CSCI 3104 Summer 2017 - Assignment # 1

Due: June 9th 2:30pm

Submit any code to the Assignment 1 Code submission link on moodle.
Be sure to justify all of your work.

1. *Asymptotic Analysis*
   a) Show that $\ln^3 n = \Theta(\log_2^3 n)$.
   b) Is $3^{2n} = \mathcal{O}(3^n)$? Justify your response.
   c) Is
   $$\sum_{k=1}^{n} k(k-1) = \Theta(n^3)?$$
   Justify your response with a proof.
   d) Prove that if $f = \mathcal{O}(g)$ and $g = \mathcal{O}(h)$, then $f = O(h)$.
   e) Prove that a polynomial function $g(n)$ of degree d is $\mathcal{O}(n^d)$. You may assume positive coefficients.

2. *Asymptotic Growth Rates*
   Order the following functions in order of smallest to largest asymptotic growth rates:
   $2^{\log n}$, e^n, $2^{2^{2n+4}}$, $n^{\frac{1}{\log n}}$, n^3, $\sqrt{\log n}$, 2^n, $\log^2 n$, $\left(\frac{3}{2}\right)^n$, $(\log n)^{\log n}$.

3. You are given two ascending order sorted arrays of integers A and B. Consider the following (pseudocode) algorithm for checking whether or not A and B have an element in common:

```
def checkIfElementInCommon(A, B):
    # Assume A, B are already sorted
    for i = 0:length(A)
        for j = 0:length(B)
            if (A[i] == B[j]):
                return TRUE
    return FALSE
```

   a) What is the worst-case running time of this algorithm? Give an asymptotic bound.
   b) For $n = 5$, write down an example of input arrays A_1, B_1 that will be the best case and an example of arrays A_2, B_2 that will be the worst case for the algorithm above.
   c) Come up with an algorithm that will have a worst-case running time of $\mathcal{O}(n)$. Implement both the original algorithm and your own $\mathcal{O}(n)$ algorithm in `Python`. Plot the runtimes of both algorithms as functions of input size for various values of n, e.g. $n = 2^k$ for $k = 2, 3, \ldots, 16$. You may create your initial sorted arrays any way you choose, but your implementation of the algorithm must **not** use any non-trivial `Python` libraries. Did your experiment agree with your theoretical analysis?

4. Suppose you are given a number and a 2D matrix where every row is sorted in ascending order, and the last number in each row is no greater than the first number of the next row. Come up with and describe an $\mathcal{O}(\log mn)$ algorithm which will return TRUE if the number exists in the matrix, and FALSE if it does not (Pseudocode is fine).

5. *Recurrence relations*
   Solve the following recurrences using the expansion method:
   a) $T(n) = T(\sqrt{n}) + 1$ if $n \geq 2$, and $T(n) = 0$ if $n < 2$.
   b) $T(n) = 4T(\frac{n}{2}) + cn^2$, and $T(1) = c$, where n is a power of 2 and c is a constant.

6. *Master Method*
   Solve the following recurrence relations using the Master Theorem, if applicable. If the Master Theorem is not applicable, please explain why not.
   a) You are given the following function:

```
def hello(n):
    if n > 1:
        print("hello")
        hello(n/4)
        hello(n/4)
```

   Based on the input n, write down and solve the recurrence relation for the number of times "hello" is printed.
   b) $T(n) = 2^n T(\frac{n}{4}) + n^2$
   c) $T(n) = 4T(\frac{n}{2}) + n^2$