Stable Matching Problem

Designing the Algorithm

- Initially, everyone is unmarried.
- Suppose an unmarried man *m chooses* the woman *w who ranks highest on his preference list and proposes to* her. Can we declare immediately that *(m,w) will be one of the pairs in our* final stable matching?
- Not necessarily: at some point in the future, a man *m whom w prefers may propose to her. On the other hand, it would be* dangerous for *w to reject m right away; she may never receive a proposal* from someone she ranks as highly as *m.*
- *So a natural idea would be to* have the pair *(m, w) enter an intermediate state—engagement.*

Designing the Algorithm

- Suppose we are now at a state in which some men and women are *free*—not engaged—and some are engaged.
- An arbitrary free man *m chooses the highest-ranked woman w to whom* he has not yet proposed, and he proposes to her. If *w is also free, then m* and *w become engaged.*
- *Otherwise, w is already engaged to some other* man *m'.*
- *In this case, she determines which of m or m' ranks higher* on her preference list; this man becomes engaged to *w and the other* becomes free.

Designing the Algorithm

- Finally, the algorithm will terminate when no one is free;
- At this moment, all engagements are declared final, and the resulting perfect matching is returned.

Gale-Shapley Algorithm

```
Initially all $m \in M$ and $w \in W$ are free
While there is a man $m$ who is free and hasn't proposed to
every woman
   Choose such a man $m$
   Let $w$ be the highest-ranked woman in $m$'s preference list
      to whom $m$ has not yet proposed
   If $w$ is free then
      $(m, w)$ become engaged
   Else $w$ is currently engaged to $m'$
      If $w$ prefers $m'$ to $m$ then
         $m$ remains free
      Else $w$ prefers $m$ to $m'$
         $(m, w)$ become engaged
         $m'$ becomes free
      Endif
   Endif
Endwhile
Return the set $S$ of engaged pairs
```

Analyzing the Algorithm

- The view of a woman *w during the execution of the algorithm.*
  - *w remains engaged from the point at which she receives her first proposal; and the sequence of partners to which she is engaged gets better and better (in terms of her preference list).*

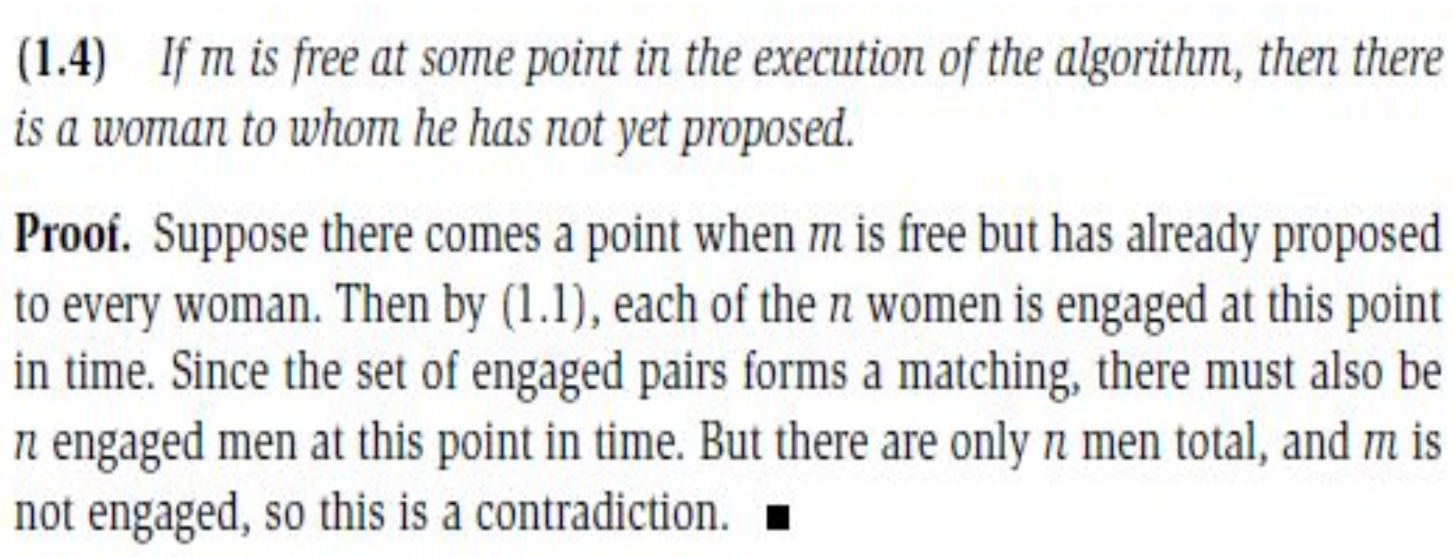
Analyzing the Algorithm

- The view of a man m *during the execution of the algorithm.*
  - *The sequence of women to whom m proposes gets worse and worse (in terms of his preference list).*

Analyzing the Algorithm

- Now we show that the algorithm terminates, and give a bound on the maximum number of iterations needed for termination.
  - *The G-S algorithm terminates after at most* n^2 *iterations of the While loop.*
  - there are only n^2 *possible pairs of men and women* in total

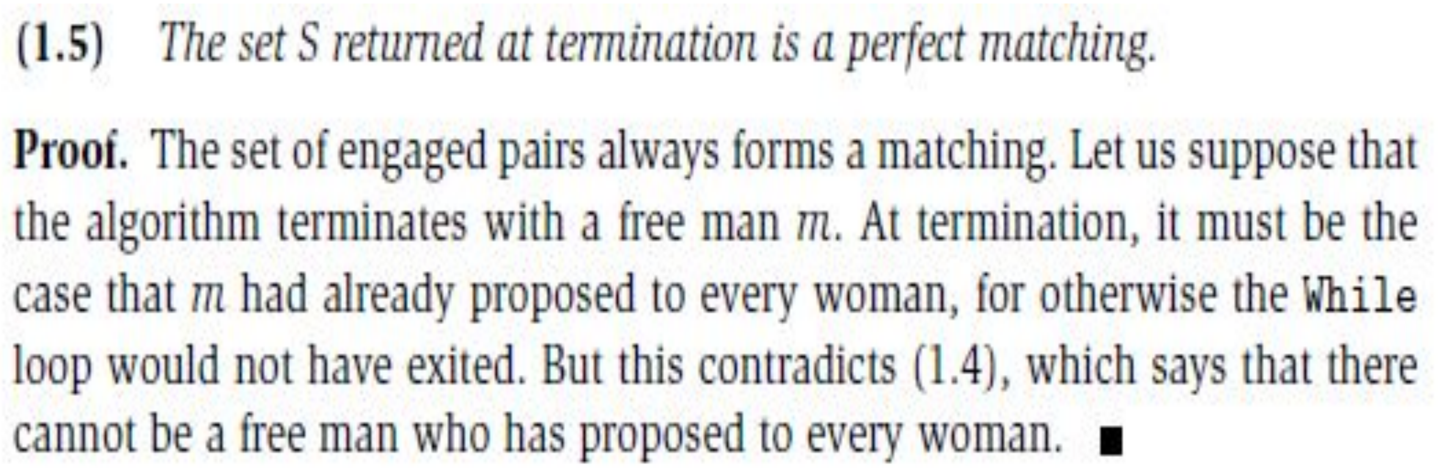
Analyzing the Algorithm



(1.4) *If m is free at some point in the execution of the algorithm, then there is a woman to whom he has not yet proposed.*

Proof. Suppose there comes a point when m is free but has already proposed to every woman. Then by (1.1), each of the n women is engaged at this point in time. Since the set of engaged pairs forms a matching, there must also be n engaged men at this point in time. But there are only n men total, and m is not engaged, so this is a contradiction. ∎

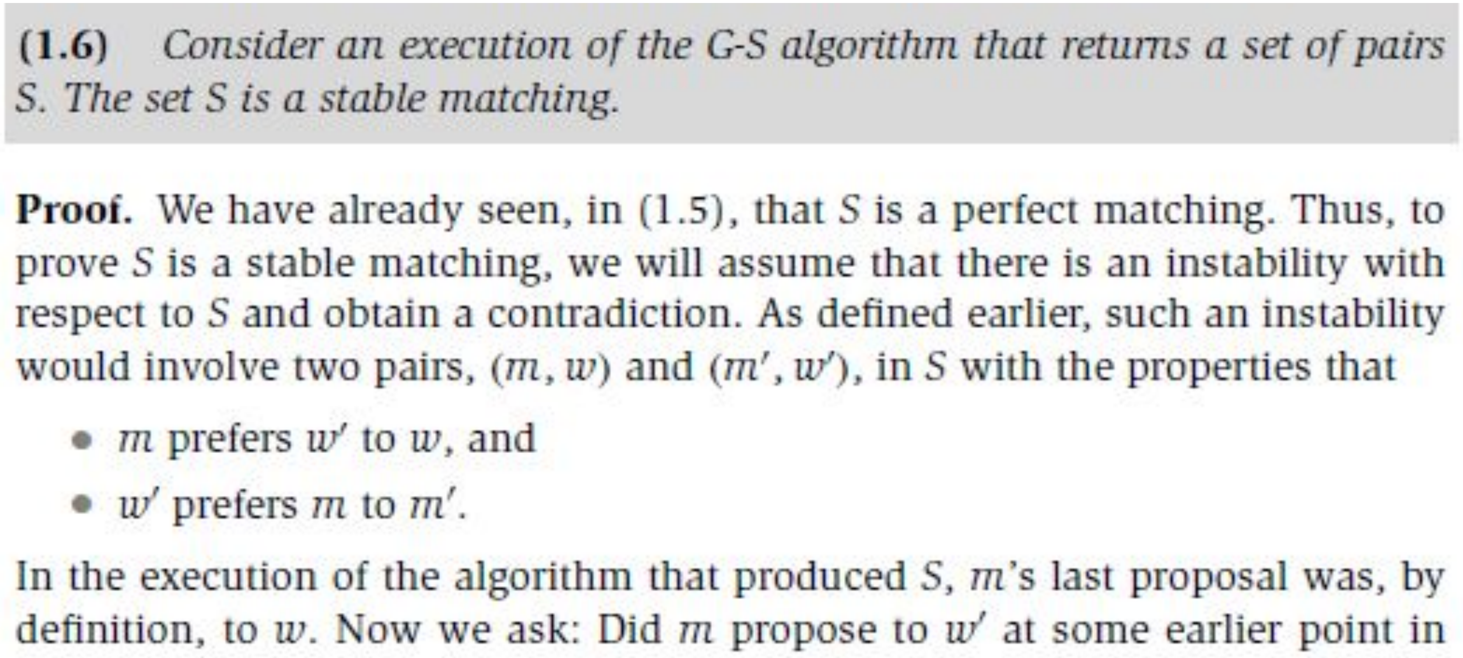
Analyzing the Algorithm



(1.5) *The set S returned at termination is a perfect matching.*

Proof. The set of engaged pairs always forms a matching. Let us suppose that the algorithm terminates with a free man m. At termination, it must be the case that m had already proposed to every woman, for otherwise the `While` loop would not have exited. But this contradicts (1.4), which says that there cannot be a free man who has proposed to every woman. ∎

Analyzing the Algorithm



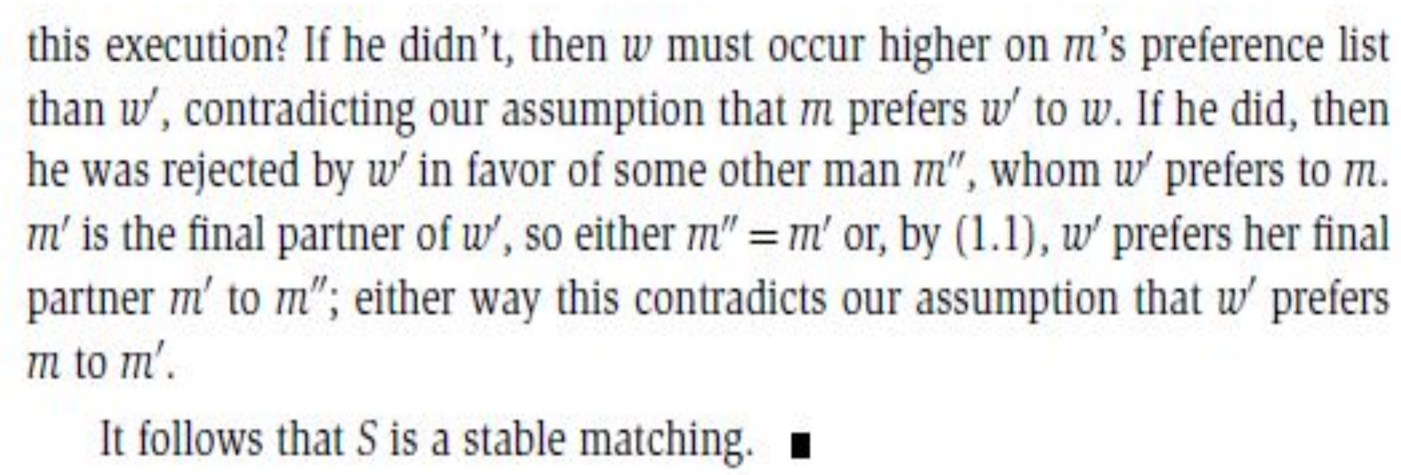
(1.6) *Consider an execution of the G-S algorithm that returns a set of pairs S. The set S is a stable matching.*

Proof. We have already seen, in (1.5), that S is a perfect matching. Thus, to prove S is a stable matching, we will assume that there is an instability with respect to S and obtain a contradiction. As defined earlier, such an instability would involve two pairs, (m, w) and (m', w'), in S with the properties that

- m prefers w' to w, and
- w' prefers m to m'.

In the execution of the algorithm that produced S, m's last proposal was, by definition, to w. Now we ask: Did m propose to w' at some earlier point in

Analyzing the Algorithm

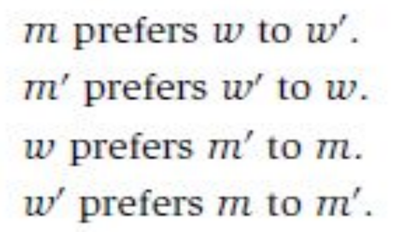


this execution? If he didn't, then w must occur higher on m's preference list than w', contradicting our assumption that m prefers w' to w. If he did, then he was rejected by w' in favor of some other man m'', whom w' prefers to m. m' is the final partner of w', so either $m'' = m'$ or, by (1.1), w' prefers her final partner m' to m''; either way this contradicts our assumption that w' prefers m to m'.

It follows that S is a stable matching. ∎

Extensions

- We now consider some further questions about the behavior of the G-S algorithm and its relation to the properties of different stable matchings

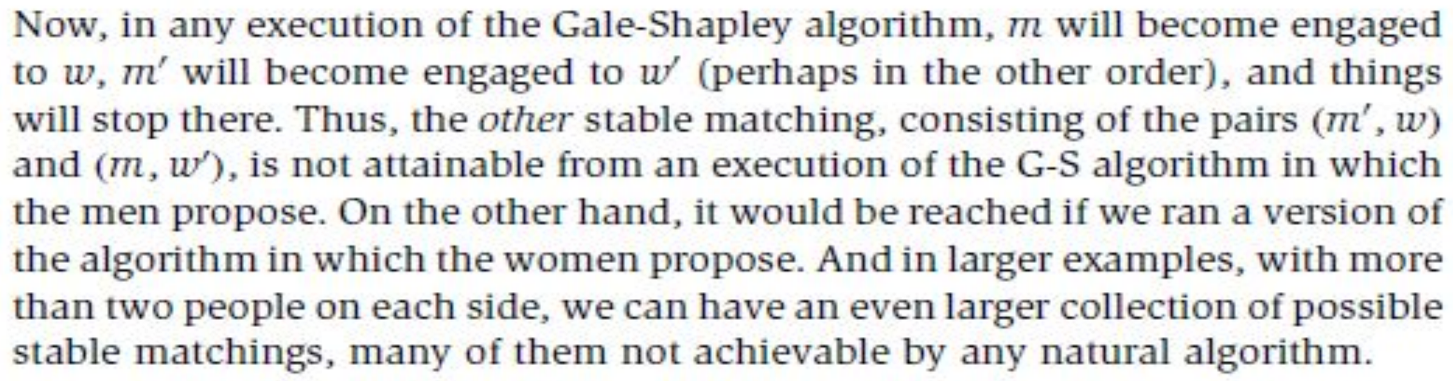


m prefers w to w'.

m' prefers w' to w.

w prefers m' to m.

w' prefers m to m'.



Now, in any execution of the Gale-Shapley algorithm, m will become engaged to w, m' will become engaged to w' (perhaps in the other order), and things will stop there. Thus, the *other* stable matching, consisting of the pairs (m', w) and (m, w'), is not attainable from an execution of the G-S algorithm in which the men propose. On the other hand, it would be reached if we ran a version of the algorithm in which the women propose. And in larger examples, with more than two people on each side, we can have an even larger collection of possible stable matchings, many of them not achievable by any natural algorithm.

Extensions

- So this simple set of preference lists compactly summarizes a world in which *someone is destined to end up unhappy: women are unhappy* if men propose, and men are unhappy if women propose.
- Different choices specify different executions of the algorithm; this is why, to be careful, we stated (1.6) as "Consider an execution of the G-S algorithm that returns a set of pairs *S,"* instead of "Consider the set *S returned by the G-S algorithm."*

Extensions

- All executions of the G-S algorithm yield the same matching.
  - First, we will say that a woman *w is a valid partner* of a man *m if there is a stable matching that contains the pair (m, w).*
  - *We will* say that *w is the best valid partner of m if w is a valid partner of m, and no* woman whom *m ranks higher than w is a valid partner of his.*
  - *We will use best(m) to denote the best valid partner of m.*

Implementing the Stable Matching Algorithm Using Lists and Arrays

- In order to asymptotically analyze the running time of an algorithm expressed in a high-level fashion.
- For example: one doesn’t have to actually program, compile, and execute it, but one does have to think about how the data will be represented and manipulated in an implementation of the algorithm, so as to bound the number of computational steps it takes.

Implementing the Stable Matching Algorithm Using Lists and Arrays

- The algorithm terminates in at most n^2 *iterations.*
- Our implementation here provides a corresponding worst-case running time of $O(n^2)$*, counting actual* computational steps rather than simply the total number of iterations.

Implementing the Stable Matching Algorithm Using Lists and Arrays

- To get such a bound for the Stable Matching algorithm, we will only need to use two of the simplest data structures: lists and arrays.
- Thus, our implementation also provides a good chance to review the use of these basic data structures as well.

Implementing the Stable Matching Algorithm Using Lists and Arrays

- In the Stable Matching Problem, each man and each woman has a ranking of all members of the opposite gender.
- The very first question we need to discuss is how such a ranking will be represented.
- Further, the algorithm maintains a matching and will need to know at each step which men and women are free, and who is matched with whom.
- In order to implement the algorithm, we need to decide which data structures we will use for all these things.

Implementing the Stable Matching Algorithm Using Lists and Arrays

- The choice of data structure is up to the algorithm designer;
- for each algorithm we will choose data structures that make it efficient and easy to implement.
- In some cases, this may involve preprocessing the input to convert it from its given input representation into a data structure that is more appropriate for the problem being solved.

Arrays and Lists

- Focus on a single list, such as the list of women in order of preference by a single man.
- Maybe the simplest way to keep a list of n elements is to use an array A of length n, and have A[i] be the i^{th} element of the list.

Arrays and Lists

- An array is simple to implement in essentially all standard programming languages, and it has the following properties.
  - We can answer a query of the form “What is the i^{th} element on the list?” in O(1) time, by a direct access to the value A[i].
  - To determine whether a particular element e belongs to the list (i.e., whether it is equal to A[i] for some i), we need to check the elements one by one in O(n) time, assuming we don’t know anything about the order in which the elements appear in A.

Arrays and Lists

- An array is simple to implement in essentially all standard programming languages, and it has the following properties.
  - If the array elements are sorted in some clear way (either numerically or alphabetically), then we can determine whether an element *e belongs* to the list in *O(log n) time using binary search.*

Arrays and Lists

- An array is less good for dynamically maintaining a list of elements that changes over time, such as the list of free men in the Stable Matching algorithm;
- since men go from being free to engaged, and potentially back again, a list of free men needs to grow and shrink during the execution of the algorithm.
- It is generally cumbersome to frequently add or delete elements to a list that is maintained as an array.

Arrays and Lists

- An alternate, and often preferable, way to maintain such a dynamic set of elements is via a linked list.
- In a linked list, the elements are sequenced together by having each element point to the next in the list.
- Thus, for each element v on the list, we need to maintain a pointer to the next element; we set this pointer to null if i is the last element.

Arrays and Lists

- We also have a pointer First that points to the first element.
- By starting at First and repeatedly following pointers to the next element until we reach null, we can thus traverse the entire contents of the list in time proportional to its length.
- Singly linked list and doubly linked list

Arrays and Lists

- While lists are good for maintaining a dynamically changing set, they also have disadvantages.
- Unlike arrays, we cannot find the ith *element of the list in O(1) time: to find the* ith *element, we have to follow the Next pointers starting* from the beginning of the list, which takes a total of *O(i) time.*

Arrays and Lists

- Given the relative advantages and disadvantages of arrays and lists, it may happen that we receive the input to a problem in one of the two formats and want to convert it into the other.
- Such preprocessing is often useful; and in this case, it is easy to convert between the array and list representations in O(n) time.
- This allows us to freely choose the data structure that suits the algorithm better and not be constrained by the way the information is given as input.

Implementing the Stable Matching Algorithm

- We will use arrays and linked lists to implement the Stable Matching algorithm.
- The algorithm terminates in at most n^2 iterations, and this provides a type of upper bound on the running time.
- If we actually want to implement the G-S algorithm so that it runs in time proportional to n^2, we need to be able to implement each iteration in constant time.

Implementing the Stable Matching Algorithm

- For simplicity, assume that the set of men and women are both {1, . . . , *n*}.
- To ensure this, we can order the men and women (say, alphabetically), and associate number *i* with the *i*th man m_i or *i*th women w_i in this order.
- This assumption (or notation) allows us to define an array indexed by all men or all women. We need to have a preference list for each man and for each woman.

Implementing the Stable Matching Algorithm

- We need to have a preference list for each man and for each woman.
- To do this we will have two arrays, one for women's preference lists and one for the men's preference lists;
- we will use ManPref[*m*, *i*] to denote the *i*th woman on man *m*'s preference list.
- WomanPref[*w*, *i*] to be the *i*th man on the preference list of woman *w*.

Implementing the Stable Matching Algorithm

- We need to consider each step of the algorithm and understand what data structure allows us to implement it efficiently.
- We need to be able to do each of four things in constant time.
  - We need to be able to identify a free man.
  - We need, for a man *m*, to be able to identify the highest-ranked woman to whom he has not yet proposed.
  - For a woman *w*, we need to decide if *w* is currently engaged, and if she is, we need to identify her current partner.
  - For a woman *w* and two men *m* and *m′*, we need to be able to decide, again in constant time, which of *m* or *m′* is preferred by *w*.

Implementing the Stable Matching Algorithm

- We need to be able to identify a free man.
  - We will do this by maintaining the set of free men as a linked list.
  - Delete engaged man from the list and Insert free man at the front of the linked list.
  - Constant time O(1).

Implementing the Stable Matching Algorithm

- We need, for a man *m*, to be able to identify the highest-ranked woman to whom he has not yet proposed.
  - To do this we will need to maintain an extra array (i.e. Next)
  - Next indicates for each man *m* the position of the next woman he will propose to on his list.
  - We initialize Next[*m*]= 1 for all men *m*.
  - If a man *m* needs to propose to a woman, he'll propose to *w* = ManPref[*m*,Next[*m*]], and once he proposes to *w*, we increment the value of Next[*m*] by one, regardless of whether or not *w* accepts the proposal.
  - Constant time O(1).

Implementing the Stable Matching Algorithm

- For a woman *w*, we need to decide if *w* is currently engaged, and if she is, we need to identify her current partner.
  - We can do this by maintaining an array (i.e. Current) Current of length *n*.
  - where Current[*w*] is the woman *w*’s current partner *m*.
  - We set Current[*w*] to a special null symbol when we need to indicate that woman *w* is not currently engaged; at the start of the algorithm.
  - Current[*w*] is initialized to this null symbol for all women *w*.
  - Constant time O(1).

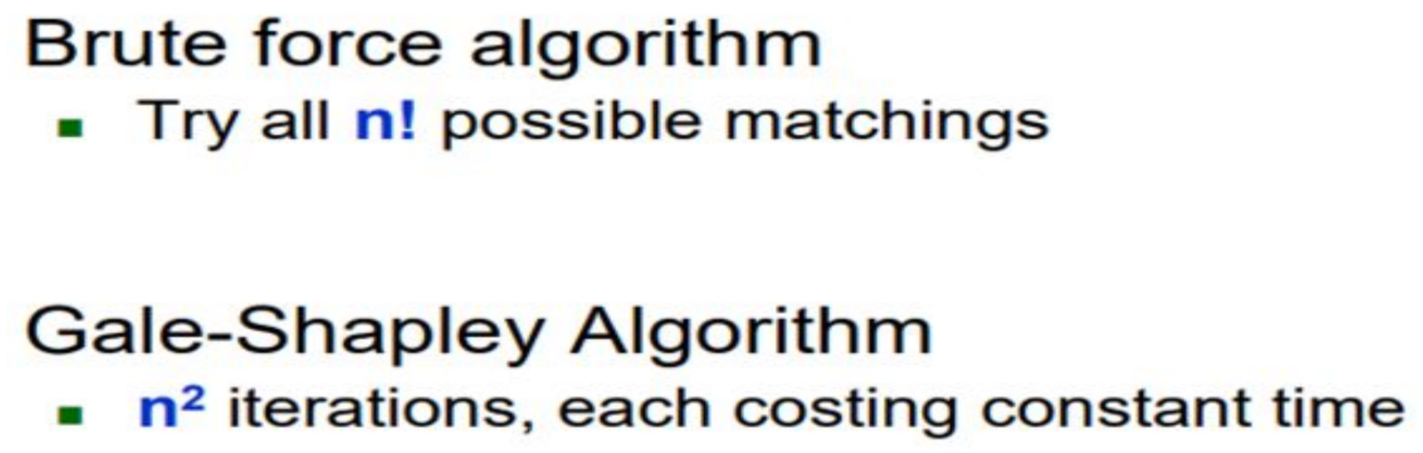
Implementing the Stable Matching Algorithm

- For a woman *w* and two men *m* and *m′*, we need to be able to decide, again in constant time, which of *m* or *m′* is preferred by *w*.
  - as we would need to walk through *w*'s list one by one, taking *O(n)* time to find *m* and *m′* on the list.
  - While *O(n)* is still polynomial, we can do a lot better if we build an auxiliary data structure at the beginning.
  - At the start of the algorithm, we create an n × n array Ranking, where Ranking[w,m] contains the rank of man m in the sorted order of w's preferences.
  - By a single pass through w's preference list, we can create this array in linear time for each woman, for a total initial time investment proportional to n^2 .
  - To decide which of m or m' is preferred by w, we simply compare the values Ranking[w,m] and Ranking[w,m'].
  - Constant time O(1).

Implementing the Stable Matching Algorithm

- Hence we have everything we need to obtain the desired running time.
- The data structures (i.e. Arrays and Lists) allow us to implement the G-S algorithm in O(n^2) time.

Implementing the Stable Matching Algorithm



Brute force algorithm

- Try all **n!** possible matchings

Gale-Shapley Algorithm

- **n^2** iterations, each costing constant time