### Question 2

For question 2, I have written my algorithm by deriving the implied constraints of the Gale-Shapley algorithm (Whitehouse). As in each step of building the stable marriages, the Gale- Shapley algorithm is taking into consideration preferences of first males and then females. This means that if a male prefers another woman instead of his wife and the other woman prefers him instead of her husband, the solution list includes unstable marriages.

Therefore, to check if the marriages are unstable, I followed the same trail in my code.

To make my algorithm faster I have used the *pandas* package and worked with data frames instead of arrays or dictionaries. I used a loop which once arrived at a couple from the solution list, where the male preferred another woman, it would check that specific woman’s relationship. If she also preferred the male in the other marriage more than her own husband, then the *is\_stable function* is determining the solution list as False. If while looping trough the list no such cases are encountered, the solution is True.

The complexity of the algorithm is O(n^3) as there are 3 nested loops with a time complexity O(n) each.

### Question 3

For question 3, the algorithm I worked on is a rather inefficient one but gives the best results since it takes into account all the possible combinations where users have their accumulated cost ≤ budget. I have adapted my algorithm based on an algorithm that generates all the permutations of a list’s elements (Salvadores, n.d.). In the algorithm, I have worked with dictionaries to store the data, and have an easy access to the users and their attributes(*cost,value, followers*).

To get the best possible combination the algorithm makes use of a function(*subset\_sum(user\_cost\_dict, target, partial =[]))* that gets all the possible combinations of users that together are less costly or equal compared to the budget(*target)*. The function (*get\_total\_value\_sum(users,user\_value\_dict,user\_follower\_dict))*, is, then iterating through all the combinations resulted from *subset\_sum().* The first combination of users is stored into a variable, together with their direct followers. In the variable to avoid double consideration of a tweeter user, that can result from one user being a follower to another, or by multiple users having common followers, only the unique users are stored. By iterating through the variable, the function calculates the *total\_value\_sum* for the respective the users. Once the iteration trough all the combinations of users goes on, if another combination has a higher value sum the *added\_users\_and\_followers*  variable is updates with the ‘new’ combination of users and the *total\_value\_sum* is updated with the higher value.

In the case where two combinations have the same total value the function *get\_users\_max\_values(combinations, user\_value\_dict,user\_follower\_dict)* is comparing the two by cost and stores into the *picked\_users* variable, the less costly between the two.

The main function *select\_advertisers(budget, followers,c,v)* has the purpose of storing the data into the right format and is using all the above mentioned functions to return the most valuable combination of users, that are affordable within the budget constraint.

# Bibliography

1.Salvadores, M. (n.d.). *Finding all possible combinations*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/4632322/finding-all-possible-combinations-of-numbers-to-reach-a-given-sum

2.Whitehouse, S. (n.d.). *Gale-Shapley Algorithm.* Retrieved from Youtube.com: https://www.youtube.com/watch?v=0m\_YW1zVs-Q