(a)

To show that there is always a stable assignment of users to servers, we need to prove the two instabilities don’t exist.

For the first type of instability, suppose there are users and , and a server . Since prefers to and is assigned to , it means that have offered a slot to before offering to . If is free at that time, will be together with . If is not free, it means that has been assigned to another server. In all situations, won’t be assigned to no server in the end. By contradiction, first type of instability doesn’t exist.

For the second type of instability, suppose that there exists a pair of (, ) that prefers to . Then must have offered a slot to before offering to . If was rejected, then must have been with a server that prefers it to at that time. In the end, must be with a server which ranks higher than in its preference list. By contradiction, second type of instability doesn’t exist. So there is always a stable assignment of users to servers.

(b)

The server can be “have available slots” or “full”. The user can be “assigned” or “free”. The algorithm is as follows:

*While some server has available slots*

*ranks first in ’s remaining preference list*

*offers a slot to user*

*If is free then*

*accepts the offer*

*Else ( is already assigned to another server )*

*If prefers to then*

*and are still together*

*Else will be assigned to*

*has one more available slot*

*has one less available slot*

(c)

Runtime complexity =

Because there are m servers and each server can offer a slot to different user for n times.

(d)

To prove the algorithm’s correctness, we must prove that 1) for all possible input, the algorithm exits; 2) when it terminates, there is always a stable matching.

The algorithm always terminates in steps because there are m servers and each server can offer a slot to different user each time for at most n times. The algorithm terminates when all slots are filled. Suppose server has available slots and there are some slots left when the algorithm terminates. Then the server with slots left need to offer slots to some users until all users are assigned to some servers. Then the total number of for m servers would be greater than m. By contradiction, it doesn’t exist so the algorithm always terminates when all slots are filled.

Everytime the algorithm terminates, there is always a stable matching if neither of the two instabilities exist. For the first type of instability, suppose there are users and , and a server . Since prefers to and is assigned to , it means that have offered a slot to before offering to . If is free at that time, will be together with . If is not free, it means that has been assigned to another server. In all situations, won’t be assigned to no server in the end. By contradiction, first type of instability doesn’t exist. For the second type of instability, suppose that there exists a pair of (, ) that prefers to . Then must have offered a slot to before offering to . If was rejected, then must have been with a server that prefers it to at that time. In the end, must be with a server which ranks higher than in its preference list. By contradiction, second type of instability doesn’t exist. So there is always a stable assignment of users to servers.

(e)

Runtime complexity =

Brute force lists all permutations. There are n users and each user has m+1 options. Each user can be assigned to m different servers or assigned to no server.

(f)

A screenshot of a cell phone

Description automatically generated

With number of servers increasing, we can see sharp increases in the runtime of brute force. There is one point in brute force that doesn’t follow the increasing trend and I guess it is an instance with small input. And since we have limited number of outputs for brute force, the trend is not that obvious here. Compared to brute force, there is a slow growth in the runtime of efficient algorithm and it looks like it’s upper bounded. With number of servers increasing, brute force has much more higher runtime than efficient algorithm.