Problem Set 1

All parts are due February 18th, 2016 at 11:59PM.

Name: Laser Nite

Collaborators: Jonathan Buschel

Part A

Problem 1-1.

(a)

$$f5, f4, f1, f2, f3$$

$f3$ is the only exponential. $f2$ is the largest polynomial; the n^4 drops down so we know $f2$ is larger than $f1$ as $log(4n)$ overtakes 4. The others are obvious; $n > log(n) > log(log(n))$

(b)

$$f4, f5, f1, f2, f3$$

$f3$ clearly grows faster than $f2$ as the $+1$ can be pulled out as a constant. Then obviously $f1$ as it is one less on the highest exponent, followed by two lower order terms $f5$ and $f4$, of which *f5* can be seen to be bigger due to increments of n by 1 increasing the total value by a larger multiple.

(c)

$$f1, f4, f2, f5, f3$$

$f3$ is much larger than $f5$, as although they both have an n^n order, $f5$ is taking the value at $1/4$ of n and multiplying it by itself $1/4$ as many times as $n!$, which means just the top quarter of $f3$ is larger than $f5$, as it's all numbers larger than $n/4$ getting multiplied by each other $n/4$ times. $f5$ and $f2$ is a tricky comparison that require simplifying expressions and lining up terms such that it can be seen $f5$ overpowers $f2$. Between $f2$ and $f4$ there's a change in order. $f1$ mostly cancels out itself such that it is on the same order as n^4, a polynomial which is clearly smaller than exponential $f4$.

Problem 1-2.

(a)
1. $O(n)$. As shown by a recursion tree, there are n steps, iterating $n-1$ until zero. Each step adds a constant c, so the total time is on the order of n times some constant c.
2. $O(n^2)$. Similar to (a), except at each step of the recursion tree cn is added, thus the total run time is on the order of $cn \times n$ or $O(n^2)$.
3. $O(log_2\ n)$ This recursion tree adds the constant c at each level like (a), except the number of levels of the tree is $log_2\ n$ as n cuts in half at each level, reaching 0 in logarithmic time. Thus the total run time is on the order of $c \times log_2\ n$.
4. $O(n)$ The constant c is added at each level of the recursion tree for each call of the recursion, so although n cuts in half at each level and reaches 0 in $log_2\ n$ time, the recursion is also called twice as many times at each level, effectively canceling out the gains in reduced depth as the width of each level of the tree doubles, starting at the constant c. Effectively, then, the total run time is on the order of $log_2\ (c * 2^n)$ which resolves to n times some constant.
5. $\theta(n\ log_2 n)$. The Master Theorem was applied.
6. $\theta(n^{log_2 3})$. The Master Theorem was applied. For practice, a recursion tree was also made that resolves to something on the order of $3^{log_2\ n}$ which is equivalent to $n^{log_2 3}$.

(b)
1. $T(n) = c$ for $n \leq 1$, and $T(n) = T(n/2) + c$ for $n > 1$
2.

Problem 1-3.

(a)

(b)

Part B

Problem 1-4.

Submit your implemented python script.

(a)

(b)

(c)

(d)

(e)