# Algorithms
# COMP3121/3821/9101/9801

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

1. INTRODUCTION

## Introduction

### What is this course about?

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

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

## Resources: textbook

### Default:

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: quite formalistic and written in a rather dry style.

### An alternative (Aleks' choice):

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at the UNSW book store
good: very readable (and very pleasant to read!); an excellent textbook;
not so good: as a reference manual for later use.

## Resources: other

### Forum on Piazza (Moderated by Song Fang)

Ask for clarifications when needed.
Share tips, exercises, other resources.
Help each other out.

### Course email (Monitored by Anahita Namvar)

`cs3821@unsw.edu.au` or `cs9801@unsw.edu.au`

### Consultation (G01, K17)

- Wednesday 2-3pm (Song)
- Thursday 7-8pm (Abdallah)
  Come and see me after class and we can walk to G01 together.
  If you are not in class but want consultation, email me before 7pm.

Song and Anahita will relay to me issues they cannot solve directly.

## Assessments

### Assignment 1 (10%)

Released: week 3 (Monday, March 2)
Due: week 4 (Monday, March 9)

### Assignment 2 (10%)

Released: towards week 7
Due: towards week 9

### Exams

Midterm (40%): week 5 (Thursday, March 19 5pm-7pm)
Final (40%): after week 10 (centrally organized)

## The role of proofs in algorithm design

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

**Only** when this is not clear by 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$(A, p, q, r)$
```

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 formalizing proofs of (essentially) obvious facts.

## 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 essentially 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; thus, proofs are **NOT** academic embellishments - in lots of cases they are **the only way** to know that the algorithm will always work!
- For that reason we will not prove the obvious (your CLRS textbook sometimes does just that, being too pedantic!) and will prove only what is genuinely nontrivial.
- **However, be very careful what you call trivial!!**

## Role of proofs - example

### Stable Matching Problem

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

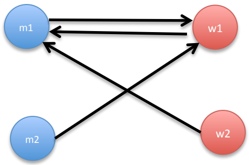
  They all attend a dinner party; after the party
  - every man gives you his ranking of all women present, **and**
  - every woman gives you her ranking of all men present;

- **Task:** Design an algorithm which produces a *stable matching*: 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 $m'$.

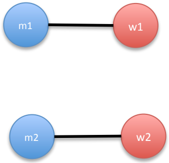
# Stable Matching Problem

Preferences:

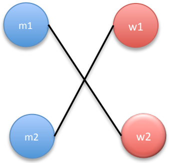


Case1:

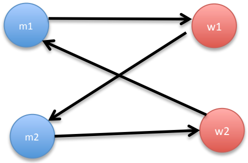
stable matching



not stable

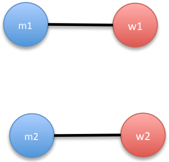


Preferences:

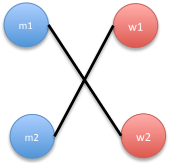


Case2:

stable matching

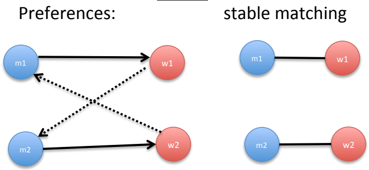


also stable!

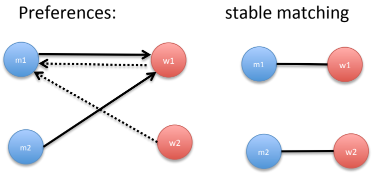


## Is there always a stable matching for any preferences of two pairs?

Case1:



Case2:



## Stable Matching Problem: Gale - Shapley algorithm (invented to pair newly graduated physicians with US hospitals for residency training)

**Question 1:** 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 exists?

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

**Question 2:** 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**.

## 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 $m'$
                        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.
```

## 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.
- But this would mean that $n$ women are paired with all of $n$ men so $m$ cannot be free. **Contradiction!**

## 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!**

## Puzzles!!!

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

### All different handshakes

- **Setting:** 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.
- **Task:**
  - How many hands did Mary shake?
  - How many hands did Tom shake?

## Puzzles!!!

### Balance puzzle

- **Setting:** We have nine coins and three of them are heavier than the remaining six.
- **Task:** Can you find the heavier coins by weighing coins on a pan balance only four times?

- How many outcomes of 4 weighings are there in total?

$$3^4 = 81$$

- How many ways are there to hide three heavier coins among six good coins?

$$\binom{9}{3} = \frac{9!}{3!6!} = \frac{7 \times 8 \times 9}{3!} = 84$$

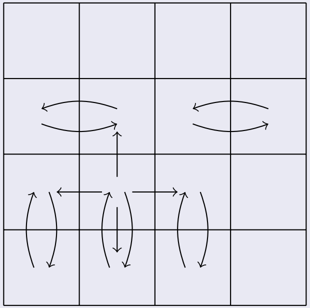
- More ways to hide than the number of all possible weighting outcomes! Thus, it is impossible to do it!

## Puzzles!!!

### Moving houses

- **Setting:**

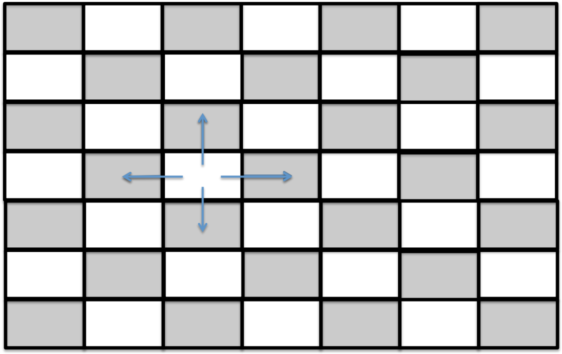
  Consider a block of houses: The inhabitant of each house thinks that all adjacent houses around them (to the left, right, top and bottom) are nicer than their house and would like to move to any of the four.



- **Task:** Can you move the inhabitants around to make them all happier, if the block is $4 \times 4$ houses?
  Can you move the inhabitants around to make them all happier, if the block is $7 \times 7$ houses?

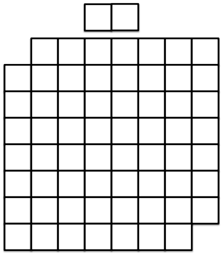
## Puzzles!!!

**Hint:**



## Puzzles!!!

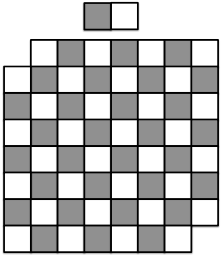
**Problem:** Consider an $8 \times 8$ board with two diagonal squares missing, and an $1 \times 2$ domino:



Can you cover the entire board with 31 such dominoes?

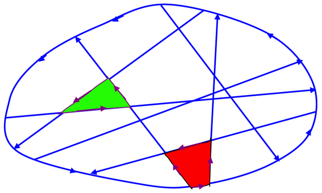
## Puzzles!!!

**Hint:**



## Puzzles!!!

**Problem:** In Elbonia all cities have a circular one-way highway around the city; see the map. All streets in the cities are one-way, and they all start and end on the circular highway (see the map).



- A block is a part of the city that is not intersected by any street.
- Design an algorithm that, given a map of a city, finds a block (just one such block, not all such blocks) that can be circumnavigated while respecting all one-way signs.

(for example, the green block has such property, but not the red one)

That's All, Folks!!