CSCI 3104 **Profs. Clauset & Grochow**
Problem Set 2 **Spring 2018, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Purely verbal or purely mathematical reasoning is typically insufficient for full credit. Instead, use mathematics to make your argument more precise and logical, and use language to explain your mathematical reasoning more clearly.

---

1. (20 pts total) Solve the following recurrence relations using any of the following methods: unrolling, tail recursion, recurrence tree (include tree diagram), or expansion. Each each case, show your work.

   (a) $T(n) = T(n-2) + C\,n$ if $n > 1$, and $T(n) = C$ otherwise

   (b) $T(n) = 3T(n-1) + 1$ if $n > 1$, and $T(1) = 3$

   (c) $T(n) = T(n-1) + 3^n$ if $n > 1$, and $T(1) = 3$

   (d) $T(n) = T(n^{1/4}) + 1$ if $n > 2$, and $T(n) = 0$ otherwise

2. (10 pts) Consider the following function:

```
def foo(n) {
     if (n > 1) {
        print( ''hello'' )
        foo(n/3)
        foo(n/3)
}}
```

   In terms of the input n, determine how many times is "hello" printed. Write down a recurrence and solve using the Master method.

3. (30 pts) Professor McGonagall asks you to help her with some arrays that are *raludominular*. A raludominular array has the property that the subarray $A[1..i]$ has the property that $A[j] > A[j+1]$ for $1 \le j < i$, and the subarray $A[i..n]$ has the property that $A[j] < A[j+1]$ for $i \le j < n$. Using her wand, McGonagall writes the following raludominular array on the board $A = [7, 6, 4, -1, -2, -9, -5, -3, 10, 13]$, as an example.

   (a) Write a recursive algorithm that takes asymptotically sub-linear time to find the minimum element of A.

(b) Prove that your algorithm is correct. (Hint: prove that your algorithm's correctness follows from the correctness of another correct algorithm we already know.)

(c) Now consider the *multi-raludominular* generalization, in which the array contains k local minima, i.e., it contains k subarrays, each of which is itself a raludominular array. Let $k = 2$ and prove that your algorithm can fail on such an input.

(d) Suppose that $k = 2$ and we can guarantee that neither local minimum is closer than $n/3$ positions to the middle of the array, and that the "joining point" of the two singly-raludominular subarrays lays in the middle third of the array. Now write an algorithm that returns the minimum element of A in sublinear time. Prove that your algorithm is correct, give a recurrence relation for its running time, and solve for its asymptotic behavior.

4. (15 pts extra credit)

Asymptotic relations like O, Ω, and Θ represent relationships between *functions*, and these relationships are transitive. That is, if some $f(n) = \Omega(g(n))$, and $g(n) = \Omega(h(n))$, then it is also true that $f(n) = \Omega(h(n))$. This means that we can sort *functions* by their asymptotic growth.[1]

Sort the following *functions* by order of asymptotic growth such that the final arrangement of functions $g_1, g_2, \ldots, g_{12}$ satisfies the ordering constraint $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$, $\ldots$, $g_{11} = \Omega(g_{12})$.

n	$n^{1.5}$	$8^{\lg n}$	$4^{\lg^* n}$	$n!$	$(\lg n)!$	$\left(\frac{5}{4}\right)^n$	$n^{1/\lg n}$	$n \lg n$	$\lg(n!)$	e^n	42

Give the final sorted list and identify which pair(s) functions $f(n), g(n)$, if any, are in the same equivalence class, i.e., $f(n) = \Theta(g(n))$.

[1]The notion of sorting is entirely general: so long as you can define a pairwise comparison operator for a set of objects $\mathcal{S}$ that is transitive, then you can sort the things in $\mathcal{S}$. For instance, for strings, we use a comparison based on lexical ordering to sort them. Furthermore, we can use any sorting algorithm to sort $\mathcal{S}$, by simply changing the comparison operators $>$, $<$, etc. to have a meaning appropriate for $\mathcal{S}$. For instance, using Ω, O, and Θ, you could apply QuickSort or MergeSort to the functions here to obtain the sorted list.