DSC 40B - Homework 02

Due: Wednesday, October 15

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.

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. Make sure to show your work by writing out the chain of inequalities to prove each bound, like we did in lecture. Note: please do *not* use limits to prove your answer (though you can use them to check your work).

a) $f(n) = 3n^2 - 8n + 4$

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

c) $f(n) = \begin{cases} n^2, & n \text{ is odd,} \\ 4n^2, & n \text{ is even} \end{cases}$

d) 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 $\Theta(n^{2.5})$.

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

g) $T_6(n) + T_4(n)/T_5(n)$

Problem 3.

In each of the problems below state the best case and worst case time complexities of the given piece of code using asymptotic notation. Note that some algorithms may have the same best case and worst case time complexities. If the best and worst case complexities *are* different, identify which inputs result in the best case and worst case. You do not otherwise need to show your work for this problem.

Example Algorithm: `linear_search` as given in lecture.

Example Solution: Best case: $\Theta(1)$, when the target is the first element of the array. Worst case: $\Theta(n)$, when the target is not in the array.

a)
```
def kth_largest(numbers, k):
    """Finds the k-th largest element in the array.
    `numbers` is an array of n numbers."""
    n = len(numbers)
    for i in range(n):
        count = 0 # Count how many numbers are larger than numbers[i]
        for j in range(n):
            if numbers[j] > numbers[i]:
                count += 1
        if count == k - 1:
            return numbers[i]
```

b)
```
def index_of_kth_largest(numbers, k):
    """`numbers` is an array of size n"""
    # the kth_largest() from above
    element = kth_largest(numbers, k)
    # the linear_search() from lecture 6 slide 11
    return linear_search(numbers, element)
```

Problem 4.

In each of the problems below compute the 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 double_coin_toss(n):
    coin_1 = np.random.rand() > 0.5
    coin_2 = np.random.rand() > 0.5
    if coin_1 and coin_2:
        i = n
        while i > 0:
            print("We got two heads!")
            i -= 1
    else:
        for i in range(n ** 2):
            print("We didn't get two heads.")
```

b)
```
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...')
```

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

    for i in range(x):
        for j in range(i):
            print("Hello!")
```

Hint: In class, we saw that the sum of the first n integers is $\frac{n(n+1)}{2}$. It turns out that the sum of the first n integers squared (so, $1^2 + 2^2 + 3^2 + \ldots + n^2$) is $\frac{n(n+1)(2n+1)}{6}$. You can (and should) use this fact, but make sure to point it out when you do.

Problem 5.

For each problem below, state the largest theoretical lower bound that you can think of and justify (that is, your bound should be "tight"). Provide justification for this lower bound. You do not need to find an algorithm that satisfies this lower bound.

Example: Given an array of size n and a target t, determine the index of t in the array.

Example Solution: $\Omega(n)$, because in the worst case any algorithm must look through all n numbers to verify that the target is not one of them, taking $\Omega(n)$ time.

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.