PS 1

① Yes, women can get a more preferable match when she lies about her preference in certain cases. Below is an example, where a, b, c are men and d, e, f are women.

In reality woman d prefers b→a→c in the order. However, she lies that she prefers b→c→a in the order of preference.

If she is honest in the situation below,

	d	e	f
a	1\2	2\1	3\1
b	2\1	1\2	3\2
c	1\3	2\3	3\3

If we follow the GS algorithm, a will be matched with d, b will be matched with e, and c will be matched with f. (set S = {a-d, b-e, c-f}).

Here, d is matched with a, her second preference. However, if she lies and declares her order in b→c→a, her situation becomes better.

	d	e	f
a	1\3	2\1	3\1
b	2\1	1\2	3\2
c	1\2	2\3	3\3

← The same situation d's lie.

If we follow GS algorithm, a will be matched with d initially, b will be matched with e initially, c will be matched with d and d breaks up with a, a will be matched with e and e breaks up with b, b will be matched with d and d breaks up with c, c will be matched up with f.

In other words, set S = {b-d, e-a, c-f}. Here, d is matched up with b, the man she preferred the most, by lying her preference. Therefore, women can get a more preferable match when she lies about her true preference.

This occurs because she lied about her preference of a and c. Since women do not have the chance to ask first, she has to rely on other men to propose to her to change her partner. If she lies to change her partner, the man who got rejected by her has to go propose to another woman that might break up existing relationship. Then, the man who got rejected has to ask to another woman, which might be the first choice of her. Therefore, this lying about preference benefits women in certain cases because they can only change their partners once they get proposed and not have the option to propose first.

```
②(a) def oneStableMatchingOnly ( Instance Of Stable Matching) :
        Conduct GS search for men and get the value stable set S1
        Conduct GS search for women and get the value stable set S2
        If set S1 == set S2 :
            return "there is one stable matching"
        else :
            return "there is more than one stable matching"
```

Above is the pseudocode. We can solve this question by getting one instance of stable matching and conduct GS algorithm (which is already published in the book so I didn't write the code out for this) in both the perspective where men propose to women and women propose to men. After getting two sets, we compare if they are the same set (if the same men matched with same women). If they are the same, we return there is one stable match, but if not, we return there is more than one stable match.

Proof:

According to the hint, the people who propose is favored and the people who are getting proposed is unfavored. For example, if men propose to women, men are in favor and can choose and propose to women in the preference they like even if they get rejected while women don't even have a chance to change if the man that she favors does not propose to her. Therefore, the GS algorithm is somewhat unfair. However, if we give women the chance to propose to men first, we are giving women the chance to be with the man that she prefers in the order of preference. If the chance for women to propose to men first produces a different result to when men propose to women, there can be two distinct stable sets, assuming that GS algorithm always returns a stable set (proven by (1.6) in text book). However, if they produce the same stable set no matter who proposes first, then there is only one stable match. Furthermore, since the GS algorithm always terminates, the function will also terminate after returning the value depending on whether set S_1 and S_2 are the same.

Here is the example that I got from lecture:

	X	Y	Z
A	2 \ 1	1 \ 2	3 \ 1
B	1 \ 2	2 \ 1	3 \ 2
C	1 \ 3	2 \ 3	3 \ 3

If A, B, C are men and X, Y, Z are women and if men propose first, the result is $S = \{Y-A, X-B, Z-C\}$.
However, if women propose first, the result is $S = \{A-X, B-Y, C-Z\}$.
Notice the two sets are different depending who asks first.

Runtime:

It is known that GS algorithm is $O(n^2)$. Therefore, 2 GS algorith will take $2O(n^2)$. The if-else statement is just a constant time c, so my algorith will take $2O(n^2)+c$. However, when calculating big O, constants disappear since it is just one statement and $2O(n^2)$ is same as $O(n^2)$, which gives my algorithm runtime of $O(n^2)$.

② (b) In the worst case situation for GS algorithm for n men and n women, the algorith will take $\Omega(n^2)$ as lower bound.

In the worst case, every men should ask the women that already has a partner (except the first man since he will automatically match with any woman) and have her to accept him as the partner and reject her previous proposal.
In this case, the rejected men goes all the way back until he gets his turn to propose again to the women that he has not proposed yet.

Here is the case: there are n men and n women.

① M_1 proposes to W_1 ($M_1 - W_1$)

② M_2 proposes to W_1 who already is matched with M_1 and W_1 rejects M_1's claim and accepts M_2 instead ($M_2 - W_1$)

③ M_3 proposes to W_1 who already is matched with M_2 and W_1 rejects M_2's claim and accepts M_3 instead ($M_3 - W_1$)

⋮

④ Continue this until you reach to the last man (M_n) and repeat so that M_n is accepted by W_1.

⑤ M_1 proposes to W_2 ($M_1 - W_2$)

⑥ M_2 proposes to W_2 and W_2 accepts M_2 and M_1 is rejected ($M_2 - W_2$)

⑦ Continue this until you reach M_{n-1} so that M_{n-1} is matched with W_2 ($M_{n-1} - W_2$)

⑧ Repeat the whole step until M_1 is matched with W_n after the cycle of rejection to $W_3, W_4, \dots, W_{n-1}$.

After steps 1-4, $M_1, M_2, \dots, M_n$ all proposed to W_1, which is total of n times.
Then, $M_1, M_2, \dots, M_{n-1}$ all proposed to W_2, which is total of n-1 times. Following this patern, we get

$$n + (n-1) + (n-2) + \dots + 2 + 1, \text{ which is equal to } (n+1) \cdot \frac{n}{2}$$

so

$$n + (n-1) + (n-2) + \dots + 2 + 1 = (n+1)\left(\frac{n}{2}\right)$$
$$= \frac{n^2+n}{2}$$

$\frac{n^2+n}{2}$ has the lower bound of n^2,

so $\Omega(n^2)$ is the worst case for GS algorithm.

③ slowest to fastest:

① $n^{1/\log n}$

② $\log_3 5n$

③ $\log^2 n$

④ $(\sqrt{2})^{\log n} = \sqrt{n} = O(\sqrt{n})$

⑥ $\log(n!)$

⑦ $\binom{n}{2} + n\log n = n^2 = O(n^2)$

⑨ $n^{\log\log n}$

⑩ 2^{2^n}

- $n^{1/\log n}$ is simply $2^{\log n/\log n} = 2$ and $\log_3 5n$ is $\log_3 5 + \log_3 n$, and since $\log_3 n$ grows faster than a constant 2, $\boxed{O(\log_3 5n) > O(n^{1/\log n})}$
- $\log_3 5n = \log_3 5 + \log_3 n$ and $\log^2 n$ is $(\log n)(\log n)$, and since $\log n$ squared grows faster than $\log_3 n$ alone added to a constant, $\boxed{O(\log^2 n) > O(\log_3 5n)}$
- $\sqrt{2}^{\log n}$ is equivalent to $2^{1/2 \log n}$ but since $\log n$ means $\log_2 n$, here, it is simplified into $2^{\log_2 \sqrt{n}}$ and 2 and $\log_2$ cancel out to leave $\sqrt{n}$, which is equivalent to $\sqrt{n}$. N to the power greater than 1 (n^x where $x > 1$) always grow faster than any log. $\boxed{O(\sqrt{n}) = O(\sqrt{2}^{\log n}) > n(\log^2 n)}$
- $\log(n!)$ is equal to $\log(n) + \log(n-1) + \log(n-2) + \dots + \log(2) + \log(1)$ and each log(value) is less than or equal to $\log(n)$ ex) $\log(n) \geq \log(n)$, $\log(n) \geq \log(n-1)$ ... There fore, $\log(n) + \log(n-1) + \dots + \log(2) + \log(1) \leq \log(n) + \log(n) + \dots + \log(n) + \log(n) = n\log(n)$. $\log(n!)$ is $O(n\log n)$ and grows faster than $\sqrt{n}$. $\boxed{O(n\log n) > O(\sqrt{n})}$
- $\binom{n}{2} + n\log n = \frac{n(n-1)}{2} + n\log n = n\left(\frac{(n-1)+\log n}{2}\right)$ which is $O(n^2)$. since n^2 has big O of $O(n^2)$, $\boxed{O(\binom{n}{2} + n\log n) = O(n^2) > O(n\log n)}$
- $n^{\log\log n} = 2^{(\log n)(\log\log n)}$ since we can substitute $n = 2^{\log_2 n}$. Here, we have a constant 2 to the power of $(\log n)(\log\log n)$. A constant to the power of variable always grows faster than any polynomial. So $\boxed{O(n^{\log\log n}) > O(n^2)}$
- 2^{2^n} grows faster than $n^{\log\log n} = 2^{(\log n)(\log\log n)}$ since we know that $2^n > \log n(\log\log n)$ since exponential grows the fastest. $\boxed{O(2^{2^n}) > O(n^{\log\log n})}$