CMPS 2200 Recitation 09

Today we'll learn more about dynamic programming using the classic problem of computing **Fibonacci Numbers**. As you know, for the nth Fibonacci number $F_n = F_{n-1} + F_{n-2}$. Here are the first 11 values:

0	1	2	3	4	5	6	7	8	9	10
0	1	1	2	3	5	8	13	21	34	55

1 Implement a recursive solution by completing `fib_recursive` and test it with `test_fib_recursive`. In addition to n, we also use an array called `counts` that keeps track of how many times each F_i is computed when computing F_n.

.
.
.

2 Write a recurrence for the **work** of this algorithm and solve it. Assume the input is n to compute F_n.

put answer in answers.md

.
.
.

3 Write a recurrence for the **span** of this algorithm and solve it.

put answer in answers.md

.
.
.

4 Inspecting the `counts` list, what interesting pattern emerges?

put answer in answers.md

.
.
.

Clearly, this implementation does a ridiculous amount of duplicate work. We should really only have to compute each F_i one time, for $i \leq n$. We'll next write two more efficient ways of computing F_n.

5 For the first efficient way, we'll implement it recursively as before except that we will also keep an additional list called `fibs`, where `fibs[i] = F_i`, to store each value we generate during the recursive solution. When the function is called for input i, we first check if F_i is in `fibs`. If so, we simply return it. Otherwise, we proceed with the recursive calls. Note that we initialize `fibs` with -1's so we can tell if F_i has been computed or not. Complete `fib_top_down` and test with `test_fib_top_down`.

.
.
.

6 When computing F_n, what is the maximum number of times that `fib_top_down(i)` will be called for any value i? Based on this, what is the **work** and **span** of this algorithm?

put answer in answers.md

.
.
.

7 Finally, we will compute a bottom-up implementation. This is a non-recursive solution that starts at F_0 and iteratively computes subsequent values of F_i until F_n is reached. To do so, store a list of $n + 1$ values, initialized to 0's, which will store the Fibonacci sequence up from F_0 to F_n. Write a for loop to fill it in, then return the last value. Complete `fib_bottom_up` and test with `test_fib_bottom_up`.

.
.
.

8 When computing F_n, what is the maximum number of times that F_i will be read from the list for any value i? Based on this, what is the **work** and **span** of `fib_bottom_up`?

put answer in answers.md

.
.
.

`fib_top_down` and `fib_bottom_up` are two simple forms of dynamic programming. Both improve over the recursive solution by **sharing** solutions to smaller problem instances in order to reduce duplicate work. `fib_top_down` does this by starting with the original problem and caching solutions to smaller problems encountered in a recursive solution. In contrast, `fib_bottom_up` creates a table of solutions to smaller problem instances, and solves them from smallest to largest. In class and in the next assignment, we will see more complicated dynamic programming solutions where the relationship between smaller and larger problem instances is more interesting.