2. a) True because the algorithm is to stop when all the women have recieved a proposal & then she goes with the guy she likes the most). Which means if it stopped on the n^{th} day, the day before that ((n-1)day), there was at least one female who did not receive a proposal; or else the algorithm would have stopped on the (n-1) day!

b) True because she will say "yes" ~~to the proposer & therefore he will keep co~~ "maybe" to the proposer she likes the best (which if there is only 1 guy, he is the best). Therefore she always had someone on a string & this person will continue to propose to her until she says no.

c) False because there is no possible preferences for females in such a way so that all of the males get their least. The reason for this is because the algorithm is done in such a way that all the pairing's "like-ness" of each other will find a middle ground. As in at least 1 guy will get above his least prefered female since males will always ask the ones the like the most first.

d) True; assume all n-males prefer different women & no more than 1 woman. Assuming for all of the female on the list, the male is her least prefered but if everyone is paired up with one person, ~~that one person~~ then the algorithm terminates & she gets her least prefered man.

e) True because a woman will always keep a man on a leash. The only time she doesn't is when no man have proposed to her previously. And if they propose to her and she keeps him on a leash, the man will continue to propose to her.

3. a)

Female	day 1	day 2	day 3	day 4	day 5	day 6
A	1 (4)	(4)	(4)	4 (3)	(3)	(3)
B	(2)	2 (1)	(1)	(1)	1 (4)	(4)
C	(3)	(3)	3 (2)	(2)	(2)	2 (1)
D						

Female	day 7	day 8	day 9	day 10
A	3 (2)	(2)	(2)	2
B	(4)	4 (3)	(3)	3
C	(1)	(1)	1 (4)	4
D				1

So the pairings is (A, 2) ; (B, 3) ; (C, 4) ; (D, 1)

3b) Prove: algorithm will terminate after $n(n+1)+1$ proposal. (unique*)

Proof: we will prove by induction.

Base Case: $n=2$. $2(1)+1=3$. This is true b/c of the following cases:

Case 1:

F_A	M_A M_B	⟶	F_A	M_A
F_B			F_B	M_B

* note it doesn't matter which female gets both proposal b/c on the first day there are 2 unique proposal & one male (either is fine) gets rejected & will propose to ~~fem~~ the second female will get the other proposal, which is the 3rd unique proposal. □

Case 2:

F_A	M_A
F_B	M_B

* again, who gets paired up with who doesn't matter & since all females received a proposal, algorithm terminates & there ~~are 2~~ are 2 unique proposal. □

Then there exists the longest case of 3 unique proposals which fits the equation.

Hypothesis: there exists k-pairs such that their preferences causes them to take the longes time that fits the equation $k(k+1)-1$.

Step: 1. Start with k couples & using hypothesis, to sort out the best optimal pairings with their current preferences will take $k(k+1)+1$ unique proposals.

2. Add in Couple M_{k+1} & W_{k+1}.

3. Change up the preferences & add in the fact that W_{k+1} is the least preferred amongst all males.

4. Note that now every Women except for W_{k+1} has at least 1 proposal & therefore at least have 1 ~~suitor~~ proposal each day until algorithm terminates.

5. Since W_{k+1} is least prefered amongst all females, M_{k+1} will propose to W_j where $W_1 \leq W_j \leq W_k$. This will add 1 to our current equation.

6. Now that woman W_j will have 2 options, she will pick one such that the other man will cause a domino affect such that the rest of the women (besides the one W_j & W_{k+1}) will receive a new proposal everyday. This will add $(k-1)$ to the list of new proposal.

7. Number 6 repeats but now ~~going to~~ starting with a new guy M_j st. $M_1 \leq M_j \leq M_{k+1}$. this means that it will add an addition $(k-1)$ to our equation.

8. After that, we've exhausted all the possibilities, and a man M_j has to propose to W_{k+1} which will terminate our algorithm since all females have a proposal now. This adds 1 to our equation.

9. Our equation now looks like this:

$$k(k-1)+1+2(k-1)+2 = k(k-1)+1+2k$$

$$= k^2-k+2k+1 = k^2+k+1 = k(k+1)+1$$

$$= ((k+1)-1)(k+1)+1 = (k+1)((k+1)-1)-1.$$

Hence proven this equation works for $k+1$. ∎

4. Prove : this algorithm results in the same pairings.

Proof : Note that males will (when starting out their proposal) propose in order. As in he will not propose his least favorite woman before all the other ones but he can propose them at the same time.

When they start proposing (whether it's M_1 or M_n or M_k) The women will repeat the same choice in selection as she did with the original algorithm.

As these women slowly choose the male best fits her, the men who were rejected will go through the same proposal algorithm as the original one. This cycle will repeat until all men & women are paired up. Since this cycle matches the original one, it should output a set of pairings the same as the original. QED.

1.
```
def existCeleb(list p, list notCeleb):
    if (# of p== 1):
        for (pn = notCeleb.get(n)):
            if (p.get(1) knows pn):
                return "no celeb"
        return "exist celeb"
    else if (# of p ==0):
        return "no celeb"
    if (p.get(1) knows p.get(2)):
        if (p.get(2) knows p.get(1)):
            notCeleb.add(p.get(2))
            p.remove(2)
        notCeleb.add(p.get(1))
        p.remove(1)
        if(# of p ==1):
            return "exist celeb"
    else   existCeleb(p, notCeleb)
    else:
        notCeleb.add(p.get(2))
        p.remove(2)
        isCeleb(p, not Celeb)
```

1. Prove: Can determine whether there is a celeberty at a party by asking at most $3k-4$ questions.

Proof: we will prove using induction.

Base Case: for $n=2$ (2 people at the party) you can figure out if there exists any celeberty by $3(2)-4 = 6-4 = 2$. Proof:

1. We ask n_1 if n_1 knows n_2.

   Case 1. n_1 doesn't know n_2. So n_2 is removed from celeb. But we need to check if n_1 is know so we ask again.

   Case 2. If n_1 knows n_2, n_1 is not a celeb; but we need to check if n_2 is a celeb by asking n_2 if n_2 knows n_1.

2. In both cases, they were asked $2x$. □

Hypothesis: there exists k people such that you can find out if there exists a celeberty in $3k-4$.

Step: Going throug just k people, we can find out if there's a celeberty ~~in $3k-4$ questions. Using our method, after $3k-4$ question, either we have no one left~~ in $2k$ questions for just this set. Either we are left with 1 person or none.

Case left w/ 0: we ask person$_{k+1}$ if he knows all of the k-people here. This will tell us if there exists a celeb or not. This is now $3k-1$ since asking himself is redundant.

Case left w/ 1: we compare Person$_k$ & Person$_{k+1}$ this only adds 2 to the algorithm.

Hence, regardless it'll take less than $3k-4$.

Since $3(k+1)-4 = 3k-1$. < less than this.