Ques 1

Part a

Processes	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	0 0 0 0	1 5 2 0
P1	1 0 0 0	1 7 5 0	0 7 5 0	
P2	1 3 5 4	2 3 5 6	1 0 0 2	
P3	0 6 3 2	0 6 5 2	0 0 2 0	
P4	0 0 1 4	0 6 5 6	0 6 4 2	

Firstly, we can execute process P0. After execution, Available becomes (1,5,3,2). Next, we can execute process P2. After execution, Available becomes (2,8,8,6). Next, we can execute process P1. After execution, Available becomes (3,8,8,6). Next, we can execute process P3. After execution, Available becomes (3,14,11,8). Next, we can execute process P4. After execution, Available becomes (3,14,12,12). Hence the processes can be safely executed <P0,P2,P1,P3,P4>.

Part b

Processes	Allocation	Max	Need	Available
	A B C D	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	0 0 0 0	1 5 2 0
P1	1 0 0 0	1 7 5 2	0 7 5 2	
P2	1 3 5 4	2 3 5 6	1 0 0 2	
P3	0 6 3 2	0 6 5 2	0 0 2 0	
P4	0 0 1 4	0 6 5 6	0 6 4 2	

Process P1 requests for (0,4,2,1). But this can not be given immediately since resource D has 0 available instances, but P1 requires 1.

Ques 2

The safety algorithm consists of two nested for loops where the first for loop runs for n iterations from 0 to n-1 and the second (nested) for loop runs for at most m iterations for each iteration of the parent for loop that is it runs for m * n times. Thus the total number of operations required would be (m * n) * n = m * n^2 .

Ques 3

Part a

All the three data structures: Max, Allocation, Need are m*n 2-D arrays; which can be stored as a global or a local variable.

Part b

The processes would modify these arrays as and when they either request resources and are granted, or when they release these resources after the completion. This would ensure that the algorithm works correctly.

To protect the data structure, we can use locks whenever the process wants to acquire the resources and modify the data structure. That is to say, the acquiring and releasing of the resources along with the modification of the data structures should be done as a part of critical section.

Ques 4

We initially allocate (m-1) resources to each of the p processes (one less than the required amount).

Total allocated resources = p * (m-1)

To start any process, we require at least 1 more resource. So, the minimum number of resources required = p * (m-1) + 1.

This is also so to prevent deadlock condition. Hence we need

$$r \geq p * (m-1) + 1$$

Ques 5

```
semaphore sem_trans; // initialized to 1

void transaction(Account from, Account to, double amount)

{

        mutex lock1, lock2;

        lock1 = get_lock(from);

        lock2 = get_lock(to);

        //Entry Section

        wait(sem_trans);

        //Critical Section

        acquire(lock1);

        acquire(lock2);

        withdraw(from, amount);

        deposit(to, amount);

        release(lock2);

        release(lock1);

        //End of Critical Section

        signal(sem_trans);

}
```