UNSW SYDNEY

1. INTRODUCTION

Aleks Ignjatović, ignjat@cse.unsw.edu.au
office: K17 504
Course Admin: Song Fang, cs3121@cse.unsw.edu.au

School of Computer Science and Engineering
UNSW Sydney

Term 2, 2023

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Asymptotic Notation

6. Puzzles

- Only prerequisite: COMP2521/9024

Prerequisite courses

- Only prerequisite: COMP2521/9024
- Highly desirable (but not officially required)
  - For undergrads (COMP3121): MATH1081 Discrete Mathematics (proofs, graphs)
  - For postgrads (COMP9101): COMP9020 Foundations of Computer Science

- Understanding of fundamental data structures covered in COMP2521/9024

- Understanding of fundamental data structures covered in COMP2521/9024
  - linked lists, arrays, priority queues, heaps, graphs, trees, balanced search trees, hash tables etc.

- Understanding of fundamental data structures covered in COMP2521/9024
  - linked lists, arrays, priority queues, heaps, graphs, trees, balanced search trees, hash tables etc.
- Understanding of fundamental algorithms covered in COMP2521/9024
  - sorting (insertion sort, merge sort, quick sort, heap sort, bubble sort, counting sort, radix sort)
  - binary search, depth first search, breath first search, hashing etc.

- Understanding of fundamental data structures covered in COMP2521/9024
  - linked lists, arrays, priority queues, heaps, graphs, trees, balanced search trees, hash tables etc.
- Understanding of fundamental algorithms covered in COMP2521/9024
  - sorting (insertion sort, merge sort, quick sort, heap sort, bubble sort, counting sort, radix sort)
  - binary search, depth first search, breath first search, hashing etc.
- Use the index to locate these notions in the textbook and/or search the Wikipedia to find articles for reviewing these notions

- Asymptotic notation and basic properties of logarithms; read the review booklet available on Moodle

- Asymptotic notation and basic properties of logarithms; read the review booklet available on Moodle
- Written communication skills

Required knowledge and skills

- Asymptotic notation and basic properties of logarithms; read the review booklet available on Moodle
- Written communication skills
  - No programming involved, emphasis on problem solving, proving correctness of algorithms and estimating efficiency of algorithms

- Asymptotic notation and basic properties of logarithms; read the review booklet available on Moodle
- Written communication skills
  - No programming involved, emphasis on problem solving, proving correctness of algorithms and estimating efficiency of algorithms
  - Smarthinking for writing help

- Tuesday 9:00 - 11:00 and Friday 11:00 - 13:00, weeks 1–10
- Face to face at Keith Burrows Theatre (K-J14-G5)
- Slides and recordings are available on Moodle

Consultation

- Friday 15:00 - 17:00, weeks 1–10
- Join live on MS Teams
- Exam consultation TBA

- Weeks 1–5, 7–10 (no tutorials in flex week)
- Most face to face, some online
- Will demonstrate problem solving and reasoning
- Prepare by reading the tutorial sheet before attending

Practice Problems

- Six sets of practice problems will be released on Moodle, with written solutions (approx weeks 1, 2, 3, 5, 7, 9).
- Discuss these with your friends and on the Ed forum.
- You can also bring these to tutorials.

- Take-home quiz
  - Released week 1, attempt before your second tutorial
  - Due before Friday at Noon in week 4
  - Incorporate tutor's feedback until task complete
  - Weighted 5% of course mark

- Take-home quiz
  - Released week 1, attempt before your second tutorial
  - Due before Friday at Noon in week 4
  - Incorporate tutor's feedback until task complete
  - Weighted 5% of course mark
- Assignments
  - 3 assignments
  - Each consists of 3 questions
  - Each assignment weighted 15% of course mark

- Final Exam
  - Section I: 5 multiple choice questions
  - Section II: 3 algorithm design problems
  - INSPERA (ON campus): more info here
  - Weighted 50% of course mark

- Final Exam
  - Section I: 5 multiple choice questions
  - Section II: 3 algorithm design problems
  - INSPERA (ON campus): more info here
  - Weighted 50% of course mark

- Forum participation
  - Up to 5 bonus marks

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

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

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

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

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

- 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*
4th edition now available at UNSW Bookshop, 3rd edition also useful

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

- 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*
4th edition now available at UNSW Bookshop, 3rd edition also useful

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

Recommended textbook

Kleinberg and Tardos: *Algorithm Design*
paperback edition available at UNSW Bookshop

- 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*
4th edition now available at UNSW Bookshop, 3rd edition also useful

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

Feedback

- Changes from last year:
  - weekly tutorials
  - fewer assignments and fewer questions per assignment
  - FAQ on Moodle and Ed
  - earlier feedback via take-home quiz

- Changes from last year:
  - weekly tutorials
  - fewer assignments and fewer questions per assignment
  - FAQ on Moodle and Ed
  - earlier feedback via take-home quiz

- Feedback is always welcome, e.g.
  - myExperience survey
  - feedback post on Ed (can post anonymously)
  - email

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Asymptotic Notation
6. Puzzles

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 can be executed *mechanically*, i.e. without intelligent decision-making.

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 can be executed *mechanically*, i.e. without intelligent decision-making.
- Designing a recipe involves creativity, executing it does not; the same is true of algorithms.

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 can be executed *mechanically*, i.e. without intelligent decision-making.
- Designing a recipe involves creativity, executing it does not; the same is true of algorithms.
- The word "algorithm" comes by corruption of the name of Muhammad ibn Musa al-Khwarizmi, a Persian scientist 780–850 AD, who wrote an important book on algebra, *"Al-kitab al-mukhtasar fi hisab al-gabr wal-muqabala"*.

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 (no parallelism);

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 (no parallelism);
- the action of each step is not randomised, so the algorithm always gives the same result for the same input.

Our goal:

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

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**

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

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**

- Can I just use Google or ChatGPT to find an algorithm for every problem?

- Can I just use Google or ChatGPT to find an algorithm for every problem?
- In a rapidly changing field new problems pop up all the time, and AI still cannot solve new problems.

Why should you study algorithms?

- Can I just use Google or ChatGPT to find an algorithm for every problem?
- In a rapidly changing field new problems pop up all the time, and AI still cannot solve new problems.
- Many programmers do not have the skills to solve new problems.

Why should you study algorithms?

- Can I just use Google or ChatGPT to find an algorithm for every problem?
- In a rapidly changing field new problems pop up all the time, and AI still cannot solve new problems.
- Many programmers do not have the skills to solve new problems.
- Interview questions at good companies often involve asking you to solve a tricky problem.

Why should you study algorithms?

- Can I just use Google or ChatGPT to find an algorithm for every problem?
- In a rapidly changing field new problems pop up all the time, and AI still cannot solve new problems.
- Many programmers do not have the skills to solve new problems.
- Interview questions at good companies often involve asking you to solve a tricky problem.
- Some of the practice problems in this course are actually Google and Microsoft interview questions.

Problem

Alice and Bob have robbed a warehouse and have to split a pile of items. There is no objective valuation of the items (e.g. price tags); instead, Alice and Bob each have a valuation in mind, and they might not agree.
Design an algorithm to split the pile so that each thief values their own pile as at least half the loot.

Example: Two Thieves

Problem

Alice and Bob have robbed a warehouse and have to split a pile of items. There is no objective valuation of the items (e.g. price tags); instead, Alice and Bob each have a valuation in mind, and they might not agree.
Design an algorithm to split the pile so that each thief values their own pile as at least half the loot.

Solution

Alice splits the pile in two parts, so that she believes that both parts are of equal value. Bob then chooses the part that he believes is no worse than the other.

Note

We are assuming that it's always possible to split up the loot into whatever fraction we like. With discrete items this is more complicated than it might appear!

Example: Two Thieves

Note

We are assuming that it's always possible to split up the loot into whatever fraction we like. With discrete items this is more complicated than it might appear!

Question

If there are n items, and Alice values the ith at v_i dollars, can Alice efficiently split the loot into two equal piles?

Note

We are assuming that it's always possible to split up the loot into whatever fraction we like. With discrete items this is more complicated than it might appear!

Question

If there are n items, and Alice values the ith at v_i dollars, can Alice efficiently split the loot into two equal piles?

Answer

There is no known algorithm that is significantly more efficient than the brute force (try all choices, of which there are 2^n).

Example: Three Thieves

Problem

Alice, Bob and Carol have robbed a warehouse and have to split a pile of items. Each thief has their own valuation of the items. Design an algorithm to split the pile so that each thief values their own pile as at least one third of the loot.

Example: Three Thieves

- The problem is much harder with 3 thieves!

Example: Three Thieves

- The problem is much harder with 3 thieves!
- Let us try do the same trick as in the case of two thieves. Say Alice splits the loot into three piles which she thinks are of equal value; then Bob and Carol each choose which pile they want to take.

Example: Three Thieves

- The problem is much harder with 3 thieves!
- Let us try do the same trick as in the case of two thieves. Say Alice splits the loot into three piles which she thinks are of equal value; then Bob and Carol each choose which pile they want to take.
- If they choose different piles, they can each take the piles they have chosen and Alice gets the remaining pile; in this case clearly each thief thinks that they got at least one third of the loot.

Example: Three Thieves

- But what if Bob and Carol choose the same pile?

Example: Three Thieves

- But what if Bob and Carol choose the same pile?
- One might think that in this case, Alice can pick either of the other two piles, after which the remaining two piles are put together for Bob and Carol to split them as in the earlier problem with only two thieves.

- But what if Bob and Carol choose the same pile?
- One might think that in this case, Alice can pick either of the other two piles, after which the remaining two piles are put together for Bob and Carol to split them as in the earlier problem with only two thieves.
- Unfortunately this does not work!

- Suppose that Alice splits the loot into three piles X, Y, Z, and that Bob thinks that

$$X = 50\%, Y = 40\%, Z = 10\%$$

of the total value, while Carol thinks that

$$X = 50\%, Y = 10\%, Z = 40\%.$$

- Clearly both Bob and Carol choose pile X, so Alice can choose pile Y or Z.

- Clearly both Bob and Carol choose pile X, so Alice can choose pile Y or Z.
- However, if Alice picks pile Y, then Bob will object that (in his eyes) only 60% of the loot remains, so he is guaranteed to get only 30% of the loot i.e. he is not guaranteed to get at least one-third of the total.

- Clearly both Bob and Carol choose pile X, so Alice can choose pile Y or Z.
- However, if Alice picks pile Y, then Bob will object that (in his eyes) only 60% of the loot remains, so he is guaranteed to get only 30% of the loot i.e. he is not guaranteed to get at least one-third of the total.
- If instead Alice picks pile Z, then Carol will object for the same reason.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.
  - Alice then asks Carol whether she thinks that $X \leq 1/3$.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.
  - Alice then asks Carol whether she thinks that $X \leq 1/3$.
  - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.

Algorithm

- Alice makes a pile X which she believes is $1/3$ of the whole loot.
- Alice proceeds to ask Bob whether he agrees that $X \leq 1/3$.
- If Bob says YES, then he would be happy to split the remainder of the loot (worth $\geq 2/3$) with one other thief.
  - Alice then asks Carol whether she thinks that $X \leq 1/3$.
  - If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
  - If Carol says YES, then Alice takes X, and Bob and Carol split the rest.

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.
- Bob asks Carol whether she thinks that $X \leq 1/3$.

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.
- Bob asks Carol whether she thinks that $X \leq 1/3$.
- If Carol says NO, then Carol takes X, and Alice and Bob split the rest.

Algorithm (continued)

- What if Bob says NO? Then Alice values pile X at $1/3$ of the total, but Bob believes it to be $> 1/3$.
- Now we ask Bob to reduce the pile X until he believes it to be $1/3$ of the total. Alice values the new pile as $< 1/3$, so she is happy to split the remainder of the loot (worth $> 2/3$) with one other thief.
- Bob asks Carol whether she thinks that $X \leq 1/3$.
- If Carol says NO, then Carol takes X, and Alice and Bob split the rest.
- If Carol says YES, then Bob takes X, and Alice and Carol split the rest.

Exercise

Try generalising this to n thieves.

Exercise

Try generalising this to n thieves.

Hint

There is a *nested recursion* happening even with 3 thieves!

Table of Contents

1. Admin
2. Solving problems using algorithms
3. Proofs
4. An example of the role of proofs
5. Asymptotic Notation
6. Puzzles

Question

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?

Question

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?

Answer

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!

Algorithm

We recursively apply the following algorithm to the subarray $A[\ell..r]$.

If $\ell = r$, i.e. the subarray has only one element, we simply exit. This is the base case of our recursion.

Otherwise:

- first define $m = \left\lfloor \frac{\ell+r}{2} \right\rfloor$, the midpoint of the subarray,
- then apply the algorithm recursively to $A[\ell..m]$ and $A[m+1..r]$, and
- finally merge the subarrays $A[\ell..m]$ and $A[m+1..r]$.

- The depth of recursion in MERGE-SORT is $\log_2 n$.

Example: Merge-Sort

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

Example: Merge-Sort

- The depth of recursion in Merge-Sort is $\log_2 n$.
- On each level of recursion, merging all the intermediate arrays takes $O(n)$ steps in total.
- Thus, Merge-Sort always terminates, and in fact it terminates in $O(n \log_2 n)$ steps.

Example: Merge-Sort

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

- The depth of recursion in MERGE-SORT is $\log_2 n$.
- On each level of recursion, merging all the intermediate arrays takes $O(n)$ steps in total.
- Thus, MERGE-SORT always terminates, and in fact it terminates in $O(n \log_2 n)$ steps.
- Merging two sorted arrays always produces a sorted array, thus, the output of MERGE-SORT will be a sorted array.
- 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.

- 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.

- 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.

- 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.

- 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, BE VERY CAREFUL what you call trivial!!**

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Asymptotic Notation

6. Puzzles

- Suppose there are n hospitals in the state, and n new doctors have graduated from university. Each hospital wants to employ exactly one new doctor. (This is to keep the problem simpler; one can design an algorithm which works for more general cases where there are more doctors than hospitals and each hospital employs more than one doctor.)

- Suppose there are n hospitals in the state, and n new doctors have graduated from university. Each hospital wants to employ exactly one new doctor. (This is to keep the problem simpler; one can design an algorithm which works for more general cases where there are more doctors than hospitals and each hospital employs more than one doctor.)

- Every hospital submits a list of preferences, which ranks all the doctors, **and** every doctor submits a list of preferences, which ranks all the hospitals.

- Suppose there are n hospitals in the state, and n new doctors have graduated from university. Each hospital wants to employ exactly one new doctor. (This is to keep the problem simpler; one can design an algorithm which works for more general cases where there are more doctors than hospitals and each hospital employs more than one doctor.)
- Every hospital submits a list of preferences, which ranks all the doctors, **and** every doctor submits a list of preferences, which ranks all the hospitals.
- We'd like to assign all n doctors to different hospitals in order to "make everyone happy" in some sense.

Definition

A *matching* is an assignment of doctors to hospitals so that no doctor has more than one job and no hospital has more than one new employee.

Definition

A *matching* is an assignment of doctors to hospitals so that no doctor has more than one job and no hospital has more than one new employee.

Definition

A *perfect matching* is a matching involving *all* doctors and *all* hospitals.

Definition

A *matching* is an assignment of doctors to hospitals so that no doctor has more than one job and no hospital has more than one new employee.

Definition

A *perfect matching* is a matching involving *all* doctors and *all* hospitals.

Our task is to design a perfect matching that somehow satisfies the hospitals and doctors, with regards to their preferences.

Question

Can we assign every doctor and every hospital their first preference?

Question

Can we assign every doctor and every hospital their first preference?

Answer

Not necessarily!

Question

Can we assign every doctor and every hospital their first preference?

Answer

Not necessarily!

- We'll have to lower our expectations. Let's aim for an allocation that doesn't make anyone too unhappy.

Question

Can we assign every doctor and every hospital their first preference?

Answer

Not necessarily!

- We'll have to lower our expectations. Let's aim for an allocation that doesn't make anyone too unhappy.
- If a hospital and doctor were both very unhappy with their allocations, they might leave our system and arrange hiring directly.

Definition

A *stable matching* is a perfect matching in which there are no two pairs (h, d) and (h', d') such that:

- hospital h prefers doctor d' to doctor d, **and**
- doctor d' prefers hospital h to hospital h'.

Definition

A *stable matching* is a perfect matching in which there are no two pairs (h, d) and (h', d') such that:

- hospital h prefers doctor d' to doctor d, **and**
- doctor d' prefers hospital h to hospital h'.

Stable matchings are self-enforcing; no hospital h and no doctor d' will leave the system to do better outside of it.

Definition

A *stable matching* is a perfect matching in which there are no two pairs (h, d) and (h', d') such that:

- hospital h prefers doctor d' to doctor d, **and**
- doctor d' prefers hospital h to hospital h'.

Stable matchings are self-enforcing; no hospital h and no doctor d' will leave the system to do better outside of it.

We will design an algorithm which produces a stable matching.

$h_1 : d_1, d_2$

$h_2 : d_1, d_2$

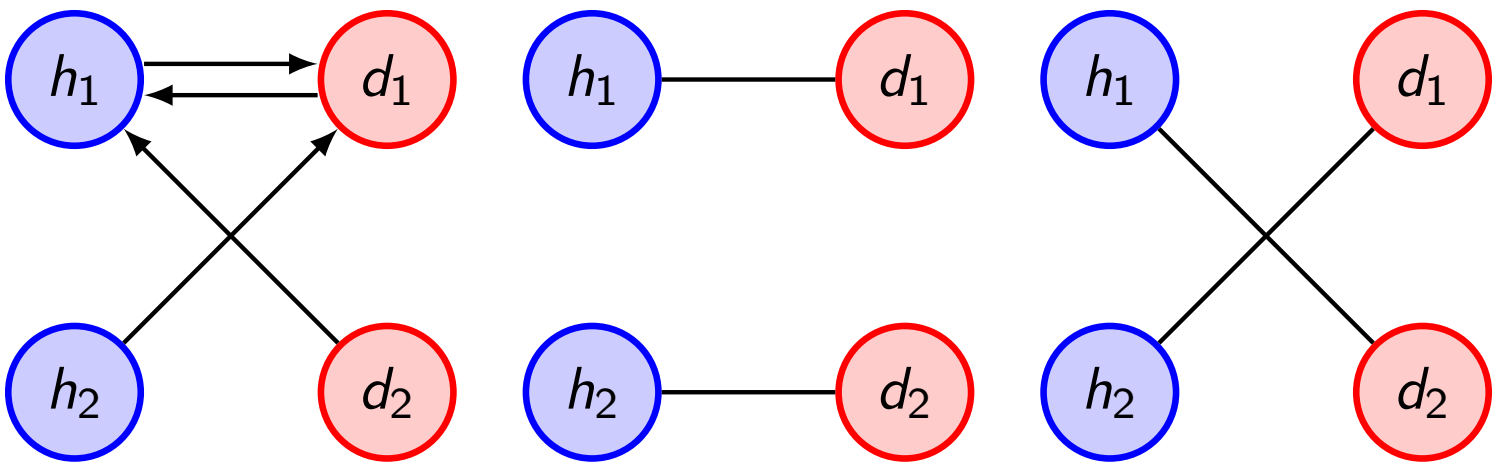
$d_1 : h_1, h_2$

$d_2 : h_1, h_2$

$h_1 : d_1, d_2$ $\quad$ $d_1 : h_1, h_2$

$h_2 : d_1, d_2$ $\quad$ $d_2 : h_1, h_2$

Preferences Stable Not stable



Stable Matching Problem: Example 2

$h_1 : d_1, d_2$

$h_2 : d_2, d_1$

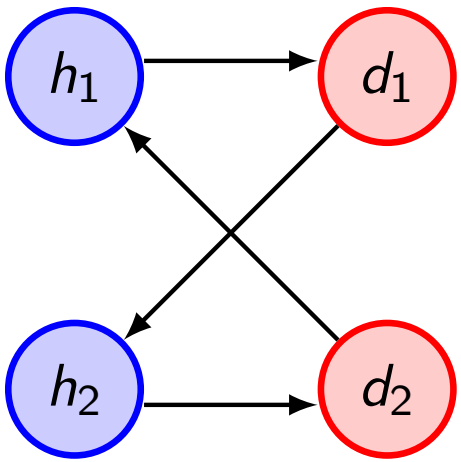
$d_1 : h_2, h_1$

$d_2 : h_1, h_2$

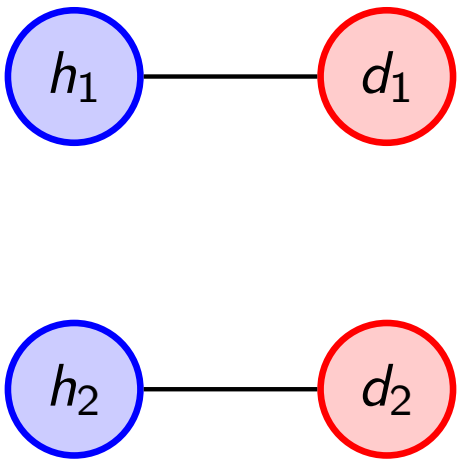
$h_1 : d_1, d_2$ $\quad$ $d_1 : h_2, h_1$

$h_2 : d_2, d_1$ $\quad$ $d_2 : h_1, h_2$

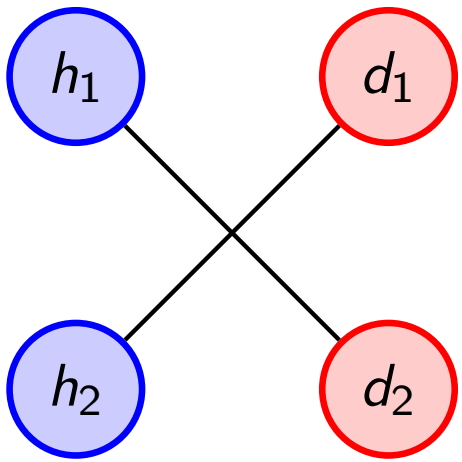
Preferences



Stable



Stable



Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

Answer

$n!$

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

Answer

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

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

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 hospitals, and n lists of preferences provided by all doctors, a stable matching always exists?

Question

Given n hospitals and n doctors, how many ways are there to match them, without regard for preferences?

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 hospitals, and n lists of preferences provided by all doctors, a stable matching always exists?

Answer

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

Question

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

Question

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

Answer

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

Question

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

Answer

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

- Produces pairs in stages, with possible revisions

Question

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

Answer

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

- Produces pairs in stages, with possible revisions
- A hospital which is not currently employing a doctor will be called *vacant*.

Question

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

Answer

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

- Produces pairs in stages, with possible revisions
- A hospital which is not currently employing a doctor will be called *vacant*.
- Hospitals will be offering jobs to doctors. Doctors will decide whether they accept a job offer or not.

Question

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

Answer

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

- Produces pairs in stages, with possible revisions
- A hospital which is not currently employing a doctor will be called *vacant*.
- Hospitals will be offering jobs to doctors. Doctors will decide whether they accept a job offer or not.
- Start with all hospitals vacant.

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.
    - If d is not yet employed, she accepts the job (at least tentatively).

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.
  - If d is not yet employed, she accepts the job (at least tentatively).
  - Otherwise, if d is already employed at a different hospital h':

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.
  - If d is not yet employed, she accepts the job (at least tentatively).
  - Otherwise, if d is already employed at a different hospital h':
    - if d prefers the new hospital h to her current hospital h', she quits h' and joins h (making h' vacant)

- While there is a vacant hospital which has not offered jobs to all doctors, pick any such vacant hospital and call it h.
- Hospital h offers a job to the highest ranking doctor d on its list, ignoring any doctors to whom it has already offered a job.
  - If d is not yet employed, she accepts the job (at least tentatively).
  - Otherwise, if d is already employed at a different hospital h':
    - if d prefers the new hospital h to her current hospital h', she quits h' and joins h (making h' vacant)
    - otherwise, since d prefers her current hospital h', she rejects the offer from h and stays at h'.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.
- Thus, every hospital can make at most n offers.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.
- Thus, every hospital can make at most n offers.
- There are n hospitals, so in total they can make $\leq n^2$ offers.

Claim 1

The algorithm terminates after $\leq n^2$ rounds.

Proof

- In every round of the algorithm, one hospital offers a job to one doctor.
- Every hospital can make an offer to a doctor at most once.
- Thus, every hospital can make at most n offers.
- There are n hospitals, so in total they can make $\leq n^2$ offers.
- Thus there can be no more than n^2 many rounds.

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.
- This means that h has already offered a job to every doctor.

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.
- This means that h has already offered a job to every doctor.
- A doctor is unemployed only if no hospital has offered them a job, so all doctors must have a job.

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.
- This means that h has already offered a job to every doctor.
- A doctor is unemployed only if no hospital has offered them a job, so all doctors must have a job.
- But this would mean that n doctors are paired with all of n hospitals, so h cannot be vacant.

Claim 2

The algorithm produces a perfect matching, i.e., every hospital is eventually paired with a doctor (and thus also every doctor is paired to a hospital).

Proof

- Assume that the algorithm has terminated, but hospital h is still vacant.
- This means that h has already offered a job to every doctor.
- A doctor is unemployed only if no hospital has offered them a job, so all doctors must have a job.
- But this would mean that n doctors are paired with all of n hospitals, so h cannot be vacant.
- This is a contradiction, completing the proof.

Claim 3

The matching produced by the algorithm is stable.

Claim 3

The matching produced by the algorithm is stable.

Proof

We will prove that the matching is stable using *proof by contradiction*.

Claim 3

The matching produced by the algorithm is stable.

Proof

We will prove that the matching is stable using *proof by contradiction*.

Assume that the matching is not stable. Thus, there are two pairs (h, d) and (h', d') such that:

Claim 3

The matching produced by the algorithm is stable.

Proof

We will prove that the matching is stable using *proof by contradiction*.

Assume that the matching is not stable. Thus, there are two pairs (h, d) and (h', d') such that:

- h prefers d' over d and
- d' prefers h over h'.

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
  - rejected h because they were already at a hospital they prefer to h, or

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
  - rejected h because they were already at a hospital they prefer to h, or
  - accepted h only to later rescind this and accept an offer from a hospital they prefer to h.

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
  - rejected h because they were already at a hospital they prefer to h, or
  - accepted h only to later rescind this and accept an offer from a hospital they prefer to h.
- In both cases d' would now be at a hospital which they prefer over h.

Proof (continued)

- Since h prefers d' over d, it must have made an offer to d' before offering the job to d.
- Since h is paired with d, doctor d' must have either:
  - rejected h because they were already at a hospital they prefer to h, or
  - accepted h only to later rescind this and accept an offer from a hospital they prefer to h.
- In both cases d' would now be at a hospital which they prefer over h.
- This is a contradiction, completing the proof.

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Asymptotic Notation

6. Puzzles

- **"Big Oh" notation:** $f(n) = O(g(n))$ is an abbreviation for:

  "There exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$".

- **"Big Oh" notation:** $f(n) = O(g(n))$ is an abbreviation for:

  "There exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$".

- In this case we say that $g(n)$ *is an asymptotic upper bound for* $f(n)$.

- **"Big Oh" notation:** $f(n) = O(g(n))$ is an abbreviation for:

  "There exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$".

- In this case we say that $g(n)$ *is an asymptotic upper bound for* $f(n)$.

- $f(n) = O(g(n))$ means that $f(n)$ does not grow substantially faster than $g(n)$ because a multiple of $g(n)$ eventually dominates $f(n)$.

- **"Big Oh" notation:** $f(n) = O(g(n))$ is an abbreviation for:

  "There exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0$".

- In this case we say that $g(n)$ *is an asymptotic upper bound for* $f(n)$.

- $f(n) = O(g(n))$ means that $f(n)$ does not grow substantially faster than $g(n)$ because a multiple of $g(n)$ eventually dominates $f(n)$.

- Clearly, multiplying constants c of interest will be larger than 1, thus "enlarging" $g(n)$.

- **"Omega" notation:** $f(n) = \Omega(g(n))$ is an abbreviation for:

  "There exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$."

- **"Omega" notation:** $f(n) = \Omega(g(n))$ is an abbreviation for:

  "There exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$."

- We say that $g(n)$ *is an asymptotic lower bound for* $f(n)$.

- **"Omega" notation:** $f(n) = \Omega(g(n))$ is an abbreviation for:

  "There exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$."

- We say that $g(n)$ *is an asymptotic lower bound for* $f(n)$.

- $f(n) = \Omega(g(n))$ says that $f(n)$ grows at least as fast as $g(n)$, because $f(n)$ eventually dominates a multiple of $g(n)$.

- **"Omega" notation:** $f(n) = \Omega(g(n))$ is an abbreviation for:

  "There exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$."

- We say that $g(n)$ *is an asymptotic lower bound for* $f(n)$.

- $f(n) = \Omega(g(n))$ says that $f(n)$ grows at least as fast as $g(n)$, because $f(n)$ eventually dominates a multiple of $g(n)$.

- Since $c\, g(n) \leq f(n)$ if and only if $g(n) \leq 1/c\, f(n)$, we have $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

- **"Omega" notation:** $f(n) = \Omega(g(n))$ is an abbreviation for:

  "There exists positive constants c and n_0 such that $0 \leq c\, g(n) \leq f(n)$ for all $n \geq n_0$."

- We say that $g(n)$ *is an asymptotic lower bound for* $f(n)$.

- $f(n) = \Omega(g(n))$ says that $f(n)$ grows at least as fast as $g(n)$, because $f(n)$ eventually dominates a multiple of $g(n)$.

- Since $c\, g(n) \leq f(n)$ if and only if $g(n) \leq 1/c\, f(n)$, we have $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$.

- **"Theta" notation:** $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$; thus, $f(n)$ and $g(n)$ have the same asymptotic growth rate.

- Note: $f(n) = O(g(n))$ is NOT a linear ordering; there are functions $f(n)$ and $g(n)$ such that it is neither $f(n) = O(g(n))$ nor $g(n) = O(f(n))$.

- Note: $f(n) = O(g(n))$ is NOT a linear ordering; there are functions $f(n)$ and $g(n)$ such that it is neither $f(n) = O(g(n))$ nor $g(n) = O(f(n))$.

- Example: $f(n) = n^2 + n$ and $g(n) = n^2$. Note that $g(n) \leq f(n)$; thus $g(n) \leq c\, f(n)$ with $c = 1$, and so $g(n) = O(f(n))$.

- Note: $f(n) = O(g(n))$ is NOT a linear ordering; there are functions $f(n)$ and $g(n)$ such that it is neither $f(n) = O(g(n))$ nor $g(n) = O(f(n))$.

- Example: $f(n) = n^2 + n$ and $g(n) = n^2$. Note that $g(n) \leq f(n)$; thus $g(n) \leq c\, f(n)$ with $c = 1$, and so $g(n) = O(f(n))$.

  However $f(n) \leq 2\, g(n)$; thus $f(n) \leq c\, g(n)$ with $c = 2$. So also $f(n) = O(g(n))$. Thus $f(n) = \Theta(g(n))$.

- Why such a definition?

- Why such a definition?
- We would like an efficiency measure independent of the kind of processor that will execute the algorithm.

- Why such a definition?
- We would like an efficiency measure independent of the kind of processor that will execute the algorithm.
- But different processors have different instruction sets “hard wired”.

- Why such a definition?
- We would like an efficiency measure independent of the kind of processor that will execute the algorithm.
- But different processors have different instruction sets “hard wired”.
- Thus, a single instruction on one processor might have to be implemented as a sequence of a few hard wired operations implemented on another processor.

- Why such a definition?
- We would like an efficiency measure independent of the kind of processor that will execute the algorithm.
- But different processors have different instruction sets “hard wired”.
- Thus, a single instruction on one processor might have to be implemented as a sequence of a few hard wired operations implemented on another processor.
- Also, for small inputs the efficiency might depend on operating system overhead.

- Why such a definition?
- We would like an efficiency measure independent of the kind of processor that will execute the algorithm.
- But different processors have different instruction sets “hard wired”.
- Thus, a single instruction on one processor might have to be implemented as a sequence of a few hard wired operations implemented on another processor.
- Also, for small inputs the efficiency might depend on operating system overhead.
- Asymptotic estimates of efficiency make these facts irrelevant.

Table of Contents

1. Admin

2. Solving problems using algorithms

3. Proofs

4. An example of the role of proofs

5. Asymptotic Notation

6. Puzzles

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

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!!