1. ***Climbing Stairs -*** You are climbing a staircase. It takes n steps to reach the top of the stairs. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

    Example 1:

        Input: 2
        Output: 2
        Explanation: There are two ways to climb to the top.
            1. 1 step + 1 step
            2. 2 steps

    Example 2:

        Input: 3
        Output: 3
        Explanation: There are three ways to climb to the top.
            1. 1 step + 1 step + 1 step
            2. 1 step + 2 steps
            3. 2 steps + 1 step

    a. Describe the structure of an optimal solution, as defined in CLRS, and use a cut and paste argument to show that the problem has optimal substructure.

b. Create a recursive algorithm that solves the problem. Don't worry about creating a memo table to efficiently solve the algorithm.

c. Modify the algorithm that you created in part b to utilize a memo table that helps solve the problem more efficiently. You should only need to add and replace a few lines of code.

2. Let's say that you're given two strings, S_1 and S_2. You can perform the following 3 operations anywhere on S_1 : add a character, delete a character, and replace a character . What is the least number of operations it would take to transform S_1 into S_2 ?

   For example: S_1 = "horse" and S_2 = "ros"
   Replace "h" with "r": "rorse"
   Delete "r" : "rose"
   Delete "e" : "ros"

   a. Describe the structure of an optimal solution, as defined in CLRS, and use a cut and paste argument to show that the problem has optimal substructure.

   b. Create a recursive algorithm that solves the problem. Don't worry about creating a memo table to efficiently solve the algorithm. [*Hint: I) Break the problem down into smaller subproblems. II) What effect does each operation have on the string, and more importantly, on the values that you are checking after?]*

   c. The table below is the start of an example memo table that will eventually be created for this problem. Each cell contains two strings, representing their respective strings. As you travel to the right of the x axis, the amount of letters in the first string increases, and as you travel down the y axis, the amount of letters in the second string increase (it's represented in this way since this is often how matrices are represented in computer science). Complete the table.

"" ""	"h" ""	"ho" ""	"hor" ""	"hors" ""	"horse" ""
"" "r"	"h" "r"				
"" "ro"					
"" "ros"					"horse" "ros"

   d. We want to find how many operations it would take to convert the first string into the second one, so the table below is meant to represent this. For the top most row and left most column, it is clearly trivial. However, it becomes less clear as the string size increases. Start at the top left most empty cell (labeled in green) in the table. Then, fill out the table below using the following algorithm:

Starting at the top left most empty cell, work your way left to right, top to bottom. If the last character in each string (of the corresponding table above) is the same, then copy the value that is at the top left corner of the cell. Otherwise, find the smallest value among the values directly above, to the left, and top left corner of the cell you're at, increment the value by one, then copy it into the cell.

0	1	2	3	4	5
1	1				
2					
3					

e. Modify the algorithm that you created in part b to utilize a memo table that helps solve the problem more efficiently. You should only need to add and replace a few lines of code. [*Hint: Think about how the indices of the matrix. How does that correspond to the given string?*]

f. Based on the example memo table that you created and the memoized solution that you created in part e, write a bottom up dynamic programming algorithm.

3. Let's say that you're given a dictionary of words that you can look up in constant time and a string of characters. Find if there exists some way that you could partition the string such that each partition exists in the dictionary.

Dictionary: {"ab", "c"}

- Input: "abc"
  Output: True

  Explanation: You can break the string into the partition: "ab" and "c"

- Input: "ababcab"
  Output: True

  Explanation: You can break the string into the partition: "ab" "ab" "c" "ab"

- Input: "bac"
  Output: False

  Explanation: You can't break the string into any partition (without changing the order of the characters) and have each partition exist in the dictionary.

a. Describe the structure of an optimal solution, as defined in CLRS, and use a cut and paste argument to show that the problem has optimal substructure.

b. Create a recursive algorithm that solves the problem. Don't worry about creating a memo table to efficiently solve the algorithm. [*Note: Assume that you have some function CheckDict*(*dictionary*, *key*) *such that it returns True if it exists in the dictionary, and false otherwise]*

c. Modify the algorithm that you created in part b to utilize a memo table that helps solve the problem more efficiently. You should only need to add and replace a few lines of code.

d. Based on the example memo table that you created and the memoized solution that you created in part c, write a bottom up dynamic programming algorithm.