FEU INSTITUTE OF TECHNOLOGY

Fake Coin Detection with Random Weights (Decrease by a Constant Factor)

CS0007 - ALGORITHMS

Dr. Hadji J. Tejuco

Acuña, Amiel Josiah - Leader
Nuyda, Matthew
Lizardo, Gideon
Viacrusis, Sean Matthew

February 27, 2025

TABLE OF CONTENTS

Introduction	**3**
Project Overview	**3**
Requirements Analysis	**4**
System Design	**5**
Implementation	**7**
Testing	**8**
User Manual	**11**
Challenges and Solutions	**11**
Future Enhancements	**12**
Conclusion	**12**
References	**12**
Appendices	**12**

INTRODUCTION

Purpose

This project simulates the fake coin scenario using a divide-and-weigh algorithm. It demonstrates how to efficiently identify a lighter counterfeit coin among genuine ones, whilst having recursive problem-solving with exception handling.

Objectives

The objectives of the program would want to accomplish simulating the fake coin scenario in a modified manner. To do this the program must;

- Generate random weights for N coins, ensuring one is lighter (with a 50% chance of no fake).
- Implement a divide-and-conquer strategy to locate the fake coin.
- Display step-by-step comparisons during the weighing process.
- Handle edge cases (e.g., no fake coin, invalid inputs).

Scope

The program scope would show where the limit of the program supports. The program can process any user-specified N (≥1), Handles both even and odd group divisions. Does not handle multiple fake coins or non-integer weights, or graphical user interfaces.

PROJECT OVERVIEW

Problem Statement

The project should be able to show a program that can identify a single lighter fake coin in a set of N coins using minimal weighing. The program should minimize the number of weighings by implementing a divide-and-weigh strategy that recursively reduces the search space by a constant. If no fake exists, detect this edge case.

Key Features

- Random weight generation (10–20 for genuine, 1–9 for fake).
- Shows each iteration as visualization of group comparisons.
- Recursive algorithm reducing problem size by 1/2 or 1/3.
- Input validation and clear output messages.

REQUIREMENTS ANALYSIS

Functional Requirements

- User specifies N, Randomly generates a list of N coins with weights between 10 and 20. Program validates input ensuring one coin is lighter.
- Generate weights with one fake (50% chance of none).
- Implement the divide-and-weigh algorithm, Divide coins into groups, compare weights, and recurse.
- Display iterations, groups, and final result (index/weight or "no fake").

Non-Functional Requirements

Performance

The algorithm should efficiently reduce the problem size denoted by; O(log N) time complexity in each step.

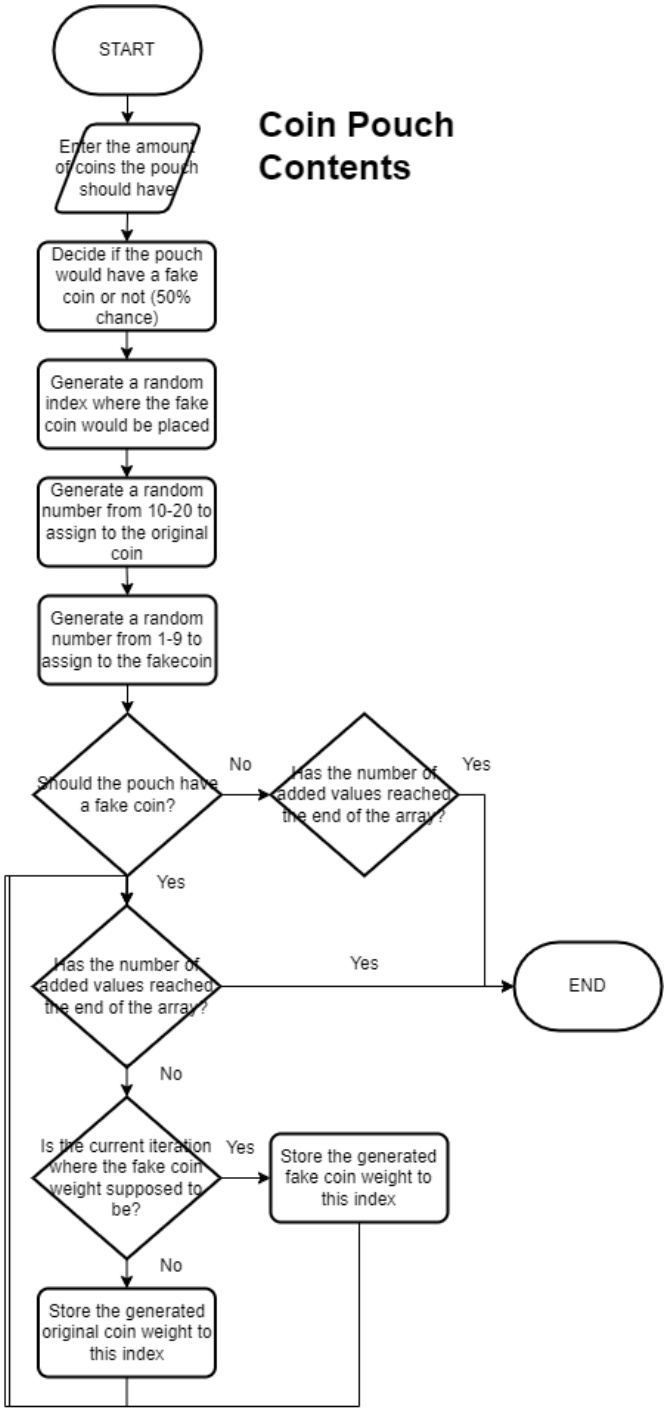
Usability

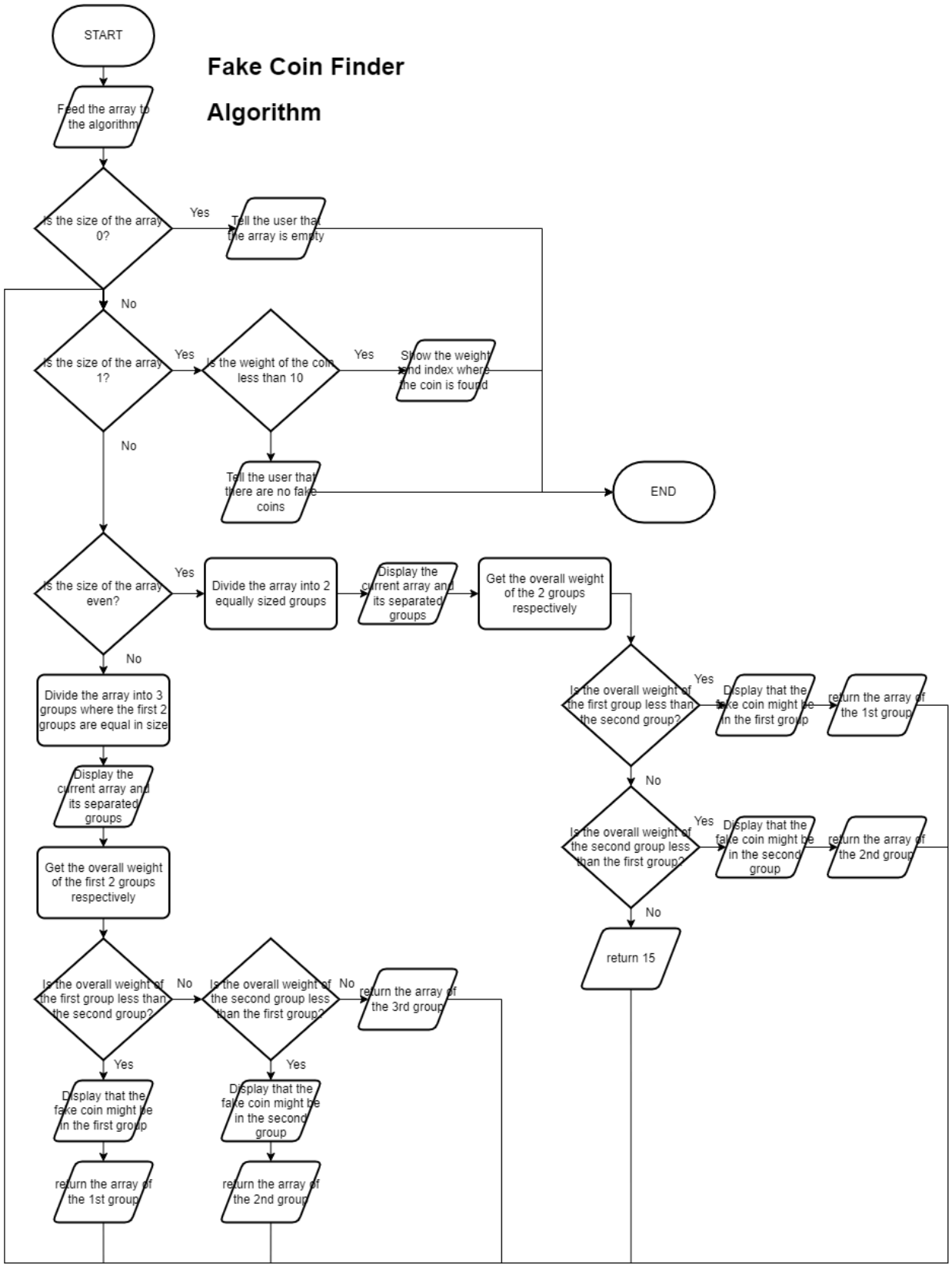
The simulation should be easy to understand and have a clear output for each step.

Error Handling

Invalid input detection (e.g., non-integer N).

SYSTEM DESIGN




Fake Coin Finder
Algorithm
START
Feed the array to the algorithm
Is the size of the array 0?
Yes
Tell the user that the array is empty
No
Is the size of the array 1?
Yes
Is the weight of the coin less than 10
Yes
Show the weight and index where the coin is found
No
Tell the user that there are no fake coins
END
Is the size of the array even?
Yes
Divide the array into 2 equally sized groups
Display the current array and its separated groups
Get the overall weight of the 2 groups respectively
No
Divide the array into 3 groups where the first 2 groups are equal in size
Display the current array and its separated groups
Get the overall weight of the first 2 groups respectively
Is the overall weight of the first group less than the second group?
No
Is the overall weight of the second group less than the first group?
No
return the array of the 3rd group
Yes
Display that the fake coin might be in the first group
return the array of the 1st group
Yes
Display that the fake coin might be in the second group
return the array of the 2nd group
Is the overall weight of the first group less than the second group?
Yes
Display that the fake coin might be in the first group
return the array of the 1st group
No
Is the overall weight of the second group less than the first group?
Yes
Display that the fake coin might be in the second group
return the array of the 2nd group
No
return 15

IMPLEMENTATION

Technologies Used

Programming Language:

C++, OOP - Programming Language Paradigm

Standard Libraries, Methods, Ect.

- <iostream>, <vector>, <cstdlib>, <ctime>
- rand() for weight generation.
- Classes: Weigh, CoinPouch, ErrorManagement.

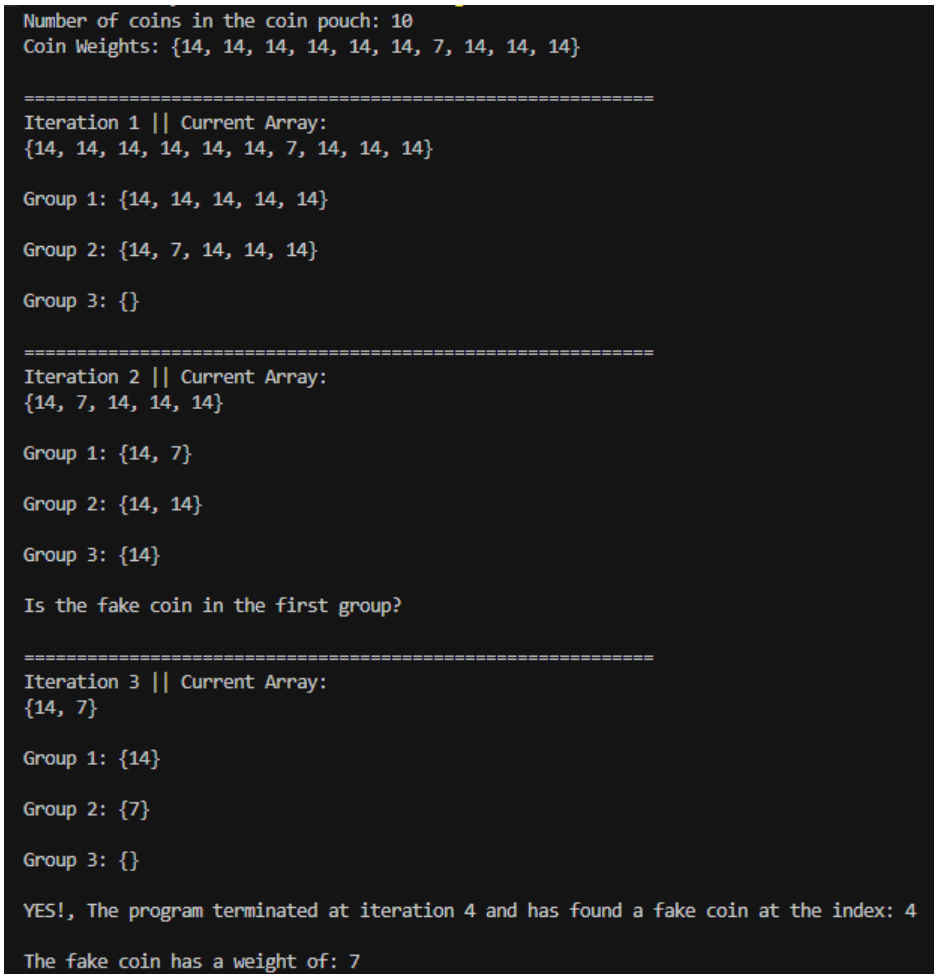
Development Tools

Compiler - g++

Text Editor/IDE - Visual Studio Code

Version Control - GitHub

Screenshots:



TESTING

FAKE COIN IN LIST

```
Number of coins in the coin pouch: 12
Coin Weights: {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 1}

============================================================
Iteration 1 || Current Array:
{14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 1}

Group 1: {14, 14, 14, 14, 14, 14}

Group 2: {14, 14, 14, 14, 14, 1}

Group 3: {}

============================================================
Iteration 2 || Current Array:
{14, 14, 14, 14, 14, 1}

Group 1: {14, 14, 14}

Group 2: {14, 14, 1}

Group 3: {}

============================================================
Iteration 3 || Current Array:
{14, 14, 1}

Group 1: {14}

Group 2: {14}

Group 3: {1}

Is the fake coin in the third group?

YES!, The program terminated at iteration 4 and has found a fake coin at the index: 6

The fake coin has a weight of: 1
```

INVALID INPUT

```
Number of coins in the coin pouch: a
Invalid input.
 Please enter a valid number: /
Invalid input.
 Please enter a valid number: 6
Coin Weights: {12, 12, 12, 12, 12, 12}

============================================================
Iteration 1 || Current Array:
{12, 12, 12, 12, 12, 12}

Group 1: {12, 12, 12}

Group 2: {12, 12, 12}

Group 3: {}

NO! There are no fake coins in this coin pouch
```

NO FAKE COIN IN LIST

```
Number of coins in the coin pouch: 4
Coin Weights: {10, 10, 10, 10}

============================================================
Iteration 1 || Current Array:
{10, 10, 10, 10}

Group 1: {10, 10}

Group 2: {10, 10}

Group 3: {}

NO! There are no fake coins in this coin pouch
```

EVEN LIST

```
Number of coins in the coin pouch: 12
Coin Weights: {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 1}

============================================================
Iteration 1 || Current Array:
{14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 1}

Group 1: {14, 14, 14, 14, 14, 14}

Group 2: {14, 14, 14, 14, 14, 1}

Group 3: {}

============================================================
Iteration 2 || Current Array:
{14, 14, 14, 14, 14, 1}

Group 1: {14, 14, 14}

Group 2: {14, 14, 1}

Group 3: {}

============================================================
Iteration 3 || Current Array:
{14, 14, 1}

Group 1: {14}

Group 2: {14}

Group 3: {1}

Is the fake coin in the third group?

YES!, The program terminated at iteration 4 and has found a fake coin at the index: 6

The fake coin has a weight of: 1
```

ODD LIST

```
Number of coins in the coin pouch: 7
Coin Weights: {10, 10, 10, 10, 10, 10, 6}

=========================================================
Iteration 1 || Current Array:
{10, 10, 10, 10, 10, 10, 6}

Group 1: {10, 10, 10}

Group 2: {10, 10, 10}

Group 3: {6}

Is the fake coin in the third group?

YES!, The program terminated at iteration 2 and has found a fake coin at the index: 6

The fake coin has a weight of: 6
```

USER MANUAL

Installation

1. Ensure g++ compiler is installed on your system
2. Open a terminal window and go to the directory where fakeCoinAlgo.cpp is installed
3. Compile with terminal command: g++ fakeCoinAlgo.cpp -o fakeCoin.
4. Run ./fakeCoin (Linux) or fakeCoin.exe (Windows).

Usage

- Enter N when prompted.
- View generated weights and step-by-step comparisons.
- Result shows fake coin index/weight or "NO fake."

CHALLENGES AND SOLUTIONS

Challenges Faced

- Handling odd-sized groups: Split into three partitions.
- Ensuring 50% chance of no fake coin.

Solutions Implemented

- Recursive weighing logic with index tracking.
- Create a method for weight generation and use a decide flag in addCoinToArr().

FUTURE ENHANCEMENTS

Potential Improvements to be made in the future for a much more interactive program could be;

- Support multiple fake coins.
- Visualize comparisons with a GUI.
- Optimize group division for faster convergence.

CONCLUSIONS

Summary

The program successfully simulated and simulates the fake coin problem, showing efficient divide-and-conquer strategies and error handling. This was done through implementing a divide-and-weigh algorithm to detect a fake coin from a set of coins.

Lessons Learned

- Importance of recursion in reducing problem complexity and algorithm efficiency.
- Implementing edge-case handling (e.g., no fake coin) for reliability.

REFERENCES

- https://github.com/TechTutorialHub/AlgorithmCodes/tree/main
- https://www.cs.uni.edu/~wallingf/teaching/cs3530/sessions/session17.html

APPENDICES

Appendix A:
N/A
Appendix B:

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

class ErrorManagement{
   public:
```

```
        void numGatekeep(int& num);
};

class Utility{
    public:
        void viewArr(vector<int> arr);
};

class Weigh : public Utility{
private:
    vector<int> coinPouch;

    void viewPartitions(vector<int> p1, vector<int> p2, vector<int> p3);
    void calculatePartitionWeight(vector<int> p1, vector<int> p2, int& w1, int& w2);

public:
    Weigh(const vector<int>& coins){
        this->coinPouch = coins;
    }

    int findFakeCoin(vector<int>& arr, int& iter, int& index);
};

class CoinPouch : public Utility{
private:
    int generate(int min, int max);

public:
    CoinPouch(){
        srand(static_cast<unsigned int>(time(0)));
    }

    void addCoinToArr(vector<int>& arr, int n);
    void displayAnswer(int w, int iter, int index);
};

int Weigh::findFakeCoin(vector<int>& arr, int& iter, int& index){
    iter++;
    int n = arr.size();
    //handles if the array is empty
    if (n == 0){
```

```
        cout << "Array is empty";
        return -1;
    }
    //base case to return the last coin found in the algorithm
    if (n == 1)
        return arr[0];

    //handle the array if it's size is even
    if (n%2 == 0) {
        //split the original array into 2 halves
        int half = n/2;
        vector<int> part1(arr.begin(), arr.begin() + half);
        vector<int> part2(arr.begin() + half, arr.end());

        //display the array of the current iterations and its partitions
        cout <<
"============================================================\n";
        cout << "Iteration "<< iter <<" || Current Array: \n";
        viewArr(arr);
        viewPartitions(part1, part2, {});

        //get the weight of the each partition
        int part1Weight = 0, part2Weight = 0;
        calculatePartitionWeight(part1, part2, part1Weight, part2Weight);

        /*weighing logic:
            if one side of the array weighs less than the other, feed it to the recursive call
            if the weight of the 2 sides are equal, then return 15 which would indicate that there are
no fake coins in the array
        */

        if (part1Weight < part2Weight) {
            return findFakeCoin(part1, iter, index);
        }
        else if (part2Weight < part1Weight) {
            index += 2;
            return findFakeCoin(part2, iter, index);
        }
        else
            return 15;
    }
```

```
    //handles the array if n is odd

    //split size of the array into 3 partitions where the first 2 partitions have equal sizes
    int third = n/3;
    int remainder = n%3;
    int midPart1 = third + (remainder > 0 ? 1:0);
    int midPart2 = midPart1 + third + (remainder > 0 ? 1:0);

    vector<int> part1(arr.begin(), arr.begin() + midPart1);
    vector<int> part2(arr.begin() + midPart1, arr.begin() + midPart2);
    vector<int> part3(arr.begin() + midPart2, arr.end());

    //display the array of the current iteration and its partitions
    cout << "============================================================\n";
    cout << "Iteration "<< iter <<" || Current Array: \n";
    viewArr(arr);
    viewPartitions(part1, part2, part3);

    //get the weights of the 1st and 2nd partitions
    int part1Weight = 0, part2Weight = 0;
    calculatePartitionWeight(part1, part2, part1Weight, part2Weight);

    /*weighing logic:
        if one side of the array weighs less than the other, feed it to the recursive call
        if the weight of the 2 sides are equal, then the fake coin must be in the 3rd partition, feed it
to the recursive call
    */

    if (part1Weight < part2Weight){
        cout << "Is the fake coin in the first group?\n\n";
        return findFakeCoin(part1, iter, index);
    }
    else if (part2Weight < part1Weight){
        cout << "Is the fake coin in the second group?\n\n";
        index+=midPart1;
        return findFakeCoin(part2, iter, index);
    }
    else{
        cout << "Is the fake coin in the third group?\n\n";
        index+=midPart2;
```

```
        return findFakeCoin(part3, iter, index);
    }
}

void Utility::viewArr(vector<int> arr){
    //go through the array and print each of its contents
    int n = arr.size();
    cout << "{";
    for(int i = 0; i < n; i++){
        cout << arr[i];
        if (i != n-1)
            cout << ", ";
    }
    cout << "} \n\n";
};

void Weigh::viewPartitions(vector<int> p1, vector<int> p2, vector<int> p3){
    cout << "Group 1: ";
    viewArr(p1);

    cout << "Group 2: ";
    viewArr(p2);

    cout << "Group 3: ";
    viewArr(p3);
}

void Weigh::calculatePartitionWeight(vector<int> p1, vector<int> p2, int& w1, int& w2){
    //get weight by adding the n and n+1 of the array
    for (int i = 0; i < p1.size(); i++){
        w1 += p1[i];
    }
    for (int i = 0; i < p2.size(); i++){
        w2 += p2[i];
    }
}

int CoinPouch::generate(int min, int max){
    //generate random number withe the rand functuion
    return rand() % (max - min + 1) + min;
};
```

```
void CoinPouch::addCoinToArr(vector<int>& arr, int n){
   /*there is a 50% chance thet there are no fake coins in the array
      this is generating a random number from 0-1
      if it returns 1 then the if statement would allow the for loop to add a fake coin in the array
with a random index
   */
   int decide = generate(0, 1);
   int fakeCoinIndex = generate(0, n-1);
   int original = generate(10, 20);

   for(int i = 0; i < n; i++){
      if(i == fakeCoinIndex && decide == 1)
         arr[i] = generate(1, 9);
      else
         arr[i] = original;
   }
}

void CoinPouch::displayAnswer(int w, int iter, int index){
   if(w >= 10){
      cout << "NO! There are no fake coins in this coin pouch";
      return;
   }

   cout << "YES!, The program terminated at iteration " << iter
       << " and has found a fake coin at the index: " << index << endl
       <<"\nThe fake coin has a weight of: " << w ;
}

void ErrorManagement::numGatekeep(int& num){
         bool loop = true;

         //If there is an invalid input, clear the flag and ask the user to input a new value
         while (loop) {
      cin >> num;

      if (cin.fail()) {
         cin.clear();
                        cin.ignore();
         cout << "Invalid input.\n Please enter a valid number: ";
```

```
        } else {
            loop = false;
        }
    }
}

int main() {
    ErrorManagement err;
    int n = 0, iter = 0, index = 0;
    cout << "Number of coins in the coin pouch: ";
    err.numGatekeep(n);

    vector<int> coins(n);
    CoinPouch pouch;

    //add random numbers to the coins array
    pouch.addCoinToArr(coins, n);

    //display the weights added to the array
    cout << "Coin Weights: ";
    pouch.viewArr(coins);

    //process the array to show what the weight of the fake coin is
    Weigh findFake(coins);
    int fakeWeight = findFake.findFakeCoin(coins, iter, index);

    pouch.displayAnswer(fakeWeight, iter, index);

    return 0;
}
```

Appendix C:
N/A