Problem Set 1

CSCI 3104 Spring 2014
Cristobal Salazar
07/22

Problem 1

a) True. To compare the functions, we take the limit as $n \to \infty$ of $n + 3/n^3$. Using L'Hospital's rule, we show $\lim_{n\to\infty} n + 3/n^3$, is equal to $\lim_{n\to\infty} 1/3n^2$. This limit is equal to 0, and therefore n^3 is growing larger than $n + 3$ and therefore can act as the upperbound of the algorithm, but the bound is not as tight as it could be.

b) False. The $\lim_{n\to\infty} 3^{2n}/3^n = \lim_{n\to\infty} 3^{2n-n}/1 = \lim_{n\to\infty} 3^n/1 = \infty$, therefore 3^n grows slower and cannot be the upper bound to 3^{2n}.

c) False. o is an upper-bound such that for all constants $k > 0$, you can find a constant a such that the inequality $f(x) < k * g(x)$ holds for all $x > a$. n^n grows larger than $n!$ as $n \to \infty$, therefore $o(n!)$ cannot be the upper-bound of this algorithm.

d) True. The $\lim_{n\to\infty}(1/3n)/1 = 0$ which means $o(1)$ grows larger and is the upper-bound for any constant c(refering to the previous problem) and therefore it is the correct.

e) Fasle. The $\lim_{n\to\infty} ln^3(n)/lg^3(n) = 0$. $lg^3(n)$ grows faster and therefore can be the upper bound, but cannot act as the lower bound. Therefore $\Theta(lg^3(n))$ cannot be the Θ of the algorithm $ln^3(n)$.

Problem 2

a) $d/dt(3t^4 + 1/3t^3 - 7)$ is equal to $12t^3 + t^2$ using the power rule.

b) This simplifies to the sum of $2^1 + 2^2 + ... + 2^{k-3} + 2^{k-2} + 2^{k-1} + 2^k$

c)This summation diverges to infinity at $n \to \infty$. Inside the summation is $1/k$, so the exponent of k is 1, and by using the p-test, we can prove it diverges, therefore going to infinity. So, $\theta(\infty)$ is the result, which means the program will never end.

Problem 3

This algorithm will traverse tree T using recursion starting at the root. This will look like:

```
function sort(currentnode){
    if(currentnode has no children)
        add currentnode to array
    if(currentnode has left child){
        sort(currentnode's left child)
        add currentnode to array
    }
    if(currentnode has right child)
        sort(currentnode's right child)
}
```

The $T(n) = 2T(n/2) + O(1)$ because each time we recurse, the tree is cut in half so it takes $T(n/2)$-time, but we recurse twice(once for both the left and right child), so it is multiplied by 2. Then adding it to the array take constant $O(1)$-time. This recurrence relation can be solved for the algorithm having $O(n)$-time, using the Master Theorem(cited below).

Problem 4

a) They should hire Flitwick because the current algorithm takes 1.99^{41} microseconds which is 20.72 days. Flitwick's algoritm takes 41^3 microseconds which is .0689 seconds, plus the 17 days to make it, which is still roughly 17 days. So they will save roughly 3.72 days if they hire Flitwick.

b) They should not hire Flitwick. Their current algorithm takes $1,000,000^{2.00}$ microseconds, which is 11.57 days. Flitwick's algorithm takes $1,000,000^{1.99}$ microseconds which is 10.08 days plus the 2 days to create it, makes a total of 12.08 days. So the company should stick to their current algorithm.

Problem 5

a) No. $\lim_{n\to\infty} 2^{nk}/2^n = \lim_{n\to\infty} 2^{nk-n}/1 = \lim_{n\to\infty} 2^{n(k-1)}/1 = \infty$, for $k > 1$. Therefore 2^n grows slower and cannot be the upper bound to the algorithm 2^{nk} for $k > 1$, so it is not a valid big-O.

b) Yes. The $\lim_{n\to\infty} 2^{n+k}/2^n = \lim_{n\to\infty} 2^{n+k-n}/1 = \lim_{n\to\infty} 2^k = 2^k$. This limit converges at 2^k, therefore $O(2^n)$ can act as the upper bound to the algorithm 2^{n+k}.

Problem 6

An array that is in sorted order from smallest to largest will always also be a min-heap. Heaps are arrays that store the parent node at i, its left child at $2i+1$ and its right child at $2i+2$. A min-heap is a heap such that every parent node is smaller than both of its children, but the children have no relation to eachother. So if an array is in ascending order, the parents will always be smaller than the children because $2i+1$ and $2i+2$(the children) are further down the array than i(the parent), and therefore will always be larger, thus satisfying the definition of a min-heap.

Sources

- http://stackoverflow.com/questions/1364444/difference-between-big-o-and-little-o-notation
  - Definition of little-o
- http://en.wikipedia.org/wiki/Master_theorem
  - Solving recurrence relations using Master Theorem