Homework #1

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Time Complexity

In class, we learned about big-O analysis and we talked about the pros and cons of using big-O for runtime representation. Reason over the two scenarios below, both of which make you think about the implication of these trafeoffs.

(a) There exists a very popular sorting algorithm called Timsort, the default sorting algorithm in both Python and Java. This sort is a combination of two different sorting algorithms: Merge sort, and Insertion sort. Recall that the Big-O of Merge sort is $O(nlogn)$ and the Big-O of Insertion sort is $O(n^2)$. What advantage would Timsort have to combine the two algorithms if merge-sort has a better Big-O metric?

(b) Consider two algorithms: $f(n)$ and $g(n)$. You run both algorithms with an input $n = 10,000$. You find that $f(n)$ takes 10 ms while $g(n)$ takes 1 min to run. Which of these has a better (i.e. smaller) Big-O metric?

Problem 2: Algorithms and Decisions

You are given 9 identical looking balls and told that one of them is slightly heavier than the others. Your task is to identify the defective ball. All you have is a balanced scale that can tell you which of two balls is heavier.

(a) Show how to identify the heavier ball in just 2 weighings.

(b) Justify why it is not possible to determine the defective ball in fewer than 2 weighings.

Problem 3: Algorithm Analysis

Answer the following questions based on the following psuedocode for the function "foo":

Algorithm 1: foo

```
mystery()
foreach i ← 1 to n do
    mystery()
    foreach j ← 1 to i do
        if i ≤ j then
            mystery()
```

A Determine the exact number of times mystery() is called in terms of n.

B Assume the mystery function is in $O(log(n) * n^2)$. Determine the complexity of "foo" in O-notation.

Problem 4: Algorithm Analysis

Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$—that is, the sum $A[i]+A[i+1]+\cdots+A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.) Below is a simple algorithm to solve this problem:

Algorithm 2: Simple Approach

```
foreach i ←1 to n do
    foreach j ←i + 1 to n do
        Add entries A[i] through A[j]
        Store the results in B[i,j]
```

(a) Give a function $f(n)$ that is an asymptotically tight bound on the running time of the algorithm above. Using the pseudocode above, argue that the algorithm is, in fact $\Theta(f(n))$.

(b) Although the algorithm you analyzed above is the most natural way to solve the problem, it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem with an asymptotically better running time than the provided algorithm.

(c) What is the running time of your new algorithm?