Self Assessment

Matt Forbes

December 8, 2010

Introduction

Thinking and working through problems this quarter has made been very satisfying. Challenge has been very prominent in just about every aspect of this course. Not only have the actual problems been difficult, but keeping my time managed while not losing quality has been as well. Trying to wrap my head around problems in this class was at first intimidating, but throughout the quarter I've developed a toolset and way of thinking that makes them much more managable.

Class Participation

After a rather tough bi-weekly assignment, I was chosen to present and explain the idea behind one of my solutions to the class. This is easier said than done. Actually getting up in front of a full class and trying to convey a hard-to-visualize thought is a bit nerve-racking. Luckily, I noticed that 'excellent' was written (and underlined twice!) next to my solution, so I was nearly positive I was going up. Knowing in advance gave me a bit of time to mentally prepare, but I don't have much experience in front of a class.

My solution was 'excellent' for the reason that I made sure to not use any assumptions without showing they were true. The problem was a divide and conqueror algorithm, but to prove that you could split the problem in half relied on an extra piece of information that I provided.

Context (from homework 2):

> If we take the median card from both hands, and call card **a** the smaller of the two, and **b** the greater. There are at most $n - \frac{n}{2}$ cards greater than **a** in its own hand. In **b**'s hand, there are at most $n - \frac{n}{2} - 1$ cards greater than **a**, because **a** > **b**. In total, there are at most $n - \frac{n}{2} + n - \frac{n}{2} - 1 = 2n - n - 1 = n - 1$ cards greater than **a**. Therefore, **a** must be greater than or equal to the absolute median of the 2n cards.
> Likewise for **b**.

Asymptotic Analysis

'Big-O' notation wasn't brand new to me coming in to this class, but my understanding was at a very simple level. It is generally easy to analyze basic algorithms like for loops, but I came to appreciate Big-O when analyzing more complicated forms like recursion.

Also from the card problem in homework 2, I used Big-O analysis to determine the runtime of my recursive algorithm. To bound the running time of my algorithm, I used a recurrence, and then bounded that recurrence from above with a 'Big-O value.' Demonstrating asympotic analysis with a recurrence:

> Let T(n) be the running time of this algorithm, where n is the total number of cards we are searching through.
>
> $$T(n) = \left\{ \begin{array}{cc} T(\frac{n}{2}) + O(1), & \text{for n} > 2 \\ \text{O}(1), & \text{for n} \leq 2 \end{array} \right\}$$
>
> An upper bound guess for T is $T(n) \leq c\log_2(n) + O(1)$
>
> $$\begin{aligned} T(n) &\leq c\log_2(\frac{n}{2}) + O(1) \\ &\leq c\log_2(n) - \log_2(2) + O(1) \\ &= c\log_2(n) + O(1) \end{aligned}$$
>
> So, $T(n)$ is $O(\log_2(n))$.

Finally an example of setting up a summation to find an upper bound of a running time. From homework 3: problem 3, in the first part I was required to find the running time of searching some kind of array of sorted arrays. So from my writeup:

> Each array A_i has 2^i elements, so a binary search on that array would have a running time of $\log_2(2^i) = i$. The total running time t of the search procedure would be:
>
> $$\begin{aligned} \text{let k} &= \lceil logn \rceil \\ t &= \sum_{i=0}^{k} \log_2 2^i \\ &= \sum_{i=0}^{k} i \\ &= \frac{k(k+1)}{2} \\ &= \frac{1}{2}\left(\log^2 n + \log n\right) \\ &= O(\log^2 n) \end{aligned}$$

Loop Invariants

When I was first introduced to loop invariants, they almost seemed overly verbose and overkill. Their merit didn't seem unwarranted, but they didn't look very fun to me at all. Throughout the quarter we used a few of these to prove some less intuitive algorithms were correct. There was no way to just look at these algorithms we proved and be able to infer their correctness, so having a tool that could strongly state this was appealing.

My only attempt at writing a loop invariant didn't go as well as I might have hoped. My problem was ambiguity, it was not immediately obvious why my invariant worked (if it did at all.) The idea was right, but the execution wasn't.

Even though my own invariant wasn't perfect, I was able to correctly prove a pre-written one. An example of this would be on homework two problem one.