This is false. Consider, in fact, one of the examples from the text, with two men and two women.

m prefers w to w'.
m' prefers w' to w.
w prefers m' to m.
w' prefers m to m'.

Note that there are no pairs at all where each ranks the other first, so clearly no such pair can show up in a stable matching.

[1] ex359.539.196

This is true. Indeed, consider such a pair (m, w) and consider a perfectmatching containing pairs (m, w') and (m', w), and, hence, not (m, w). Then since m and w each rank the other first, they each prefer the other to their partners in this matching, and so this matching cannot be stable.

[1] ex742.158.311

There is not always a stable pair of schedules. Suppose Network $\mathcal{A}$ has two shows $\{a_1, a_2\}$ with ratings 20 and 40; and Network $\mathcal{D}$ has two shows $\{d_1, d_2\}$ with ratings 10 and 30.

Each network can reveal one of two schedules. If in the resulting pair, a_1 is paired against d_1, then Network $\mathcal{D}$ will want to switch the order of the shows in its schedule (so that it will win one slot rather than none). If in the resulting pair, a_1 is paired against d_2, then Network $\mathcal{A}$ will want to switch the order of the shows in its schedule (so that it will win two slots rather than one).

[1] ex468.481.560

The algorithm is very similar to the basic Gale-Shapley algorithm from the text. At any point in time, a student is either "committed" to a hospital or "free." A hospital either has available positions, or it is "full." The algorithm is the following:

```
While some hospital h_i has available positions
  h_i offers a position to the next student s_j on its preference list
  if s_j is free then
    s_j accepts the offer
  else (s_j is already committed to a hospital h_k)
    if s_j prefers h_k to h_i then
      s_j remains committed to h_k
    else s_j becomes committed to h_i
         the number of available positions at h_k increases by one.
         the number of available positions at h_i decreases by one.
```

The algorithm terminates in $O(mn)$ steps because each hospital offers a positions to a student at most once, and in each iteration, some hospital offers a position to some student.

Suppose there are $p_i > 0$ positions available at hospital h_i. The algorithm terminates with an assignment in which all available positions are filled, because any hospital that did not fill all its positions must have offered one to every student; but then, all these students would be committed to some hospital, which contradicts our assumption that $\sum_{i=1}^{m} p_i < n$.

Finally, we want to argue that the assignment is stable. For the first kind of instability, suppose there are students s and s', and a hospital h as above. If h prefers s' to s, then h would have offered a position to s' before it offered one to s; from then on, s' would have a position at *some* hospital, and hence would not be free at the end — a contradiction.

For the second kind of instability, suppose that (h_i, s_j) is a pair that causes instability. Then h_i must have offered a position to s_j, for otherwise it has p_i residents all of whom it prefers to s_j. Moreover, s_j must have rejected h_i in favor of some h_k which he/she preferred; and s_j must therefore be committed to some h_ℓ (possibly different from h_k) which he/she also prefers to h_i.

[1] ex304.339.892

(a) The answer is Yes. A simple way to think about it is to break the ties in some fashion and then run the stable matching algorithm on the resulting preference lists. We can for example break the ties lexicographically — that is if a man m is indifferent between two women w_i and w_j then w_i appears on m's preference list before w_j if $i < j$ and if $j < i$ w_j appears before w_i. Similarly if w is indifferent between two men m_i and m_j then m_i appears on w's preference list before m_j if $i < j$ and if $j < i$ m_j appears before m_i.

Now that we have concrete preference lists, we run the stable matching algorithm. We claim that the matching produced would have no strong instability. But this latter claim is true because any strong instability would be an instability for the match produced by the algorithm, yet we know that the algorithm produced a stable matching — a matching with no instabilities.

(b) The answer is No. The following is a simple counterexample. Let $n = 2$ and m_1, m_2 be the two men, and w_1, w_2 the two women. Let m_1 be indifferent between w_1 and w_2 and let both of the women prefer m_1 to m_2. The choices of m_2 are insignificant. There is no matching without weak stability in this example, since regardless of who was matched with m_1, the other woman together with m_1 would form a weak instability.

[1] ex734.923.393

For each schedule, we have to choose a *stopping port*: the port in which the ship will spend the rest of the month. Implicitly, these stopping ports will define truncations of the schedules. We will say that an assignment of ships to stopping ports is *acceptable* if the resulting truncations satisfy the conditions of the problem — specifically, condition (†). (Note that because of condition (†), each ship must have a distinct stopping port in any acceptable assignment.)

We set up a stable marriage problem involving ships and ports. Each ship ranks each port in chronological order of its visits to them. Each port ranks each ship in reverse chronological order of their visits to it. Now we simply have to show:

(1) *A stable matching between ships and ports defines an acceptable assignment of stopping ports.*

Proof. If the assignment is not acceptable, then it violates condition (†). That is, some ship S_i passes through port P_k after ship S_j has already stopped there. But in this case, under our preference relation above, ship S_i "prefers" P_k to its actual stopping port, and port P_k "prefers" ship S_i to ship S_j. This contradicts the assumption that we chose a stable matching between ships and ports. ■

[1] ex873.532.244

This is closely analogous to the previous problem, with input and output wires playing the roles of ships and ports.

A *switching* consists precisely of a perfect matching between input wires and output wires — we simply need to choose which input stream will be switched onto which output wire.

From the point of view of an input wire, it wants its data stream to be switched as early (close to the source) as possible: this minimizes the risk of running into another data stream, that has already been switched, at a junction box. From the point of view of an output wire, it wants a data stream to be switched onto it as late (far from the source) as possible: this minimizes the risk of running into another data stream, that has not yet been switched, at a junction box.

Motivated by this intuition, we set up a stable marriage problem involving the input wires and output wires. Each input wire ranks the output wires in the order it encounters them from source to terminus; each output wire ranks the input wires in the reverse of the order it encounters them from source to terminus. Now we show:

(1) *A stable matching between input and output wires defines a valid switching.*

Proof. To prove this, suppose that this switching causes two data streams to cross at a junction box. Suppose that the junction box is at the meeting of Input i and Output j. Then one stream must be the one that originates on Input i; the other stream must have switched from a different input wire, say Input k, onto Output j. But in this case, Output j prefers Input i to Input k (since j meets i downstream from k); and Input i prefers Output j to the output wire, say Output ℓ, onto which it is actually switched — since it meets Output j upstream from Output ℓ. This contradicts the assumption that we chose a stable matching between input wires and output wires. ■

Assuming the meetings of inputs and outputs are represented by lists containing the orders in which each wire meets the other wires, we can set up the preference lists for the stable matching problem in time $O(n^2)$. Computing the stable matching then takes an additional $O(n^2)$ time.

[1] ex852.589.348

Assume we have three men m_1 to m_3 and three women w_1 to w_3 with preferences as given in the table below. Column w_3 shows true preferences of woman w_3, while in column w_3' she pretends she prefers man m_3 to m_1.

m_1	m_2	m_3	w_1	w_2	w_3	(w_3')
w_3	w_1	w_3	m_1	m_1	m_2	m_2
w_1	w_3	w_1	m_2	m_2	m_1	m_3
w_2	w_2	w_2	m_3	m_3	m_3	m_1

First let us consider one possible execution of the G-S algorithm with the true preference list of w_3.

m_1	w_3			w_3
m_2		w_1		w_1
m_3			$[w_3][w_1]w_2$	w_2

First m_1 proposes to w_3, then m_2 proposes to w_1. Then m_3 proposes to w_2 and w_1 and gets rejected, finally proposes to w_2 and is accepted. This execution forms pairs (m_1, w_3), (m_2, w_1) and (m_3, w_2), thus pairing w_3 with m_1, who is her second choice.

Now consider execution of the G-S algorithm when w_3 pretends she prefers m_3 to m_1 (see column w_3'). Then the execution might look as follows:

m_1	w_3		$-$	w_1			w_1
m_2		w_1		$-$	w_3		w_3
m_3			w_3		$-$	$[w_1]w_2$	w_2

Man m_1 proposes to w_3, m_2 to w_1, then m_3 to w_3. She accepts the proposal, leaving m_1 alone. Then m_1 proposes to w_1 which causes w_1 to leave her current partner m_2, who consequently proposes to w_3 (and that is exactly what w_3 wants). Finally, the algorithm pairs up m_3 (recently left by w_3) and w_2. As we see, w_3 ends up with the man m_2, who is her true favorite. Thus we conclude that by falsely switching order of her preferences, a woman may be able to get a more desirable partner in the G-S algorithm.

[1]ex562.302.864