DSC 40B - Homework 02

Due: Wednesday, April 19

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 at 11:59 p.m.

Problem 1.

State the growth of the function below using Θ notation in as simplest of terms possible, and prove your answer by finding constants which satisfy the definition of Θ notation.

E.g., if $f(n)$ were $3n^2 + 5$, we would write $f(n) = \Theta(n^2)$ and not $\Theta(3n^2)$.

$$f(n) = \frac{n^3 - n^2 + n + 1000}{(n-1)(n+2)}$$

Problem 2.

Suppose $T_1(n), \ldots, T_6(n)$ are functions describing the runtime of six algorithms. Furthermore, suppose we have the following bounds on each function:

$$\begin{aligned}
T_1(n) &= \Theta(n^{2.5}) \\
T_2(n) &= O(n \log n) \\
T_3(n) &= \Omega(\log n) \\
T_4(n) &= O(n^4) \text{ and } T_4 = \Omega(n^2) \\
T_5(n) &= \Theta(n) \\
T_6(n) &= \Theta(n \log n) \\
T_7(n) &= O(n^{1.5} \log n) \text{ and } T_7 = \Omega(n \log n)
\end{aligned}$$

What are the best bounds that can be placed on the following functions?

For this problem, you do not need to show work.

Hint: watch the supplemental lecture at `https://youtu.be/tmR-bIN2qw4`.

Example 1: $T_1(n) + T_2(n)$.

Solution: $T_1(n) + T_2(n)$ is $\Theta(n^3)$.

Example 2: $f(n) = 2 \cdot T_4(n)$.

Solution: $f(n) = 2 \cdot T_4(n)$ is $O(n^4)$ and $\Omega(n^2)$.

a) $T_1(n) + T_5(n)$

b) $T_2(n) + T_6(n)$

c) $T_2(n) + T_4(n)$

d) $T_7(n) + T_4(n)$

e) $T_3(n) + T_1(n)$

f) $T_1(n) \times T_4(n)$

Problem 3.

In each of the problems below compute the average case time complexity (or expected time) of the given code. State your answer using asymptotic notation. Show your work for this problem by stating what the different cases are, the probability of each case, and how long each case takes. Also show the calculation of the expected time.

a)
```
def foo(n):
    # randomly choose a number between 0 and n-1 in constant time
    k = np.random.randint(n)

    if k < np.sqrt(n):
        for i in range(n**2):
            print(i)
    else:
        print('Never mind...')
```

b)
```
def bogosearch(numbers, target):
    """search by randomly guessing. `numbers` is an array of n numbers"""
    n = len(numbers)

    while True:
        # randomly choose a number between 0 and n-1 in constant time
        guess = np.random.randint(n)
        if numbers[guess] == target:
            return guess
```

In this part, you may assume that the numbers are distinct and that the target is in the array.

Hint: if $0 < b < 1$, then $\sum_{p=1}^{\infty} p \cdot b^{p-1} = \frac{1}{(1-b)^2}$.

Problem 4.

Provide a tight theoretical lower bound for the problems given below. Provide justification for your answer.

a) Given a list of size n containing `True`s and `False`s, determine whether `True` or `False` is more common (or if there is a tie).

b) Given a list of n numbers, all assumed to be integers between 1 and 100, sort them.

c) Given an $\sqrt{n} \times n$ array whose rows are sorted (but whose columns may not be), find the largest overall entry in the array.

For example, the array could look like:

$$\begin{pmatrix} -2 & 4 & 7 & 8 & 10 & 12 & 20 & 21 & 50 \\ -30 & -20 & -10 & 0 & 1 & 2 & 3 & 21 & 23 \\ -10 & -2 & 0 & 2 & 4 & 6 & 30 & 31 & 35 \end{pmatrix}$$

This is an $\sqrt{n} \times n$ array, with $n = 9$ (there are 3 rows and 9 columns). Each row is sorted, but the columns aren't.