Ashlyn Goh 1002840 Cl02

50.005 – Lab 3

Q1: Implement a basic bank system (20 marks)

The output is as follows:

```
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]
[2, 2, 2]
[6, 0, 0]
[0, 1, 1]
[4, 3, 1]
```

Q2: Implementing a Safety Check algorithm (20 marks)

The output is as follows:

```
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]
[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]
```

Q3: Discuss about the complexity of Banker's algorithm (10 marks)

Let the number of customers (processes) be n and the number of resources be m. In this analysis, we will be using the pseudo-code given in the Lab 3 handout.

At the start of the algorithm, the variables are initialised and assigned values.

```
1  boolean checkSafe(customerNumber, request){
2      temp_avail = available - request;
3      temp_need(customerNumber) = need - request;
4      temp_allocation(customerNumber) = allocation + request;
5      work = temp_avail;
6      finish(all) = false;
7      possible = true;
```

In line 2, `temp_avail` is an array of size m so the subtraction will take O(m).
In line 3-4, `temp_need` and `temp_allocation` are 2-d arrays of size n*m so the operation will take O(n*m).
In line 5, `work` is an array of size m so the assignment will take O(m).
In line 6, `finish` is an array of size n so the assignment will take O(n).
In line 7, it takes O(1) time.

In total, the operations before the while loop takes O**(n*m).**

```
9      while(possible){
10         possible = false;
11         for (customer Ci = 1:n){
12             if (finish(Ci) == false && temp_need(Ci) <= work){
13                 possible = true;
14                 work += temp_allocation(Ci);
15                 finish(Ci) = true;
16             }
17         }
18     }
19     return (finish(all) == true);
20 }
```

From the code snippet above, we observe that the for-loop repeats for **n times** (once for each customer). Within the for-loop in line 12, we check if `temp_need(Ci) <= work`. Since we are comparing between integer arrays of size m, the time complexity for the comparison is O(m). Within the if clause in line 14, we have `work += temp_allocation(Ci).` Since work is an array of size m, the element-wise addition would take O(m). In total, within the for-loop, we have O(m+m) = O(m). The time complexity for lines 11 to 17 would thus be O(n*m).

Lastly, this for loop is nested within a while loop. In the worst case, the while loop would run for n times because possible is set to true each time inside the if clause. Hence, the total time complexity for lines 9 to 18 would be **O(n*n*m)**.

In line 19, since finish is an array of size n it would take O(n) time to check if the values are true.

Therefore, the time complexity of Banker's algorithm is **O(n*n*m)** where the non-dominant terms are ignored.