DSC 190 - Discussion 06

Problem 1.

Consider the following problem: given an $n \times n$ grid of unique numbers and a positive integer k, pick k numbers from the grid so that the sum of the numbers is maximized. Clearly, the greedy strategy of picking the k largest numbers is guaranteed to be optimal.

Argue that *any* optimal solution to this problem must include the largest number in the grid using a proof by contradiction.

Solution: Let m be the largest number in the grid. Suppose $x_1 > x_2 > ... > x_n$ is an optimal solution. and further suppose (for a contradiction) that $x_1 \neq m$. Then m cannot be any one of x_2, x_3, ..., x_n because x_1 is larger than all of them. But then replacing x_1 with m yields a new solution whose total is larger than $x_1 + x_2 + ... + x_n$. Therefore $x_1, x_2, ..., x_n$ is not optimal, which is a contradiction. Thus m must be in the solution.

Problem 2.

The *Fibonacci sequence* is a sequence of numbers where the next number is determined by adding the two previous numbers. The sequence is famous because the ratio of consecutive elements converges to the golden ratio.

The formal definition of the Fibonacci sequence is recursive:

$$x_n = \begin{cases} 1 & \text{if } n \in \{0, 1\} \\ x_{n-1} + x_{n-2} & \text{otherwise} \end{cases}$$

a) As a warmup, write out the first six elements of the sequence, starting with $n = 0$

Solution: 1,1,2,3,5,8

b) Write a recursive function which takes in n and computes the nth Fibonacci number. How large can n be before your function starts to take more than a second or so to finish?

Solution:

```python
def fibonacci(n):
    '''Compute the nth Fibonacci number'''
    if (n==0) or (n==1):
        return 1
    return fibonacci(n-1) + fibonacci(n-2)
```

c) Draw the recursion tree that results from running your function with $n = 5$. Are there overlapping subproblems?

d) Your recursive code for computing the Fibonacci sequence is, in a sense, a *backtracking* algorithm. Modify it by adding a cache to create a top-down dynamic programming solution for computing the nth Fibonacci number.

Solution:

```
def fibonacci_dp(n, cache=None):
    if cache is None:
        cache = [None]*(n+1)

    if(cache[n] is not None):
        return cache[n]

    if (n==0) or (n ==1):
        cache[n] = 1
        return 1

    fib_n = fibonacci_dp(n-1, cache) + fibonacci_dp(n-2, cache)
    cache[n] = fib_n

    return fib_n
```

e) Create a bottom-up, iterative version of your function.

Solution:

```
def fibonacci_bottom_up(n):
    fib_seq = [None]*(n+1)

    fib_seq[0] = 1

    if(n >= 1):
        fib_seq[1] = 1

        for i in range(2, n+1):
            fib_seq[i] = fib_seq[i-1] + fib_seq[i-2]

    return fib_seq[n]
```