```
shiinx@LAPTOP-LU3JM1KO:~/50.005/50005Lab2/BankersAlgorithmLab/StarterCode_Java$ java TestBankQ2 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]
[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]
```

```
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]

shiinx@LAPTOP-LU3JM1KO:~/50.005/50005Lab2/BankersAlgorithmLab/StarterCode_Java$
```

Considering time complexity of only Banker's Algorithm, we will only be looking at the function checkSafe() where the algorithm lie. Other functions are largely for request, release and setting up and will be ignored when computing the time complexity.

Code snippet taken from Banker.java:

```
private synchronized boolean checkSafe(int customerIndex, int[] request) {
    int[] t_avail = available.clone();
    int[][] t_need = copy2DArray(need);
    int[][] t_allocation = copy2DArray(allocation);


    for(int i = 0; i < numberOfResources; i++){
      t_avail[i] -= request[i];
      t_need[customerIndex][i] -= request[i];
      t_allocation[customerIndex][i] += request[i];
    }

    int[] work = t_avail.clone();
    boolean[] isFinished = new boolean[numberOfCustomers];
    Arrays.fill(isFinished, Boolean.FALSE);
    boolean isPossible = true;

    while(isPossible){
      isPossible = false;
      for(int i = 0; i < numberOfCustomers; i++){
        if(!isFinished[i] && compareArray(t_need[i],work)){
          isPossible = true;
          for(int j = 0; j < numberOfResources; j++){
            work[j] += t_allocation[i][j];
          }
          isFinished[i] = true;
        }
      }
    }

    return areAllTrue(isFinished);
  }
```

Main loop and code to inspect for worst case time complexity

Inner for loop "`for(int j = 0; j < numberOfResources; j++)`" , contributes time complexity **O(numberOfResoures)**.

Outer for loop "`for(int i = 0; i < numberOfCustomers; i++)`", contributes time complexity **O(numberOfCustomers)**.

From online sources, "If there are n processes, then in the worst case the processes are ordered such that each iteration of the banker's algorithm must evaluate all remaining processes before the last one satisfies". Hence, the while loop will also contribute **O(numberOfCustomers)**.

As this is a **for loop nested within a for loop that is nested within a while loop**, all complexity can be multiplied together to get the time complexity.

Therefore, the time complexity for the Banker algorithm is **O(numberOfCustomers*numberOfCustomers*numberOfResources).**