Prove the correctness of Dekker's algorithm:
(a) Prove that mutual exclusion is enforced.
Proof 1: (Direct proof based on the definition of mutual exclusion)
To show mutual exclusion is enforced in Dekker's algorithm, we prove that no any other process can get into its critical section after one of the processes is already in its critical section.

Suppose process P_1 enters its critical section first and remains there. According to the algorithm, flag[1] must be true. Since P_1 is the only one to get in, another process, say P_0, that wants to get in as well must be executing a statement outside its critical section. Since the while loop is the only control structure that may block its access, we need only to discuss the case when P_0 is executing the while statement. According to P0(), when flag[1] is true, P_0 won't be permitted to exit the while loop and thus enter the critical section, whatever turn is.

Hence, at most one process may be in its critical section at a time. Mutual exclusion is enforced indeed in Dekker's algorithm.

Proof 2: (Using contradition)
Suppose two processes, P_0 and P_1, are in their respective critical sections in the purpose of contradiction.
According to Dekker's algorithm, we know:
• For P_0, flag[0] is set to true and **then** flag[1] is checked and confirmed to be false, which may be denoted by:

```
...
t1: flag[0] = true
...
t2 flag[1] = false
...
```

• For P_1, flag[1] is set to true and **then** flag[0] is checked and confirmed to be false, which may be denoted by:

```
...
t3 flag[1] = true
...
t4 flag[0] = false
...
```

Note that based on the above conditions, we cannot directly jump to contradiction.

Suppose P_0 entered the critical section no later than P_1, which means $t_2 <= t_4$, noting that the confirmation of the falsity of the condition in the outer while loop is the last action a process takes before it enters the critical section. We can actual use t2<t4 instead of $t_2 <= t_4$, if one processor (CPU) is considered.

Then we compare t_2 and t_3. Since once flag[1] is set to true at t_3, it will by no means become false, we know $t_2 < t_3$ must hold. Thus we know
$t_1 < t_2 < t_3 < t_4$

However once flag[0] is set to true at t_1, it will by no means become false before P_0 finally exits the critical section, noting we suppose that both P_0 and P_1 are in the critical sections at the present time. Thus it is impossible to have checked flag[0] to be false at t_4, which according to our assumption has happened.

Thus a contradiction is drawn. We cannot assume two processes are in the critical sections at any moment, meaning at most one process may access the exclusive resources at one time. So mutual exclusion is enforced in Bekker's algorithm.

(b) Bounded waiting: Prove that a process requiring access to its critical section will not be delayed indefinitely.
That is to show there is no starvation.
Proof:
To show there is no starvation in Dekker's algorithm, we prove that any process can eventually enter its critical section if it wants.

Suppose process P_1 wants to enter its critical section. Since the while loop is the only control structure that may block its access, we simply suppose it is executing the while statement.
• **If** flag[0] **=** false:
P_1 will surely fail in checking for while (flag[0]) and thus get into the critical section. One exception is P_1 may not be able to reach while (flag[0]), due to being trapped at while (turn == 0), i.e., it enters the while (flag[0) loop but trapped inside at while (turn == 0). If it is being blocked there, flag[1] has been set to be false to show courtesy. Thus P_1 will in no way present P_0 from accessing the exclusive resources, which will enable P_0 to enter the critical section and finally set turn to 0 and flag[0] to false. Thus P_1 may eventually get out of the loop of while (turn == 0).

We need not discuss the case when P_0 is not running, because for P_1 to arrive at while (turn == 0), flag[0] must have once been true and P_0 must have been active, which shows turn and flag[1] have obtained or will have eventually the values giving the green light for P_1 to enter the critical section.

• **Or otherwise** flag[0] **=** true:
Similar to the ending part of the discussion in the first case, turn and flag[1] have been or will be eventually given 1 and false respectively, which guarantees P_1 can exit the while loop at last and get into the critical section. In either case, P_1 will finally be able to enter its critical section. So due to the equivalence of all the individual processes, any process, if it wants, can eventually visit the exclusive resources. Hence no starvation is in Dekker's algorithm.

(c) Making progress: If no process is in its critical section, any process that wants to access the critical section can get in the critical section. This is straightforward. Due to the property (b), process will not be blocked indefinitely at the while loop. Thus the only other possible situation to violate the making progress property is that one process (say P0) is in its remainder section, and the other process (say P1) is blocked at the while loop waiting for the finish of P0. This is impossible because when P0 is in its reminder section, flag[0] is set to zero, and thus P1 will not be blocked at while (flag[0]==true).