CS:3330 Homework 3, Spring 2018

Due in class on Thu, Feb 8

1. Consider the following code fragment. Analyze its running time and express your answer as $\Theta(f(n))$, for some function f of n.

   It is possible to solve this problem by precisely figuring out how many times the `print` statement is executed. This will involve coming up with a sum and finding a closed form solution to it. I would like you to follow a less tedious approach, outlined below:

   (a) Show that the running time is $O(n^3)$. You can do this by "overestimating" the work done by the code fragment, e.g., by allowing j and k variables to travel through a bigger range of values.

   (b) Show that the running time is $\Omega(n^3)$. You can do this by "underestimating" the work done by the code fragment, e.g., by restricting your attention to when i travels from 1 to $n/3$ and j travels from $2n/3$ to n.

   (c) Using (a) and (b), conclude that the running time of this code fragment is $\Theta(n^3)$.

```
for i ← 1 to n do
    for j ← i + 1 to n do
        for k ← i to j do
            print("hello")
```

2. Implement the following Primality Testing algorithm, based on Fermat's Little Theorem, efficiently. Note that k in the code is some small

   positive integer parameter, that you can either hardcode or pass as argument to the function. Also note that you should use the efficient implememtation of the function `moduloPower` from Homework 1 to compute $a^{n-1} \bmod n$.

Input: a positive integer n
Algorithm: FLTPrimalityTest
repeat k times
  pick an integer a at random from $[1, n-1]$
  if $a^{n-1} \bmod n \neq 1$, output `composite` and exit the program.
output `prime`

   (a) Use your implementation of FLTPrimalityTest algorithm to determine which of the 5 numbers in the posted text file (`primalityTest.txt`) are prime and which are composite.

   (b) As you know, the FLTPrimalityTest algorithm can incorrectly classify composites as primes. Set $k = 5$ in the FLT Primality Test algorithm and determine the number of integers in the range $[500, 100000]$ that are, on average, incorrectly classified as primes by the algorithm. Since the algorithm is randomized, it will likely behave differently each time it is executed. So run the algorithm 10 times and report the average number of integers in the range $[500, 100000]$ that are incorrectly classified To complete this task, your program would have to be able to *correctly* identify primes/composites and the easiest way to do this is to simply implement and use the naive primality testing algorithm.

(c) Re-run the experiment in (b) with $k = 15$. You should see fewer incorrect classifications now (compared with $k = 5$). Once again, report the average number of integers in the range $[500, 100000]$ that are incorrectly classified.

(d) Now set $k = 20$ and produce as output all integers in the range $[500, 100000]$ that are incorrectly classified as primes from one run of FLP PRIMALITY TEST. Compare this output with the list of Carmichael numbers less than 100000. See

`http://www.chalcedon.demon.co.uk/rgep/cartable.html`

for lists of Carmichael numbers. Are you seeing any non-Carmichael composites classified as primes?

3. We are given a length-n binary list L. In other words, $L[j] \in \{0, 1\}$ for all $j = 1, 2, \ldots, n$. Our problem is to count the number of 0's in L. We could of course solve this problem in $\Theta(n)$ time by simply scanning L, but we want to solve it faster and so we turn to randomization. The randomized algorithm we use is the following.

```
function CountZeroes(L)
    n ← length(L)
    count ← 0

    for j ← 1 to 100 do
        Comment: pick a random index i between 1 and n
        i ← random(1, n)
        if L[i] = 0 then
            count ← count + 1

    return n · count/100
```

(a) What is the running time of this algorithm? Please show all your work.

(b) Suppose that $n = 10^6$ and exactly a quarter of the elements in L are 0. What is the probability that the CountZeroes function returns 0?

(c) A random variable X has *binomial distribution* with parameters m (a positive integer) and p (a real number between 0 and 1) if

$$\Pr[X = k] = \binom{m}{k} \cdot p^k \cdot (1-p)^{m-k}. \tag{1}$$

The way to think about X being binomially distributed is that we perform m independent random trials and each trial can either succeed or fail. The "success" probability of a trial is p and so the failure probability of a trial is $1 - p$. Then the random variable X represents the number of successful trials, out of m total random trials. To understand the formula in (1) note that the probability of k successful trials is p^k, the probability of $(m - k)$ unsuccessful trials is $(1 - p)^{m-k}$, and there are $\binom{m}{k}$ ways of distributing the k successful trials among the m random trials.

As in (b) suppose that $n = 10^6$ and exactly a quarter of the elements in L are 0. Using the above expression for a binomial distribution, write down the expression for the probability that CountZeroes returns the correct answer, namely $10^6/4$.