Lab 8. Introduction to algorithms

This is eight homework for CS 566.

Task 1. Solve the problem "Coin Change" from https://leetcode.com/problems/coin-change/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
import sys
from typing import List

class Solution:
    def coinChange(self, coins: List[int], amount: int) -> int:
        MAX = amount+1
        dp = [MAX] * MAX
        dp[0] = 0

        for a in range(1, amount+1):
            for c in coins:
                if a-c>=0:
                    dp[a] = min(dp[a],1+dp[a-c])
        # print(dp)

        if dp[amount] != amount+1:
            return dp[amount]
        else:
            return -1
```

Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

```
#test_case_1
coins = [1,2,5]
amount = 11
actual = Solution().coinChange(coins, amount)
expected = 3
assert actual==expected, "Mistake in test case 1"
print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Aymptotic Notation O. (1 point)

Memory Analysis: O(m), where n is number of coins and m is amount

Time Analysis: O(n*m), where n is number of coins and m is amount

Task 2. Solve the problem "Longest Increasing Subsequence" from https://leetcode.com/problems/longest-increasing-subsequence/description/ using Python3.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

```
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        dp = [1] * len(nums)
        for i in range(1, len(nums)):
            for j in range(0, i):
                if nums[j] < nums[i]:
                    dp[i] = max(dp[i], 1+dp[j])
        return max(dp)
```

```
nums = [10,9,2,5,3,7,101,18]
expected = 4
actual = Solution().lengthOfLIS(nums)
assert expected == actual, "Mistake in test case 1"
print("OK")
```

```
OK
```

Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O(n)

Time Analysis: O(n^2)

Theoretical Question.

Design $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence (LIS) of a sequence of n numbers. Use the convention from the textbook:

.

- Define DP State
- Define Transition
- Define Initialization

Prove that your complexity is indeed $O(n^2)$

DP State:

We have array dp, where $dp[i]$ represents the length of longest increasing subsequence that ends with the element at index i in the sequence.

Transition:

Use nested for loop, where outer for loop i goes through 2nd element to end and inner loop j goes through all previous elements before i.

If the current element $numbers[i]$ is greater than a previous element $numbers[j]$, we can extend the increasing subsequence ending at j by including i. In other words, if $numbers[j] < numbers[i]$, then $dp[i] = max(dp[i], dp[j] + 1)$.

Initialization:

dp = [1] * n

Algorithm

See above problem 2

Proof of $O(n^2)$

The outer loop runs n times, and the inner loop runs at most n-1 times. Hence worst case of running n*(n-1) times, we get O(n^2) time complexity.