1 Problem 1

a) So to find a counter example we can use the simple case of having 2 x's and 2 y's s.t $x \in X$, and $y \in Y$. The members can be x, x', y, and y'. Each of the members has a preference list for the other group. So let's say the preference list of each is...

x: (y, y')	y: (x',x)
x': (y',y)	y': (x,x')

So this means that we can have matches between x and y', and x' and y for a stable matching. This is because no people in the pairs prefer each other more than their current partners. So no instablities. This is a counter example because in both of the couples, neither of them are nice couples or ranked first on each other's list.

b) Let's take an m ∈ M, and w ∈ W such that m and w are a nice couple. We need to show that (m,w) can make a stable matching and futhermore that they are a pairing.

Assume for contradiction, that m and w are not paired together. Let m be paired to w' and w paired to m' where m' and w' are both lower on m and w's preferences lists then each other. The preference lists would look something like (m... m' ...) and (w... w'...).

Although, we can see that this creates an instablity. This is because m and w prefer to be with each other more than their current partners. This will happen for any m' and w'. Meaning, that m and w, that make a nice couple, always have to be paired together. This means that we can "ignore" them since no matter what we cannot change their pairing.

We can use the GS algorithm to continue with the rest of the men pairing with the women. As we've proved in class, the GS algorithm always gives a stable matching. Meaning that a nice couple will always give a stable matching.

c) There is quite a simple example. We can use our example from above actually. There will be 2 disjoint couples in this instance. Using the same preferences...

x: (y', y)	y: (x',x)
x': (y,y')	y': (x,x')

There is only one way to pair this. This is because from part b) we know that if there is a nice couple, then that nice couple is a part of S. If we have n disjoint nice couples, then all of those are part of S = stable matching, and make up the only pairs in the instance.

We know this is unique from part b) because the nice couples don't want to be with anyone else since they are already with their top preference meaning that this is the only arrangment.

d) This problem is suggesting that if there is a unique stable matching then the instance I is composed of n pairs of disjoint nice couples.

We can use induction on this statement. Assume that we have a unique stable matching and that it only consists of n disjoint couples. Then we can show that with another couple, to give a total of n+1 pairs, that the last couple when added will still have the matching to be unique.

So we have n nice couples. We know that from part b) these n nice couples won't bring any instablities because they are all with their top choices and don't prefer anyone else over each other.

There are two cases for the last couple. They are nice or not. If they are not nice they have to be paired with each other no matter what anyway. For example, let's say that this new man m_{n+1} has w_k at the top of his list, he won't get her since she is paired with her top choice. Now, we notice that with a nice couple they are always paired since they are each other's top choice and won't give an instablity.

So no matter what, if the last couple is nice or not, the matching still remains unique.

2 Problem 2

So we can take the case where n = 2. So there are 4 people and the preference lists contain 3 people each.

Let's call the people a,b,c, and d. So we notice that there are three possible pairings. (ab,cd), (ac, bd), and (ad,bc). This is because we don't care about the order of the hotels.

Now, let's construct preference lists in order to make it so that the pairings above can never happen.

Let's just make a's preference list (b,c,d) for simplicity. b is the top preference and goes down.

Then let's start off with the pairing (ab,cd). We notice that a cannot be on the top of b's preference list otherwise we get a nice couple which could lead to a stable matching.

So it's better to keep a on the second spot in b's preference list. It would look like this, (,a,). Now, we want to create an instability. Let's say b and c will make the instability, and that they want to be with each other.

Then we can make b's preference list look like (c,a,) and c's like (,b,). Let's tackle what would happen with pairs ac and bd.

Let's make the instability between a and b. So they want each other more than their current partners. So the preference lists would look something like, a:(b,c,d) and b:(c,a,d). c is paired with d currently, but they want a more. So let's make the prefernce list for c:(a,b,d). Now let's check the final pairing. If we just make d's preference list (b,c,a) then we notice that there is an instablity. a and c both want to be with each other more than their current partners. Even if we pair a and c together, we notice that their pairing gives another instablity as shown above!

To recap, here are the preferences.

a: (b,c,d)
b: (c,a,d)
c: (a,b,d)
d: (b,c,a)

3 Problem 3

Let's illustrate with an example. So let's say that there are 5 days in a month and that there are 2 ships and ports each.

Let ship 1 schedule look like this: P1, sea, P2, sea, sea Let ship 2 scheudle look like this: sea P1 sea sea P2.

We notice that the only truncations that we could do is truncating ship 1 at P2. If we stop ship 1 at P1 then ship 2 wouldn't even be able to make it to a port. It follows that ship 2 is truncated at P2.

This gives a good idea on what the algorithm would be. We can model it after the GS algorithm.

Let s be the ports, and let p be the ports.

Initially, the ships have not set sail and the ports are free.

```
while there is a free port p
    Choose a p in the set of ports
    Let s be the last to visit port p


    if s hasn't been truncated yet then
        s is truncated at port p

    Else s has been truncated then
        pick the next free ship s' that comes before ship s, which is to visit port p
        s' will be truncated at port p

Return the set S of truncated pairs between ships and ports
```

The above algorithm is essentially the GS algorithm, which we have proved in class always gives a stable matching as the output.

4 Problem 4

To illustrate, we can first use an example.

x: (y, y')	y: (x',x)
x': (y',y)	y': (x,x')

With those preferences, we can see that there are two stable matchings and they will be our X and Y. The matches are where (m',w') and (m, w) are paired. The next one is (m',w) and (m,w') are paired. So if we use the rules for constructing Z, we get the pairings (m,w) and (m,w') which is a stable matching. Let's do another with n = 3 this time.

m: (w",w,w')	w: (m,m',m")
m": (w,w",w')	w': (m',m",m)
m": (w',w",w)	w": (m",m,m')

With the above preferences we can create the pairs, (m,w"), (m',w), (m",w'). For the second matching we have (m,w), (m',w'), (m",w") that works too. With using the algorithm to compute Z, we get the pairs (m,w"), (m',w), (m",w') which was the matching that we got earlier.

We notice that for Z', and n = 3, using the algorithm for Z' also works.

Let's first try to prove for Z that this is true.

Assume for contradiction, that that Z is not a stable matching. Then there is some couple in Z that creates an instablity. Let's say this couple is (m,w). Let m be paired with w' and w is paired with m'. By definition, m and w prefer each other more than their current partners. Although this is a contradiction! Above is what I have so far, I'm pretty sure the argument is flawed but I am not sure on how to fix it