**Time and Space Complexity**

Finding the time and space complexity of a dynamic programming algorithm may sound like a daunting task. However, this task is usually not as difficult as it sounds. Furthermore, justifying the time and space complexity in an explanation is relatively simple as well. One of the main points with DP is that we never repeat calculations, whether by tabulation or memoization, we only compute a state once. Because of this, the time complexity of a DP algorithm is directly tied to the number of possible states.

If computing each state requires F time, and there are n*n* possible states, then the time complexity of a DP algorithm is  *O*(*n*⋅*F*). With all the problems we have looked at so far, computing a state has just been using a recurrence relation equation, which is  *O*(1). Therefore, the time complexity has just been equal to the number of states. To find the number of states, look at each of your state variables, compute the number of values each one can represent, and then multiply all these numbers together.

Let's say we had 3 state variables: i, k, and holding for some made up problem. i is an integer used to keep track of an index for an input array nums, k is an integer given in the input which represents the maximum actions we can do, and holding is a boolean variable. What will the time complexity be for a DP algorithm that solves this problem? Let n = nums.length and K be the maximum actions possible given in the input. i can be from 0 to nums.length,  k can be from 0 to K, and holding can be true or false. Therefore, there are n⋅K⋅2 states. If computing each state is *O*(1), then the time complexity will be *O*(*n*⋅*K*⋅2)=*O*(*n*⋅*K*).

Whenever we compute a state, we also store it so that we can refer to it in the future. In bottom-up, we tabulate the results, and in top-down, states are memoized. Since we store states, the space complexity is equal to the number of states. That means that in problems where calculating a state is *O*(1), the time and space complexity are the same. In many DP problems, there are optimizations that can improve both complexities - we'll talk about this later.

The three main components of the framework are:

The choice between bottom-up and top-down

The base cases

The recurrence relation

The number of dimensions

The time complexity

A data structure and a way of visiting each DP state

Commit these to memory! Understanding the components of the framework allows us to approach dynamic programming problems in a step by step way.

Generally, the time and space complexity of dynamic programming algorithms is directly related to:

Whether the algorithm is top-down or bottom-up

The number of possible states

Whether an array or hashtable is used for caching results

In many cases, the space complexity is the number of states, and the time complexity is the number of states multiplied by the cost of calculating a state.