Analysis of Algorithms 2

3-8-2002

Opening Discussion

- What did we talk about last class? Do you have any questions about the assignment?
- Did anyone pay much attention to the example problem in section 6.3? I think that problem is very instructive as to how we could look at problems to make better solutions. The best one is not at all straightforward.

Not Always the Same

- Code rarely does the same number of instructions, even on the same input size. Most of the time we have conditionals that cause the path to vary depending on the exact nature of the input.
- For this reason we sometimes discuss different orders to the same algorithm.
  - Best-case, worst-case, average case

Best and Worst Case

- Some problems have certain inputs that can cause them to behave very good or very poorly. For example a flagged bubble sort is O(n) on sorted data.
- The case where the algorithm does the fewest operations is called the best-case. Because it is rare we typically don't care about it.
- The worst-case is also rare, but we do worry about it because it can "break" our code if a worst-case input is given and it takes too long to complete.

Average-Case

- Most of the time what we worry more about is the average case performance.
- For this we look at every "step" in the program and decide how many operations it will perform with "no particular" input.
- This is what we will typically look at though in some cases when the worst-case is different we will also mention it.

Logarithms

- The log function is the inverse of the exponential function. It is a function that we often like to see when doing analysis in the log function. The log function is a very slowly growing function. In fact, log n is of lower order than n^x for any x>0.
- We typically arrive at orders involving logs when we divide the "size" of a problem repeatedly.

Example - Searching

- One of the things that we frequently want to do with computers is to search through data. This is actually something that you have already had extensive experience with from assignment #2 and assignment #3.
- What order this operation is depends on the data structure and the data in it. For a linked list it is always O(n).

Unsorted Data

- If the data is unsorted then it doesn't matter what the data structure is because you always have to keep searching until you find it. This means that in the worst case you will look at all n elements of the collection. In the average case you might find it in n/2, but that is still O(n).

Binary Search

- If data is sorted and we have some way of accessing data from the "middle" of the sort order, then we can do searches faster.
- We are able to do this because we can look at data in the middle and determine if what we are looking for is above or below that. The part it isn't in gets ignored. This gives up O(log n) time.

Minute Essay

- What do you intend to do over spring break?
- I'll be posting code for assignment #1 and #2 before the end of the day. Right after spring break I will do the same for #3.
- Have a great spring break!