DSC 40B - Discussion 02

Problem 1.

Consider the code below where heights is an array of n elements:

```
for i in range(n):
        for j in range(2*i):
                height = heights[i] + heights[j]
```

What is the time complexity of the code?

Solution:

We can see that outer iteration runs n times, for inner iteration:
On outer iter 1, inner body runs 0 times
On outer iter 2, inner body runs 2 times
On outer iter 3, inner body runs 4 times
Hence,
On outer iter α, inner body runs $(2\alpha - 2)$ times

$$\sum_{\alpha=1}^{n} f(\alpha) = \sum_{\alpha=1}^{n} 2 * \alpha - 2$$

$$\sum_{\alpha=1}^{n} 2 * \alpha - 2 = 0 + 2 + 4 + ... + (2n - 4) + (2n - 2)$$

This is an arithmetic series and we know the formula for sum of an arithmetic series is:

$$Sum = \frac{n}{2} * (a_1 + a_n)$$

Where n is the number of terms, a_1 is the first term in the series and a_n is the last term. Therefore sum of the series:

$$Sum = \frac{n}{2} * (0 + 2n - 2) = n(n-1) = n^2 - n$$

Therefore the time complexity can be given as $\theta(n^2)$.

Problem 2.

Are the following asymptotic bounds true?

1. $n \log n = O(n^2)$
2. $\left(\frac{1}{n}\right)^3 = O(\frac{1}{n})^4$
3. $n^{\frac{1}{2}} = \Omega(n^{\frac{1}{3}})$
4. $e^n = O(n!)$
5. $\log n = \Omega(\sqrt{n})$

Solution:

1. $n \log n = O(n^2)$: True. $\frac{n \log n}{n^2} = \frac{\log n}{n} \to 0$ as $n \to \infty$, so you can use the criterion from slides 6 of lecture 3.

2. $\left(\frac{1}{n}\right)^3 = O\left(\frac{1}{n}\right)^4$: False $\frac{(\frac{1}{n})^3}{O(\frac{1}{n})^4} = n \to \infty$ as $n \to \infty$

3. $n^{\frac{1}{2}} = \Omega(n^{\frac{1}{3}})$: True: $\frac{n^{\frac{1}{2}}}{n^{\frac{1}{3}}} = n^{\frac{1}{6}} \to \infty$ as $n \to \infty$

4. $e^n = O(n!)$ True. An easy way to see it is to write the following

$$\begin{aligned}\frac{e^n}{n!} &= \frac{e \times e \times e \ldots \times e}{1 \times 2 \times 3 \ldots \times n} \\ &= \frac{e}{1} \times \frac{e}{2} \times \frac{e}{3} \times \underbrace{\frac{e}{4}}_{<1} \times \underbrace{\frac{e}{5}}_{<1} \ldots \times \underbrace{\frac{e}{n}}_{<1} \quad \text{(remember that } e < 4) \\ &\leq \frac{e}{1} \times \frac{e}{2} \times \frac{e}{3} = \frac{e^3}{6} \text{ which is a constant}\end{aligned}$$

5. $\log n = \Omega(\sqrt{n})$: False: $\frac{\log n}{\sqrt{n}} \to 0$ as $n \to \infty$

Problem 3.

a) Let $f(n) = 12log_2(3^{n^2-2n} + 2^{logn} - 10n^2 - log_3 n)$. Which of the following asymptotic bounds on f is true?

1. $f(n) = \Omega(1)$
2. $f(n) = O(1)$
3. $f(n) = O(\log n)$
4. $f(n) = \Omega(\log n)$
5. $f(n) = \Theta(\log n)$
6. $f(n) = O(n)$
7. $f(n) = O(e^n)$
8. $f(n) = \Theta(n)$
9. $f(n) = \Theta(n^2)$

Solution: $f(n) = \Theta(n^2)$
it follows from that that:

1. $f(n) = \Omega(1)$ (ie f is bounded below by a constant after some time)
2. $f(n) \neq O(1)$ (ie f is not bounded)
3. $f(n) \neq O(\log n)$
4. $f(n) = \Omega(\log n)$
5. $f(n) \neq \Theta(\log n)$ (because of 3)
6. $f(n) \neq O(n)$
7. $f(n) = O(e^n)$
8. $f(n) \neq \Theta(n)$ (because of 6)
9. $f(n) = \Theta(n^2)$

This question is meant to develop some intuition for finding asymptotic bounds on tricky functions, so we won't write up the formal proof (although it is possible, just difficult algebraically.) We want to start off by finding a simpler expression which is approximately equal to the expression inside the log for large values of n:

$3^{n^2-2n} + 2^{logn} - 10n^2 - log_3 n \approx 3^{n^2-2n}$ as n grows to infinity, since 3^{n^2-2n} is the highest order term. Furthermore, since n^2 grows much faster than $-2n$, $3^{n^2-2n} \approx 3^{n^2}$. So, we have

$f(n) \approx 12log_2(3^{n^2})$.

Applying log rules, we can bring down the exponent to get $12log_2(3^{n^2}) = n^2 \cdot 12log_2(3)$. Since $12log_2(3)$ is a constant with respect to n, we now know that this function is approximately $\theta(n^2)$.

b) What is the best case time complexity of the following function?

```
def foo(arr):
''' arr is a sorted array of size n'''
        i = 0
        j = len(arr) - 1

        while i < j:
                current_sum = arr[i] + arr[j]

                if current_sum == 5:
                        return sum(arr)
                elif current_sum < 5:
                        i += 1
                else:
                        j -= 1

        return False
```

Solution: $\Theta(n)$

There are 2 cases to consider:

1. We reach

   `return sum(arr)`

   at some point in the loop. In that case, the best case time complexity is if we reach it on the first iteration, which gives us $\Theta(n)$.
2. We never reach

   `return sum(arr)`

   . In this case the while loop runs approximately $n/2$ times, so the complexity is still $\Theta(n)$.

Both those cases have the same complexity, so the best-case complexity is $\Theta(n)$

Problem 4.

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 5.

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 $\Omega(1)$.