Algorithms

COMP3121/9101

Aleks Ignjatović, ignjat@cse.unsw.edu.au
office: 504 (CSE building)
Course Admin: Anahita Namvar, cs3121@cse.unsw.edu.au

School of Computer Science and Engineering
University of New South Wales Sydney

1. INTRODUCTION

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is an algorithm?

An algorithm is a collection of precisely defined steps that are executable using certain specified mechanical methods.

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is an algorithm?

An algorithm is a collection of precisely defined steps that are executable using certain specified mechanical methods.

By "mechanical" we mean the methods that do not involve any creativity, intuition or even intelligence. Thus, algorithms are specified by detailed, easily repeatable "recipes".

What is this course about?

It is about **designing algorithms** for solving practical problems.

What is an algorithm?

An algorithm is a collection of precisely defined steps that are executable using certain specified mechanical methods.

By "mechanical" we mean the methods that do not involve any creativity, intuition or even intelligence. Thus, algorithms are specified by detailed, easily repeatable "recipes".

The word "algorithm" comes by corruption of the name of *Muhammad ibn Musa al-Khwarizmi*, a Persian scientist 780-850, who wrote an important book on algebra, *"Al-kitab al-mukhtasar fi hisab al-gabr wal-muqabala"*. You are encouraged to read about him in Wikipedia.

In this course we will deal only with sequential deterministic algorithms which means that:

- they are given as sequences of steps, thus assuming that only one step can be executed at a time;
- the action of each step gives the same result whenever this step is executed for the same input.

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithm **design techniques**

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithm **design techniques**
- particular algorithms will be mostly used to illustrate design techniques

Why should you study algorithms design?

Can you find every algorithm you might need using Google?

Our goal:

To learn **techniques** which can be used to solve **new, unfamiliar** problems that arise in a rapidly changing field.

Course content:

- a survey of algorithm **design techniques**
- particular algorithms will be mostly used to illustrate design techniques
- emphasis on development of your algorithm design **skills**

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

- excellent: very readable textbook (and very pleasant to read!);

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use

An alternative textbook:

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
preferably the third edition, should also be available at the bookstore

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use

An alternative textbook:

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
preferably the third edition, should also be available at the bookstore

- excellent: to be used later as a reference manual;

Textbooks

Textbook:

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store

- excellent: very readable textbook (and very pleasant to read!);
- not so good: as a reference manual for later use.

An alternative textbook:

Cormen, Leiserson, Rivest and Stein: *Introduction to Algorithms*
preferably the third edition, should also be available at the bookstore

- excellent: to be used later as a reference manual;
- not so good: somewhat formalistic and written in a rather dry style.

Problem:

Two thieves have robbed a warehouse and have to split a pile of items without price tags on them. Design an algorithm for doing this in a way that ensures that each thief **believes** that he has got at least one half of the loot.

Examples of Algorithms

Problem:

Two thieves have robbed a warehouse and have to split a pile of items without price tags on them. Design an algorithm for doing this in a way that ensures that each thief **believes** that he has got at least one half of the loot.

The solution:

One of the two thieves splits the pile in two parts, so that he believes that both parts are of equal value. The other thief then chooses the part that he believes is no worse than the other.

Problem:

Two thieves have robbed a warehouse and have to split a pile of items without price tags on them. Design an algorithm for doing this in a way that ensures that each thief **believes** that he has got at least one half of the loot.

The solution:

One of the two thieves splits the pile in two parts, so that he believes that both parts are of equal value. The other thief then chooses the part that he believes is no worse than the other.

The hard part: how can a thief split the pile into two equal parts? Remarkably, this turns out that, most likely, there is no more efficient algorithm than the brute force: we consider all partitions of the pile and see if there is one which results in two equal parts.

Problem:

Three thieves have robbed a warehouse and have to split a pile of items without price tags on them. How do they do this in a way that ensures that each thief believes that he has got at least one third of the loot?

- Remarkably, the problem with 3 thieves is much harder than the problem with 2 thieves!

Problem:

Three thieves have robbed a warehouse and have to split a pile of items without price tags on them. How do they do this in a way that ensures that each thief believes that he has got at least one third of the loot?

- Remarkably, the problem with 3 thieves is much harder than the problem with 2 thieves!

- Let us try do the same trick as in the case of two tieves. Say the first thief splits the loot into three piles which he thinks are of equal value; then the remaining two thieves choose which pile they want to take.

Problem:

Three thieves have robbed a warehouse and have to split a pile of items without price tags on them. How do they do this in a way that ensures that each thief believes that he has got at least one third of the loot?

- Remarkably, the problem with 3 thieves is much harder than the problem with 2 thieves!

- Let us try do the same trick as in the case of two tieves. Say the first thief splits the loot into three piles which he thinks are of equal value; then the remaining two thieves choose which pile they want to take.

- If they choose different piles, they can each take the piles they have chosen and the first thief gets the remaining pile; in this case clearly each thief thinks that he got at least one third of the loot.

Examples of Algorithms

Problem:

Three thieves have robbed a warehouse and have to split a pile of items without price tags on them. How do they do this in a way that ensures that each thief believes that he has got at least one third of the loot?

- Remarkably, the problem with 3 thieves is much harder than the problem with 2 thieves!

- Let us try do the same trick as in the case of two tieves. Say the first thief splits the loot into three piles which he thinks are of equal value; then the remaining two thieves choose which pile they want to take.

- If they choose different piles, they can each take the piles they have chosen and the first thief gets the remaining pile; in this case clearly each thief thinks that he got at least one third of the loot.
- But what if the remaining two thieves choose the same pile?

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:

- after the first thief splits the loot into three piles A, B, C, it might happen, for example, that the second thief thinks that

$$A = 50\%, B = 40\%, C = 10\%$$

of the total value, while the third thief thinks that

$$A = 50\%, B = 10\%, C = 40\%.$$

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:
- after the first thief splits the loot into three piles A, B, C, it might happen, for example, that the second thief thinks that

$$A = 50\%, B = 40\%, C = 10\%$$

of the total value, while the third thief thinks that

$$A = 50\%, B = 10\%, C = 40\%.$$

- Thus, if the first thief picks pile B, then the second thief will object that the first thief is getting 40% while he will likely get only $60\%/2 = 30\%$.

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:

- after the first thief splits the loot into three piles A, B, C, it might happen, for example, that the second thief thinks that

$$A = 50\%, B = 40\%, C = 10\%$$

of the total value, while the third thief thinks that

$$A = 50\%, B = 10\%, C = 40\%.$$

- Thus, if the first thief picks pile B, then the second thief will object that the first thief is getting 40% while he will likely get only $60\%/2 = 30\%$.
- If the first thief picks pile C then the third thief will object for the same reason.

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:

- after the first thief splits the loot into three piles A, B, C, it might happen, for example, that the second thief thinks that

$$A = 50\%, B = 40\%, C = 10\%$$

of the total value, while the third thief thinks that

$$A = 50\%, B = 10\%, C = 40\%.$$

- Thus, if the first thief picks pile B, then the second thief will object that the first thief is getting 40% while he will likely get only $60\%/2 = 30\%$.

- If the first thief picks pile C then the third thief will object for the same reason.
- What would be a correct algorithm?

- One might think that in this case the first thief can pick any of the two piles that the second and the third thief did not choose; the remaining two piles are put together and the two remaining thieves split them as in Problem 1 with only two thieves.
- Unfortunately this does not work:
- after the first thief splits the loot into three piles A, B, C, it might happen, for example, that the second thief thinks that

$$A = 50\%, B = 40\%, C = 10\%$$

of the total value, while the third thief thinks that

$$A = 50\%, B = 10\%, C = 40\%.$$

- Thus, if the first thief picks pile B, then the second thief will object that the first thief is getting 40% while he will likely get only $60\%/2 = 30\%$.
- If the first thief picks pile C then the third thief will object for the same reason.
- What would be a correct algorithm?
- Let the thieves be T_1, T_2, T_3;

Algorithm:

T_1 makes a pile P_1 which he believes is 1/3 of the whole loot;

T_1 proceeds to ask T_2 if T_2 agrees that $P_1 \leq 1/3$;

If T_2 says YES, **then** T_1 asks T_3 if T_3 agrees that $P_1 \leq 1/3$;

  If T_3 says YES, **then** T_1 takes P_1;

   T_2 and T_3 split the rest as in Problem 1.

  Else if T_3 says NO, **then** T_3 takes P_1;

   T_1 and T_2 split the rest as in Problem 1.

Else if T_2 says NO, **then** T_2 reduces the size of P_1 to $P_2 < P_1$ such that T_2 thinks $P_2 = 1/3$;

  T_2 then proceeds to ask T_3 if he agrees that $P_2 \leq 1/3$;

   If T_3 says YES **then** T_2 takes P_2;

    T_1 and T_3 split the rest as in Problem 1.

   Else if T_3 says NO then T_3 takes P_2;

    T_1 and T_2 split the rest as in Problem 1.

<u>**Algorithm:**</u>

T_1 makes a pile P_1 which he believes is $1/3$ of the whole loot;
T_1 proceeds to ask T_2 if T_2 agrees that $P_1 \leq 1/3$;
If T_2 says YES, **then** T_1 asks T_3 if T_3 agrees that $P_1 \leq 1/3$;
    If T_3 says YES, **then** T_1 takes P_1;
        T_2 and T_3 split the rest as in Problem 1.
    Else if T_3 says NO, **then** T_3 takes P_1;
        T_1 and T_2 split the rest as in Problem 1.
Else if T_2 says NO, **then** T_2 reduces the size of P_1 to $P_2 < P_1$ such that T_2 thinks $P_2 = 1/3$;
    T_2 then proceeds to ask T_3 if he agrees that $P_2 \leq 1/3$;
      If T_3 says YES **then** T_2 takes P_2;
        T_1 and T_3 split the rest as in Problem 1.
      Else if T_3 says NO then T_3 takes P_2;
        T_1 and T_2 split the rest as in Problem 1.

Homework: Try generalising this to n thieves! (a bit harder than with three thieves!)

Algorithm:

T_1 makes a pile P_1 which he believes is $1/3$ of the whole loot;

T_1 proceeds to ask T_2 if T_2 agrees that $P_1 \leq 1/3$;

If T_2 says YES, **then** T_1 asks T_3 if T_3 agrees that $P_1 \leq 1/3$;

  If T_3 says YES, **then** T_1 takes P_1;

   T_2 and T_3 split the rest as in Problem 1.

  Else if T_3 says NO, **then** T_3 takes P_1;

   T_1 and T_2 split the rest as in Problem 1.

Else if T_2 says NO, **then** T_2 reduces the size of P_1 to $P_2 < P_1$ such that T_2 thinks $P_2 = 1/3$;

  T_2 then proceeds to ask T_3 if he agrees that $P_2 \leq 1/3$;

   If T_3 says YES **then** T_2 takes P_2;

    T_1 and T_3 split the rest as in Problem 1.

   Else if T_3 says NO then T_3 takes P_2;

    T_1 and T_2 split the rest as in Problem 1.

Homework: Try generalising this to n thieves! (a bit harder than with three thieves!)

Hint: there is a *nested recursion* happening even with 3 thieves!

The role of proofs in algorithm design

When do we need to give a **mathematical proof** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

The role of proofs in algorithm design

When do we need to give a **mathematical proof** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

When this is not obvious by inspecting the algorithm using common sense!

The role of proofs in algorithm design

When do we need to give a **mathematical proof** that an algorithm we have just designed terminates and returns a solution to the problem at hand?

When this is not obvious by inspecting the algorithm using common sense!

Mathematical proofs are **NOT** academic embellishments; we use them to justify things which are not obvious to common sense!

Example: MERGESORT

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. MERGE-SORT(A, p, q)
4. MERGE-SORT$(A, q + 1, r)$
5. MERGE(A, p, q, r)

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. Merge-Sort(A, p, q)
4. Merge-Sort$(A, q+1, r)$
5. Merge(A, p, q, r)

1. The depth of recursion in Merge-Sort is $\log_2 n$.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

1. **if** $p < r$
2. **then** $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$
3. Merge-Sort(A, p, q)
4. Merge-Sort$(A, q+1, r)$
5. Merge(A, p, q, r)

1. The depth of recursion in Merge-Sort is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.

Example: MERGESORT

Merge-Sort(A,p,r) *sorting A[p..r]*

if $p < r$

  then $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$

    MERGE-SORT(A, p, q)

    MERGE-SORT$(A, q+1, r)$

    MERGE(A, p, q, r)

1. The depth of recursion in MERGE-SORT is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MERGESORT always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

if $p < r$

  then $q \leftarrow \lfloor \frac{p+r}{2} \rfloor$

    Merge-Sort(A, p, q)

    Merge-Sort$(A, q+1, r)$

    Merge(A, p, q, r)

1. The depth of recursion in Merge-Sort is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MergeSort always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.
4. Merging two sorted arrays always produces a sorted array, thus, the output of MergeSort will be a sorted array.

Example: MergeSort

Merge-Sort(A,p,r) *sorting A[p..r]*

```
if p < r
  then q ← ⌊(p+r)/2⌋
       Merge-Sort(A, p, q)
       Merge-Sort(A, q + 1, r)
       Merge(A, p, q, r)
```


1. The depth of recursion in MERGE-SORT is $\log_2 n$.
2. On each level of recursion merging intermediate arrays takes $O(n)$ steps.
3. Thus, MERGESORT always terminates and, in fact, it terminates in $O(n \log_2 n)$ many steps.
4. Merging two sorted arrays always produces a sorted array, thus, the output of MERGESORT will be a sorted array.
5. The above is essentially a proof by induction, but we will never bother formalising proofs of (essentially) obvious facts.

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.
- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.
- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; in a lot of cases they are **the only way** to know that the algorithm does the job.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.
- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; in a lot of cases they are **the only way** to know that the algorithm does the job.
- For that reason we will **NEVER** prove the obvious (the CLRS textbook sometimes does just that, by sometimes formulating and proving trivial little lemmas, being too pedantic!). We will prove only what is genuinely nontrivial.

The role of proofs in algorithm design

- However, sometimes it is **NOT** clear from a description of an algorithm that such an algorithm will not enter an infinite loop and fail to terminate.
- Sometimes it is **NOT** clear that an algorithm will not run in exponentially many steps (in the size of the input), which is usually almost as bad as never terminating.
- Sometimes it is **NOT** clear from a description of an algorithm why such an algorithm, after it terminates, produces a desired solution.
- Proofs are needed for such circumstances; in a lot of cases they are **the only way** to know that the algorithm does the job.
- For that reason we will **NEVER** prove the obvious (the CLRS textbook sometimes does just that, by sometimes formulating and proving trivial little lemmas, being too pedantic!). We will prove only what is genuinely nontrivial.
- **However, <u>BE VERY CAREFUL</u> what you call trivial!!**

The Stable Matching Problem

The Stable Matching Problem

- Assume that you are running a dating agency and have n men and n women as customers;

The Stable Matching Problem

- Assume that you are running a dating agency and have n men and n women as customers;
- They all attend a dinner party; after the party:
  - every man gives you a list with his ranking of all women present, and
  - every woman gives you a list with her ranking of all men present;

Role of proofs - example

The Stable Matching Problem

- Assume that you are running a dating agency and have n men and n women as customers;
- They all attend a dinner party; after the party:
  - every man gives you a list with his ranking of all women present, **and**
  - every woman gives you a list with her ranking of all men present;
- Design an algorithm which produces a *stable matching*, which is:

Role of proofs - example

The Stable Matching Problem

- Assume that you are running a dating agency and have n men and n women as customers;
- They all attend a dinner party; after the party:
  - every man gives you a list with his ranking of all women present, **and**
  - every woman gives you a list with her ranking of all men present;
- Design an algorithm which produces a *stable matching*, which is: a set of n pairs $p = (m, w)$ of a man m and a woman w so that the following situation never happens:

The Stable Matching Problem

- Assume that you are running a dating agency and have n men and n women as customers;
- They all attend a dinner party; after the party:
  - every man gives you a list with his ranking of all women present, **and**
  - every woman gives you a list with her ranking of all men present;
- Design an algorithm which produces a *stable matching*, which is: a set of n pairs $p = (m, w)$ of a man m and a woman w so that the following situation never happens:

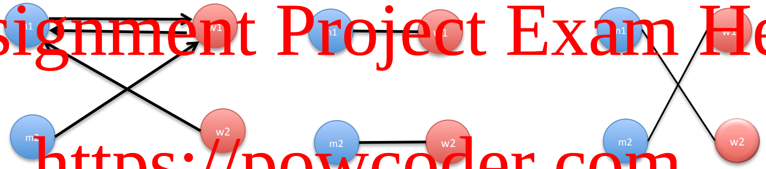
for two pairs $p = (m, w)$ and $p' = (m', w')$:

- man m prefers woman w' to woman w, **and**
- woman w' prefers man m to man m'.

Stable Matching Problem: examples

Case1:

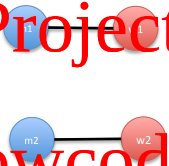
Preferences: | stable matching | not stable



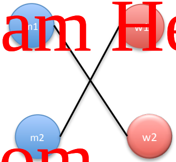
Case1:

Preferences:

stable matching

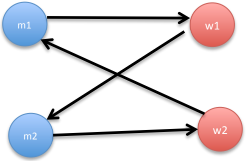


not stable

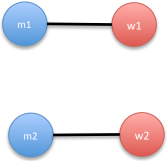


Case2:

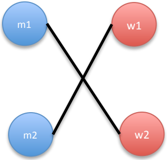
Preferences:



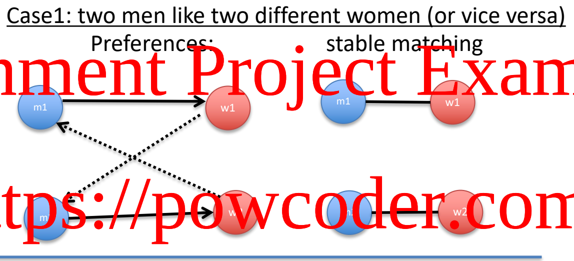
stable matching



also stable!



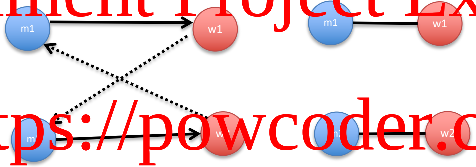
Is there always a stable matching for any preferences of two pairs?



Is there always a stable matching for any preferences of two pairs?

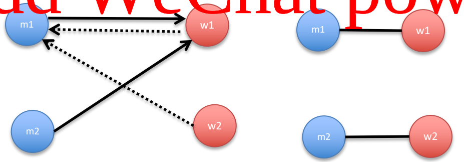
Case1: two men like two different women (or vice versa)

Preferences: stable matching



Case2: men like the same woman and women like the same man

Preferences: stable matching



Stable Matching Problem: Gale - Shapley algorithm

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Stable Matching Problem: Gale - Shapley algorithm

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n!$

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Can we find a stable matching in a reasonable amount of time??

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Can we find a stable matching in a reasonable amount of time??

Answer: **YES**, using the **Gale - Shapley algorithm**.

Question: Is it true that for every possible collection of n lists of preferences provided by all men, and n lists of preferences provided by all women, a stable matching always exists?

Answer: **YES**, but this is **NOT** obvious!

Given n men and n women, how many ways are there to match them, i.e., just to form n couples?

Answer: $n! \approx (n/e)^n$ - more than exponentially many in n ($e \approx 2.71$);

Can we find a stable matching in a reasonable amount of time??

Answer: **YES**, using the **Gale - Shapley algorithm**.

Originally invented to pair newly graduated physicians with US hospitals for residency training.

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
  pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.

- Start with all men free;

While there exists a free man who has not proposed to all women

pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

If no one has proposed to w yet

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.

- Start with all men free;

While there exists a free man who has not proposed to all women

pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

If no one has proposed to w yet

she always accepts and a pair $p = (m, w)$ is formed;

Else she is already in a pair $p' = (m', w)$;

Stable Matching Problem: Gale - Shapley algorithm

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women

pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

If no one has proposed to w yet

she always accepts and a pair $p = (m, w)$ is formed;

Else she is already in a pair $p' = (m', w)$;

If m is higher on her preference list than m'

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women
    pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;
    If no one has proposed to w yet
        she always accepts and a pair $p = (m, w)$ is formed;
    Else she is already in a pair $p' = (m', w)$;
        If m is higher on her preference list than m'
            the pair $p' = (m', w)$ is deleted;

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.
- Start with all men free;

While there exists a free man who has not proposed to all women

  pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

  If no one has proposed to w yet

    she always accepts and a pair $p = (m, w)$ is formed;

  Else she is already in a pair $p' = (m', w)$;

    If m is higher on her preference list than m'

      the pair $p' = (m', w)$ is deleted;

      m' becomes a free man;

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.

- Start with all men free;

While there exists a free man who has not proposed to all women

pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

If no one has proposed to w yet

she always accepts and a pair $p = (m, w)$ is formed;

Else she is already in a pair $p' = (m', w)$;

If m is higher on her preference list than m'

the pair $p' = (m', w)$ is deleted;

m' becomes a free man;

a new pair $p = (m, w)$ is formed;

- Produces pairs in stages, with possible revisions;
- A man who has not been paired with a woman will be called *free*.
- Men will be proposing to women. Women will decide if they accept a proposal or not.

- Start with all men free;

While there exists a free man who has not proposed to all women
  pick such a free man m and have him propose to the highest ranking woman w on his list to whom he has not proposed yet;

  If no one has proposed to w yet
    she always accepts and a pair $p = (m, w)$ is formed;
  Else she is already in a pair $p' = (m', w)$;

    If m is higher on her preference list than m'
      the pair $p' = (m', w)$ is deleted;
      m' becomes a free man;
      a new pair $p = (m, w)$ is formed;
    Else m is lower on her preference list than m';
      the proposal is rejected and m remains free.

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Proof:

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Proof:

- Assume that the while *While* loop has terminated, but m is still free.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.

Stable Matching Problem: Gale - Shapley algorithm

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop

Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)

Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.
- But this would mean that n women are paired with all of n men so m cannot be free.

Claim 1: Algorithm terminates after $\leq n^2$ rounds of the *While* loop
Proof:

- In every round of the *While* loop one man proposes to one woman;
- every man can propose to a woman at most once;
- thus, every man can make at most n proposals;
- there are n men, so in total they can make $\leq n^2$ proposals.

Thus the *While* loop can be executed no more than n^2 many times.

Claim 2: Algorithm produces a matching, i.e., every man is eventually paired with a woman (and thus also every woman is paired to a man)
Proof:

- Assume that the while *While* loop has terminated, but m is still free.
- This means that m has already proposed to every woman.
- Thus, every woman is paired with a man, because a woman is not paired with anyone only if no one has made a proposal to her.
- But this would mean that n women are paired with all of n men so m cannot be free. **Contradiction!**

Claim 3: The matching produced by the algorithm is stable.

Claim 3: The matching produced by the algorithm is stable.
Proof:

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

m prefers w' over w;
w' prefers m over m'.

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$

$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;

Claim 3: The matching produced by the algorithm is stable.

Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable; thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;
- In both cases she would now be with m' whom she prefers over m.

Stable Matching Problem: Gale - Shapley algorithm

Claim 3: The matching produced by the algorithm is stable.
Proof: Note that during the *While* loop:

- a woman is paired with men of increasing ranks on her list;
- a man is paired with women of decreasing ranks on his list.

Assume now the opposite, that the matching is not stable;
thus, there are two pairs $p = (m, w)$ and $p' = (m', w')$ such that:

$$m \text{ prefers } w' \text{ over } w;$$
$$w' \text{ prefers } m \text{ over } m'.$$

- Since m prefers w' over w, he must have proposed to w' before proposing to w;
- Since he is paired with w, woman w' must have either:
  - rejected him because she was already with someone whom she prefers, or
  - dropped him later after a proposal from someone whom she prefers;
- In both cases she would now be with m' whom she prefers over m.
- **Contradiction!**

A Puzzle!!!

Why puzzles? It is a fun way to practice problem solving!

Problem : Tom and his wife Mary went to a party where nine more couples were present.

- Not every one knew everyone else, so people who did not know each other introduced themselves and shook hands.
- People who knew each other from before did not shake hands.
- Later that evening Tom got bored, so he walked around and asked all other guests (including his wife) how many hands they had shaken that evening, and got 19 different answers.
- How many hands did Mary shake?
- How many hands did Tom shake?

That's All, Folks!!