Milestone 2: Algorithm Design

# **Milestone 1: Understanding the problems**

**GitHub Repository Link:** <https://github.com/Angellsh/COP4533-Group-13.git>

**Team members:**

‣ *Name:* Loubna Benchakouk  
 *Email:* l.benchakouk@ufl.edu  
 *GitHub Username:* loubnaB023

‣ *Name:* Anhelina Liashynska  
 *Email:* aliashynska@ufl.edu  
 *GitHub Username*: Angellsh

‣ *Name:* Jacob Ramos *Email:* jacob.ramos@ufl.edu *GitHub Username:* JacobR678

‣ *Name:* Dani Brown *Email:* brown.d@ufl.edu *GitHub Username:* danBrownGithub

**Member Roles:**

* **Jacob Ramos**: Problem 1 & 2 completed both tasks (100% progress)
* **Loubna Benchakouk**: Problem 3 completed (100% progress)
* **Anhelina Liashynska**: GitHub setup
* **Dani Brown**: Gantt Chart
* Roles for upcoming milestones will be assigned later

**Communication methods:** We use Discord for regular communication and Google Docs for document collaboration.

**Project Gantt Chart:**

# **Problem 1**

We are given a matrix of stock prices where each row represents a different stock and each column will represent a different day. We calculate the maximum potential profit for each specific stock using a 1-based index.

Each stock/day combination maximum profit: (1,2,5,15) (2,1,3,9) (3,1,2,2) (4,2,5,7)

Answer: Stock/Day Combination Maximum Profit: (1,2,5,15)

Explanation:

* Stock 1 yields the maximum when bought on the 2nd day and sold on the 5th day for a profit of 15.
* Stock 2 yields the maximum profit when bought on day 1 and sold on day 3 yielding a profit of 9.
* Stock 3 yields the maximum potential profit when bought on day 1 and sold on day 2 yielding a profit of 2.
* Finally, stock 4 yields the maximum potential profit when bought on day 2 and sold on day 5 yielding a profit of 7.

# **Problem 2**

We are given a matrix where each row represents a different stock and each column will represent a different day. We are given an integer k which will represent the maximum number of non-overlapping transactions permitted, in this case k = 3. For each transaction we must buy and sell one stock.

Answer: (4,1,2), (2,2,3), (1,3,5) total profit = 90

Explanation:

1. Stock 4: Buy on the 1st day at price 5, sell on the 2nd day at price 50 for a profit of 45.
2. Stock 2: Buy on the 2nd day at price 20, sell on the 3rd day at price 30 for a profit of 10.
3. Stock 1: Buy on the 3rd day at price 15, sell on the 5th day at price 50 for a profit of 35.
4. Total profit = 45 + 10 + 35 = 90

# **Problem 3**

**Problem Statement**

We are given a matrix where each row represents a different stock and each column will represent a different day. Additionally, we are given an integer c which will represent a cooldown period where we cannot buy any stock for c days after selling any stock. If a stock is sold on day i, the next stock will not be eligible for purchase until day i + c + 1. For this example, c = 2.

Answer: (3,1,3), (3,6,7) total profit = 4 + 7 = 11

Explanation:

1. First transaction we buy stock 3 on day 1 and sell on day 3 for a profit of 4
2. Since the stock was sold on day 3 we cannot purchase another stock till day 6
3. On day 6, we buy stock 3 again and sell on day 7 for a profit of 7
4. The total profit is 11

**Transaction rules:**

1. We can only buy before we sell, and only once per transaction.

2. Resting period: after we sell on day j2 we need to wait until (j2+c+1) day to buy.

3. We can perform multiple transactions on any stock while following the cooldown rule.

4. Main objective is to maximize the total profit across all valid transactions.

**Input:**

We have a matrix A where each:

Row = one stock

Column = one day

A[i][j] = price of stock(i + 1) on day(j + 1)

Matrix A:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Stock\_1 | 7 | 1 | 5 | 3 | 6 | 8 | 9 |
| Stock\_2 | 2 | 4 | 3 | 7 | 9 | 1 | 8 |
| Stock\_3 | 5 | 8 | 9 | 1 | 2 | 3 | 10 |
| Stock\_4 | 9 | 3 | 4 | 8 | 7 | 4 | 1 |
| Stock\_5 | 3 | 1 | 5 | 8 | 9 | 6 | 4 |

Cooldown period: c = 2  
  
To solve this problem we need to find all profitable transactions for each stock(row in the matrix)

1. Choose a buy day and then try all sell days that come after that buy day
2. For each(buy, sell) day, check if the price on the sell day is higher than the price on the buy day.
3. Keep just the profitable pairs(i, j, l)

**Step 1: Identify All Possible Profitable Transactions**

For each stock, we need to check all (buy, sell) pairs where buyDay < sellDay and profit > 0:

**Stock 1: [7, 1, 5, 3, 6, 8, 9]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BuyDay | SellDay | BuyPrice | SellPrice | Profit | NextBuy | ValidTransaction |
| 1 | 2 | 7 | 1 | -6 |  |  |
| 3 | 7 | 5 | -2 |  |  |
| 4 | 7 | 3 | -4 |  |  |
| 5 | 7 | 6 | -1 |  |  |
| 6 | 7 | 8 | 1 | Day9(6+2+1) | No |
| 7 | 7 | 9 | 2 | Day10(7+2+1) | No |
| 2 | 3 | 1 | 5 | 4 | Day6(3+2+1) | (6,7) |
| 4 | 1 | 3 | 2 | Day7(4+2+1) | (7,7) |
| 5 | 1 | 6 | 5 | Day8(5+2+1) | No |
| 6 | 1 | 8 | 7 | Day9(6+2+1) | No |
| 7 | 1 | 9 | 8 | Day10(7+2+1) | No |
| 3 | 4 | 5 | 3 | -2 |  |  |
| 5 | 5 | 6 | 1 | Day8(5+2+1) | No |
| 6 | 5 | 8 | 3 | Day9 | No |
| 7 | 5 | 9 | 4 | Day10 | No |
| 4 | 5 | 3 | 6 | 3 | Day8 | No |
| 6 | 3 | 8 | 5 | Day9 | No |
| 7 | 3 | 9 | 6 | Day10 | No |
| 5 | 6 | 6 | 8 | 2 | Day9 | No |
| 7 | 6 | 9 | 3 | Day10 | No |
| 6 | 7 | 8 | 9 | 1 | Day10 | No |

From the table we see that the best combination for Stock 1: (2,7) with profit = 8

**Stock 2: [2, 4, 3, 7, 9, 1, 8]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BuyDay | SellDay | BuyPrice | SellPrice | Profit | NextBuy | ValidTransaction |
| 1 | 2 | 2 | 4 | 2 | Day5(2+2+1) | (5, 6); (5, 7); (6, 7) |
| 3 |  | 3 | 1 | Day6 | (6, 7) |
| 4 |  | 7 | 5 | Day7 | (7, 7) |
| 5 |  | 9 | 7 | Day8 | No |
| 6 |  | 1 | -1 |  |  |
| 7 |  | 8 | 6 | Day10 | No |
| 2 | 3 | 4 | 3 | -1 |  |  |
| 4 |  | 7 | 3 | Day7 | (7, 7) |
| 5 |  | 9 | 5 | Day8 | No |
| 6 |  | 1 | -3 |  |  |
| 7 |  | 8 | 4 | Day10 | No |
| 3 | 4 | 3 | 7 | 4 | Day7 | (7, 7) |
| 5 |  | 9 | 6 | Day8 | No |
| 6 |  | 1 | -2 |  |  |
| 7 |  | 8 | 5 | Day10 | No |
| 4 | 5 | 7 | 9 | 2 | Day8 | No |
| 6 |  | 1 | -6 |  |  |
| 7 |  | 8 | 1 | Day10 | No |
| 5 | 6 | 9 | 1 | -8 |  |  |
| 7 |  | 8 | -1 |  |  |
| 6 | 7 | 1 | 8 | 7 | Day9 | No |

Best single transaction for Stock 2: (1,5) with profit = 7

**Stock 3: [5, 8, 9, 1, 2, 3, 10]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BuyDay | SellDay | BuyPrice | SellPrice | Profit | NextBuy | ValidTransaction |
| 1 | 2 | 5 | 8 | 3 | Day5 | (5, 6); (5, 7); (6, 7) |
| 3 |  | 9 | 4 | Day6 | (6, 7) |
| 4 |  | 1 | -4 |  |  |
| 5 |  | 2 | -3 |  |  |
| 6 |  | 3 | -2 |  |  |
| 7 |  | 10 | 5 | Day10 | No |
| 2 | 3 | 8 | 9 | 1 | Day6 | (6, 7) |
| 4 |  | 1 | -7 |  |  |
| 5 |  | 2 | -6 |  |  |
| 6 |  | 3 | -5 |  |  |
| 7 |  | 10 | 2 | Day10 | No |
| 3 | 4 | 9 | 1 | -8 |  |  |
| 5 |  | 2 | -7 |  |  |
| 6 |  | 3 | -6 |  |  |
| 7 |  | 10 | 1 | Day10 | No |
| 4 | 5 | 1 | 2 | 1 | Day8 | No |
| 6 |  | 3 | 2 | Day9 | No |
| 7 |  | 10 | 9 | Day10 | No |
| 5 | 6 | 2 | 3 | 1 | Day9 | No |
| 7 |  | 10 | 8 | Day10 | No |
| 6 | 7 | 3 | 10 | 7 | Day10 | No |

Best single transaction for Stock 3: (4,7) with profit = 9

**Stock 4: [9, 3, 4, 8, 7, 4, 1]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BuyDay | SellDay | BuyPrice | SellPrice | Profit | NextBuy | ValidTransaction |
| 1 | 2 | 9 | 3 | -6 |  |  |
| 3 |  | 4 | -5 |  |  |
| 4 |  | 8 | -1 |  |  |
| 5 |  | 7 | -2 |  |  |
| 6 |  | 4 | -5 |  |  |
| 7 |  | 1 | -8 |  |  |
| 2 | 3 | 3 | 4 | 1 | Day6(3+2+1) | (6, 7) |
| 4 |  | 8 | 5 | Day7 | (7, 7) |
| 5 |  | 7 | 4 | Day8 | No |
| 6 |  | 4 | 1 | Day9 | No |
| 7 |  | 1 | -2 |  |  |
| 3 | 4 | 4 | 8 | 4 | Day7 | (7, 7) |
| 5 |  | 7 | 3 | Day8 | No |
| 6 |  | 4 | 0 | Day9 | No |
| 7 |  | 1 | -3 |  |  |
| 4 | 5 | 8 | 7 | -1 |  |  |
| 6 |  | 4 | -4 |  |  |
| 7 |  | 1 | -7 |  |  |
| 5 | 6 | 7 | 4 | -3 |  |  |
| 7 |  | 1 | -6 |  |  |
| 6 | 7 | 4 | 1 | -3 |  |  |

Best single transaction for Stock 4: (2,4) with profit = 5

**Stock 5: [3, 1, 5, 8, 9, 6, 4]**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| BuyDay | SellDay | BuyPrice | SellPrice | Profit | NextBuy | ValidTransaction |
| 1 | 2 | 3 | 1 | -2 |  |  |
| 3 |  | 5 | 2 | Day6(3+2+1) | (6, 7) |
| 4 |  | 8 | 5 | Day7 | (7, 7) |
| 5 |  | 9 | 6 | Day8 | No |
| 6 |  | 6 | 3 | Day9 | No |
| 7 |  | 4 | 1 | Day10 | No |
| 2 | 3 | 1 | 5 | 4 | Day6 | (6, 7) |
| 4 |  | 8 | 7 | Day7 | (7, 7) |
| 5 |  | 9 | 8 | Day8 | No |
| 6 |  | 6 | 5 | Day9 | No |
| 7 |  | 4 | 3 | Day10 | No |
| 3 | 4 | 5 | 8 | 3 | Day7 | (7, 7) |
| 5 |  | 9 | 4 | Day8 | No |
| 6 |  | 6 | 1 | Day9 | No |
| 7 |  | 4 | -1 |  |  |
| 4 | 5 | 8 | 9 | 1 | Day8 | No |
| 6 |  | 6 | -2 |  |  |
| 7 |  | 4 | -4 |  |  |
| 5 | 6 | 9 | 6 | -3 |  |  |
| 7 |  | 4 | -5 |  |  |
| 6 | 7 | 6 | 4 | -2 |  |  |

Best single transaction for Stock 5: (2,5) with profit = 8

Since we know the best individual transactions per stock. Now we check if we can combine some of them to build a valid sequence.

Starting with stock 1, the best transaction is: buy on day 2, sell on day 7 with profit = 8. After applying the cooldown rule the next valid buy day is day 10 but our max day is 7. Therefore, we can’t combine it with any other transaction

* Sequence (1, 2, 7) with total profit = 8

Stock 2: The best transaction is to buy on day 1 and sell on day 5 with profit = 7 and since the next buy day is day 8 we can’t make an extra transaction.

* Sequence (2, 1, 5) with total profit = 7

but we have another transaction with a smaller profit of 2 if we buy on day 1 and sell on day 2, after the resting period we can buy stock 3 on day 5, sell on day 7 with profit =8

* Sequence (2, 1, 2), (3, 5, 7) with total profit = 10

Stock 3 we found that the best transaction is to buy on day 4, sell on day 7 with profit = 9 and since we need to wait for day10 (invalid) to make another transaction

* Sequence (3, 4, 7) with total profit = 9

But if we buy on day 1 and sell on day 3 with profit = 4, we can combine it with Stock 2 on day 6 after the cooldown period, we buy on day 6 and sell on day 7 with profit = 7

* Sequence (3, 1, 3), (2, 6, 7) with total profit = 11

Stock 4, we have the best profit = 5 if we buy on day 2 and sell on day 4, since the next valid buy day is day 7 and there is no available transaction starting day 7

* Sequence (4, 2, 4) with total profit = 5

For Stock 5 the best transaction is when we buy on day 2 and sell on day 5 with profit = 8, after applying the cooldown rule, we don’t get a valid day

* Sequence (5, 2, 5) with total profit = 8

From the above, the maximum profit = 11 from the sequence (3, 1, 4), (2, 6, 7)

* To achieve the maximum profit, buy 3rd stock on day 1, sell it on day 3. buy 2nd stock on day 6 and sell it on day 7 adhering to 2 days waiting period

# **Milestone 2: Algorithm Design**

2.1 Task-1: Brute Force Algorithm for Problem1 O(m·n2).

The goal is to find the maximum profit from a single buy/sell transaction on the same stock with one buy before one sell, only one transaction, and return stock index, buy day, sell day, and max profit.

**Assumptions & variable definitions:**

* m: number of stocks (rows in matrix A)
* n: number of days (columns in A)
* A transaction is defined as buying a stock on day j1 and selling it later on day j2, where j1 < j2.
* The transaction must be on the same stock (row).
* The result should be a tuple: (i, j1, j2, profit) where:
* i: index of the chosen stock (1-based index)
* j1: day to buy
* j2: day to sell
* profit = A[i][j2] - A[i][j1]

**Pseudocode:**

Algorithm MaxProfitBruteForce(A, m, n)

Input: matrix A(m × n) representing stock prices

Output: tuple (i, j₁, j₂, profit) representing the best stock and days to buy/sell and max profit

maxProfit ← 0

bestStock ← 0

bestBuyDay ← 0

bestSellDay ← 0

for i ← 0 to m − 1 do

for j₁ ← 0 to n − 2 do

for j₂ ← j₁ + 1 to n − 1 do

profit ← A[i][j₂] − A[i][j₁]

if profit > maxProfit then

maxProfit ← profit

bestStock ← i + 1

bestBuyDay ← j₁ + 1

bestSellDay ← j₂ + 1

if maxProfit = 0 then

return (0, 0, 0, 0) // no profitable transaction

else

return (bestStock, bestBuyDay, bestSellDay, maxProfit)

The algorithm checks all possible pairs of days (j1, j2) for each stock to calculate potential profit, it loops over every stock (m stocks) and for each stock, compares every pair of buy/ sell days.

The algorithm keeps track of the max profit found so far and stores its stock index and days. If no profitable transaction exists (all negative or 0), it returns (0, 0, 0, 0).

2.2 Task-2: Greedy Algorithm forProblem1 O(m·n)

**Assumptions & variable definitions:**

* Only one transaction is allowed per stock
* Buy must occur before sell (j₁ < j₂)
* Each stock is evaluated independently
* Return (0, 0, 0, 0) if no profit is possible

**Pseudocode:**

Algorithm MaxProfitGreedy(A, m, n)

Input: matrix A(m × n) representing stock prices

Output: tuple (i, j₁, j₂, profit) representing the best stock and days to buy/sell and max profit

maxProfit ← 0

bestStock ← 0

bestBuyDay ← 0

bestSellDay ← 0

for i ← 0 to m − 1 do

minPrice ← A[i][0]

minDay ← 0

for j ← 1 to n − 1 do

if A[i][j] − minPrice > maxProfit then

maxProfit ← A[i][j] − minPrice

bestStock ← i + 1

bestBuyDay ← minDay + 1

bestSellDay ← j + 1

if A[i][j] < minPrice then

minPrice ← A[i][j]

minDay ← j

if maxProfit = 0 then

return (0, 0, 0, 0)

else

return (bestStock, bestBuyDay, bestSellDay, maxProfit)

In this greedy version of the algorithm, we optimize the search for the max profit from a single buy/ sell transaction. For each stock, the algorithm keeps the minimum price observed so far (minPrice) and the corresponding day (minDay). As it iterates through the remaining days, it computes the current potential profit by subtracting minPrice from the price on the current day. If this profit exceeds the previously recorded maxProfit, the algorithm updates the optimal transaction details (stock index, buy day, and sell day). If the current day's price is less than minPrice, it becomes the new minPrice. If no profitable transaction exists, the algorithm returns the default tuple (0, 0, 0, 0).

**Time Complexity:** O(m · n)

The Algorithm iterates over each of the (m) stocks

For each stock, it performs a single pass over (n) daily prices

In each pass, comparisons and updates are done in constant time

* Total = O(m · n)

2.3 Task-3: Dynamic Programming Algorithm for Problem1 O(m·n)

**Assumptions & variable definitions:**

**Pseudocode:**

2.4 Task-4: Dynamic Programming Algorithm for Problem2 O(m·n2k)

**Assumptions & variable definitions:**

**Pseudocode:**

2.5 Task-5: Dynamic Programming Algorithm for Problem2 O(m·n·k)

**Assumptions & variable definitions:**

**Pseudocode:**

2.6 Task-6: Dynamic Programming Algorithm for Problem3 O(m·n2)

**Assumptions & variable definitions:**

**Pseudocode:**

2.7 Task-7: Dynamic Programming Algorithm for Problem3 O(m·n)

**Assumptions & variable definitions:**

**Pseudocode:**