Algorithm
Sort C so that C's with earliest last meeting are paired 1st
If there's a C that starts later than C' finishes
C is paired with it's earliest E
while there exists a free E
E meets with C
if C has not been paired yet
(C,E) get paired
else (C,E') are paired
if C prefers E' to E
E remains free
else
(C,E) get paired and E' is free

☆ prefers can be defined as "has the latest available meeting with"

Proof:

Say we have a schedule where a C finishes before C' starts, by pairing C' with it's first E, C' cannot be stood up.

Then by sorting the C's we are able to pair C's in such a way that the C's who ~~finish first~~ have the earliest last meeting get paired first. So now for any C who finishes after, they cannot be paired with that E. As the ~~set~~ # of pairs increase the possible pairs for C' diminish. Knowing that no E' can meet with 2 Cs at any time, we can see that the meeting between C' and E' is in a unique time slot.

If C' and E' meet before C' is supposed to meet any ~~other~~ E" then there is no conflict and C' is not stood up.

If C' and E' meet after C' is supposed to meet any E than C' gets stood up but knowing that E has already been paired with another C ~~meaning that~~ in an earlier time slot than E is not available to C'. So C' must meet E' before C' meets with any other E's that might stand C' up

Analysis:

Sort $\underline{C}$ $O(n^2)$

If $C_{start} > C'_{finish}$ $O(n)$

C is paired with C's first E

while there exists a free E $O(n^2)$

E meets with C $O(1)$

if C has not been paired yet

(C, E) get paired

else (C, E') are paired

if C prefers E' to E

E remains free

else

(C, E) get paired and E' is free $O(1)$

Sorting takes $O(n^2)$ bc we check each C and ~~then~~ Check the time slots in C

Comparing $C_s > C'_f$ is $O(n^2)$ because we check each C and the last time slot in each C

Pairing E to C takes $O(n^2)$ bc we need to pair each E to a C

- use a loop to go through E's and a nested loop for C's