CS:3330 Homework 4, Spring 2018

Due in class on Thu, Feb 15

1. Suppose that the input is an array L of numbers with the "promise" that either 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? Explain your reasoning.

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

   (c) Suppose that the input L is positive. Calculate a value p such that the probability that the output of the above algorithm is `"negative"` is at most p. 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 an upper bound on the probability of each of the events (A), (B), (C), and (D) happening, given that L is positive. The sum of these probabilities is an upper bound on the probability that the algorithm will output `"negative"`, even though L is positive.

   (d) Describe in 1-2 sentences how you might modify the code fragment above, without increasing its asymptotic running time, to reduce the probability that the algorithm makes an error.

2. I have posted a file `words.dat` containing 5,757 5-letter words. (This file was created by Donald Knuth, a Turing award winning computer and these were all valid 5-letter English words at the time at which the file was created.) Now suppose that you want to create a hash table to store these words.

   (a) Using ideas from our example of constructing a hash function for storing 250 IP addresses (in lecture, Tue Feb 6th), construct a hash function for storing these 5-letter words in a hash table. Recall that the first step in designing your hash function is deciding how large you want the hash table to be. Describe your hash function as clearly as you can.

   (b) Write a program that that reads from `words.dat` and uses your hash function to hash words into hash table slots. You don't have to explicitly store words in the hash table. I am only interested in knowing if your hash function has been able to spread the words out in the hash table. To help me understand this, output the average number of words that hashed to a non-empty slot and the maximum number of words that hashed to a slot.

3. Analyze the running time of the following code fragment and express your answer using Θ notation, as a function of n. Please show your work in order to receive partial credit.

```
j ← n
for i ← 1 to n do
    k ← 1
    while k ≤ j do
        print("hello")
        k ← k + 1
    j ← j/2
```

4. Solve Problem 2.12 from the textbook.
