Some quibbles.

The induction principle works on the natural numbers.
Proves statements of form: \( \forall n \in \mathbb{N}, P(n) \).

Yes.

What if the statement is only for \( n \geq 3 \)?
\( \forall n \in \mathbb{N}, (n \geq 3) \implies P(n) \)

Restate as:
\( \forall n \in \mathbb{N}, \exists Q(n) \) where \( Q(n) = \neg(n \geq 3) \implies P(n) \)

Base Case: Typically start at 3.
Since \( \forall n \in \mathbb{N}, Q(n) \implies Q(n + 1) \) is trivially true before 3.

Can you do induction over other things? Yes.

Any set where any subset of the set has a smallest element.
In some sense, the natural numbers.

Strong Induction and Recursion.

Thm: For every natural number \( n \geq 12, n = 4x + 5y \).
Instead of proof, let’s write some code!

```python
def find-x-y(n):
    if (n==12): return (3,0)
    elif (n==13): return (2,1)
    elif (n==14): return (1,2)
    elif (n==15): return (0,3)
    else:
        (x’,y’) = find-x-y(n-4)
        return (x’+1,y’)
```

Base cases: \( P(12) , P(13) , P(14) , P(15) \). Yes.

Strong Induction step:
Recursive call is correct: \( P(n-4) \implies P(n) \).
\( n = 4x + 5y \implies n = 4(x+1) + 5(y) \)

Slight differences: showed for all \( n \geq 16 \) that \( \sum_{i=1}^{n} P(i) \implies P(n) \).

Stable Matching Problem

▶ \( n \) candidates and \( n \) jobs.
▶ Each job has a ranked preference list of candidates.
▶ Each candidate has a ranked preference list of jobs.

How should they be matched?
Count the ways..

- Maximize total satisfaction.
- Maximize number of first choices.
- Maximize worse off.
- Minimize difference between preference ranks.

The best laid plans..

Consider the pairs...

- (Anthony) Davis and Pelicans
- (Lonzo) Ball and Lakers

Davis prefers the Lakers.
Lakers prefer Davis.
Uh...oh. Sad Lonzo and Pelicans.

So..

Produce a pairing where there are no crazy moves!

**Definition:** A pairing is disjoint set of n job-candidate pairs.
Example: A pairing $S = \{(\text{Lakers, Ball}), (\text{Pelicans, Davis})\}.$

**Definition:** A rogue couple $b, g^*$ for a pairing $S$:
$b$ and $g^*$ prefer each other to their partners in $S$.
Example: Davis and Lakers are a rogue couple in $S$.

A stable pairing??

Given a set of preferences.
Is there a stable pairing?
How does one find it?

Consider a single type version: stable roommates.

A B C D
B C A D
C A B D
D A B C

The Propose and Reject Algorithm.

Each Day:
1. Each job proposes to its favorite candidate on its list.
2. Each candidate rejects all but their favorite proposer (whom they put on a string.)
3. Rejected job crosses rejecting candidate off its list.

Stop when each job gets exactly one proposal.
Does this terminate?
...produce a pairing?
....a stable pairing?
Do jobs or candidates do “better”?

Example.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C A B</td>
</tr>
<tr>
<td>B</td>
<td>X X 3</td>
</tr>
<tr>
<td>C</td>
<td>X 1 3</td>
</tr>
<tr>
<td></td>
<td>A C B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
<th>Day 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>A</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Terminal.

Every non-terminated day a job crossed an item off the list.
Total size of lists? n jobs, n length list. \( n^2 \)
Terminates in \( \leq n^2 \) steps!

Pairing when done.

Lemma: Every job is matched at end.
(Launch Proof poll.)

Proof:
If not, a job \( b \) must have been rejected \( n \) times.
Every candidate has been proposed to by \( b \),
and Improvement lemma

\( \Rightarrow \) each candidate has a job on a string,
and each job is on at most one string.

\( n \) candidates and \( n \) jobs. Same number of each.
\( \Rightarrow \) \( b \) must be on some candidate's string!
Contradiction.

Pairing is Stable.

Lemma: There is no rogue couple for the pairing formed by traditional marriage algorithm.

Proof:
Assume there is a rogue couple; \( (b, g') \)

\[ b' \rightarrow g' \quad b \text{ prefers } g' \text{ to } g. \]

\[ b \rightarrow g \quad g' \text{ prefers } b \text{ to } b'. \]

Job \( b \) proposes to \( g' \) before proposing to \( g \).
So \( g' \) rejected \( b \) (since he moved on)
By Improvement lemma, \( g' \) prefers \( b' \) to \( b \).
Contradiction!

Good for jobs? candidates?

Is the Job-Proposes better for jobs? for candidates?
Definition: A pairing is \( x \)-optimal if \( x \)'s partner
is its best partner in any stable pairing.

Definition: A pairing is \( x \)-pessimal if \( x \)'s partner
is its worst partner in any stable pairing.

Definition: A pairing is job optimal if it is \( x \)-optimal for all jobs \( x \).

Claim: The optimal partner for a job must be first in its preference list.
True? False? False!

Subtlety here: Best partner in any stable pairing.
As well as you can be in a globally stable solution!

Question: Is there a job or candidate optimal pairing?
Is it possible:
\( b \)-optimal pairing different from the \( b' \)-optimal pairing!
Yes? No?
Understanding Optimality: by example.

A: 1,2 1: A,B
B: 1,2 2: B,A
Consider pairing: (A,1), (B,2).

Stable? Yes.
Optimal for B?
Notice: only one stable pairing.
So this is the best B can do in a stable pairing.
So optimal for B.
Also optimal for A, 1 and 2.
Also pessimal for A, B, 1 and 2.

A: 1,2 1: B,A
B: 2,1 2: A,B
Pairing S: (A,1), (B,2).
Stable? Yes.
Pairing T: (A,2), (B,1). Also Stable.
Which is optimal for A? S
Which is optimal for B? S
Which is optimal for 1? T
Which is optimal for 2? T

Job Propose and Candidate Reject is optimal!

For jobs? For candidates?

Theorem: Job Propose and Reject produces a job-optimal pairing.
Proof:
Assume not: there is a job b does not get optimal candidate, g.
There is a stable pairing S where b and g are paired.
Let f be first day job b gets rejected
by its optimal candidate g who it is paired with in stable pairing S.
b∗ - knocks b off of g's string on day t ⇒ g prefers b∗ to b
By choice of t, b∗ likes g at least as much as optimal candidate.
⇒ b∗ prefers g to its partner g∗ in S.
Rogue couple for S.
So S is not a stable pairing. Contradiction.

Notes:
S - stable.
(b∗, g∗) ∈ S.
But (b∗, g) is rogue couple!
Used Well-Ordering principle...Induction.

How about for candidates?

Theorem: Job Propose and Reject produces candidate-pessimal pairing.

T – pairing produced by JPR.
S – worse stable pairing for candidate g.
In T, (g, b) is pair.
In S, (g, b∗) is pair.
g prefers b to b∗.
T is job optimal, so b prefers g to its partner in S.
(g, b) is Rogue couple for S
S is not stable.
Contradiction.

Notes:
Not really induction.
Structural statement: Job optimality ⇒ Candidate pessimality.

Quick Questions.

How does one make it better for candidates?
Propose and Reject - stable matching algorithm. One side proposes.
Jobs Propose ⇒ job optimal.
Candidates propose. ⇒ optimal for candidates.

Residency Matching..

The method was used to match residents to hospitals.
Hospital optimal...until 1990’s...Resident optimal.
Another variation: couples.

Takeaways.

Analysis of cool algorithm with interesting goal: stability.
“Economic”: different utilities.
Definition of optimality: best utility in stable world.
Action gives better results for individuals but gives instability.
Induction over steps of algorithm.
Proofs carefully use definition:
Optimality proof:
contradiction of the existence of a better pairing.