Report: Programming Assignment 1

By Vinay Shah, vss452

a) Show that there is always a stable assignment of users to servers.

Let us assume not. Then according to the definition of instability, at least one of the following exists: Case 1 (as defined in the problem), and/or case 2 (as defined in the problem). We have n users and m servers, and the total number of jobs that can be run on a server are less than or equal to the number of users. By that definition, we know that m <= n. In our problem the servers are analogous to the “men” that “propose” to the users, while the users are the “women”.

If case 1 has occurred, that means that there exists a user **u** that is assigned to server **s**, and a user **u’** that is not assigned to a server, where **u’** is ranked higher than **u** by **s**. This means that **s** prefers **u’**. This would mean that **s** proposed to **u’** at an earlier point, and that **u’** rejected **s** because **u’** was matched to a server they preferred more. But once users match with a server, they don’t un-match from the server. This is a contradiction because **u’** was not assigned to any server.

If case 2 has occurred, that means that there exists a user **u** that is assigned to server **s**, and a user **u’** assigned to a server **s’**, where **s’** prefers **u** and **u’** prefers **s**. If **s** prefers **u’**, that means that **s** proposed to **u’** at some earlier point and was rejected because **u’** was matched to a server **s’’** that was ranked higher on **u**’s preference list. This means that **s’’** is either ranked above **s.** Since **s’** is the final match for **u’**, that must mean that **s’** is ranked higher than **s’’** and **s**. Either way, this is a contradiction because **u’** prefers **s** according to the assumption.

Since we have shown that neither instability can exist, we have shown that there is always a stable assignment of users to servers.

b) Give an algorithm in pseudocode (either an outline or paragraph works) to find a stable assignment that is server optimal.

Hint: it should be very similar to the Gale-Shapley algorithm, with servers taking the role of the men, and users of the women.

```
Initially, all the servers are empty and the users are unassigned
while There exists a server that has a free slot and has not proposed to every user
    do Choose such a server s
        Let u be the highest ranked user on preference list of s to whom s has not
        proposed
        If u is unassigned
            Then (s, u) become matched and one of the server slots of s decreases
        Else u is currently matched with some other server s’
            If u prefers s’ to s
                Then s continues to have a free slot
            Else u prefers s to s’
                (s, u) become matched and one of the server slots of s decreases,
                and s’ frees up one slot
Return the set S of matches
```

c) Give the runtime complexity of your algorithm in Big O notation and explain why.

The runtime complexity is O(m*n). This is because there are m servers and n users, and no server proposes to a user more than once. Since every server could propose to every user, there would be m*n proposals which could be written as some $O(n^2)$.

d) Give a proof of your algorithm’s correctness. Remember that you must prove both that your algorithm terminates and gives a correct result.

The algorithm terminates because it can only run when there exist free slots in servers and there are users to whom the servers have not yet proposed. The loop exhausts all **n** users while each of the **m** servers **s** has free slots. It will terminate after at most m*n iterations.

The algorithm gives a correct result if the resulting matching is a stable matching. We will show that the algorithm results in a stable matching.

Because the number of users is always greater than or equal to the number of server slots, there could be some users who remain unmatched. But every server slot will be matched with a user.

Every server’s slots will be matched with users at the termination of the algorithm. We will prove this by contradiction. Assume that there exists a server with an empty slot. This means that there is at least one user who is not matched to the server. If there exists an empty slot and an unmatched user, that means that the server has not proposed to the user yet, and therefore the algorithm has not yet terminated. This is a contradiction to our initial assumption.

Next, we show that there are no instabilities. There can be instabilities in two ways as specified in the project description. We will prove that they cannot exist when running our algorithm by contradiction.
For case 1, we will assume that such an instability exists. This means that there exists a user **u** that is assigned to server **s**, and a user **u’** that is not assigned to a server, where **u’** is ranked higher than **u** by **s**. This means that **s** prefers **u’**. This would mean that **s** proposed to **u’** at an earlier point, and that **u’** rejected **s** because **u’** was matched to a server they preferred more. But once users match with a server, they don’t un-match from the server. This is a contradiction because **u’** was not assigned to any server.

For case 1, we will assume that such an instability exists. This means that there exists a user **u** that is assigned to server **s**, and a user **u’** assigned to a server **s’**, where **s’** prefers **u** and **u’** prefers **s**. If **s** prefers **u’**, that means that **s** proposed to **u’** at some earlier point and was rejected because **u’** was matched to a server **s’’** that was ranked higher on **u’**s preference list. This means that **s’’** is either ranked above **s.** Since **s’** is the final match for **u’**, that must mean that **s’** is ranked higher than **s’’** and **s**. Either way, this is a contradiction because **u’** prefers **s** according to the assumption.

Since we have shown that neither instability can exist, we have shown that there is always a stable assignment of users to servers.

We have therefore proven that every server’s slots end up filled with users, and that such a matching is indeed stable. We have also shown that it terminates after a finite number of steps. Therefore, the algorithm is correct and terminating.

e) Consider a Brute Force Implementation of the algorithm where you find all combinations of possible matchings and verify if they are a stable marriage one by one. Give the runtime complexity of this brute force algorithm in Big O notation and explain why.

The brute force implementation would have to try all permutations of the matchings in order to find a stable matching. Since there are a total of **m** servers and **n** users, there will be **mn!** matchings, and therefore O(mn!) runtime complexity.

f) In the following two sections you will implement code for a brute force solution and an efficient solution. In your report, use the provided data files to plot the number of servers (x-axis) against the time in ms it takes for your code to run (y-axis). There are four small data files and four large data files included in the input provided. The large data files may be too large for the brute force algorithm to finish running on your machine. If that is the case, do not worry about plotting the brute force results for the large data files. Your plot should therefore contain 8 points from your efficient algorithm and 4-8 points from the brute force algorithm. Please make sure the points from different algorithms are distinct so that you can easily compare the runtimes from the brute force algorithm and your efficient algorithm. Scale the plot so that the comparisons are easy to make (we recommend a logarithmic scaling). Also take note of the trend in run time as the number of servers increases.

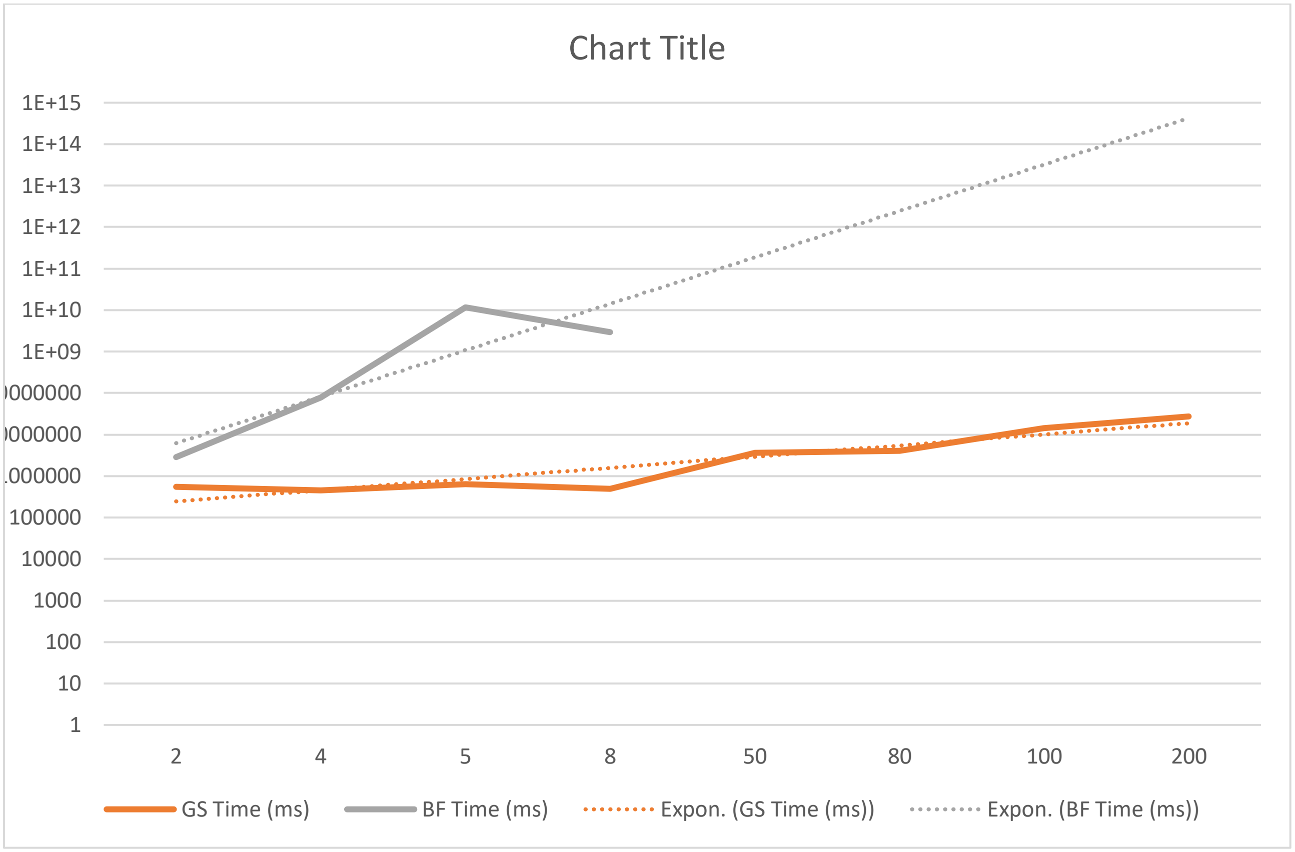
Chart Title
1E+15
1E+14
1E+13
1E+12
1E+11
1E+10
1E+09
GS Time (ms)
BF Time (ms)
Expon. (GS Time (ms))
Expon. (BF Time (ms))