# Milestone 1

## Group Members

**Anthony Mezzatesta**
Amezzatesta@ufl.edu
Github - AnthonyMezzatesta
Role - Developer

**Jonathan Palma**
palma.jonathan@ufl.edu
Github - gitttjonzzz
Role - Developer

**Luis Gonzalez**
lgonzaherman@gmail.com
Github - gonzaherman99
Role - Developer

**Mert Dayi**
imert.dayi@ufl.edu
Github - mertdayi01
Role - Project Manager / Developer

## Communication Method

Slack

## Github Repository Link

https://github.com/gonzaherman99/C4533-GroupProject

# Gantt Chart

| | TASK TITLE | TASK OWNER | START DATE | DUE DATE | DURATION | PCT OF TASK COMPLETE | WEEK 1 | | | | | WEEK 2 | | | | | WEEK 3 | | | | | WEEK 4 | | | | | WEEK 5 | | | | | WEEK 6 | | | | | WEEK 7 | | | | | WEEK 8 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F |
| 4 | Milestone 1: Understanding the problems | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4.1 | Problem 1 | Anthony M | 6/9/25 | 6/15/25 | 6 | 100% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4.2 | Problem 2 | Jonathan P | 6/9/25 | 6/15/25 | 6 | 100% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4.3 | Problem 3 | Luis G | 6/9/25 | 6/15/25 | 6 | 100% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4.4 | TODO | Mert D | 6/9/25 | 6/15/25 | 6 | 100% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | Milestone 2: Algorithm Design | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Meeting | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Task 1 | Anthony M | 6/16/25 | 7/6/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Task 2 | Jonathan P | 6/16/25 | 7/6/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Task 3 | Luis G | 6/16/25 | 7/6/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5.1 | TODO | Mert D | 6/16/25 | 7/6/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | Milestone 3: Algorithm Implementation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Meeting | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Algorithm 1 | Anthony M | 7/7/25 | 7/27/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Algorithm 2 | Jonathan P | 7/7/25 | 7/27/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Algorithm 3 | Luis G | 7/7/25 | 7/27/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6.1 | TODO | Mert D | 7/7/25 | 7/27/25 | 20 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | Presentation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Create Slides | | 7/28/25 | 8/3/25 | 5 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Record Presentation | | 7/28/25 | 8/3/25 | 5 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Submit | | 7/28/25 | 8/3/25 | 5 | 0% | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Step 1)
Input Matrix:

```
A =   [12, 1, 5, 3, 16],     // Stock 1
      [4, 4, 13, 4, 9],      // Stock 2
      [6, 8, 6, 1, 2],       // Stock 3
      [14, 3, 4, 8, 10]      // Stock 4
```

Step 2)
STOCK 1:

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | -11 |
| 1 | 3 | -7 |
| 1 | 4 | -9 |
| 1 | 5 | 4 |
| 2 | 3 | 4 |
| 2 | 4 | 2 |
| 2 | 5 | 15 |
| 3 | 4 | -2 |
| 3 | 5 | 11 |
| 4 | 5 | 13 |

STOCK 2:

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 0 |
| 1 | 3 | 9 |
| 1 | 4 | 0 |
| 1 | 5 | 5 |
| 2 | 3 | 9 |
| 2 | 4 | 0 |
| 2 | 5 | 5 |
| 3 | 4 | -9 |
| 3 | 5 | -4 |
| 4 | 5 | 5 |

STOCK 3:

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 2 |
| 1 | 3 | 0 |
| 1 | 4 | -5 |
| 1 | 5 | -4 |
| 2 | 3 | -2 |
| 2 | 4 | -7 |
| 2 | 5 | -6 |
| 3 | 4 | -5 |
| 3 | 5 | -4 |
| 4 | 5 | 1 |

STOCK 4:

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | -11 |
| 1 | 3 | 10 |
| 1 | 4 | -6 |
| 1 | 5 | -4 |
| 2 | 3 | 1 |
| 2 | 4 | 5 |
| 2 | 5 | 7 |
| 3 | 4 | 4 |
| 3 | 5 | 6 |
| 4 | 5 | 2 |

Step 3)
Stock 1 most profitable transaction:
Buy at day 2 for 1 and sell at day 5 for 16. Profit = 15

Stock 2 most profitable transaction:
Buy at day 1 for 4 and sell at day 3 for 13. Profit = 9

Stock 3 most profitable transaction:
Buy at day 1 for 6 and sell at day 2 for 8. Profit = 2

Stock 4 most profitable transaction:
Buy at day 2 for 3 and sell at day 5 for 10. Profit = 7

Step 4)
Maximum profit is 15 from Stock 1 – Buy day 2 and sell day 5
OUTPUT: (1, 2, 5, 15)

# Problem Statement 2

**You are given a matrix A of dimensions m × n, where each element represents the predicted prices of m different stocks for n consecutive days. Additionally, you are given an integer k (1 ≤ k ≤ n). Your task is to manually find a sequence of at most k transactions, each involving the purchase and sale of a single stock, that yields the maximum profit.**

# Step 1)

**Input Matrix:**

```
A = [[25, 30, 15, 40, 50],    // Stock 1
     [10, 20, 30, 25,  5],    // Stock 2
     [30, 45, 35, 10, 15],    // Stock 3
     [ 5, 50, 35, 25, 45]]    // Stock 4
```

# Step 2)

**STOCK 1:**

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 5 |
| 1 | 3 | -10 |
| 1 | 4 | 15 |
| 1 | 5 | 25 |
| 2 | 3 | -15 |

| | | |
|---|---|---|
| 2 | 4 | 10 |
| 2 | 5 | 20 |
| 3 | 4 | 25 |
| 3 | 5 | 35 |
| 4 | 5 | 10 |

**STOCK 2:**

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 10 |
| 1 | 3 | 20 |
| 1 | 4 | 15 |
| 1 | 5 | -5 |
| 2 | 3 | 10 |
| 2 | 4 | 5 |
| 2 | 5 | -15 |
| 3 | 4 | -5 |
| 3 | 5 | -25 |
| 4 | 5 | -20 |

**STOCK 3:**

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 15 |
| 1 | 3 | 5 |
| 1 | 4 | -20 |
| 1 | 5 | -15 |
| 2 | 3 | -10 |
| 2 | 4 | -35 |
| 2 | 5 | -30 |
| 3 | 4 | -25 |
| 3 | 5 | -20 |
| 4 | 5 | 5 |

**STOCK 4:**

| BUY DAY | SELL DAY | PROFIT |
|---|---|---|
| 1 | 2 | 45 |
| 1 | 3 | 30 |
| 1 | 4 | 20 |
| 1 | 5 | 40 |
| 2 | 3 | -15 |

| | | |
|---|---|---|
| 2 | 4 | -25 |
| 2 | 5 | -5 |
| 3 | 4 | -10 |
| 3 | 5 | 10 |
| 4 | 5 | 20 |

# Step 3)

**Finding optimal non-overlapping transactions (k=3):**

**Best profitable transactions:**

1. Stock 4: (1,2) → Profit = 45
2. Stock 1: (3,5) → Profit = 35
3. Stock 4: (1,5) → Profit = 40
4. Stock 1: (1,5) → Profit = 25
5. Stock 1: (3,4) → Profit = 25

**Selected non-overlapping transactions:**

- Transaction 1: Stock 4, Buy Day 1, Sell Day 2 → Profit = 45
- Transaction 2: Stock 1, Buy Day 3, Sell Day 5 → Profit = 35

**Total Maximum Profit = 80**

**Output: [(4,1,2), (1,3,5)]**

**Problem 3**

**Problem Statement**

You are given a matrix A of dimensions $m \times n$, where each element represents the predicted prices of m different stocks for n consecutive days. Additionally, you are given an integer c ($1 \leq c \leq n - 2$). Your task is to determine the maximum profit achievable under the given trading restrictions, where you cannot buy any stock for c days after selling any stock. If you sell a stock on day i, you are not allowed to buy any stock until day $i + c + 1$.

**Input**

| | **Day 1** | **Day 2** | **Day 3** | **Day 4** | **Day 5** | **Day 6** | **Day 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: $c = 2$

First let's define best(t) as the maximum profit achievable with one buy, sell between days t and 7.

| **t** | **Window** | **Best trade** | **$best_1(t)$** |
|---|---|---|---|
| **1–4** | Days 1 to 7 | buy Stock 3 on day 4 at 1 then sell on day 7 at 10 | 9 |
| **5** | Days 5 to 7 | buy Stock 3 on day 5 at 2 then sell on day 7 at 10 | 8 |
| **6** | Days 6 to 7 | buy Stock 2 on day 6 at 1 then sell on day 7 at 8 | 7 |
| **7** | day 7 | no transaction | 0 |

Now for each possible first trade (i, j, l), let's compute $Profit_1 = P[\, i , l \,] - P[\, i , j \,]$, earliest next buy $B = l + 3$, $Profit_2 = best_1(B)$, and $Total = Profit_1 + Profit_2$. Here are the top contenders:

| **Buy j** | **First trade ( i , l )** | **$Profit_1$** | **Next B=l +3** | **$Profit_2$** | **Total** |
|---|---|---|---|---|---|
| **j = 1** | Stock 2: buy | 7 | 8 to 0 | 0 | 7 |

| | | | | | |
|---|---|---|---|---|---|
| | 1, sell 5 | | | | |
| **j = 2** | Stock 1: buy 2, sell 3 | 4 | 6 to 7 | 7 | 11 |
| **j = 3** | Stock 2: buy 3, sell 5 | 6 | 8 to 0 | 0 | 6 |
| **j = 4** | Stock 3: buy 4, sell 7 | 9 | 10 to 0 | 0 | 9 |
| **j = 5** | Stock 3: buy 5, sell 7 | 8 | 10 to 0 | 0 | 8 |
| **j = 6** | Stock 2: buy 6, sell 7 | 7 | 10 to 0 | 0 | 7 |

The best total (11) come from:

1. Stock 1: buy on day 2 at 1 then sell on day 3 at 5 ($Profit_1 = 4$) with the cooldown until day 6 (3 + 2 + 1)

2. Stock 2: buy on day 6 at 1 then sell on day 7 at 8 ($Profit_2 = 7$)

Total profit = 4 + 7 = 11.

**Final Answer**

Maximum profit: 11

Trades: (i = 1, j = 2, l = 3) and (i = 2, j = 6, l = 7)

# Milestone 2

## Group Members

**Anthony Mezzatesta**
Amezzatesta@ufl.edu
Github - AnthonyMezzatesta
Role - Developer

**Jonathan Palma**
palma.jonathan@ufl.edu
Github - gitttjonzzz
Role - Developer

**Luis Gonzalez**
lgonzaherman@gmail.com
Github - gonzaherman99
Role - Developer

**Mert Dayi**
imert.dayi@ufl.edu
Github - mertdayi01
Role - Project Manager / Developer

## Communication Method

Slack

## Programming Language

Python

## Github Repository Link

https://github.com/gonzaherman99/C4533-GroupProject

# Gantt Chart

| | TASK TITLE | TASK OWNER | START DATE | DUE DATE | DURATION | PCT OF TASK COMPLETE |
|---|---|---|---|---|---|---|
| 4 | Milestone 1: Understanding the problems | | | | | |
| 4.1 | Problem 1 | Anthony M | 6/9/25 | 6/15/25 | 6 | 100% |
| 4.2 | Problem 2 | Jonathan P | 6/9/25 | 6/15/25 | 6 | 100% |
| 4.3 | Problem 3 | Luis G | 6/9/25 | 6/15/25 | 6 | 100% |
| 4.4 | TODO | Mert D | 6/9/25 | 6/15/25 | 6 | 100% |
| 5 | Milestone 2: Algorithm Design | | | | | |
| | Meeting | | | | | |
| | Task 1 | Anthony M | 6/30/25 | 7/20/25 | 20 | 100% |
| | Task 2 | Jonathan P | 6/30/25 | 7/20/25 | 20 | 100% |
| | Task 3 | Luis G | 6/30/25 | 7/20/25 | 20 | 100% |
| 5.1 | TODO | Mert D | 6/30/25 | 7/20/25 | 20 | 100% |
| 6 | Milestone 3: Algorithm Implementation | | | | | |
| | Meeting | | | | | |
| | Algorithm 1 | Anthony M | 7/21/25 | 8/3/25 | 12 | 0% |
| | Algorithm 2 | Jonathan P | 7/21/25 | 8/3/25 | 12 | 0% |
| | Algorithm 3 | Luis G | 7/21/25 | 8/3/25 | 12 | 0% |
| 6.1 | TODO | Mert D | 7/21/25 | 8/3/25 | 12 | 0% |
| 7 | Presentation | | | | | |
| | Create Slides | | 7/28/25 | 8/3/25 | 5 | 0% |
| | Record Presentation | | 7/31/25 | 8/3/25 | 3 | 0% |
| | Submit | | 8/2/25 | 8/3/25 | 1 | 0% |

| WEEK 1 | | | | | WEEK 2 | | | | | WEEK 3 | | | | | WEEK 4 | | | | | WEEK 5 | | | | | WEEK 6 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F | M | T | W | R | F |

Brute Force Problem 1)

```
function brute_force_prob1(A):
    m = rows in A
    n = columns in A
    best_profit = 0
    best_stock = 0
    best_buy = 0
    best_sell = 0

    # Iterate over each stock
    for i from 0 to m-1:
        # Buy day index
        for j from 0 to n-2:
            # Sell day index (must be after buy)
            for k from j+1 to n-1:
                profit = A[i][k] - A[i][j]
                if profit > best_profit:
                    best_profit = profit
                    best_stock = i
                    best_buy = j
                    best_sell = k
    # No profitable transaction
    if best_profit <= 0:
        return (0, 0, 0, 0)
    else:
        # Convert to 1-based indexing
        return (best_stock+1, best_buy+1, best_sell+1, best_profit)
```

Greedy Problem 1)

```
function greedy_prob1(A):
   m = rows in A
   n = columns in A
   best_profit = 0
   best_stock = -1
   best_buy = -1
   best_sell = -1

        # Iterate over each stock
        for i from 0 to m-1:
                min_price = A[i][0]
                min_index = 0
                current_profit = 0
                current_buy = 0
                current_sell = 0

                # Start from day 1
                for j from 1 to n-1:
                        # Found new minimum price
                        if A[i][j] < min_price:
                                min_price = A[i][j]
                                min_index = j
                        else:
                                profit = A[i][j] – min_price
                                # Update if better profit
                                if profit > current_profit:
                                        current_profit = profit
                                        current_buy = min_index
                                        current_sell = j

                # Update global best
                if current_profit > best_profit:
                        best_profit = current_profit
                        best_stock = i
                        best_buy = current_buy
                        best_sell = current_sell

        if best_profit <= 0:
                return (0, 0, 0, 0)
        else:
                return (best_stock+1, best_buy+1, best_sell+1, best_profit)
```

DP Problem 1)

```
function dp_prob1(A):
    m = rows in A
    n = columns in A
    best_profit = 0
    best_stock = -1
    best_buy = -1
    best_sell = -1

    # Iterate over each stock
    for i from 0 to m-1:
        min_price = A[i][0]
        min_index = 0
        max_profit_i = 0
        buy_i = 0
        sell_i = 0

        # Start from day 1
        for j from 1 to n-1:
            # Update minimum price if lower found
            if A[i][j] < min_price:
                min_price = A[i][j]
                min_index = j
            # Calculate profit if selling today
            profit = A[i][j] - min_price
            if profit > max_profit_i:      # Update if better profit
                max_profit_i = profit
                buy_i = min_index
                sell_i = j

        # Update global best
        if max_profit_i > best_profit:
            best_profit = max_profit_i
            best_stock = i
            best_buy = buy_i
            best_sell = sell_i

    if best_profit <= 0:
        return (0, 0, 0, 0)
    else:
        return (best_stock+1, best_buy+1, best_sell+1, best_profit)
```

Dynamic Programming Algorithm for Problem 2 ($O(m \cdot n^2k)$ Time)

```
function dp_prob2_n2k(A):
    m = number of rows in A
    n = number of columns in A
    k = [number of  operations]

    DP = 3D array of size [m+1][n+1][k+1], initialized to -infinity
    DP[0][0][0] = 0  # or base_case_value

    for i from 1 to m:
        for j from 1 to n:
            for x from 0 to k:
                for p from 1 to n:
                    for q from 0 to k:
                        if valid_transition(p, q, j, x):
                            DP[i][j][x] = max(
                                DP[i][j][x],
                                DP[i-1][p][q] + cost(B, i, j, x)
                            )

    answer = maximum DP[m][j][x] over all valid j, x
    return answer
```

Dynamic Programming Algorithm for Problem 2 ($O(m \cdot n \cdot k)$ Time)

```
function dp_prob2_nk(A):
    m = number of rows in A
    n = number of columns in A
    k = [number of operations]

    DP = 3D array of size [m+1][n+1][k+1], initialized to -infinity
    DP[0][0][0] = 0  # or base_case_value

    for i from 1 to m:
        for j from 1 to n:
            for x from 0 to k:
                DP[i][j][x] = max(
                    DP[i-1][j][x],
                    DP[i-1][j-1][x-1]
                ) + cost(B, i, j, x)

    answer = maximum DP[m][j][x] over all valid j, x
    return answer
```

```
function dp_prob3_slow(A, c):
    m = rows in A
    n = columns in A
    dp = array of size n initialized to 0
    choice = array of size n initialized to null

    for i from 1 to n-1:
        dp[i] = dp[i-1]
        choice[i] = null

        for j from 0 to i-1:
            prev_profit = (j-c-1 >= 0) ? dp[j-c-1] : 0

            for stock from 0 to m-1:
                profit = A[stock][i] - A[stock][j]
                total = prev_profit + profit

                if total > dp[i]:
                    dp[i] = total
                    choice[i] = (stock, j, prev_profit)

    if dp[n-1] <= 0:
        return []

    result = []
    i = n-1
    while i > 0 and choice[i] != null:
        stock, buy_day, prev_profit = choice[i]
        result.add((stock+1, buy_day+1, i+1))

        j = buy_day - c - 1
        while j >= 0 and dp[j] != prev_profit:
            j = j - 1
        i = j

    reverse(result)
    return result


function dp_prob3_fast(A, c):
    m = rows in A
    n = columns in A

    hold = -infinity
```

```
sold = 0
rest = 0

transactions = []
hold_info = null

for i from 0 to n-1:
   prev_hold = hold
   prev_sold = sold
   prev_rest = rest

   rest = max(prev_rest, prev_sold)

   sold = prev_hold
   sell_stock = -1
   for stock from 0 to m-1:
      profit = prev_hold + A[stock][i]
      if profit > sold:
         sold = profit
         sell_stock = stock

   if sell_stock != -1 and sold > prev_sold:
      if hold_info != null:
         transactions.add((hold_info.stock+1, hold_info.day+1, i+1))
      hold_info = null

   hold = prev_hold
   buy_stock = -1
   buy_profit = hold

   for stock from 0 to m-1:
      profit = rest - A[stock][i]
      if profit > buy_profit:
         buy_profit = profit
         buy_stock = stock

   if buy_stock != -1:
      hold = buy_profit
      hold_info = (stock: buy_stock, day: i)

final_profit = max(sold, rest)
if final_profit <= 0:
   return []
```

return transactions