Problem Solution - ALG1

Problem Definition:

We are given a array of price predictions for m stocks for n consecutive days. The price of stock i for day j is A[i][j] for i = 1,...,m and j = 1,...,n. You are tasked with finding the maximum possible profit by buying and selling stocks. The predicted price at any day will always be a non-negative integer. You can hold only one share of one stock at a time. You are allowed to buy a stock on the same day you sell another stock. More formally,

Problem 1: Given a matrix A of m × n integers (non-negative) representing the predicted prices of m stocks for n days, find a single transaction (buy and sell) that gives maximum profit.

Problem 2: Given a matrix A of m × n integers (non-negative) representing the predicted prices of m stocks for n days and an integer k (positive), find a sequence of at most k transactions that gives maximum profit. [Hint :- Try to solve for k = 2 first and then expand that solution.]

Task:

Algorithm1: Design a $\Theta(m * n^2)$ time brute force algorithm for solving Problem 1.

Solution:

For the brute force algorithm, the approach is the iterate over all the stocks and find the maximum approach taking into account all the possible buy and sell permutation. The pair for each stock j = 1, …, m will be (a, b) where a = 1, … , n - 1 and b = a + 1, … , n.

Proof of correctness:

This algorithm checks all the possible pairs for each stock and find the maximum out of all those pairs. Hence, this algorithm will always find the correct desired output and find the maximum possible profit always. Brute force algorithms are always finds the correct solution.

Time complexity analysis:

Time complexity for the given task can be calculated as follows: First iterating over all the stocks i.e. m. Then for all the pairs for that stock in n days will be from a = 1 to n-1 and from b = a+1 to n. So, for each stock on day 1 there will be n - 1 pairs for day 2 there will be n - 2 pairs and so on. Hence there will be (n-1) + (n-2) + … + 1 pairs for each stock i.e. (n-1)(n) / 2 pairs. Hence there will be m * (n(n-1)/2) pairs.

So the time complexity for this algorithm will be: $O(m * (n^2 - n) / 2)$ = **$O(m * n^2)$**

Space Complexity analysis:

The algorithm stores the stocks and their prices in a 2d array of size m*n. It also stores the least price, profit, buy and sell as a integer.

So the space complexity for this algorithm will be: O (m * n + c) = **O (m * n)**
[Here c is a constant]

Pseudo Code:

```
Step 1 : Input A[m][n]

Step 2 : profit = 0, stock = 0, buy = -1, sell = -1

Step 3 : For I = 1 to m :

                For j = 1 to n - 1:

                    For k = j + 1 to n:

                    if (A[i][k] - A[i][j] > profit)

                        profit = A[i][k] - A[I][j]

                        stock = i

                        buy = j

                        sell = k

Step 4: Return {Profit, stock, buy, sell}
```