(2) Stable matching variants. Recall that in the Gale-Shapley algorithm all proposals are made only on one side (classically the men's side), and that the results favors the proposer's side (as showed in class on Wednesday, and a;sp see claim (1.7) on page 10 of the textbook). A more fair way to try to find a stable matching is to allow both sides to make proposals. Here is what such an algorithm may look like:

```
maintain a 2n long array CURRENT.
   For each person i (man or woman) maintain a pointer: CURRENT[i] which is the current
       matched partner of i (nill if i is not matched)
maintain two n-by-n tables REJECTED. For a man M and woman W, REJECTED[M,W]=REJECTED[W,M]
       are both set to 1 if either M rejected W or W rejected M, and 0 otherwise,
While there is a person i (man or woman), and a possible partner j such that
   REJECTED[i,j]=0, and i is currently unmatched or i prefers j to his/her
   current partner CURRENT[i] then
       i makes a proposal to the highest ranked j of this form
       if j is unmatched or prefers i to his/her current partner, then
           j rejects CURRENT[j] (if not nil) and tentatively matches with i, that is:
               if CURRENT[j]=x not nil, then
                   update CURRENT[x]=nil and REJECTED[j,x]=REJECTED[x,j]=1
               if CURRENT[i]=y not nil, then
                   update CURRENT[y]=nil and REJECTED[i,y]=REJECTED[y,i]=1
               set CURRENT[j]=i, CURRENT[i]=j,
       else
           j rejects i, that is
               set REJECTED[i,j]=REJECTED[j,i]=1
endwhile
Return current tentative matching
```

in class we showed that the Gale-Shapley algorithm has the following useful properties: there were only at most n^2 proposals, and the algorithm was guaranteed to output a stable perfect matching. For each of these claims either prove that the claim remains true in the new version also, or show a counter example that this claim can fail (that is, an input to the stable matching problem along with a way to run the algorithm above that makes the claim fail). Concretely, answer the following questions

(a) (2 points) Is the new symmetric algorithm above guaranteed to end after at most $O(n^2)$ proposals (assuming that there are n men and n women).
(b) (2 points) is the algorithm guaranteed to output a perfect matching?
(c) (3 points) If the resulting matching is perfect, is it guaranteed to be also stable?

Solution:

(a) Yes, this is true: there are $2n$ participants, and each can make at most n offers (as no repeat offers allowed), so there are only $2n^2$ offers.
(b) the algorithm is **not** guaranteed to output a perfect matching. For example with $n = 2$ if preferences are

```
2
0 1
1 0
1 0
0 1
```

man m_0 proposes to woman w_0, then woman w_0 proposes to man m_1 (and rejects m_0), now man m_1 proposes to woman w_1 (and rejects w_0), and finally woman w_1 proposes to men m_0 (and rejects m_1). Now the algorithm is done, and our matching is just $\{(m_0, w_1)\}$, not perfect.

(c) even if the output is a perfect matching, it is **not** guaranteed to be stable. For example with $n = 3$ if preferences are

```
3
0 1 2
1 0 2
0 1 2
1 0 2
1 0 2
0 1 2
```

man m_0 proposed to woman w_0, then woman w_0 proposes to man m_1 (and rejects m_0), now men m_1 proposes to woman w_1 (and rejects w_0), man m_2 proposes to woman w_0, and finally woman w_2 proposes to men m_0. The remaining proposals that can be made: m_0 to w_1 and w_0 to m_1 will get rejected. This gives us the perfect matching $\{(m_1, w_1), (m_0, w_2), (m_2, w_0)\}$, that is perfect, but not stable (as (m_0, w_0) form an instability).

(d) (3 points) Suppose you run the above algorithm but give priority to one side (say the man), that is whenever possible the person i selected is a man, and only let women propose once no man can propose. Answer questions (a-c) for this version of the algorithm also.

Solution: this way we ran exactly the Gale-Shapley algorithm (with men proposing) to its end, before women get to propose at all. The Gale-Shapley algorithm guarantees that the solution is stable, as we proved in class
With a current stable matching all proposal by woman will get rejected: a woman w wants to propose only to men m she prefers to her current partner. But (m, w) is not an instability, so m most prefer to his current partner to w, so will reject the proposal.

(3) Scheduling check-ups. (10 points) You are helping a supervisor in a sport facility. With the year just starting they are hiring a lot of new helpers in the gym for various activities. Each helper works for one shift each day. The supervisor would like to visit the gym a few times the first day, and make sure she meets all new helpers. This question is asking you to give an efficient algorithm for this problem. To be more formal, assume there are n new helpers, and helper i has a shift for the time interval $[s_i, f_i]$. If the supervisor visits at time t, she meets all helpers whose interval satisfies $s_i \leq t \leq f_i$.

Give a polynomial time algorithm that finds a set of visit times T with $|T|$ as small as possible so that the supervisor meets all helpers. For full credit your algorithm should run in $O(n \log n)$ time.

Solution: Sort all intervals by finish time, and assume that intervals are numbered in this order. Consider the helpers in this order. When considering helper i, if so far no visit time is scheduled that meets person i, then add time f_i to the set T.

Sorting takes $O(n \log n)$ time. To decide if a student with finish time f_i has already been visited, all we need to check is if the most recent visit $t \in T$ has $s_i \leq t$. If this is not true, add f_i to T.

Running time: Sorting takes $O(n \log n)$ time. To decide if a student with finish time f_i has already been visited, all we need to check is if the most recent visit $t \in T$ has $s_i \leq t$. If this is not true, add f_i to T. We can go one-by-one on the sorted list of intervals, and check if they satisfy $s_i \leq t$. Since we only consider each interval ones, this is only $O(n)$ additional time after sorting.

Correctness by "Greedy Stays ahead" Consider the algorithm's visit schedule, and let $t_1, \ldots, t_k$ be the set of times selected. Let an optimal visit schedule use times $o_1, \ldots, o_k$ also sorted in increasing order.

We will prove by induction that $t_i \geq o_i$ for all i.

base case. This must be true as we have $t_1 = f_1$, the earliest finish time, and there must be a visit in the $[s_1, f_1]$ interval, so clearly $o_1 \leq t_1$.

induction. Assume the claim is true for $i \leq k$ and we want to prove it for $k+1$. Let t_{k+1} be the next visit time of the algorithm, and consider the schedule $[s_j, f_j]$ of the helper that caused this selection, so $t_{k+1} = f_j$. Recall that we have that $t_k < s_j$ as this helper was not yet visited. By the induction hypothesis $o_k \leq t_k$, so the optimal schedule o most also have a visit in the $[s_j, f_j]$ interval, so $o_{k+1} \leq f_j = t_{k+1}$.

showing optimality of schedule. Note that this also o=implies that o must have as many visit times as our schedule. You can show this by contradiction: suppose o ends after some k visits. We know that $o_k \leq t_k$ by the above. But our schedule continued as there was an interval not yet visited, that is an interval starting after t_k. Optimum must visit this interval also, so cannot end yet.

Correctness by "Exchange argument" Among all possible optimal visit schedules, consider the schedule O that agreed with our schedule for as many steps as possible. Let $o_1 < \ldots < o_k$ be this schedule, and assume the step where the two schedules differ is step j (so $o_1 = t_1, o_2 = t_2, \ldots$ till $o_{j-1} = t_{j-1}$, but $o_j \neq t_j$). Recall the reason the algorithm choose the visit time t_j: this is the end of the earliest ending interval (green interval on teh picture) that is not visited by the first $j-1$ visit times. Optimum must also visit this interval, so most be $o_j \leq t_j$. Now we can change to a new solution, but

Figure 1: Exchange argument for problem 3.

deleting o_j and replacing it with t_j.

Proof by minimax There is a much less standard and more creative proof of optimality. For each visit t_i, it happened at the end of a person's interval, let's call this person j_i. Note that the selected people have disjoint intervals: after the first k times, we only consider people whose interval is to the right of the end of these selected intervals for people $j_1, \ldots, j_k$. So if our algorithm selected k times, then the k corresponding people have disjoint intervals, so each needs a separate visit time! This immediately shows that less than k visits cannot possibly be enough. This is usually summarized is *The minimum number of visit times equal the maximum number of disjoint intervals* (hence the name above) and would also constitute a proof of our interval scheduling from lecture being optimal.