DSC 40B - Discussion 02

Problem 1. (Lab Problems)

a) Let $f(n) = (n+1)(n-1) \times 2^{log_2(3^n)}$. Which of the following asymptotic bounds on f is true?

> **Solution:** $\Theta(n^2.3^n)$

b) True or False. If $f_1 = \Theta(g_1(n))$ and $f_2 = \Omega(g_2(n))$ then $\frac{f_1}{f_2} = \Theta(\frac{g_1}{g_2})$

> **Solution:** False.
>
> Try breaking it with a counterexample.
>
> What if $f_1 = n^3$, $f_2 = n^2$, $g_1 = n^3$ and $g_2 = n$? All of the conditions are satisfied, but $\frac{f_1}{f_2} = n$, while $\frac{g_1}{g_2} = n^2$, so $\frac{f_1}{f_2}$ is not $\Theta(\frac{g_1}{g_2})$.

c) What is the worst case time complexity of the following function?

```
def foo(arr):
        ''' arr is an array of size n'''
        for x in arr:
                for y in arr:
                        if (x+y) == 5:
                                return sum(arr)
        return False
```

> **Solution:** $\Theta(n^2)$
> In the worst case, we never find $x + y == 5$, and go through all $\Theta(n^2)$ iterations just to return False at the end.

Problem 2.

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

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

> **Solution:** This question has infinitely many solutions. One such solution is shown below.
>
> Upper Bound : $f(n) \leq c_2 \cdot g(n)$

$$\begin{aligned}\frac{n^3-n^2+n+1000}{(n-1)(n+2)} &\leq \frac{1002n^3}{n^2+n-2} \\ &\leq \frac{1002n^3}{n^2+(n-2)}(n \geq ?) \\ &\leq \frac{1002n^3}{n^2}(n \geq 2) \\ &\leq 1002n\end{aligned}$$

Lower Bound : $c_1 \cdot g(n) \leq f(n)$

$$\begin{aligned}\frac{n^3-n^2+n+1000}{(n-1)(n+2)} &\leq \frac{n^3-n^2}{n^2+n} \\ &\leq \frac{0.5n^3+(0.5n^3-n^2)}{n^2+n}(n \geq ?) \\ &\leq \frac{0.5n^3}{n^2+n}(n \geq 2) \\ &\leq \frac{0.5n^3}{n^2+n^2} \\ &\leq \frac{1}{4}n\end{aligned}$$

We have $n \geq 2$ for both the upper and lower bound. Hence, $N = 2$.
Therefore $f(n) = \theta(n)$ with constants $N = 2, c_1 = 0.25, c_2 = 1002$

Problem 3.

Consider the algorithm below.

```
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
```

We will set up the analysis of the expected time complexity of this algorithm.

a) What are the cases? How many are there?

> **Solution:** Case α occurs when the target is found on iteration α. In principle, the algorithm can run forever, although this is very unlikely (it happens with probability zero). As such, there are infinitely-many cases.

b) What is the probability of case α?

> **Solution:** $P(\alpha)$ is the probability of guessing wrong $\alpha - 1$ times and guessing right on the αth time:
>
> $$(1 - 1/n)^{\alpha-1} \cdot (1/n)$$

c) What is the running time in case α?

> **Solution:** We perform α iterations in case α, each taking constant time, c. The total work in case α is therefore $c\alpha$.

Problem 4.

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

a) Given an array of n numbers, find the sum of the numbers in the array.

> **Solution:** Adding all the elements in the array requires you to visit all the elements at least once. Therefore, the lower bound is $\Omega(n)$.

b) Given a sorted array of $n \geq 2$ numbers, find the second largest number in the array.

> **Solution:** As the array is sorted, we just need to check the second last element in the array to get the second largest number in the array. This takes constant time. Therefore, the lower bound is $\Omega(1)$.