**Problem Solution - ALG3**

**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:**

Algorithm3: Design a Θ(m ∗ n) time dynamic programming algorithm for solving Problem 1.

**Solution:**

For the dynamic programming, for each stock, it checks if the minimum price yet is greater then the current price of stock then it sets minimum price as the current price. Then it calculates the profit by selling the stock at current price after buying it at the minimum price. It checks if the current profit is greater than previous profit then it sets the max profit to current profit and calls the next iteration with the new minimum price or new max profit. So it basically finds the maximum profit for each stock and outputs the maximum out of those.

Def: OPT(j) - Maximum profit for astock for days 1 to j.

Goal: Max{ OPTi (n) } where i = 1 to m

Case 1: OPT(j) sells the stock on day j

- Current profit will be price of stock on day j minus the buying price

- If current profit is greater than the max profit so far, set max profit to current profit

Case 2: OPT(j) buys the stock on day j

- Set buying price as the price on the stock on day j

Case 3: OPT(j) does not buy or sell the stock on day j

- Current profit must be less than the max profit made from so far.

Bellman Equation:

OPT(j) = 0 if j = 0

= max(max(current profit, max profit so far), OPT(j - 1)) if j > 0

**Proof of correctness:**

**Invariant:** OPT(j) gives maximum profit from the stocks after j days.

**Pf. :** Proof by induction

**Base Case:** OPT(0), OPT(1) return 0

**Next Case:** OPT(2) = max{OPT(1) which is 0 and since n is 2 there are just 2 days for all stocks and hence just the maximum difference between second day and first day for all stock will be the maximum profit}

**Inductive Hypothesis:** Assuming OPT(j) gives maximum profit.

For OPT(j + 1):

* If the price of the stock is less the current buying price it will be updated.
* If the current price - buying price is less than the max profit so far it will return same as the last iteration else it will update the maximum profit and selling date.

Hence for any stock on any day j it will give the maximum profit.

**Time complexity analysis:**

Time complexity for the given task can be calculated as follows: First iterating over all the stocks i.e. m. For each stock, it calculates new buying price or the maximum profit buy doing at most 3 operations and calculates the maximum profit till the current day for each day n.

So the time complexity for this algorithm will be: **O (m \* n)**

**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.

For memoization: It uses 2 extra arrays of size n for each stock for storing the minimum prices till the day and the index of the minimum price. So the space complexity for the algorithm will be:

O (m \* n \* n \* n + c) = **O (m \* n3)** [Here c is a constant]

For top down approach: It doesn’t use any extra memory except storing the minimum price so fas and the maximum profit so far. So the space complexity for this algorithm will be:

O (m \* n + c) = **O (m \* n)** [Here c is a constant]

**Pseudo Code:**

**Bottom-up Approach:**

**OPT(A[], index, profit, minPrice)**

**Step1:** if A[index] < minPrice

minPrice = A[index]

**Step 2:** currentProfit = A[index] - minPrice

**Step 3:** if currentProfit > profit

profit = currentProfit

**Step 4:** return max{ profit, OPT(A, index - 1, profit, minPrice) }

**Memoization:**

**OPT(A, n)**

**Step 1:** dp[n]**,** buyIndex[n], profit = 0

**Step 2:** For j = 1 to n :

if A[j] < dp[j - 1]

dp[j] = A[j]

buyIndex[j] = j

else

dp[j] = dp[j - 1]

buyIndex[j] = buyIndex[j - 1]

**Step 3:** For j = 1 to n :

if profit < (A[j] - dp[j])

profit = A[j] - dp[j]

**Step 4:** return profit

**MAIN()**

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

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

**Step 3 :** For i = 1 to m :

result = OPT(A[i], n, profit, INT\_MAX) // *BOTTOM UP*

result = OPT(A[i], n) // *MEMOIZATION*

if result > profit

profit = result

**Step 4:** Return profit