King Saud University College of Computer and Information Sciences Computer Science Department	
CSC311 Algorithms Analysis & Design	Second Semester 1445

Optimal Portfolio Allocations

Students		
Name	**ID**	**Individual Contributions**
Fay Almajed	443200541	File Reading Implementation – Report Writing - Debugging
Hessa Alshaya	443200834	Optimal Portfolio Methods - Debugging
Layan Alomari	443200423	Asset Class – Time Complexity – Testing and Validation

A) Pseudocode of The Algorithm

```
Initialize OptimalSolutions as an empty list

Initialize maxExpectedReturn as 0

Initialize RiskLevel as the maximum possible value

For i from 0 to assets.get(0).quantity:

    For j from 0 to min(assets.get(1).quantity, totalInvestment - i):

        Let k = totalInvestment - i - j

        If k > assets.get(2).quantity, continue to the next iteration

        Calculate expectedReturn as:

            calcExReturn(assets.get(0).expectedReturn, i, totalInvestment) +

            calcExReturn(assets.get(1).expectedReturn, j, totalInvestment) +

            calcExReturn(assets.get(2).expectedReturn, k, totalInvestment)

        Calculate riskLevel as:

            calcRisk(assets.get(0).riskLevel, i, totalInvestment) +

            calcRisk(assets.get(1).riskLevel, j, totalInvestment) +

            calcRisk(assets.get(2).riskLevel, k, totalInvestment)

        If riskLevel <= risktoleranceLevel and expectedReturn > maxExpectedReturn:

            Update maxExpectedReturn to expectedReturn

            Update RiskLevel to riskLevel

            Clear OptimalSolutions

            Add new Asset to OptimalSolutions for each asset:

                Asset(assets.get(0).id, assets.get(0).expectedReturn, assets.get(0).riskLevel, i)

                Asset(assets.get(1).id, assets.get(1).expectedReturn, assets.get(1).riskLevel, j)

                Asset(assets.get(2).id, assets.get(2).expectedReturn, assets.get(2).riskLevel, k)

Return Optimalportfolio(OptimalSolutions, maxExpectedReturn, RiskLevel)
```

A) Time Complexity

1- Information Extraction:

   Worst Case Scenario: O(n).

   Best Case Scenario: O(1).

2- Risk Calculation:

   Worst Case Scenario: O(1).

   Best Case Scenario: O(1).

3- Expected Return Calculation:

   Worst Case Scenario: O(1).

   Best Case Scenario: O(1).

4- Quantity Calculation:

   Worst Case Scenario: O(n).

   Best Case Scenario: O(n).

5- Optimal Allocation Finding:

   Worst and Best Case Scenario: O(n^2).

   **Note: The algorithm searches for all possible allocations, from 0 to n, for the most optimal allocation. This exhaustive search is essential regardless of the case since every potential allocation needs to be considered. Therefore, whether the best allocation is found early or not does not matter.

B) Experimental Results

```
AAPL : 0.07 : 0.04 : 1100
GOOGL : 0.1 : 0.05 : 600
MSFT : 0.06 : 0.025 : 900
Total investment is 900 units
Risk tolerance level is 0.038
```

```
Optimal Allocation:
AAPL: 1 units
GOOGL: 467 units
MSFT: 432 units
Expected Portfolio Return: 0.081
Portfolio Risk Level: 0.038
Algorithm Execution Time: 83 milliseconds
```

```
AAPL : 0.07 : 0.04 : 900
GOOGL : 0.1 : 0.05 : 400
MSFT : 0.06 : 0.025 : 700
Total investment is 900 units
Risk tolerance level is 0.038
```

```
Optimal Allocation:
AAPL: 113 units
GOOGL: 400 units
MSFT: 387 units
Expected Portfolio Return: 0.079
Portfolio Risk Level: 0.038
Algorithm Execution Time: 63 milliseconds
```

C) Screenshots

Getting the information from the .txt file:

```
try {
    //accessing the text file
    FileReader fReader = new FileReader(path);
    BufferedReader bReader = new BufferedReader(fReader);

    String line;
    while ((line = bReader.readLine()) != null) {
        if (line.startsWith(prefix:"Total investment is")) totalInvestment = Integer.parseInt(line.split(regex:" ")[3]);
        else if (line.startsWith(prefix:"Risk tolerance level is")) toleranceLevel = Double.parseDouble(line.split(regex:" ")[4]);
        else {
            String[] info = line.split(regex:"\\s*:\\s*");
            id = info[0];
            expectedReturn = Double.parseDouble(info[1].trim());
            riskLevel = Double.parseDouble(info[2].trim());
            quantity = Integer.parseInt(info[3].trim());
            assetLine++;

            Asset asset = new Asset(id, expectedReturn,riskLevel, quantity);
            if(!(assets.add(asset))) {
                System.out.println("Failed to add asset at line: " + assetLine);
            }
        }


    }
```

Searching for the best allocation:

```
public static Optimalportfolio FindOptimalAllocation(LinkedList<Asset> assets, int totalInvestment,
    List<Asset> OptimaSolutions = new ArrayList<>();
    double maxExpectedReturn = 0;
    double RiskLevel = Double.MAX_VALUE;

    for (int i = 0; i <= assets.get(index:0).quantity; i++) {
        for (int j = 0; j <= Math.min(assets.get(index:1).quantity, totalInvestment - i); j++) {
            int k = totalInvestment - i - j;

            if (k > assets.get(index:2).quantity) {
                continue;// move to the next
            }

            double expectedReturn = calcExReturn(assets.get(index:0).expectedReturn, i, totalInvestment)
                    + calcExReturn(assets.get(index:1).expectedReturn, j, totalInvestment)
                    + calcExReturn(assets.get(index:2).expectedReturn, k, totalInvestment);

            double riskLevel = calcRisk(assets.get(index:0).riskLevel, i, totalInvestment)
                    + calcRisk(assets.get(index:1).riskLevel, j, totalInvestment)
                    + calcRisk(assets.get(index:2).riskLevel, k, totalInvestment);

            // Check if the cureent allocation < or = to the risk tolerance level and has
            // higher expected return than the pre
            if (riskLevel <= risktoleranceLevel && expectedReturn > maxExpectedReturn) {
                maxExpectedReturn = expectedReturn;
                RiskLevel = riskLevel;
                OptimaSolutions.clear();
                OptimaSolutions
                        .add(new Asset(assets.get(index:0).id, assets.get(index:0).expectedReturn, assets.get(index:0).riskLevel, i));
                OptimaSolutions
                        .add(new Asset(assets.get(index:1).id, assets.get(index:1).expectedReturn, assets.get(index:1).riskLevel, j));
                OptimaSolutions
                        .add(new Asset(assets.get(index:2).id, assets.get(index:2).expectedReturn, assets.get(index:2).riskLevel, k));
            }
        }
    }
    return new Optimalportfolio(OptimaSolutions, maxExpectedReturn, RiskLevel);
}
```

Other methods that helped searching for the best allocation:

```
// calculate risk for each asset
public static double calcRisk(double risk, double Punit, double total) {
    return (Punit / total) * risk;
}

// calculate Expected return for each asset
public static double calcExReturn(double ExReturn, double Punit, double total) {
    return (Punit / total) * ExReturn;
}

// case if the investment may exeed the available quantity
public static int calcAllQuantity(List<Asset> assets) {
    int numOfAsset = assets.size();
    int size = 0;
    for (int i = 0; i < numOfAsset; i++)
        size += assets.get(i).getQuantity();
    return size;

}
```

D) Source Code

```
import java.io.*;
import java.util.*;

public class demo {

    public static void main(String[] args) {

        //variables
        String id;
        double expectedReturn, riskLevel, toleranceLevel = 0;
        int quantity, totalInvestment = 0, assetLine = 0;

        //assets list
        LinkedList<Asset> assets = new LinkedList<Asset>();

        //saving the assets text file location
        String path = "assets.txt";

        try {
            //accessing the text file
            FileReader fReader = new FileReader(path);
            BufferedReader bReader = new BufferedReader(fReader);

            String line;
            while ((line = bReader.readLine()) != null) {
                if (line.startsWith("Total investment is")) totalInvestment =
Integer.parseInt(line.split(" ")[3]);
                else if (line.startsWith("Risk tolerance level is")) toleranceLevel =
Double.parseDouble(line.split(" ")[4]);
```

```
                else {
                    String[] info = line.split("\\s*:\\s*");
                    id = info[0];
                    expectedReturn = Double.parseDouble(info[1].trim());
                    riskLevel = Double.parseDouble(info[2].trim());
                    quantity = Integer.parseInt(info[3].trim());
                    assetLine++;

                    Asset asset = new Asset(id, expectedReturn,riskLevel, quantity);
                    if(!(assets.add(asset))) {
                        System.out.println("Failed to add asset at line: " + assetLine);
                    }
                }


            }

            if (totalInvestment > calcAllQuantity(assets))
                System.out.println("can't give you an optimal allocation.");
            else {
                Optimalportfolio result = FindOptimalAllocation(assets, totalInvestment,
toleranceLevel);
                System.out.println("Optimal Allocation:");
                for (Asset asset : result.getAsset()) {
                    System.out.println(asset.id + ": " + asset.quantity + " units");
                }

                System.out.println("Expected Portfolio Return: " + String.format("%.3f",
result.getEPR()));
                System.out.println("Portfolio Risk Level: " + String.format("%.3f",
result.getPRL()));
            }
            // closing the file
            // bReader.close();
        } catch (Exception e) {
            System.out.println("Couldn't access file: " + e.getMessage());
        }

    }// end main

    // calculate risk for each asset
    public static double calcRisk(double risk, double Punit, double total) {
        return (Punit / total) * risk;
    }

    // calculate Expected return for each asset
    public static double calcExReturn(double ExReturn, double Punit, double total) {
        return (Punit / total) * ExReturn;
    }

    // case if the investment may exeed the available quantity
    public static int calcAllQuantity(List<Asset> assets) {
        int numOfAsset = assets.size();
        int size = 0;
        for (int i = 0; i < numOfAsset; i++)
            size += assets.get(i).getQuantity();
        return size;

    }

    // brute force algorithm to find Optimal Allocation
    public static Optimalportfolio FindOptimalAllocation(LinkedList<Asset> assets, int
totalInvestment,
            double risktoleranceLevel) {
        List<Asset> OptimaSolutions = new ArrayList<>();
        double maxExpectedReturn = 0;
```

```
        double RiskLevel = Double.MAX_VALUE;

        for (int i = 0; i <= assets.get(0).quantity; i++) {
            for (int j = 0; j <= Math.min(assets.get(1).quantity, totalInvestment - i);
j++) {
                int k = totalInvestment - i - j;

                if (k > assets.get(2).quantity) {
                    continue;// move to the next
                }

                double expectedReturn = calcExReturn(assets.get(0).expectedReturn, i,
totalInvestment)
                        + calcExReturn(assets.get(1).expectedReturn, j, totalInvestment)
                        + calcExReturn(assets.get(2).expectedReturn, k, totalInvestment);

                double riskLevel = calcRisk(assets.get(0).riskLevel, i, totalInvestment)
                        + calcRisk(assets.get(1).riskLevel, j, totalInvestment)
                        + calcRisk(assets.get(2).riskLevel, k, totalInvestment);

                // Check if the cureent allocation < or = to the risk tolerance level and
has
                // higher expected return than the pre
                if (riskLevel <= risktoleranceLevel && expectedReturn >
maxExpectedReturn) {
                    maxExpectedReturn = expectedReturn;
                    RiskLevel = riskLevel;
                    OptimaSolutions.clear();
                    OptimaSolutions
                            .add(new Asset(assets.get(0).id,
assets.get(0).expectedReturn, assets.get(0).riskLevel, i));
                    OptimaSolutions
                            .add(new Asset(assets.get(1).id,
assets.get(1).expectedReturn, assets.get(1).riskLevel, j));
                    OptimaSolutions
                            .add(new Asset(assets.get(2).id,
assets.get(2).expectedReturn, assets.get(2).riskLevel, k));
                }
            }
        }
        return new Optimalportfolio(OptimaSolutions, maxExpectedReturn, RiskLevel);
    }
}// end class
```

E) Challenges & Solutions

We were presented with several challenges:

6- Optimizing asset allocation efficiency.
7- Calculating total risk and return.
8- Testing and validation.
9- Time management.

We overcame these challenges by researching similar algorithms and learning how they were implemented, as well as ensuring reliability and accuracy of outputs by using the sample run provided in the project file. Breaking the program into smaller problems helped a lot with time management and understanding every aspect of the project.

E) Evaluation Rubic

Team Work			
Criteria	**Fay**	**Hessa**	**Layan**
Work division: Contributed equally to the work	**1**	**1**	**1**
Student succeeds in smoothly forming /joining group within time	**1**	**1**	**1**
Peer evaluation: Level of commitments (Interactivity with other team members), and professional behavior towards team & TA	**1**	**1**	**1**
Project Discussion: Accurate answers, understanding of the presented work, good listeners to questions	**1**	**1**	**1**
Time management: Attending on time, being ready to start the demo, good time management in discussion and demo.	**1**	**1**	**1**
Total/3	**3**	**3**	**3**