DSC 40B - Homework 02

Due: Tuesday, 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. Note: please do *not* use limits to prove your answer (though you can use them to check your work).

a) $f(n) = 5n^2 - 10n + 4$

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

c) $f(n) = \begin{cases} n^2, & n \text{ is odd,} \\ 3n^2, & n \text{ is even} \end{cases}$

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:

$$
\begin{aligned}
&T_1(n) = \Theta(n^4) \\
&T_2(n) = O(n^2 \log(n)) \text{ and } T_2(n) = \Omega(n) \\
&T_3(n) = \Omega(n^3) \\
&T_4(n) = O(n^6) \text{ and } T_4 = \Omega(n^2) \\
&T_5(n) = \Theta(n) \\
&T_6(n) = O(n^2) \text{ and } T_6(n) = \Omega(\log(n)) \\
&T_7(n) = \Theta(\log(n))
\end{aligned}
$$

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

Solution: $T_1(n) + T_2(n)$ is $\Theta(n^4)$.

a) $T_1(n) + T_2(n) + T_5(n)$

b) $T_4(n) + T_6(n)$

c) $T_7(n) + T_3(n)$

d) $T_1(n) \cdot T_7(n) \ / \ T_5(n)$

e) $T_1(n) + T_4(n)$

f) $T_2(n) + T_6(n)$

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)
```python
def first_repeated(data):
    """Finds the first repeated element in `data`.
    Returns `None` if no elements are repeated.
    `data` is an array of size n."""
    n = len(data)
    for i in range(n):
        for j in range(i + 1, n):
            if data[i] == data[j]:
                return data[i]
    return None
```

b)
```python
def f_2(data):
    """`data` is an array of n numbers"""
    m = data[0]
    c = 1
    for ix in range(1, len(data)):
        if c == 0:
            m = data[ix]
            c = 1
        elif m == data[ix]:
            c += 1
            if c > len(data) // 2:
                return m
        else:
            c -= 1

    return m
```

c)
```python
def kth_smallest(numbers, k):
    """Finds the k-th smallest 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 less than numbers[i]
        for j in range(n):
            if numbers[j] < numbers[i]:
                count += 1
        if count == k - 1:
            return numbers[i]
```

d)
```python
def index_of_kth_smallest(numbers, k):
    """`numbers` is an array of size n"""
    # the kth_smallest() from above
    element = kth_smallest(numbers)
    # the linear_search() from lecture
    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)
```python
def foo(n):
    # randomly choose a number between 0 and n-1 in constant time
    k = np.random.randint(n)

    if k > n/10:
        for i in range(n**2):
            print(i)
    else:
        for i in range(n):
            print(i)
```

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

    if k > 10: # <--- here is the difference!
        for i in range(n**2):
            print(i)
    else:
        for i in range(n):
            print(i)
```

Note: this code is *almost* the same as the code in the previous part, with one small difference that has been highlighted with a comment.

c)
```python
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 an array of n numbers, find the second largest number.

b) Given an array of n numbers, check to see if there are any duplicates.

c) Given an array of n integers (with $n \geq 2$), check to see if there is a pair of elements that add to be an even number.