TCSS 343 - Week 1 - Tuesday

Jake McKenzie

December 22, 2018

Asymptotics with Mathematical Induction

"Programs must be written for people to read, and only incidentally for machines to execute".

. . .

Harold Abelson

"You shouldn't try optimizing something that you can't measure."

. . .

Elecia White

Exact answers are nice when you can find them. Often times we can't or don't care to find them. As computer scientists we care about the behaivor of the algorithms we employ. If we run into a recurrence relation or summation that expressed loosly the behaivor of these algorithms we'd like to know something about their runtime.

Asymptotic Notation in Seven Words:
suppress constant factors and lower-order terms

Constant factors are things that will be language and system dependent while lower order terms are irrelevant for large inputs.

Now consider these definitions:
Big-O Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0$ then $f(n) \in O(g(n))$

This is another way of saying that $f(n) \leqslant c \cdot g(n)$ for some position c.
Big-Ω Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty$ then $f(n) \in \Omega(g(n))$

This is another way of saying that $f(n) \geqslant c \cdot g(n)$ for some position c.
Big-Θ Notation
If $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to k$ where k is a positive finite number then $f(n) \in \Theta(g(n))$

This is another way of saying that $f(n) \in O(g(n))$ and $f(n) \in \Theta(g(n))$.
Now while we're in this course, we will typically need to show that these relationships are true when prompted, but it's always good to know that the following is true where $0 < \epsilon < 1 < c$.

$$1 < \log\log n < \log n < n^{\epsilon} < n^{c} < n^{logn} < c^{n} < n^{n} < c^{c^{n}}$$

This asymptotic pecking order above is from Don Knuth's Concrete Mathematics.

From Tim Roughgarden's Algorithms Illuminated:

$$\max\{f(n), g(n)\} \leqslant f(n) + g(n) \wedge 2 \cdot \max\{f(n), g(n)\} \geqslant f(n) + g(n)$$

(reminder that $\wedge$ is the logical "and" symbol.)

0. Show the following by using the definition of big Theta and the information immediately above to show that $\max\{f(n), g(n)\} \in \Theta(f(n)+g(n))$

1. Arrange the following functions in order of increasing growth rate, with $g(n)$ following $f(n)$ in your list if and only if $f(n) \in O(g(n))$. That means using the limit definitions I gave you or by proof technique you've seen in class. You cannot simply use Don's asymptotic pecking order I stated but I strongly suggest you use the asymptotic pecking order as a guide to the neighborhood of a correct answer.

   a) $\sum_{i=0}^{n} 2^i$

   b) n^2

   c) $n^{0.9999999} \log n$

   d) 1.00001^n

   e) $4^{\log n}$

   f) $100^{\sqrt{n}}$

   g) $\log 2^{\frac{n}{2}}$

   h) $1000000n$

2. Arrange the following functions in order of increasing growth rate, with $g(n)$ following $f(n)$ in your list if and only if $f(n) \in O(g(n))$. That means using the limit definitions I gave you or by proof technique you've seen in class. You cannot simply use Don's asymptotic pecking order I stated but I strongly suggest you use the asymptotic pecking order as a guide to the neighborhood of a correct answer.

   a) $n^{\frac{5}{3}}$

   b) $\sum_{i=0}^{n}(i+1)$

   c) $n\sqrt{n}$

   d) $\log n^n$

   e) $(\frac{n}{\log n})^3$

   f) $2^{\frac{n}{2}}$

   g) $\log \log n$

   h) $n^{1.5}$

3. Prove by induction that: $\sum_{k=1}^{n} \frac{k}{2^k} = 2 - 2^{1-n} - n2^{-n}$.

4. The **pigeonhole principle** (in one of its more simple variations states: If you have n pigeons in h holes, then "some hole" has $\geqslant \lceil \frac{n}{h} \rceil$ pigeons. Prove this principle by strong induction. (Hint: What does $\frac{n}{h}$ represent mathematically?)

```
Algorithm Convert_to_Binary (n) ;
Input: n (a positive integer).
Output: b (an array of bits corresponding to the binary representation of n).

begin
    t := n ; { we use a new variable t to preserve n }
    k := 0 ;
    while t > 0 do
        k := k + 1 ;
        b[k] := t mod 2 ;
        t := t div 2 ;
end
```

Figure 2.6 Algorithm *Convert_to_Binary*.

5. For the following problem we will explore an induction and the notion of an **invariant** of an algorithm, which put simply, is a statement about a variable correct independent of the number of times a loop is executed. For the purposes of this algorithm the expression: $n = t \times 2^k + m$ is a loop invariant. Loop invariants can be hard to find but are often the heart of any algorithm.
   Induction Hypothesis: if m is the integer represented by the binary array $b[1 \ldots k]$, then $n = t \times 2^k + m$. To prove the correctness of this algorithm we must prove three conditions:

   (a) The hypothesis is true at the beginning of the loop.

   (b) The truth of the hypothesis at steps k implies the step $k + 1$.

   (c) When the loop terminates, the hypothesis implies the correctness of the algorithm.

   Attempt to complete the proof given the information I've given you to start.