Q.1.a)

Efficient G-S Algorithm

For every free woman in the set

Pair with the highest priority man in her preference list

If that man is already paired with another woman’ then free that woman’

From the paired man’s preference list Remove all woman of lower or same priority

For each of those women removed, remove the paired man from the preference list of each of those women

End

Detailed Psuedo Code

Arrange all women preference lists in decreasing order of priority of men starting from index 1

The index 0 has a -1 if free or the index of man that she is paired with

Arrange all men preference lists in decreasing order of priority of women starting from index 1

The index 0 has a -1 if free or the index of woman that he is paired with

//If there are are ties then force a sequential priority on the ties so {2,1,1} will become {-1,2,3,1}

//{1,1,1} becomes {-1,1,2,3}, {1,2,1} becomes {-1,1,3,2}

//Whenever a man is paired with a woman then his preference list is shortened so that it only contains //women with a higher priority than the one he is presently paired with

//whenever any woman is removed from the man’s preference list then that man should also be //removed from that woman’s preference list

If matching-list(woman-idx) is free then say w2

Womans-man-idx = man in w-preference-list[woman-idx].[1] say m5

//Check if the man can pair with this woman. She exists in the mans preference list

If m-preference-list[womans-man-idx + n].contains(woman-idx + 1)

//pair the woman in matching-list(woman-idx) with womans-man-idx w2-m5

Matching-list(woman-idx) = womans-man-idx

//check if that man is already paired and free that woman if paired m5-w4

womans-mans-woman-idx = m-preference-list[womans-man-idx].[0] w4

if womans-mans-woman-idx not = -1 it is 4

matching-list(womans-mans-woman-idx) = -1 w4 free

end-if

//free all lower or same priority women from this man womans-man-idx preference list

//and remove this man from those removed womens preference lists

womans-man-idx.sublist(womans-mans-woman-idx + 1,preference-list[womans-man-idx + n],size()).clear()

frees w4 and others of lower priority

for (i = womans-man-idx, i < n + 1;i++){

w-preference-list[i].remove(value of womans-man-idx + 1)

}

End

w-preference-list[woman-idx].[0] = womans-man-idx

m-preference-list[womans-man-idx].[0] = woman-idx + 1

End

The code above is still not a O(n2) but it is efficient because the men’s preference & women’s list gets shorter in every iteration.

Q.1.b)

In any iteration of a free woman, in the worst case at least one man will be paired with the woman which is his highest priority, thus removing that woman-pair pair from the next iteration.

Thus a worst case would be n + (n-1) + (n-2) +……1 iterations. Which is n(n+1)/2 which O(n2).

However this assumes that data structures can be designed which can delete the unwanted woman-man pairs in a constant amount of time.

However

Q.1.c)

A brute force algorithm will generate all the pairs so that will be O(n2).

For stability check the pseudo code looks like

For each woman w in the list n

Check if any other man m’ is higher in her preference list than the man m is paired with n

Check the preference list of m’ to see if w is higher than the woman w’ he is paired with n

However the it will take O(n3) run time to check if each pair is stable or not. Thus the overall run time complexity would be O(n3)

Q.1.d)

