EL9343 Homework 10

1. **A robot is located at the top-left corner of a m x n grid (marked 'Start' in the diagram below).The robot canonly move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid (marked 'Finish' in the diagram below).How many possible unique paths are there?**

The algorithm of this problem woule be like this:

```
def uniquePaths(self, m: int, n: int) -> int:
    dp = [[1 for i in range(n)] for j in range(m) ]
    for i in range(1,m):
        for j in range(1,n):

            dp[i][j] = dp[i][j-1] + dp[i-1][j]
    return dp[m - 1][n - 1]
```

however, in the mathematic way of find the solution in this problem, we are going to get the solution :

how many possible unique paths (m+n−2)! // (m - 1)! * (n - 1)!

2. Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem and argue that your algorithm is correct.

Algorithm:

If we take an item with v 1v1, w 1w1, and remove an item with v 2v2, w 2w2, and w 1 > w 2w1>w2, v 1 v 2v1v2, and w 1 > w 2w1>w2, v 1 v 2v1v2, we may replace 11 with 22 and obtain a better solution. As a result, we should always select the most valuable products.

3. Design a greedy algorithm for making change consisting of quarters, dimes, nickels, and pennies. (a)Provide the pseudocode. (b)Write down the running time of your algorithm

(a)

```
Result = {}
Coins = [25, 10, 5, 1]
Face_value

For i in range(coins):

        While face_value > coins[i]:
                Face_value -= coins[i]
                Result[coins[i]] += 1


Return result
```

(b)
The time complexity would be O(N)

4. The Fibonacci numbers are defifined by recurrence CLRS(3.22). Give an O(n)-time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the subproblemgraphfor n=4. How many vertices and edges are in the graph?

```
def fib(self, N: int) -> int:
        if N <= 1:
            return N

        cache = [0] * (N + 1)
        cache[1] = 1
        for i in range(2, N + 1):
            cache[i] = cache[i - 1] + cache[i - 2]

        return cache[N]
```

there are 9 vertices and 8 edges in the graph.

Here is the graph: