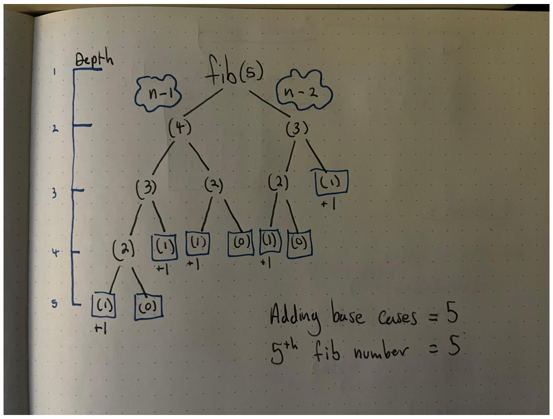
1. **Explain how the larger problem of solving for the nth Fibonacci number, can be solved by finding the solutions to it's smaller, subproblems.**

The base cases are important because they provide a “stopping point” that tells the program when to add values. Adding each base case results in calculating the nth fibonacci number.

2. **What are the base case (s)? In this problem, why do they represent the smallest subproblem(s) we want to recurse down too?**

The base cases for this problem are 0 and 1. These represent the lowest possible values of n. In the recursive “naive” solution, the tree is split at each stage until n is either 0 or 1. In either case, each split is added together and returned up the recursive call.

3. **Draw the recursive tree for Solution #1 (recursive, non-dynamic), up to a depth of 5. Identify the overlapping subproblems.**



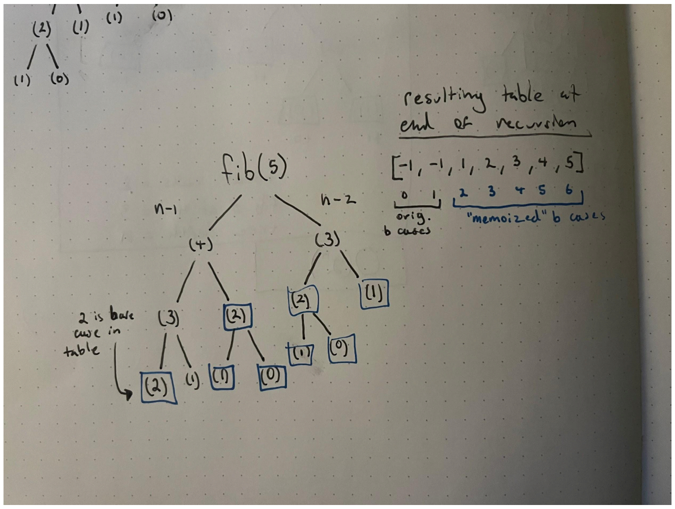
Adding base cases = 5
5^{th} fib number = 5

Notice the trees for 2 and 3 repeat. The values for these trees can be recorded and used as base cases for future recursive calls. Note that the base cases are NOT saved for the non-dynamic solution.

4. **Why is Solution #1, the non-dynamic recursive solution, so slow in terms of runtime efficiency?**

The naive approach is so slow because the tree splits 2^n times. Until n <= 1, the tree continues to split. As n increases, the runtime becomes extremely slow. For example, when I tried n = 50, it took so long I lost patience and quit the program before it could finish.

5. **Draw one possible recursive tree for Solution #2 (recursive, Memoized solution), up to a depth of 5.**



(Sorry! I forgot to add one more level! There should be another tree splitting from the last 2 on the left.)

6. **With Solution #2, the dynamic programming recursive implementation, how does Memoization work to improve the runtime efficiency?**

Memoization works to improve the runtime efficiency by recording new base cases as recursive calls are made. I made a note of the final table, which represents the value of each case corresponding to the n value by index. So, when n is 2, its value of 1 is stored in the table at index 2.