**Task 6**

1. **Detailed assumptions:**

* The program stores the weight of coins in coins dynamic array.
* The user can enter the number of coins in the problem and define the size of the array dynamically.
* There are two variables used as a memory when recursion happened.
* One variable store normal value of the coins (normal weight of non fake coin)
* The other variable stores the index of the fake coin
* The default value of the two variables is -1
* There are two functions used to calculate the fake coin index.
* the first function named balance used to compare two coins returning 1 when the right coin in the balance is heavier and -1 if the left is heavier and 0 if they are equal in size.
* the second function is getFakeIndex it returns void but can change the values of the external memory in dynamic programming.
* The function takes the first index and the second index
* The number of coins is assumed to be greater than 2 so we can detect the fake coin (n>2).
* The user is asked to enter the number of coins, the weight of non-fake coin, the weight of fake coin and the index of fake coin.
* The user data stored in the dynamic array called coins
* The function get fake index executes.
* The index is stored in fakeIndex variable, and the weight of the original coins is stored in normalValue variable that is used by function to be compared with other functions using divide and conquer and memory (dynamic programming).
* The order and index of coin is printed for user after execution a function.

1. **Problem Description:**

* There are 12 coins identical in appearance.
* either all are genuine or exactly one of them is fake.
* It is unknown whether the fake coin is lighter or heavier than the genuine one.
* You have a two-pan balance scale without weights.
* The problem is to find whether all the coins are genuine and, if not, to find the fake coin and establish whether it is lighter or heavier than the genuine ones.

1. **Explaining solution:**

Dynamic programming:

* The dynamic programming use divide and conquer technique and memory to help in optimization and ignore useless recursions after founding the fake coin if found.
* Two recursions happen for every function call dividing problem into two problems until the problem size be 3 or 2 then solving the small problems.

**Explanation:**

* The code create variable n and dynamic array coins at the beginning of the execution of the code.
* the user is asked to enter some parameters to initiate the dynamic array and the variables
* the function divides until the size of the problem is 3 or 2 then it searches for the fake coin and store the result in fakeIndex variable.
* The output after finishing the function is written in fakeIndex variable which is a variable in the public class and accessed anywhere, while the input from user is stored in fake\_coin\_index variable which is a local variable used to initialize coins array only and can’t be accessed by getFakeIndex
* After finishing the function the value of calculated fakeIndex is printed to the user.

1. **Pseudo code:**

function balance(x, y):

if x > y:

return 1

else if y > x:

return -1

else:

return 0

function getFakeIndex(startIndex, endIndex):

if fakeIndex is -1:

number\_of\_items = endIndex - startIndex + 1

if number\_of\_items is 3:

firstIndex = startIndex

secondIndex = firstIndex + 1

thirdIndex = secondIndex + 1

if normalValue is not -1:

if balance(coins[firstIndex], normalValue) is not 0:

fakeIndex = firstIndex

else if balance(coins[secondIndex], normalValue) is not 0:

fakeIndex = secondIndex

else if balance(coins[thirdIndex], normalValue) is not 0:

fakeIndex = thirdIndex

else if normalValue is -1:

if balance(coins[firstIndex], coins[secondIndex]) is 0:

normalValue = coins[firstIndex]

if balance(coins[firstIndex], coins[thirdIndex]) is not 0:

fakeIndex = thirdIndex

else if balance(coins[firstIndex], coins[thirdIndex]) is 0:

normalValue = coins[firstIndex]

fakeIndex = secondIndex

else if balance(coins[secondIndex], coins[thirdIndex]) is 0:

normalValue = coins[secondIndex]

fakeIndex = firstIndex

else if number\_of\_items is 2:

firstIndex = startIndex

secondIndex = firstIndex + 1

if normalValue is not -1:

if balance(coins[firstIndex], normalValue) is not 0:

fakeIndex = firstIndex

else if balance(coins[secondIndex], normalValue) is not 0:

fakeIndex = secondIndex

else if normalValue is -1:

thirdIndex = secondIndex + 1

if balance(coins[firstIndex], coins[secondIndex]) is 0:

normalValue = coins[firstIndex]

else if balance(coins[firstIndex], coins[thirdIndex]) is 0:

normalValue = coins[firstIndex]

fakeIndex = secondIndex

else if balance(coins[secondIndex], coins[thirdIndex]) is 0:

normalValue = secondIndex

fakeIndex = firstIndex

else:

midIndex = (startIndex + endIndex) / 2

getFakeIndex(startIndex, midIndex)

getFakeIndex(midIndex, endIndex)

function main():

user\_input\_num\_of\_coins = get input from user

user\_input\_Index\_of\_fake = get input from user

user\_input\_Weight\_of\_normal\_coin = get input from user

user\_input\_Weight\_of\_fake\_coin = get input from user

n = user\_input\_num\_of\_coins

fake\_coin\_index = user\_input\_Index\_of\_fake

normal\_coin\_weight = user\_input\_Weight\_of\_normal\_coin

fake\_coin\_weight = user\_input\_Weight\_of\_fake\_coin

coins = create an array of size n and fill it with normal\_coin\_weight

coins[fake\_coin\_index] = fake\_coin\_weight

getFakeIndex(0, n - 1)

if fakeIndex is -1:

print "There is no fake coin"

else:

if fakeIndex is 0:

print "The fake coin is the first coin"

else if fakeIndex is 1:

print "The fake coin is the second coin"

else if fakeIndex is 2:

print "The fake coin is the third coin"

else:

print "The fake coin is the " + (fake\_coin\_index + 1) + "th coin"

print "The fake coin has an index of " + fakeIndex

1. **Implementation:**

**Code:**

import java.util.Arrays;

import java.util.Scanner;

/\*user can change the values of n,NORMAL\_SIZE,HEAVY\_COIN and LIGHT\_COIN in the class Fake\_coin\_problem

\*\*user can change the values of fake\_coin\_index and fake\_coin\_weight also in the main function

\*\*changing those values will set the problem dynamically

\*/

public class Main {

static int n = 12;

/\*define dynamic array of coins weights with dynamic size n\*/

static int[] coins;

/\*define memory variables used to solve the problem using divide and conquer (dynamic programming technique)\*/

/\*memory variable used by getFakeIndex function to store the weight of not fake coin if the weight is known by function automatically\*/

static int normalValue = -1;

/\*memory variable used by getFakeIndex function to store fake index and memorize it to stop recursive calls if found\*/

static int fakeIndex = -1;

/\*function to compare two coins\*/

static int balance(int x, int y){

if(x>y){/\*if the coin on the right side of the balance is bigger return 1\*/

return 1;

}else if(y>x){/\*if the coin on the left side of the balance is bigger return 1\*/

return -1;

} else{/\*if the two coins are equal return 0\*/

return 0;

}

}

/\*the function that calculates the fake index of the coin and store it in fakeIndex variable if found

\*if the fake coin not found, leave its value at -1

\* \*/

private static void getFakeIndex(int startIndex,int endIndex) {

if(fakeIndex == -1) {/\*if the fake coin dont found do the following\*/

/\*determining the number of items that may be 3 or 2 only in every divided problem to solve other wise divide\*/

int number\_of\_items = endIndex - startIndex + 1;

if (number\_of\_items == 3) {/\*if the number of items in the divided problem = 3\*/

int firstIndex, secondIndex, thirdIndex;

firstIndex = startIndex;

secondIndex = firstIndex + 1;

thirdIndex = secondIndex + 1;

if (normalValue != -1) {

if (balance(coins[firstIndex], normalValue) != 0) {

fakeIndex = firstIndex;

} else if (balance(coins[secondIndex], normalValue) != 0) {

fakeIndex = secondIndex;

} else if (balance(coins[thirdIndex], normalValue) != 0) {

fakeIndex = thirdIndex;

}

} else if (normalValue == -1) {

if (balance(coins[firstIndex], coins[secondIndex]) == 0) {

normalValue = coins[firstIndex];

if (balance(coins[firstIndex], coins[thirdIndex]) != 0) {

fakeIndex = thirdIndex;

}

} else if (balance(coins[firstIndex], coins[thirdIndex]) == 0) {

normalValue = coins[firstIndex];

fakeIndex = secondIndex;

} else if (balance(coins[secondIndex], coins[thirdIndex]) == 0) {

normalValue = coins[secondIndex];

fakeIndex = firstIndex;

}

}

} else if (number\_of\_items == 2) {/\*if the number of items in the divided problem = 2\*/

int firstIndex = startIndex;

int secondIndex = firstIndex + 1;

if (normalValue != -1) {

if (balance(coins[firstIndex], normalValue) != 0) {

fakeIndex = firstIndex;

} else if (balance(coins[secondIndex], normalValue) != 0) {

fakeIndex = secondIndex;

}

} else if (normalValue == -1) {

/\*cant predict so we assume third one exists\*/

int thirdIndex = secondIndex+1;

if(balance(coins[firstIndex], coins[secondIndex]) == 0){

normalValue = coins[firstIndex];

}else if(balance(coins[firstIndex], coins[thirdIndex]) == 0){

normalValue = coins[firstIndex];

fakeIndex = secondIndex;

}else if(balance(coins[secondIndex], coins[thirdIndex]) == 0){

normalValue = secondIndex;

fakeIndex = firstIndex;

}

}

} else {//divide problem

int midIndex = (startIndex + endIndex) / 2;

getFakeIndex(startIndex, midIndex);

getFakeIndex(midIndex, endIndex);

}

}/\*if the fake coin found end the function\*/

}

public static void main(String[] args) {

/\*get the necessary inputs from user\*/

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of coins: ");

int user\_input\_num\_of\_coins = scanner.nextInt();

System.out.print("Enter the index of the fake coin: ");

int user\_input\_Index\_of\_fake = scanner.nextInt();

System.out.print("Enter the weight of normal coin: ");

int user\_input\_Weight\_of\_normal\_coin = scanner.nextInt();

System.out.print("Enter the weight of fake coin: ");

int user\_input\_Weight\_of\_fake\_coin = scanner.nextInt();

/\*initialize program variables with the values from user\*/

n = user\_input\_num\_of\_coins;

int fake\_coin\_index = user\_input\_Index\_of\_fake;

int normal\_coin\_weight = user\_input\_Weight\_of\_normal\_coin;

int fake\_coin\_weight = user\_input\_Weight\_of\_fake\_coin;

/\*calculating the size of the dynamic array of weight of coins from the input of user\*/

coins = new int[n];

//fill the dynamic array with the values of normal\_coin\_weight variable to fill it with non-fake coins

Arrays.fill(coins,normal\_coin\_weight);

/\*choose the index of fake coin and the weight of it \*/

coins[fake\_coin\_index] = fake\_coin\_weight;

/\*getting the fake coin\*/

getFakeIndex(0,coins.length-1);

/\*printing the fake coin after putting its index to fakeIndex variable\*/

if(fakeIndex == -1){/\*fake coin not found\*/

System.out.println("there is no fake coin");

}else{

if(fakeIndex == 0){/\*first\*/

System.out.println("the fake coin is the first coin");

} else if (fakeIndex == 1) {/\*second\*/

System.out.println("the fake coin is the second coin");

} else if (fakeIndex == 2) {/\*third\*/

System.out.println("the fake coin is the third coin");

} else{/\*other\*/

System.out.print("the fake coin is the ");

System.out.print(fake\_coin\_index+1);

System.out.println("th coin");

}

System.out.print("the fake coin has index of ");

System.out.println(fakeIndex);

}

}

}

Input and output form to solve task 6:

A screen shot of a computer

Description automatically generated with medium confidence

**When there is no fake coin:**

The weights are equal so the function searches for fake coin but the index still equals to -1 (no index found) so the user gets there is no fake coin.

A screenshot of a computer

Description automatically generated with medium confidence

1. **Time and Space Complexity:**

**Summary:**

Time complexity: O(log(n))

Space complexity: O(log(n))

**Explanation:**

The time complexity of the getFakeIndex algorithm can be analyzed as follows:

Let's assume the number of coins is n. In each recursive call to getFakeIndex, the problem is divided into two subproblems of size approximately n/2. This division continues until the base case is reached, where the number of items is either 3 or 2. Therefore, the number of recursive calls is logarithmic with base 2, which can be represented as log(n).

Within each recursive call, there are constant time operations such as comparisons, assignments, and function calls. Therefore, the time complexity within each call can be considered constant, denoted as O(1).

Hence, the overall time complexity of the getFakeIndex algorithm is O(log(n)), where n is the number of coins.

Regarding the space complexity of the algorithm, the main space-consuming factor is the recursion stack. In each recursive call, a new frame is added to the stack to store the local variables and return address. Since the maximum depth of the recursion is log(n), the space complexity of the recursion stack is O(log(n)).

Additionally, there are a few memory variables (normalValue and fakeIndex) that are used to store intermediate results. These variables require constant space, denoted as O(1).

Therefore, the overall space complexity of the getFakeIndex algorithm is O(log(n)).

It's important to note that these analyses assume that the size of the coins array (n) is a power of 2. If n is not a power of 2, the time and space complexity would still be close to O(log(n)) but with a slightly larger constant factor due to the division of uneven subproblems.

1. **Inputs and outputs:**

**Input:** number of coins, index of the fake coin, weight of normal coin and weight of fake coin.

**Output:** fake coin position if found, if not prints there is no fake coin.

1. **Conclusions:**

Our target is to reduce the time complexity from O(n) with space complexity of O(1) to time complexity of O(log(n)) and space complexity of O(log(n)) which uses small space and time to give us an effective solution in addition to using dynamicprogramming to make the best casebetter and uses lower space and memory to optimize the problem.

1. **References:**

*https://centerofmathematics.blogspot.com/2017/09/think-thursday-9-21-17-twelve-coins.html*