Homework Assignment 03

Name: **Asrar Syed**
Due by: **October 29, 2025**
Course Section: **CSC 4520-006**

PROBLEM 1. [40 POINTS]

For this problem I used a Dynamic Programming with Tabulation (bottom-up) approach where I started from the base case i.e. the smallest subproblem and we work to bigger problems.

In this problem the smallest subproblems is zero. We then loop everything after zero to n+1. We set the index of the answer array with a recurrence relation.

The recurrence relation consists of bitwise operations.

k >> 1 shifts the bits of k to the right by 1 position
Know: k >> 1 is equivalent to k // 2 for positive integers.

i & 1 checks only the least significant bit of i and does a comparison check.

PROBLEM 2. [30 POINTS]

For this problem I used a Dynamic Programming (bottom-up) approach where I started from the base case i.e. the smallest subproblem and we work to bigger problems.

The base cases being
n = 1: Sequences = a, b
n = 2: Sequences = aa, ab, ba

I do a couple if – else checks for the base case.

I then start building a for-loop to loop everything after three to n+1. I have the base cases set to their respective value and use them to build up to the final answer.

PROBLEM 3. [30 POINTS]

This was solved using a greedy approach, but we chose the smallest values after sorting them in ascending order right before it crossed the maximum value set.

Obviously, we need to sort the list and there were no sorting requirements I just used pythons default sorting algorithm i.e. Timsort because it has a worst case of O(n log n) and a best case of O(n).

The for-loop after is basic and does the math – has a O(n) so irrelevant to the sort() function.

PROBLEM 4. [20 POINTS]

I create two functions i.e. a dynamic programming approach and a fast-doubling approach.

In the dynamic programming using memorization or tabulation is irrelevant and bad for very large Fibonacci numbers because it uses to much space if we use a tabulation bottom-up approach and its slow. If we use a top-down approach which I tried using memorization we a hit pythons recursion limit (1000) which is far below the desired 2,000,000-th Fibonacci number.

My code just uses a bottom-up approach from the base case to our goal. But we only save the last two numbers, so we save on space-cost. But it's still slow with O(n) time.

In the fast-doubling method, I used the Fibonacci formulas and bitwise operations (>> and &) to make the function efficient. This method is much faster because it uses a Divide-and-Conquer approach, splitting the problem in half each time.

The base case if k == 0 tells the recursion when to stop and also provides the starting values. The second if statement (if k & 1) checks whether the current index is odd or even so the correct fast-doubling formula can be applied.

Finally, the helper function returns a tuple (F_(k), F_(k+1)), which lets each recursive call build the next pair of Fibonacci numbers. This gives a time complexity of O(log n).