**Problem Solution – ALG4**

**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:**

Algorithm 4:Design a Θ(m\*n^2k) time brute force algorithm for solving problem 2.

**Solution:**

For this task, we need to design a brute force algorithm. And as we know in brute force algorithm we will try all possible pairs and output those transactions that will result into maximum profit. First we will compute all possible pairs of transaction which for n days will be n \* (n-1) / 2 i.e. order of n2. We will compute n2 pairs in each stock and we will also keep track of previous stocks transactions so that we can filter out the most profitable transactions among the two. By filter we will see if in each transaction with same buy and sell date we will keep the most profitable one. Now all that’s left is to pick k possible pairs from this n2 transactions.

**Pseudo Code:**

PickKtransactions(transactions, index, Solution = ϕ, k) {

if k is zero or index is length of transactions then → Return Solution

current → transaction[index];

if current is Compatible with Solution then{

Solution1 → PickKtransactions(transactions, index+1, Solution U {current}, k-1)

Solution2 → PickKtransactions(transactions, index+1, Solution, k)

if profit of Solution1 is greater than Solution2 then → Return Solution1

else → Return Solution2

}

else return PickKtransactions(transactions, index+1, Solution, k)

}

BruteForce(Stocks S, Days D, Max Transactions k)

{

previousTransactions → ϕ

for i → 1 to S:

transaction → Get All transactions(i)

transactions[i] → filter\_keepMostProfitableOnes(transaction, previousTransactions)

update previousTransactions

maxprofit → ϕ

result → ϕ

for i → 1 to S:

r → GetKtransactions(transactions[i], 1, ϕ, k)

if profit or r > maxprofit then {

result → r

maxprofit → profit of r

}

return result

}

**Proof of Correctness:**

In this algorithm we try all pairs possible for m stocks and then we pick only those `atmost k` pairs with which result into maximum profit and are compatible with each other. Hence, as we tried all pairs and using comparison we kept the most profitable ones, we can say that the algorithm outputs at most k pairs with maximum profit.

**Time Complexity analysis:**

We can see that Get All transactions will result into time complexity of order of n2 also filtering and keeping the most profitable ones will also have a time complexity of order of n2, as we will just iterate and compare those two, and updating previous will result in constant time complexity. Now this results in time complexity of m\*n2, As we iterate over m stocks. Now, as for time complexity of GetKtransactions, we know that we are picking atmost k pairs from n2 choices, which means it tries all n2Ck pairs which reduces its time complexity to n2k, and since we are iterating over m stocks, the time complexity is of order of Θ(m\*n2k).

**Space Complexity analysis:**

`transactions` take a space in order of m\*n2 as we try all possible pairs, Now, PickKtransactions will try upto n2k pairs but since we pass the solution to the function and we have the upper bound of k transaction on solution it will have space complexity in order of k. Hence this gives us an upper bound of Θ(m\*n2) space.