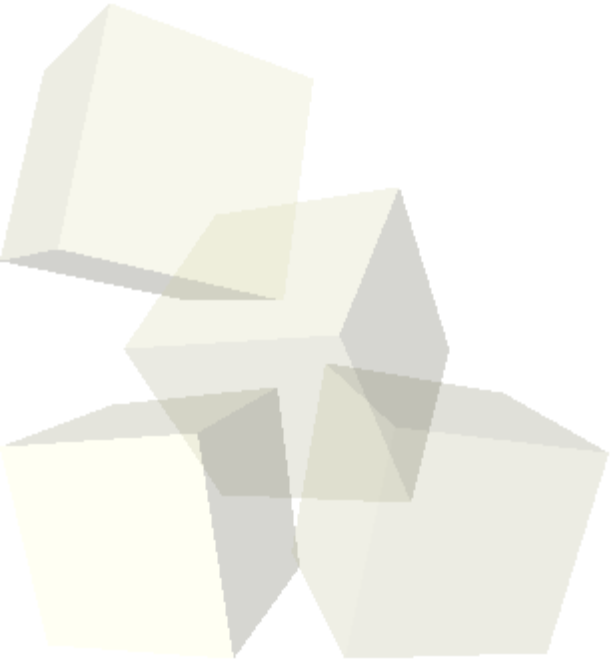
Divide and Conquer

- Divide and conquer is a common algorithm design technique. The basic idea is that we take a large subset and split it into smaller pieces, solve the pieces, and return the combine the answers.
- The trick in doing this is figuring out how to split things up and then how to put the answers back together.
- What are some common divide and conquer algorithms that you know of?



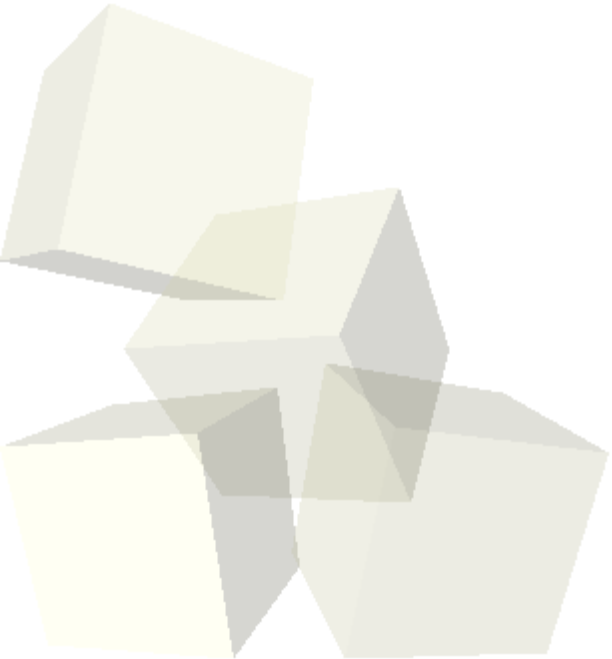
Counting Inversions

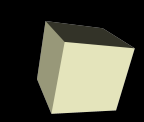
- How different are two lists of numbers? Simply number them in sorted order by one and count the number of inversions.
- We can do this with an augmented version of the merge sort.



Integer Multiplication

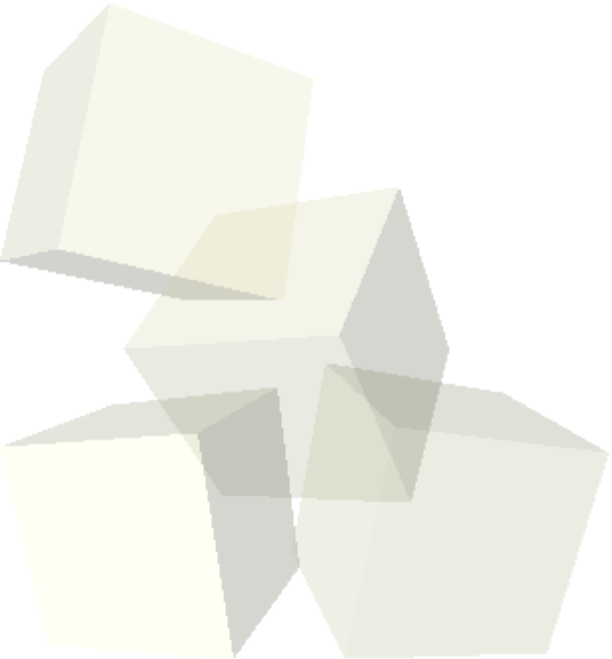
- Divide and conquer can also be used to give us a multiplication algorithm that is faster than $O(n^2)$.

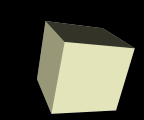




Other Algorithms

- Later in the semester we will be doing other divide and conquer algorithms for computational geometry and Fourier transforms.





Reminders

- We have canceled class for Thursday. I'm putting the finishing touches on the description for assignment #2 and it will be posted soon.