DSC 40B - Homework 01

Due: Tuesday, January 12

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope on Tuesday at 11:59 p.m.

Problem 1. *(Eligible for Redemption)*

For each of the following pieces of code, state the time complexity in terms of n using asymptotic notation. You do not need to show your work.

a)
```
def f_1(n):
    total = 0
    for i in range(n):
        for j in range(n**2):
            for k in range(n//4):
                total += max(i, j, k)
    return total
```

b)
```
def f_2(n):
    total = 0
    for i in range(n):
        for j in range(n + i):
            total += max(i, j)
    return total
```

c)
```
def mode(data):
    """
    computes the mode
    `data` is a list of size n
    """
    mode = None
    largest_frequency = 0
    for x in data:
        count = 0
        for y in data:
            if x == y:
                count += 1
        if count > largest_frequency:
            largest_frequency = count
            mode = x
    return mode
```

d)
```
def variance(data):
    """
    computes the variance of `data`
    `data` is a list of size n
    """
    mu = sum(data) / len(data)
    total = 0
    for x in data:
```

```
            total += (x - mu)**2
        return total / len(data)
```

e)
```
def also_the_variance(data):
    """
    computes the variance of `data`
    `data` is a list of size n
    """
    # compute the variance again
    total = 0
    for x in data:
        mu = sum(data) / len(data)
        total += (x - mu)**2
    return total / len(data)
```

f)
```
def f_6(n):
    for i in range(3*n + 5, n**2 + 3*n + 100):
        for j in range(int(n**0.5)):
            print(i, j)
```

g)
```
def f_7(n):
    i = 100
    while i < n:
        i *= 3.1415
        print(i)
```

h)
```
def f_8(n):
    i = 10
    while i < n**2:
        i += 3
        for j in range(n):
            print(i, j)
```

i)
```
def f_9(n):
    for x in range(3*n + 1_000_000, 5*n):
        for y in range(42):
            print(x, y)
```

Problem 2. *(Eligible for Redemption)*

As a data scientist, it is important to have a sense for which algorithms are feasible for a given problem size. For instance: How big of a data set can a quadratic algorithm crunch in one minute? That's what this problem aims to find out.

Suppose Algorithm A performs n basic operations when given an input of size n, while Algorithm Bperforms n^2 basic operations and Algorithm C performs n^3 basic operations. Suppose that your computer takes 1 nanosecond to perform a basic operation. What is the largest problem size each can solve in 1 second, 10 minutes, and 1 hour? That is, fill in the following table:

	1 sec.	10 min.	1 hr
Algorithm A	?	?	?
Algorithm B	?	?	?
Algorithm C	?	?	?

Show your work for at least one cell in each row.

Note: 1 nanosecond is a very *optimistic* estimate of the time it takes for a computer to perform a basic operation.[1]

Hint: it's a small thing, but your answers should never be decimals. It isn't possible to perform 3.7 operations, after all. Should you round up or round down?

Problem 3. *(Eligible for Redemption)*

For each of the following functions, express its rate of growth using Θ-notation, and prove your answer by finding constants which satisfy the definition of Θ-notation. Note: please do *not* use limits to prove your answer (though you can use them to check your work).

a) $f(n) = 3n^4 + 10n^2 + 4$

b) $f(n) = \dfrac{n^2 + 2n - 5}{n - 10}$

c) $f(n) = \begin{cases} n + 10, & n \text{ is odd,} \\ n - 5, & n \text{ is even} \end{cases}$

Problem 4.

In lecture, we saw the following algorithm for computing a median:

```
def median(numbers):
    min_h = None
    min_value = float('inf')
    for h in numbers:
        total_abs_loss = 0
        for x in numbers:
            total_abs_loss += abs(x - h)
        if total_abs_loss < min_value:
            min_value = total_abs_loss
            min_h = h
    return min_h
```

We said that this algorithm has quadratic time complexity, which means that the time it takes to run grows like n^2, where n is the size of the input list. Let's see if this is indeed the case by timing the function on inputs of various size.

a) Write a function named `time_median` which takes in a single argument, `n`. It should create a list of `n` numbers and call `median` ten times on that list, timing each call. Your function should return the average time it took for `median` to run over all ten calls. Provide your code.

Hint: We saw the `timeit` magic functions in lecture, but the `time` function in Python's `time` module is more useful here. To time the execution of a function call, we can write:

```
import time
start = time.time()
my_function_that_i_want_to_time()
end = time.time()
elapsed = end - start
```

[1] See Peter Norvig's essay "Teach Yourself Computer Programming in 10 Years" for a table of timings for various operations. http://norvig.com/21-days.html

b) Using your function `time_median`, time `median` on inputs of size 100, 400, 700, 1000, 1300, 1600, and 1900. Create a plot of these times, where the horizontal axis measures the input size, and the vertical axis measures the average time taken by `median` in seconds.

Include your code in your solution. You may use any tool you like to make the plot, though we recommend using `matplotlib`.

Problem 5.

Determine the time complexity of the following piece of code, showing your reasoning and your work.

```
def f(n):
i = 1
while i <= n:
    i *= 2
    for j in range(i):
        print(i, j)
```

Hint: you might need to think back to calculus to remember the formula for the sum of a geometric progression... or you can check wikipedia.[2]

[2] https://en.wikipedia.org/wiki/Geometric_progression