CS:3330 Homework 2, Fall 2015

Due in class on Tue, Sept 29

---

1. You are given an n-vertex graph $G = (V, E)$ with the property that it consists of a bunch of connected components, each of which has size $O(\log n)$. Design an algorithm for the maximum independent set problem with this input, running in polynomial time.

2. Problem 8(a) from Chapter 2 in the textbook.
   Hint: $f(n) = \Theta(\sqrt{n})$.

3. Consider the following algorithm for testing if a given positive integer n is a prime number.

```
input(n)
f ← 2
while f ≤ √n do
    if f evenly divides n then
        print("Not a prime")
        stop
    f ← f + 1

print("prime")
```

   (a) Argue that this algorithm is correct – specifically, that it is sufficient to only consider candidate factors of n in the range 2 through $\sqrt{n}$.

   (b) What is the worst case running time of this algorithm as a function of n?

   (c) Let s be the *size* of the input to this algorithm. Rewrite the worst case running time of this algorithm as a function of s. Is this a polynomial-time algorithm or an exponential-time algorithm?

   (d) Implement the above algorithm in your favorite high-level language (e.g., Java, C++, Python, Scala, etc.). This should take no more 10-15 lines of code. Try to use your program to determine if `758500183202087890352073067` is a prime. As an answer to this problem, tell us if this number is a prime, but more importantly, tell us how long your program took to determine if the input was a prime. For this part of the question, it is okay to say something like "I ran my program overnight and it did not complete and so I don't know if the input is a prime."

4. Take the following list of functions (from nonnegative integers to nonnegative integers) and arrange them in ascending order of growth. Thus, if a function g immediately follows f in the list, then $f = O(g)$. Some of these functions are described in words and some as summations. First, for every function described in words or as a summation, write it in standard form before placing it in the sorted list of functions.

   (a) $\sum_{i=1}^{n} i$

   (b) The running time of the `mergeSort` algorithm.

   (c) $(\log_2 n)^{\log_2 n}$

   (d) $n^{\log_2(\log_2 n)}$

   (e) $(\log_2 128)^n$

   (f) The running time of the brute force algorithm for the *Maximum Independent Set* problem discussed in class.

(g) $\sum_{i=0}^{n} \frac{n}{2^i}$

(h) 100

5. You are given a sorted list L of numbers (say, sorted in increasing order). The problem is to find two indices i and j, $i \neq j$ such that $L[i] + L[j] = 0$. Of course, it is possible that no such i and j exists (e.g., if all elements in L are positive). In that case, the output should indicate that no such i and j exist.

   Your task is to describe an $O(n)$ time algorithm for this problem.

   Hint: Start with two "pointers" – one pointing to the beginning of the list (i.e., to a small number) and one pointing to the end (i.e., to a large number). In each step of the algorithm, one of the two pointers will move in an appropriate direction. What exactly happens in each step is for you to figure out.

6. Suppose that the input is a list L of numbers with the "promise" that at least 90% of the numbers in the list are positive or at least 90% of the numbers in the list are negative. In the former case, we say that L is *positive* and in the latter case we say that L is *negative*. My goal in this problem is to design and analyze a fast, randomized algorithm that determines if L is positive or negative. Here is my attempt at doing this.

```
n ← length(L)
i1 ← random(1, n), i2 ← random(1, n), i3 ← random(1, n)
if at least two elements in {L[i1], L[i2], L[i3]} are positive then
    output("positive")
else
    output("negative")
```

   (a) What is the running time of this algorithm?

   (b) This algorithm can, on ocassion, produce incorrect output. Explain how this can happen.

   (c) Suppose that the input L is positive. Calculate the probability that the output of the above algorithm is `"negative."` To help you with this calculation, note that for the output to be `"negative"` it must be the case that:

   (A) $L[i_1]$ and $L[i_2]$ are negative, but $L[i_3]$ is positive or

   (B) $L[i_1]$ and $L[i_3]$ are negative, but $L[i_2]$ is positive or

   (C) $L[i_2]$ and $L[i_3]$ are negative, but $L[i_1]$ is positive or

   (D) $L[i_1]$, $L[i_2]$, and $L[i_3]$ are all negative.

   Calculate the probability of each of the events (A), (B), (C), and (D), given that L is positive. The sum of these probabilities is the probability that the algorithm will output `"negative"` even though L is positive.
