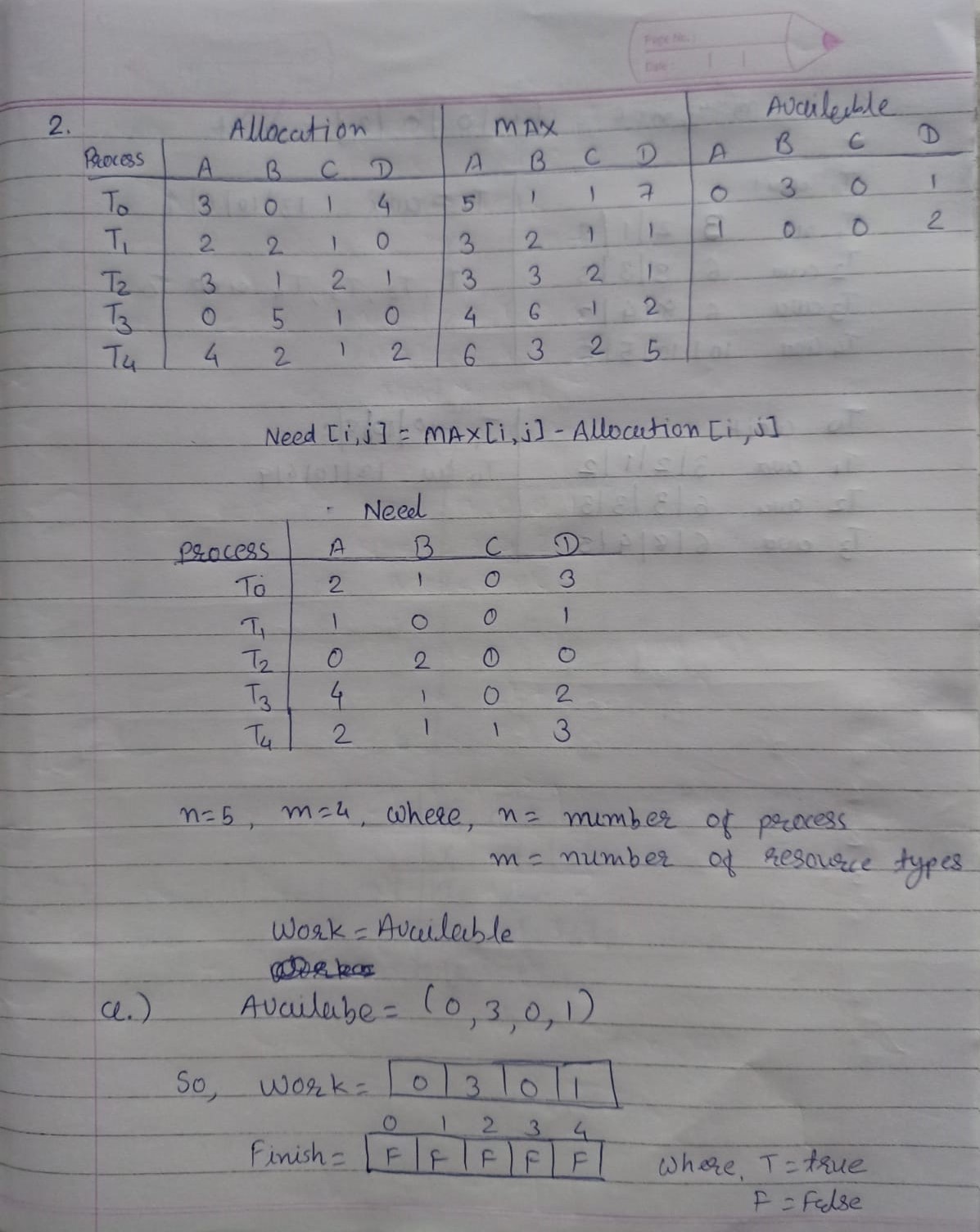
**1.**

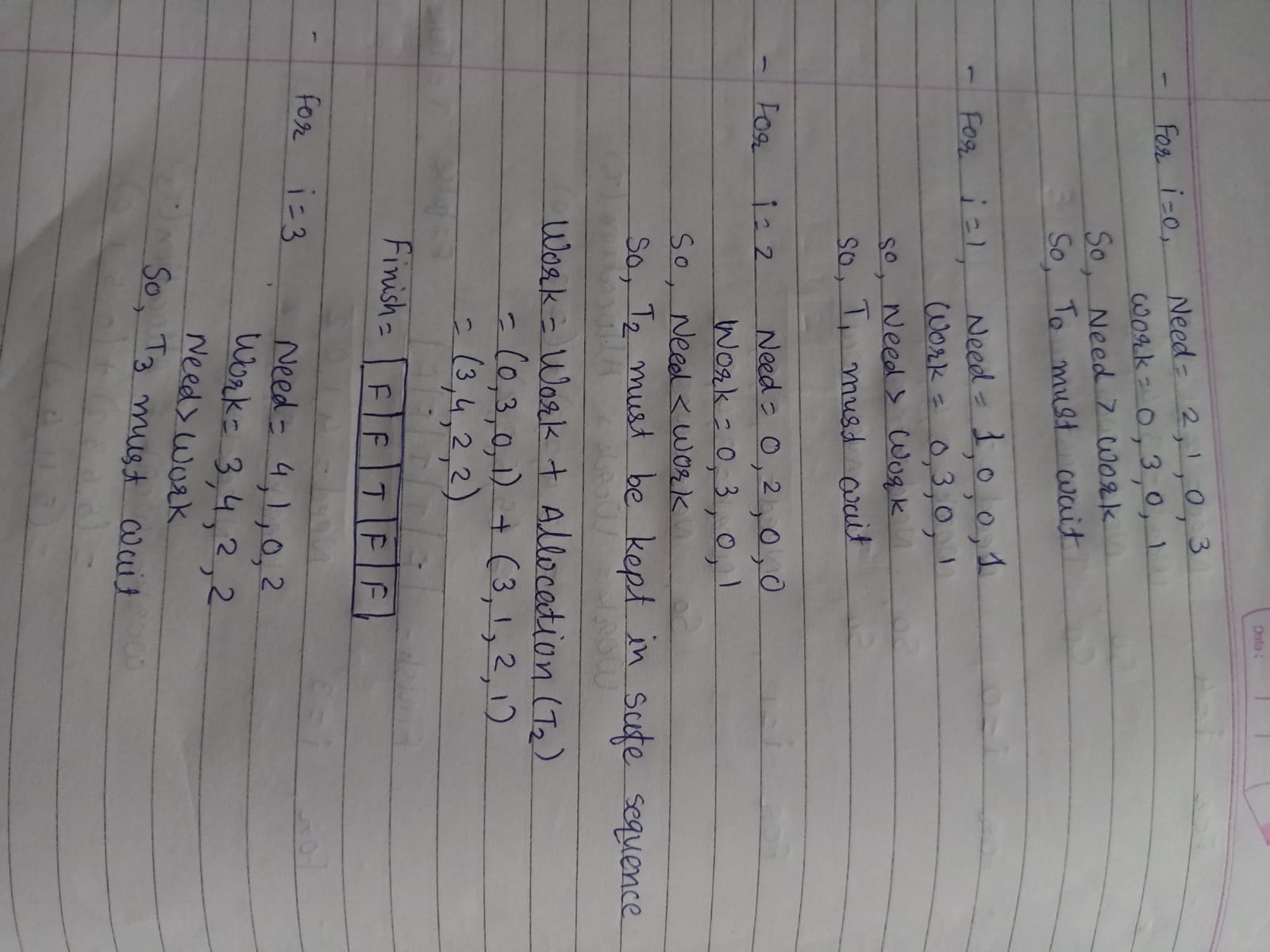
**ANS.**

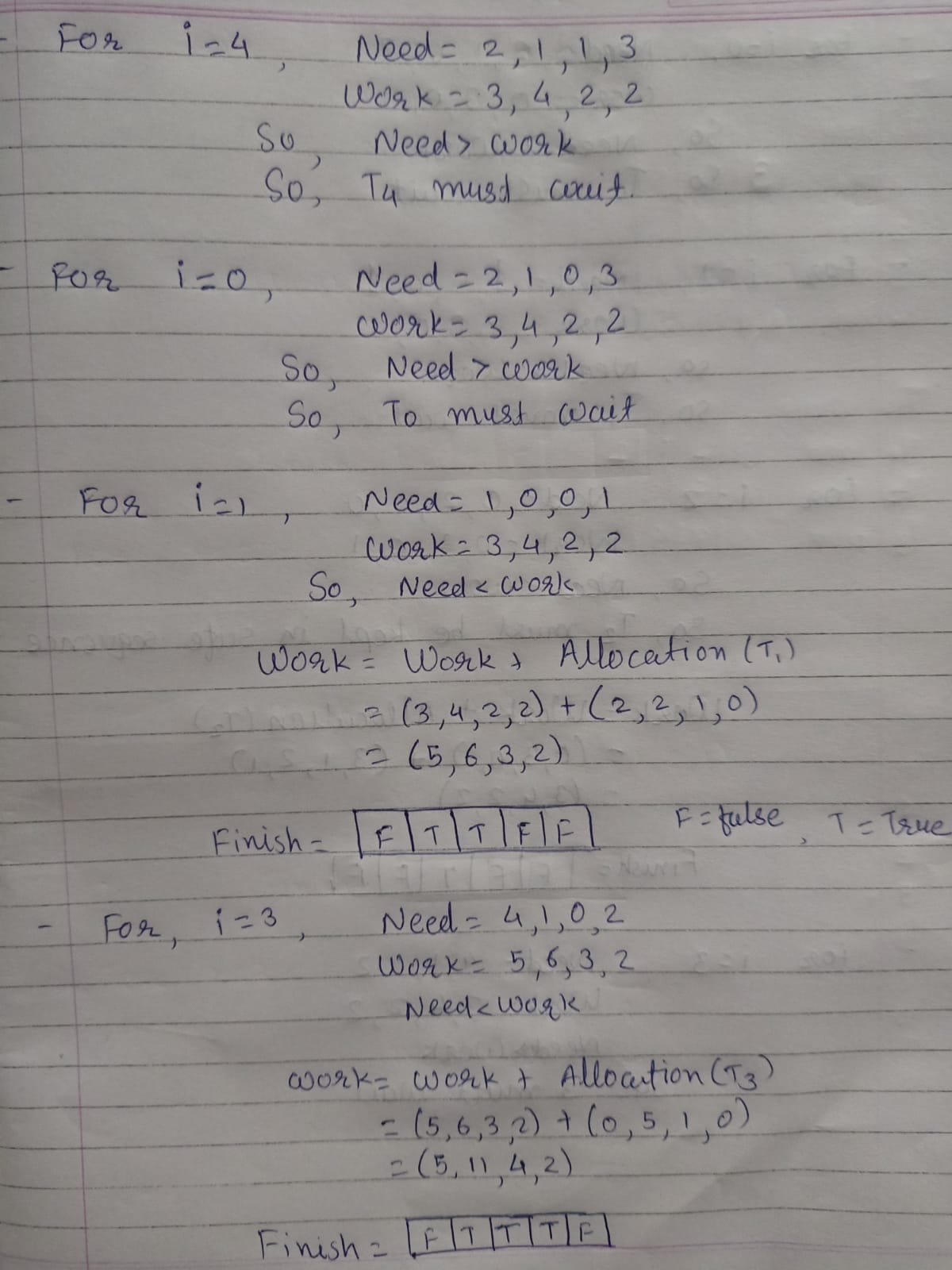
* Representing a projection of a moving object is known as trajectory.
* It's a comparison process.
* It depicts the relative motion of moving objects along with their trajectories.
* The moving object's relative motion as well as the following trajectories
* Traveling from north to east can also be done in a south-west route depending on the circumstances.
* They can be carried out in accordance with a satisfying approach. The optimization and computation are carried out in order to arrive at a conclusion.

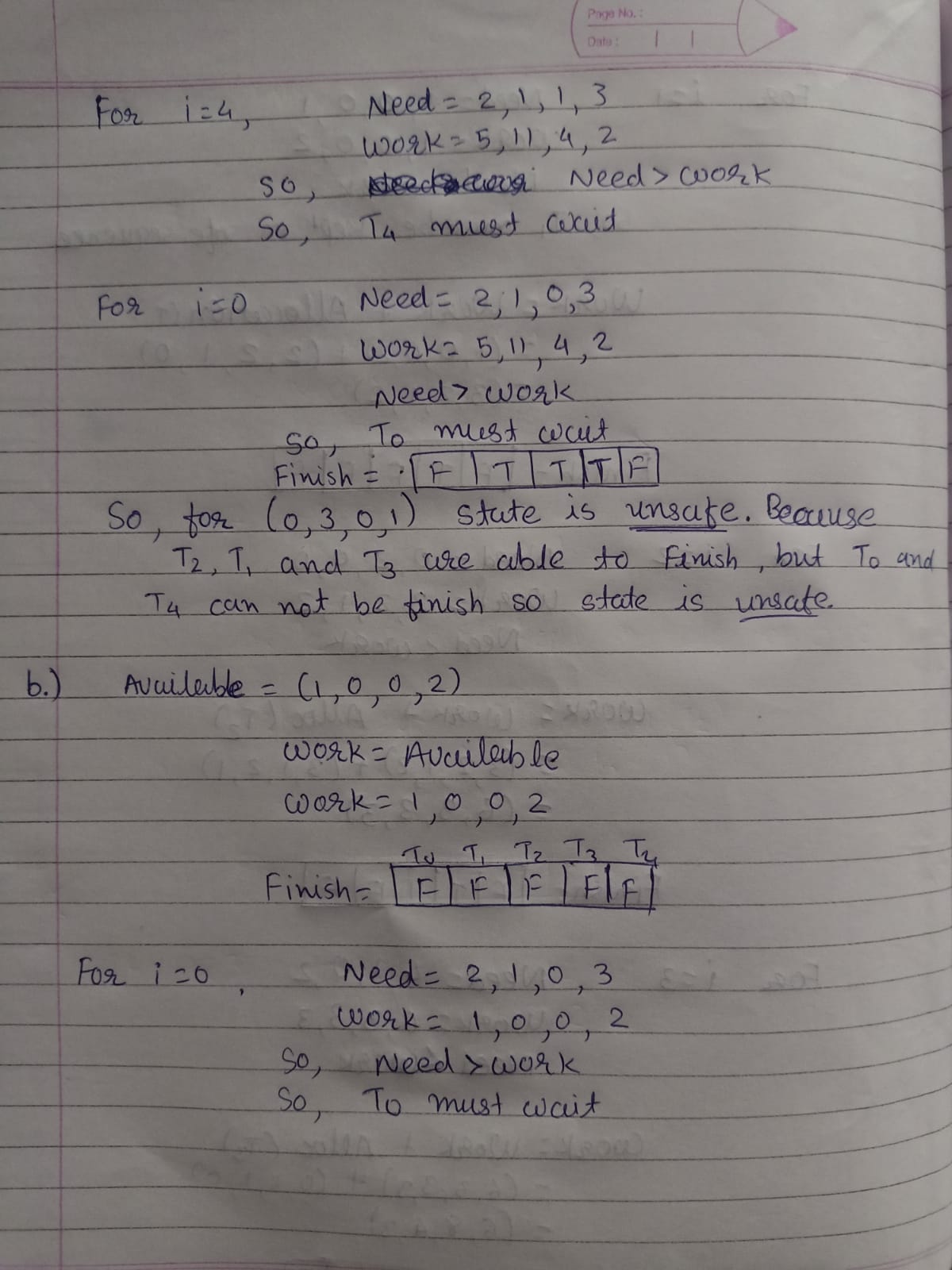
**2.**

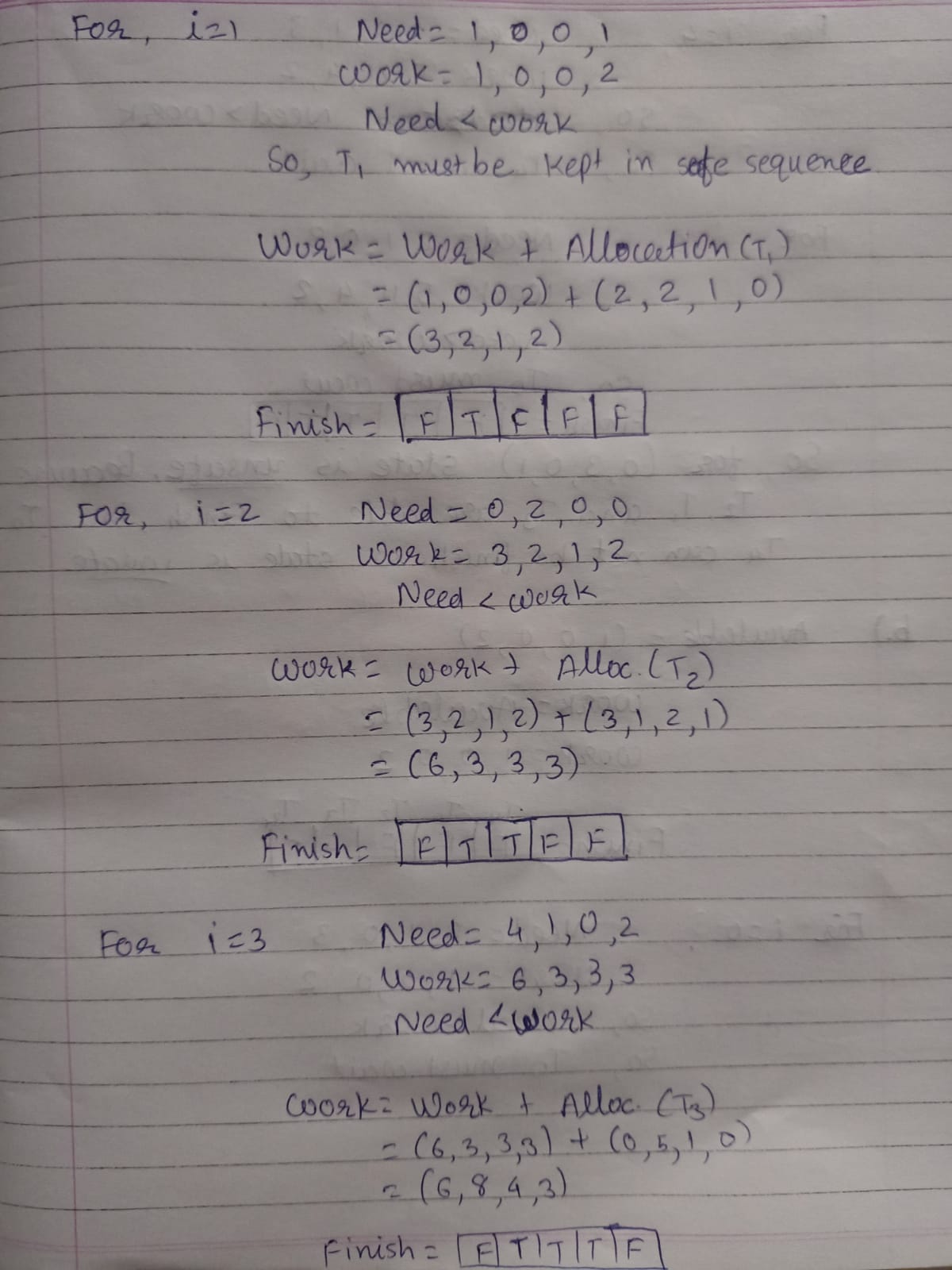
**ANS.**

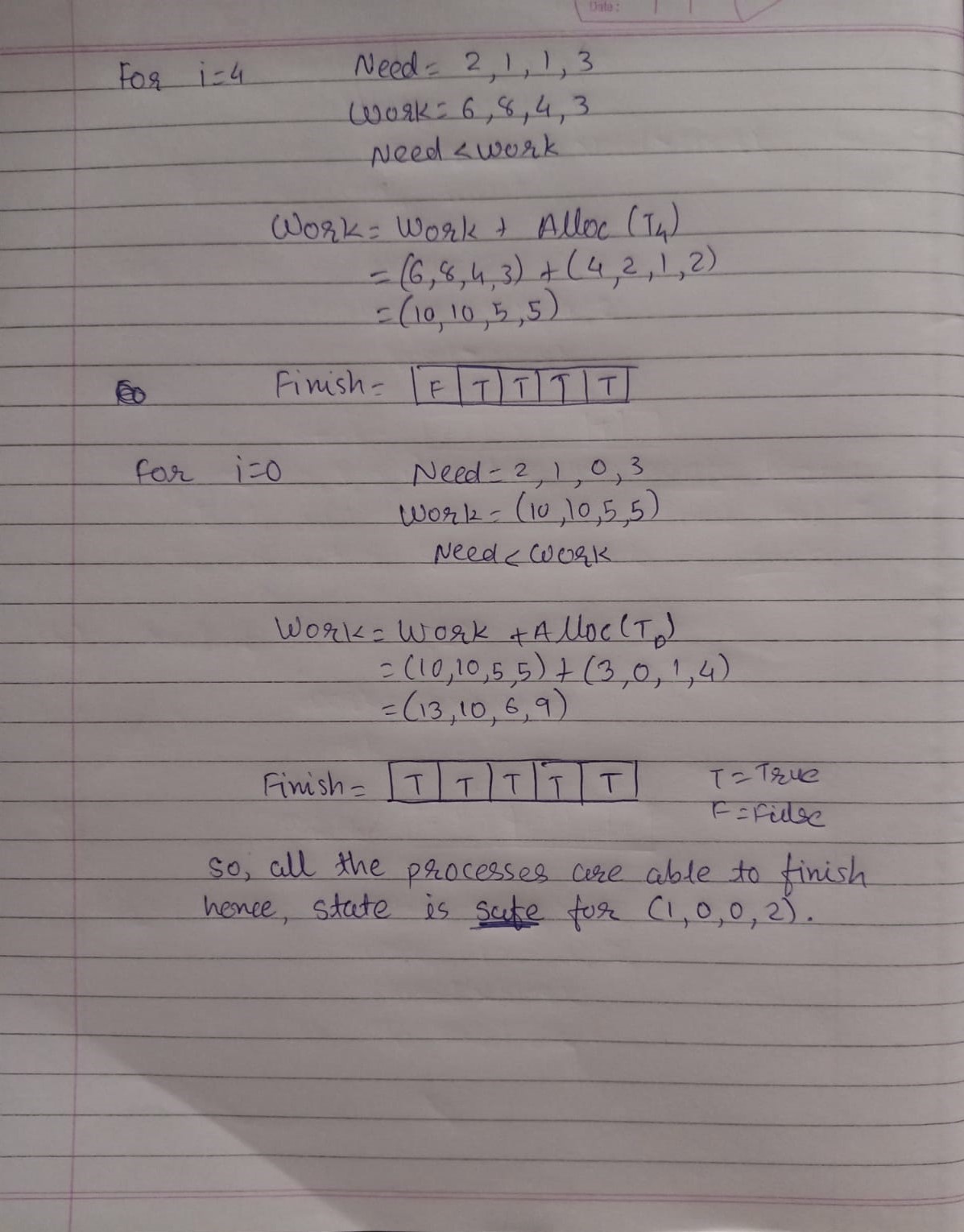
****

****

****

****

****

****

**3.**

**ANS.**

The key to avoiding deadlock is to acquire the locks in some fixed order, say in increasing numeric order. In this way, there can be no cycle of lock holds

void transfer(account \*a, account \*b) {

if (a->account\_id < b->account\_id) {

acquire(&a->mutex);

acquire(&b->mutex);

} else {

acquire(&b->mutex);

acquire(&a->mutex);

}

if (a->money > 0) {

a->money--;

b->money++;

}

release(&a->mutex);

release(&b->mutex);

}

Here we consider two account's a & b as processing objects. When accounts a & b are free then and only then both the accounts can be locked. After these accounts are locked then transfers can be performed else wait until the accounts a & b are free. Therefore, deadlock between 2 processes can be avoided.

**4.**

**ANS.**

1) Let work and finish be vectors of length ‘m’ and ‘n’ respectively.

Initialize: Work = Available

Finish [i] = false; for i=1, 2, 3, 4…..n

2) Find an I such that both

a) Finish [i] = false

b) Needi <= Work

If no such I exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

Goto step (2)

4) if Finish [i] = true for all i

Then the system is in safe state

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 Needi ≤ Work

* the 1st time, n processes have to compare if their Needi ≤ Work
* the 2nd time, n-1 processes have to be compared
* Thus, in the worst case, there are n + (n-1) + (n-2) + ... + 2 + 1 = n(n+1)/2 comparisons, which is proportional to n2 complexity
* Each vector comparison requires m individual comparisons, since there are m resource types, so total complexity is O(m\*n2)