Analysis of Algorithms

3-6-2002

Opening Discussion

- Do you have any questions about the quiz?
- What did we talk about last class? Do you have any questions on assignment #3?
- From the readings, who can tell me what algorithm analysis is and why we are interested in it. What did you get out of today's reading?

What is Algorithm Analysis?

- One of the things that we are very interested in in computer science is making programs that can handle large problems efficiently (quickly). We have a formal way of talking about how efficient an algorithm is.
- In this we quantify how the amount of work a program does varies with the input size of the program, n.

Work? Input Size?

- Typically when we do an analysis we focus on certain operations and how many times they are performed. This is not done with exactly the running time of the computer. We might count additions, multiplications, comparisons, copies, etc.
- By input size we mean some measure of how big the problem we are working on is. Often number of "elements" being worked upon.

Counting Operations

- After picking the type of operation(s) we are interested in, we then want to be able to express that as a function of n.
- For simple dependencies this can typically be done by noticing the nature of the loops in the program. This logic in the innermost loop typically dominates the order of an algorithm.
- For recursive algorithms other tricks must be employed.

O() Notation

- The way you will typically here the order of a program referred to is with the big-O notation. So an algorithm will be called O(f(n)) for some f(n).
- In simple terms this means that the amount of work done by the algorithm increases with n in the same way that f(n) increases with n.

What it Really Means

- To be more formal, define g(n) as the actual number of operations that occur for an input of size n, then g(n) is O(f(n)) iff there exists positive C and m such that g(n)<C*f(n) for n>=m.
- Note that C and m are finite constants that do not vary with n. This means that if you can find such numbers, no matter how big they are, g(n) is O(f(n)).

Notes about O()

- Notice that coefficients are thrown out and that in general we only care about the largest term. All other terms become insignificant when the input size becomes very large.
- Asymptotic Analysis: This notation matters most when the input size is large. It can actually be meaningless for runtimes if the input size of the program is small.

Other Notations

- There are other bounds you should be aware of too.

$$g(n) \equiv O(f(n)) \leftrightarrow \exists C, m \ni g(n) \leq Cf(n) \forall n \geq m$$
$$g(n) \equiv \Omega(f(n)) \leftrightarrow \exists C, m \ni g(n) \geq Cf(n) \forall n \geq m$$
$$g(n) \equiv \Theta(f(n)) \leftrightarrow g(n) \equiv O(n) \wedge g(n) \equiv \Omega(n)$$
$$g(n) \equiv o(f(n)) \leftrightarrow g(n) \equiv O(n) \wedge g(n) \not\equiv \Theta(n)$$

Minute Essay

- What is the order of a search on a linked list? What about for building a sorted linked list? What will the order be on assignment #3 if you use a linked list?
- I want to distribute a code set solving assignments #1 and #2, but once I do so I can't accept any more submissions for assignment #2.