Data Structures (810:052) Homework #2 Name:_____

Due: Sept 10 (Friday) at 5 PM

Objective: To practice a recursive dynamic programming memoization implementation.

In lecture 4 we saw a recursive "divide-and-conquer" algorithms for fibonacci. The general idea of divide-and-conquer algorithms is:

- dividing the original problem it into small problem(s)
- solving the smaller problem(s)
- combine the solution(s) to smaller problem(s) to solve the original problem

In lecture 4 we also saw a more efficient *dynamic programming* solution for fibonacci. Dynamic programming solutions eliminate the redundancy of divide-and-conquer algorithms by calculating the solutions to smaller problems first, storing their answers, and looking up their answers if later needed instead of recalculating them. In Python we could use a list to store the answers to smaller problems of the Fibonacci sequence.

To use dynamic programming when you have a recursive statement of the problem. You can do the following steps:
1) Store the solution to smallest problems, i.e., the base cases
2) Loop (no recursion needed) from the base cases up to the biggest problem of interest. On each iteration of the loop we:
   - solve the next bigger problem
   - store its result for later use so we never have to recalculate it

One tradeoff of simple dynamic programming implementations is that they can require more memory since we store solutions to **all** smaller problems. Often, we can reduce the amount of storage needed if the next larger problem (and all the larger problems) don't really need the solution to the really small problems, but just the larger of the smaller problems. In fibonacci, when calculating the next value in the sequence we only need the previous two solution. In lecture 4, we reimplement the Fib(n) function a third time to eliminate the list fibonacci and replace it with three variables.

Sometimes a solution to the ("largest") problem of interest does not require the solution to all of the smaller problems. We'd like to calculate only the solutions to smaller problems that we actually need for the problem of interest. One problem is predicting which smaller problems will be needed. One solution to this is *memoization* where you write a recursive solution like divide-and-conquer because the recursive solution starts with the largest problem and only calls smaller problems that are actually needed to solve it. To avoid the divide-and-conquer problem of recalculating smaller problems multiple times, we'll check if we previously calculated a solution to a smaller problem before doing the recursive call.

Your assignment for homework #2 is to write a memoization solution to fibonacci. The general steps for using memoization are:
1) initialize a list (/array) of **all** smaller problem solutions to some dumby value like -1 to indicate that these problems have not yet been solved.
2) in this list set the base case solutions
3) write a recursive version like divide-and-conquer, except before doing the recursive call(s) check if your list already has an answer. If it does, use the list answer instead.

You will need to complete fib_helper recursive function in the below box, and implement.

```
def fib(n):
    # Step 1 and 2:  initialize solution list and set base cases
    fibonacci = []
    fibonacci.append(0)
    fibonacci.append(1)
    for i in range(2,n+1):
        fibonacci.append(-1)

    # Step 3:
    fib_helper(n, fibonacci)

    return fibonacci[n]

def fib_helper(n, fibonacci):
```