**Problem Solution - ALG5**

**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:**

Algorithm5: Design a Θ(m ∗ n2 \* k) time dynamic programming algorithm for solving Problem2.

**Solution:**

For the dynamic programming,

Def: OPT[k][i] - Maximum profit in i days after k transactions.

Goal: OPT[K][N] -> return K tuples of transactions.

Case 1: OPT[i][j] base case

Case 2: OPT[i][j] Number of transaction is greater than number of days.

- Number of transactions cannot be greater than days so just return the pro

Case 3: OPT[i][j] Number of transaction is less than number of days.

- Now we have two choices to pick up from. First pick up the profit from previous day i.e no transactions on the current day

- Second, if the transaction occur on current day. To find that out we find the price of the all the stocks on current day, and check which stock gives the maximum profit if we buy the stock on any of the previous days.

**Bellman Equation:**

OPT[i][j] = 0 if i = 0, j = 0

= OPT[i - 1][j] if i > j

= Max{ OPT[i][j - 1], Max{ For all stocks from 0 to m (Max { For k from 0 to j-1: OPT[i-1][j-1] - A[i][k] } + A[i][j] } } if i <= j

**Proof of correctness:**

**Invariant:** OPT[i][j] gives maximum profit.

**Pf:** Proof by induction.

Base case: i = 1, j = 1 is easy Return 0

Next case: i = 1, j = 2 has 2 days for all days so just return the max of the difference between second day and first day.

Next case: i = 2, j = 2 return Max{ OPT[i][j - 1], Max{ all the stock transactions since there will only be one transaction per stock, excluding the previous one } }

Inductive Hypothesis: Assuming OPT[i][j] gives maximum profit after i transactions.

* If i + 1 > j. Number of transactions cannot be more than number or days so just return the maximum profit after i transactions. So it will return the maximum profit
* If i <= j:
  + Let profit by selling any one of the on jth  day is greater than the profit of j - 1 day. Then OPT will return that value.
  + If profit by selling is less, the OPT will return the Opt[i][j - 1] value.

**Time complexity analysis:**

Time complexity for the given task can be calculated as follows: First going from transactions i = 0 to k, then iterating over j = 0 to n days, for each stock we find the maximum profit from k = 0 to j - 1 days to find out the maximum profit for that day after i transactions.

So the time complexity for this algorithm will be: **O (m \* n2 \* k)**

**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.

Storing the OPT in a 2d array of k + 1 length and width n having a tuple of 3 integers as the each element. And the result will be stored in a 2 d array of max length k. And some iterating variables for the loops.

So the time complexity for the implementation will be:  **O ( n \* (m + k) )**

**Pseudo Code:**

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

**Step 2:** Initialize OPT[k+1][n]

**Step 3:** for i = 1 to k+1

for j = 1 to n

if( j > i)

OPT[i][j] = OPT[i-1][j]

continue to next iteration

maxProfit = OPT[I][j-1][profit]

for p = 0 to m

diff = INT\_MIN, index = 0

for l = 0 to j

currentDiff = OPT[i - 1][l][profit] - A[p][l]

if currentDiff > diff

diff = currentDiff

index = l

profit = A[p][j] + diff

if maxProfit < profit

OPT[i][j][profit] = profit

OPT[i][j][stock] = p

OPT[I][j][buy] = buy

// BACTRACKING

**Step 5:** Initialize i = k + 1, j = n

**Step 4:** while i != 1 and j != 1

if i == 1

j = j -1

else if j == 1

i = i - 1

else if OPT[i][j][profit] == OPT[I][j-1][profit]

j = j - 1

else

Add the transaction OPT[i][j] to result

**Step 6:** Return result