Cody Hubbard 004843389

CS 180 HW 1

Spring 2017

CH 2 #5

Solutions:

a. FALSE - consider the counterexample where $f(n) = 10$ and $g(n) = 1$.

In this case $f(n) = O(g(n))$ because there exists a c, say $c = 10$, such that $f(n) \leq g(n) = 10 \leq 10 \cdot 1 = 10 \leq 10$. However cosider $\log_2 10$ and$10 \cdot \log_2 1$; $\log_2 10 \geq 10 \cdot \log_2 1$ and thus $\log_2 f(n) \neq O(\log_2(g(n))$.

b. FALSE - consider the counterexample where $f(n) = 2n^2$ and $g(n) = n^2$. In this case $f(n) \leq O(g(n)$ however $2^{2n^2} = 4^{n^2} \geq 2^{n^2}$and thus $2^{f(n)} \neq O(2^{g(n)})$.

c. TRUE -

PROOF: It is given that there exists a constant c and an n_0 such that for all $n > n_0$, $f(n) \leq c \cdot g(n)$. Consider $f(n)^2$and $g(n)^2$, $g(n)^2 = (c \cdot g(n))^2 = c^2 \cdot g(n)^2$, but $c^2$is just another constant say c'. So $f(n)^2 \leq c' \cdot g(n)^2$and thus $f(n)^2 = O(g(n)^2)$ for all $n > n_0$ as to be proved.

CH 2 #6

Solutions:

a. By analyzing the supplied pseudo code we can see that the outermost loop will run n tiems, the first nested loop will run n times, the adding of elements within the first nested loop will take, at its worst, n operations and the storage of the resutlts is one operation. This gives us a running time that looks something like $n(n(n+1)) = n^3 + n^2$and thus $f(n) = O(n^3)$.

b. Consider the algorithm when processing $\frac{n}{2}$ elements. The outermost loop will run $\frac{n}{2}$ times, the inner loop will preform $\frac{n^2}{4} + \frac{n}{4}$ operations, and thus the algorithm will have in total $\frac{n}{2}(\frac{n^2}{4} + \frac{n}{4}) = \frac{n^3}{8} + \frac{n^2}{8}$ operations and will be $\Omega(n^3)$.

c.

For i = 1, 2, . . . , n
Sum = i
    For j = i + 1, i + 2, . . . , n
        Add current array entrie, A[j] to the Sum
        Store the result in B[i, j]
    Endfor
Endfor

Since this algorithm removes the unessecary iteration though all n elements in the inner loop it will reduce the computations of the inner loop to just a constant. therefore in the calcualtions for both part (a) and (b) will be reduced to something like $n(n(1+1+1))$, and thus wll result in a reduction in run time to$\Theta(n^2)$.

CH 2 #8

Soultions:

a. Take the square root of the number of rungs we have and round up. Drop a jar in intervals of that number (so for 25 rungs drop the jar at 0, 5, 10...). Once the first Jar breaks search linearly by dropping the second jar linearly at the rung starting at the beginning of the current interval to the rung the first Jar broke at (i.e. if the first jar broke at 10 drop the second jar from rungs 6-9). $\lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$ as required.

b. Preform the method used for part (a) recursively. That is when a jar is broken, take the square root of the rung it broke at, round up, and search the interval it broke at in intervals of that number. If at any point you are at your last jar, search linearly though the current interval.