Name: Kwa Li Ying (1003833)
CSE Class: CI03

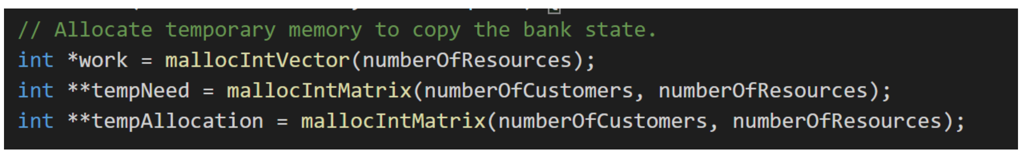
CSE Lab 2 Report

Complexity Analysis

The time complexity of the safety algorithm in the Banker's Algorithm is $O(mn^2)$, where m is the number of resources and n is the number of customers.

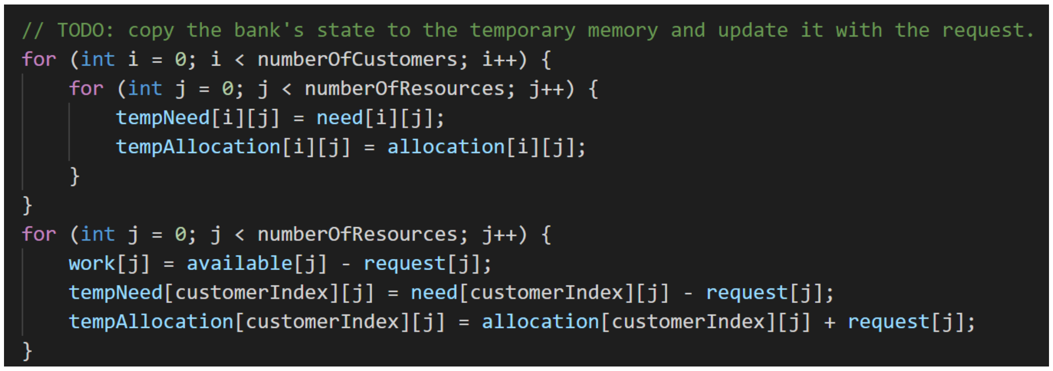
In the checkSafe() part of the code, a few things happen sequentially.

The allocation of space takes constant time and gives a complexity of O(1).



```
// Allocate temporary memory to copy the bank state.
int *work = mallocIntVector(numberOfResources);
int **tempNeed = mallocIntMatrix(numberOfCustomers, numberOfResources);
int **tempAllocation = mallocIntMatrix(numberOfCustomers, numberOfResources);
```

The first TODO (Copy the bank's state to the temporary memory and update it with the request) will give a complexity of O(mn + m) because it is just an execution of the 2 loops – m x n times for the first loop and m times for the second.

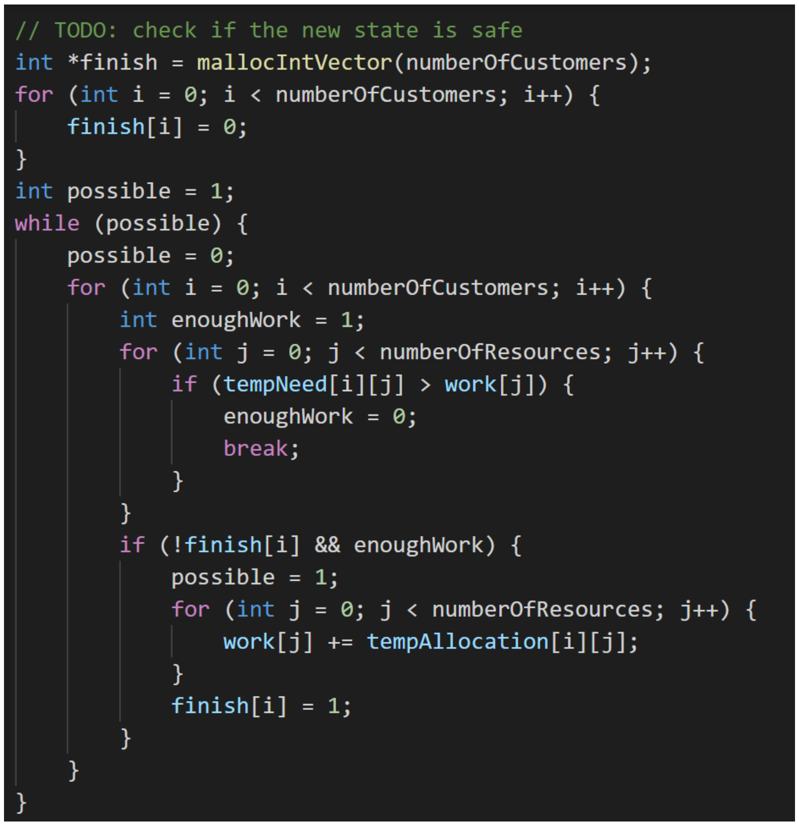


```
// TODO: copy the bank's state to the temporary memory and update it with the request.
for (int i = 0; i < numberOfCustomers; i++) {
    for (int j = 0; j < numberOfResources; j++) {
        tempNeed[i][j] = need[i][j];
        tempAllocation[i][j] = allocation[i][j];
    }
}
for (int j = 0; j < numberOfResources; j++) {
    work[j] = available[j] - request[j];
    tempNeed[customerIndex][j] = need[customerIndex][j] - request[j];
    tempAllocation[customerIndex][j] = allocation[customerIndex][j] + request[j];
}
```

The next TODO contains the main part of the safety algorithm.

Looping through the finish-array to set finish(all) = false gives a runtime of O(n).

Name: Kwa Li Ying (1003833)
CSE Class: CI03

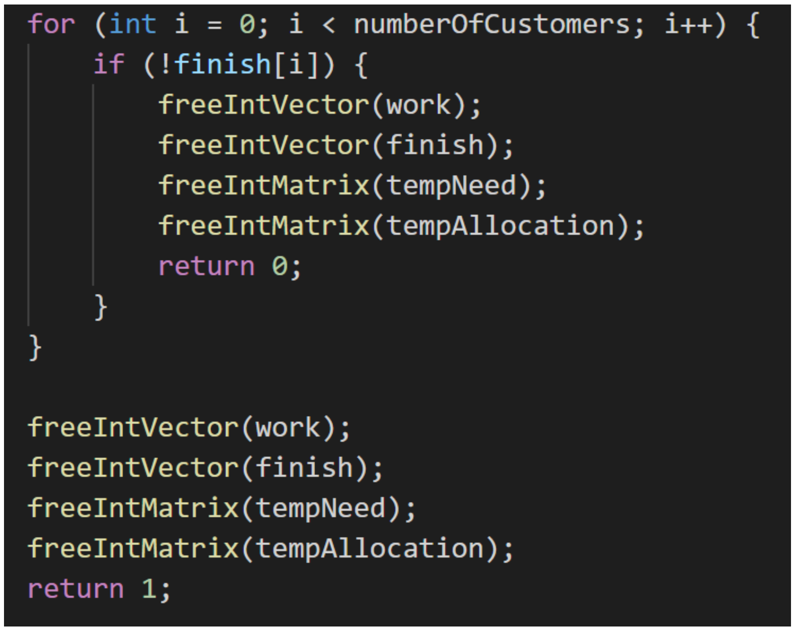


```c
// TODO: check if the new state is safe
int *finish = mallocIntVector(numberOfCustomers);
for (int i = 0; i < numberOfCustomers; i++) {
    finish[i] = 0;
}
int possible = 1;
while (possible) {
    possible = 0;
    for (int i = 0; i < numberOfCustomers; i++) {
        int enoughWork = 1;
        for (int j = 0; j < numberOfResources; j++) {
            if (tempNeed[i][j] > work[j]) {
                enoughWork = 0;
                break;
            }
        }
        if (!finish[i] && enoughWork) {
            possible = 1;
            for (int j = 0; j < numberOfResources; j++) {
                work[j] += tempAllocation[i][j];
            }
            finish[i] = 1;
        }
    }
}
```

Looking at the nested for-loop(i < numberOfCustomers) in the outer while-loop, the innermost for-loop(j < numberOfResources) that checks the enoughWork condition has a runtime of O(n), and the if-block right after it also has a runtime of O(n). These combined, give a runtime of O(2n) = O(n). When multiplied by the outer for-loop such that this is checked for each customer, this gives a complexity of O(mn).

In the worst-case runtime, every loop in the while-loop sets the possible variable to 1, while only one customer[i]'s finish[i] gets set to true each time. This allows the while-loop to run n times (one for each customer), since at each time there is just nice one customer whose condition is satisfied and sets one finish to true in the finish-array, so that there is one less while-loop to continue running. With this outer while-loop combined, the complexity becomes O(mn^2).

Name: Kwa Li Ying (1003833)
CSE Class: CI03


```
for (int i = 0; i < numberOfCustomers; i++) {
    if (!finish[i]) {
        freeIntVector(work);
        freeIntVector(finish);
        freeIntMatrix(tempNeed);
        freeIntMatrix(tempAllocation);
        return 0;
    }
}

freeIntVector(work);
freeIntVector(finish);
freeIntMatrix(tempNeed);
freeIntMatrix(tempAllocation);
return 1;
```

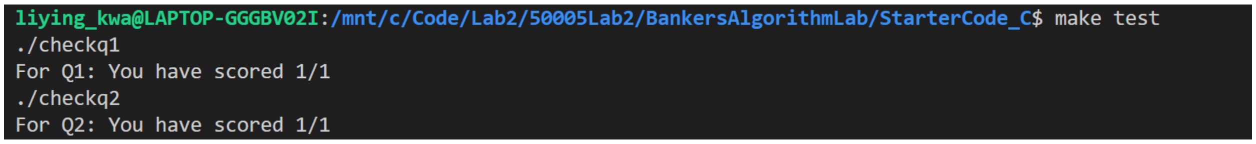
The last part that checks finish(all) == true gives a complexity of O(n), but this is added instead of multiplied as it is not nested.

Combining the runtime of the whole safety segment, we have O(mn + m + n + mn^2 + n) = O(mn^2). Therefore the time complexity of Banker's Algorithm is O(mn^2).

Name: Kwa Li Ying (1003833)
CSE Class: CI03

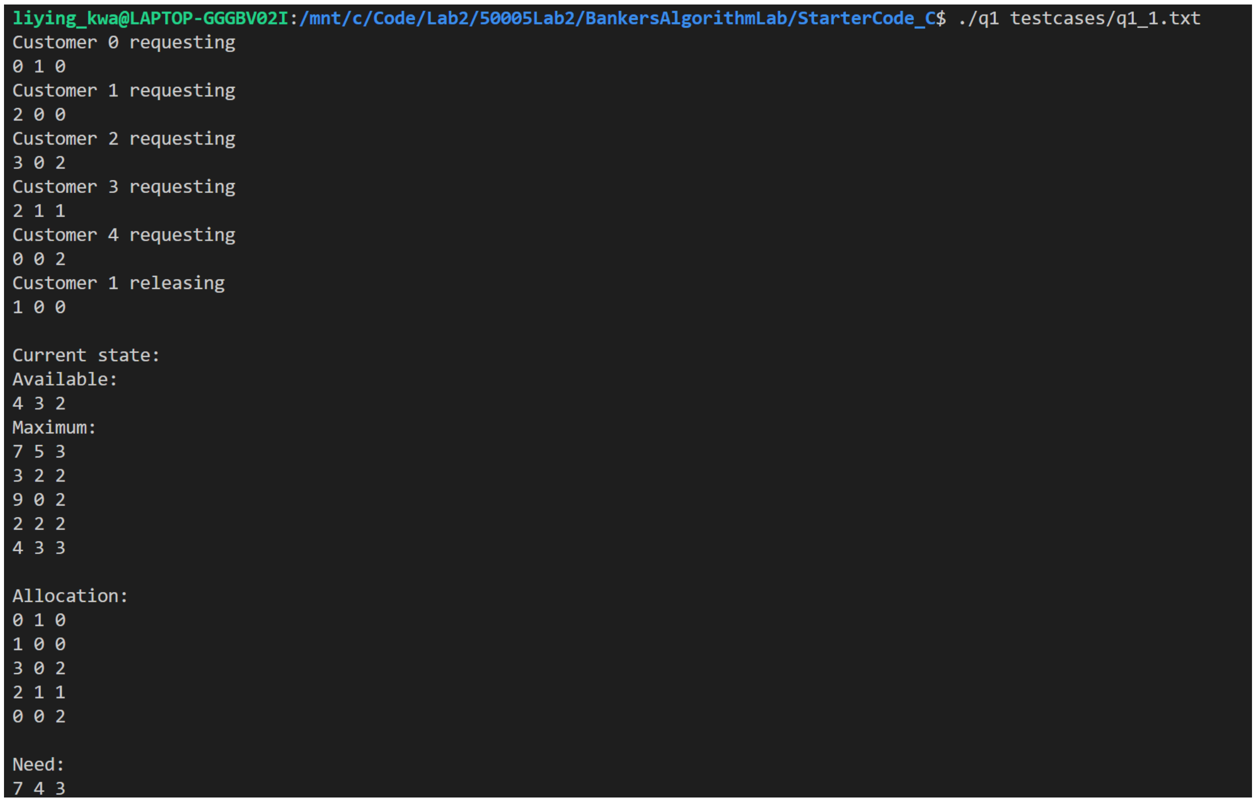
Result Screenshots

Make Test:



```
liying_kwa@LAPTOP-GGGBV02I:/mnt/c/Code/Lab2/50005Lab2/BankersAlgorithmLab/StarterCode_C$ make test
./checkq1
For Q1: You have scored 1/1
./checkq2
For Q2: You have scored 1/1
```

Q1:

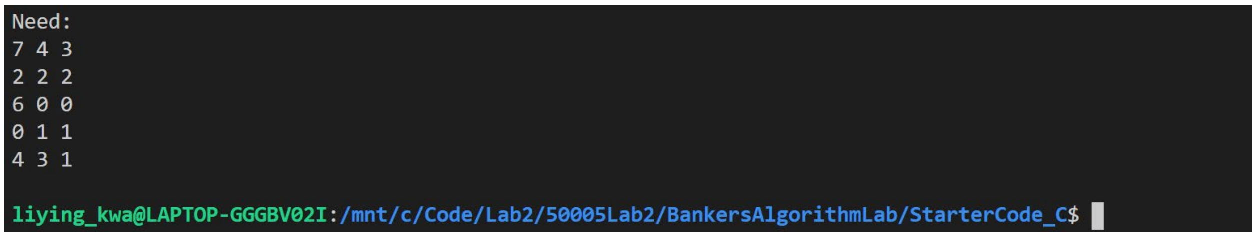


```
liying_kwa@LAPTOP-GGGBV02I:/mnt/c/Code/Lab2/50005Lab2/BankersAlgorithmLab/StarterCode_C$ ./q1 testcases/q1_1.txt
Customer 0 requesting
0 1 0
Customer 1 requesting
2 0 0
Customer 2 requesting
3 0 2
Customer 3 requesting
2 1 1
Customer 4 requesting
0 0 2
Customer 1 releasing
1 0 0

Current state:
Available:
4 3 2
Maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation:
0 1 0
1 0 0
3 0 2
2 1 1
0 0 2

Need:
7 4 3
```

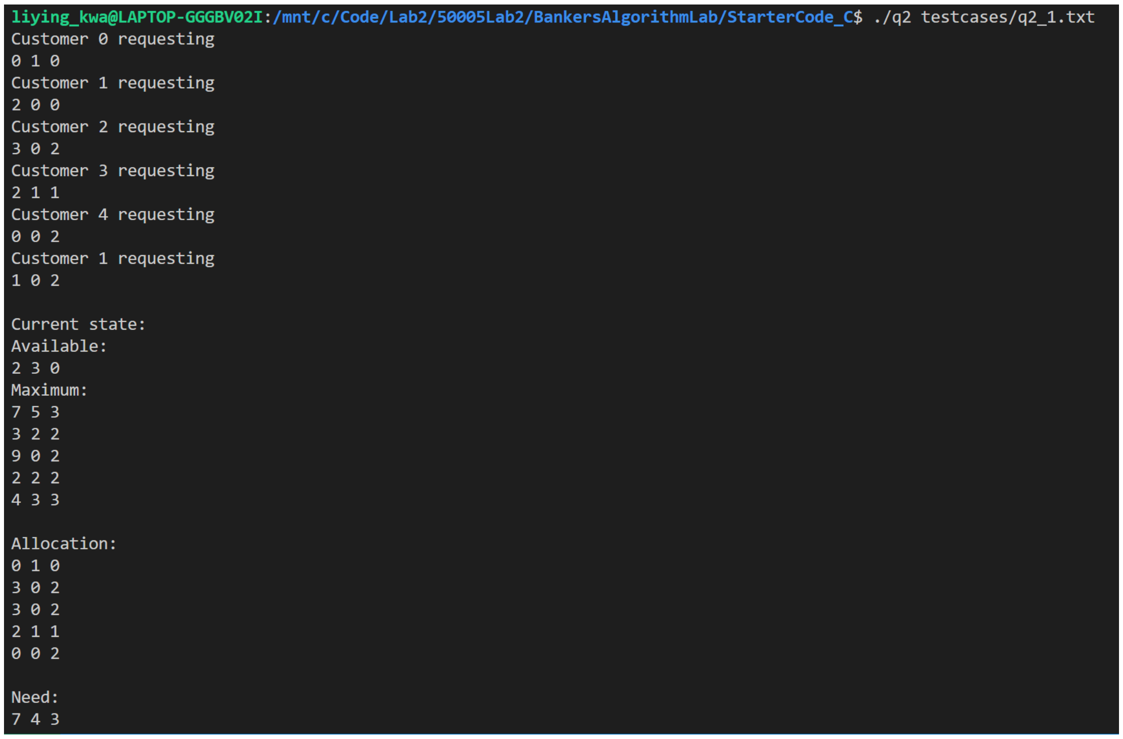


```
Need:
7 4 3
2 2 2
6 0 0
0 1 1
4 3 1

liying_kwa@LAPTOP-GGGBV02I:/mnt/c/Code/Lab2/50005Lab2/BankersAlgorithmLab/StarterCode_C$
```

Name: Kwa Li Ying (1003833)
CSE Class: CI03

Q2:

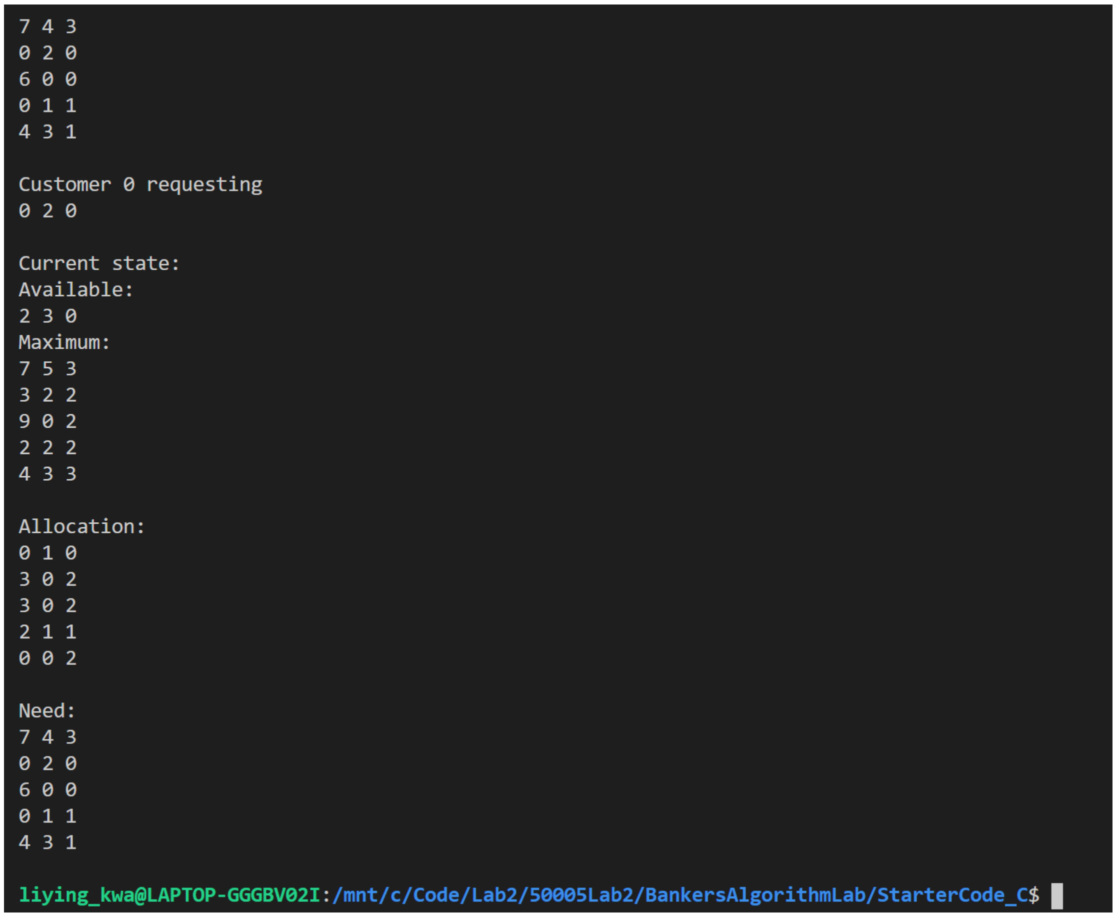


```
liying_kwa@LAPTOP-GGGBV02I:/mnt/c/Code/Lab2/50005Lab2/BankersAlgorithmLab/StarterCode_C$ ./q2 testcases/q2_1.txt
Customer 0 requesting
0 1 0
Customer 1 requesting
2 0 0
Customer 2 requesting
3 0 2
Customer 3 requesting
2 1 1
Customer 4 requesting
0 0 2
Customer 1 requesting
1 0 2

Current state:
Available:
2 3 0
Maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2

Need:
7 4 3
```



```
7 4 3
0 2 0
6 0 0
0 1 1
4 3 1

Customer 0 requesting
0 2 0

Current state:
Available:
2 3 0
Maximum:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Allocation:
0 1 0
3 0 2
3 0 2
2 1 1
0 0 2

Need:
7 4 3
0 2 0
6 0 0
0 1 1
4 3 1

liying_kwa@LAPTOP-GGGBV02I:/mnt/c/Code/Lab2/50005Lab2/BankersAlgorithmLab/StarterCode_C$
```