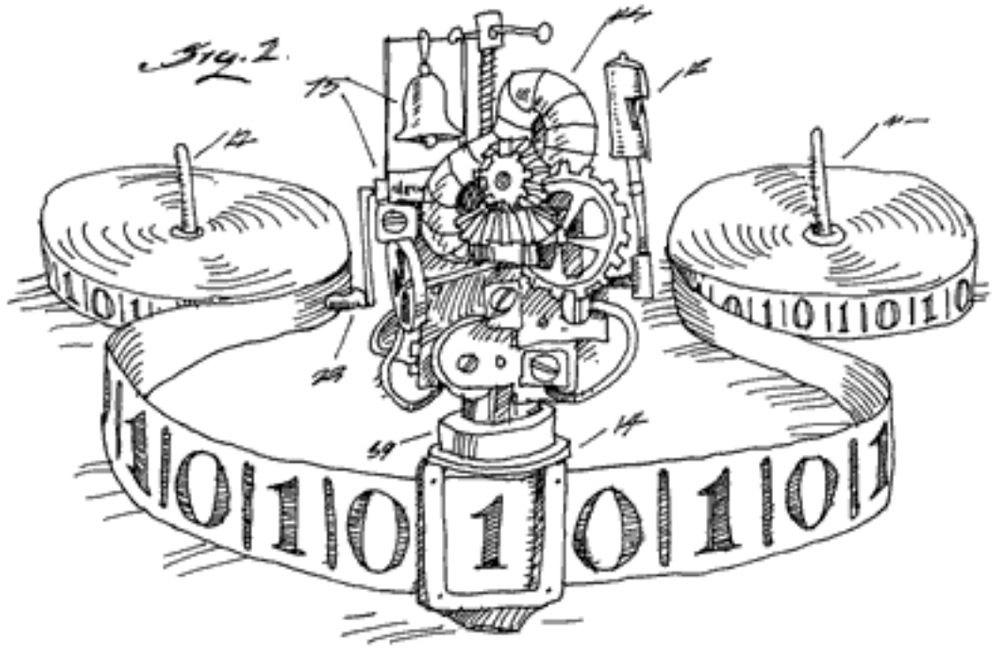
CSC4010: Introduction to Theoretical Computer Science



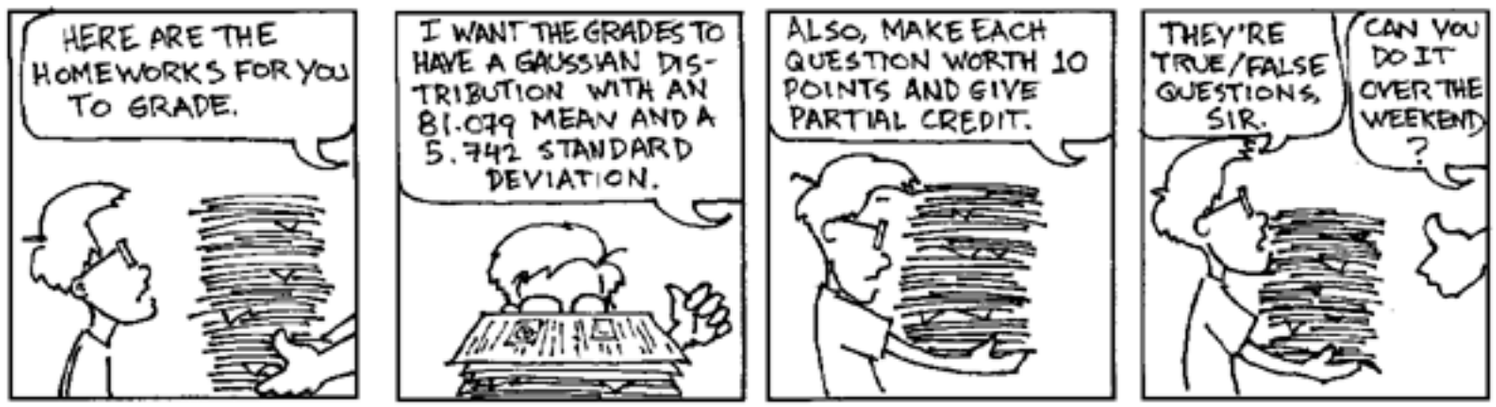
Lecture 1

Course Staff

Instructor: Aditi Dudeja (https://aditidudeja.github.io)
Email: aditidudeja@cuhk.edu.cn

Teaching Assistant: ZHU Tianran
Email: tianranzhu@link.cuhk.edu.cn

USTF: SUN Youran
Email: youransun@link.cuhk.edu.cn

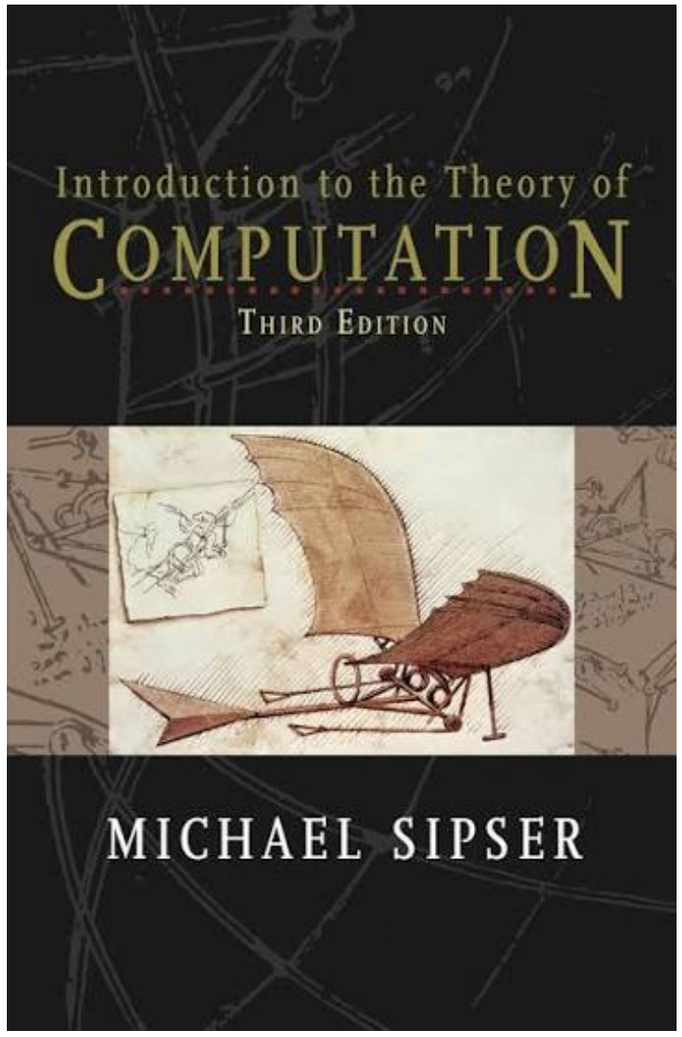


Administration

1. **Lectures:** Mon, Wed: 8:30-9:50 am
2. **Tutorial:** Tuesday: 19:00-19:50.

Attendance is not compulsory but strongly recommended.

Note: Tomorrow's Tutorial is a lecture.



Administration

The course website (https://aditidudeja.github.io/courses/csc4010-f26/) contains:

(i) **Syllabus:** This is a very important document, please read it.

(ii) Schedule

(iii) Files: Homework, (annotated) Lecture slides, pointers to references.

(iv) My Office Hours: Mon, Tue: 4-5 pm (DY504b)
Wed, Thu: 4-5 pm (on Zoom)
After class/by appointment

(v) Grades: via Blackboard

(vi) Announcements: Check email, Blackboard, webpage. Make sure you received my email and Blackboard notice. Check these regularly.

Course Components

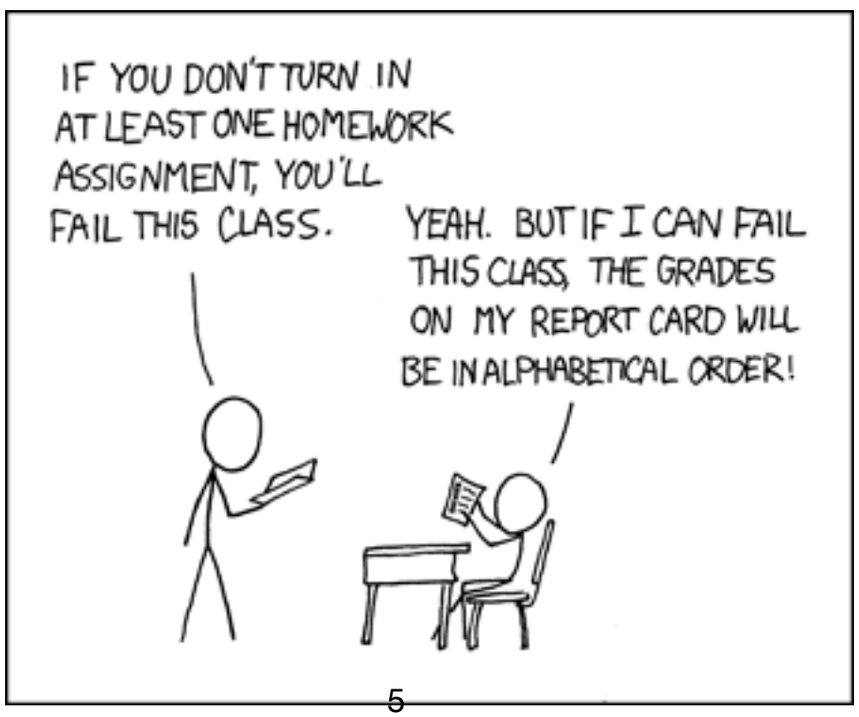
1. Homeworks:

a) These are not graded.

b) You will be asked to prove certain mathematical statements.

c) Discussed in tutorials and office hours. Try them beforehand. Try to pin down and explain what you tried and where you got stuck, we will give you hints on how to make progress.

d) Exams and in-class tests will be based on them, so it is good to solve them even if they are not graded.



Course Components

2. In-Class Tests (40 %):

a) Will be held during first 20 minutes of tutorials.

b) **Dates:** 29/9/26, 27/10/26, 24/11/26, 15/12/26

c) Multiple choice/short answer.

d) Your lowest-graded test will be dropped.

3. Midterm (30 %):

a) Based on lectures 1-14.

b) **Date:** 2/11/26 (in class).

4. Final Exam (30 %):

a) (Tentatively) based on entire course, with emphases on lecture 15 onwards.

b) **Date:** According to University Schedule. We will circle back to this.

About This Course:

"What can computers do, and what do limits on computation let us build ?"

What are the limits of computation?

a) In many CS courses, we learn how to design an algorithm. This course asks a different one: **when should we stop looking for one?**

b) We will learn:

   i. Some tasks are **solvable.**

   ii. Some tasks are **solvable but need unrealistic resources.**

   iii. And surprisingly, knowing these limits is not pessimistic. It is useful. It is a part of building **secure systems.**

About This Course

Four Modules:

- **Computability Theory:** Can it all be computed?
- **Complexity Theory:** Can it be computed with realistic resources?
- **Fine-Grained Complexity:** What is the best possible bound on the resources?
- **Cryptography:** How can hardness help us protect information?

Can it all be computed?

Suppose someone offers you a perfect program checker:

`Halt(P, x)` takes any program `P` and input `x` and returns:

- `YES` if `P(x)` eventually stops,
- `NO` if `P(x)` loops forever.

Can such a checker exist?

Can it all be computed?

Suppose someone offers you a perfect program checker:

`Halt(P, x)` takes any program `P` and input `x` and returns:

- `YES` if `P(x)` eventually stops,
- `NO` if `P(x)` loops forever.

Can such a checker exist?

Can such a checker exist in the future, when we have even more powerful hardware?

To answer this: we will mathematically define the notion of an “algorithm” or a “program” via Turing Machines. Eventually show such a checker cannot exist.

Can it be computed realistically?

Is it possible to choose true/false values for A, B, C so that:

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C)$$

Evaluates to 1.

Can it be computed realistically?

Is it possible to choose true/false values for A, B, C so that:

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C)$$

Evaluates to 1.

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Can it be computed realistically?

Is it possible to choose true/false values for A, B, C so that:

$$(A \vee B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C)$$

Evaluates to 1.

A	B	C
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

- The easiest algorithm has to check 2^3 possibilities.
- This structure also comes up in problems such as resource allocation, circuit design, scheduling.
- What if the number of variables is 30?
- In module 2, we define the notion of “efficient computation”.

How fast can it possibly be?

Imagine n student profiles, each represented by a binary vector of features. We want to know whether there are **two profiles with no features in common.**

01011

10100

Brute force: compare every pair or about $\binom{n}{2}$ checks.

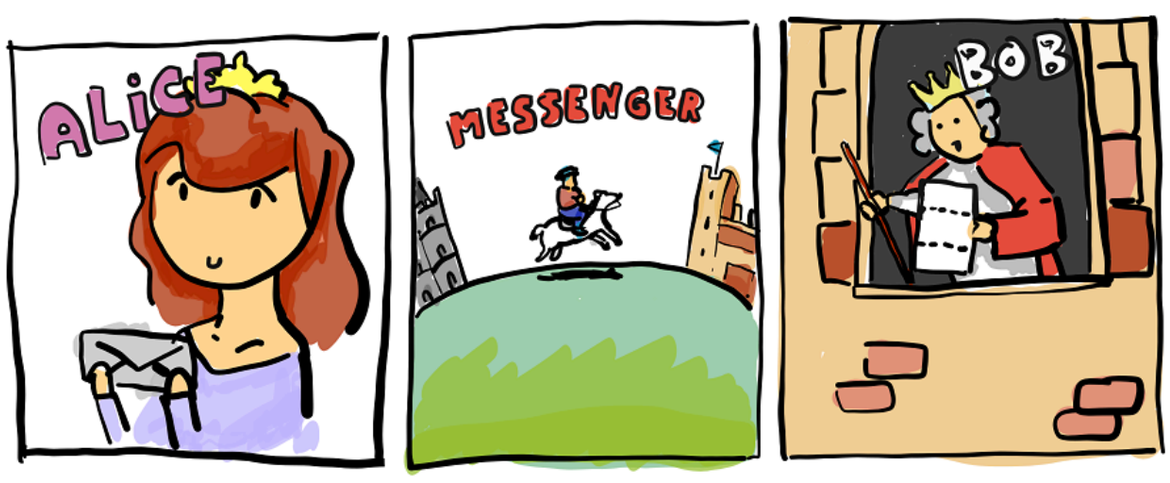
Question: can we get $n^{1.99}$ time?

Using Impossibility as Feature: Cryptography

- Suppose Alice and Bob are communicating over a network that the messenger can see completely.

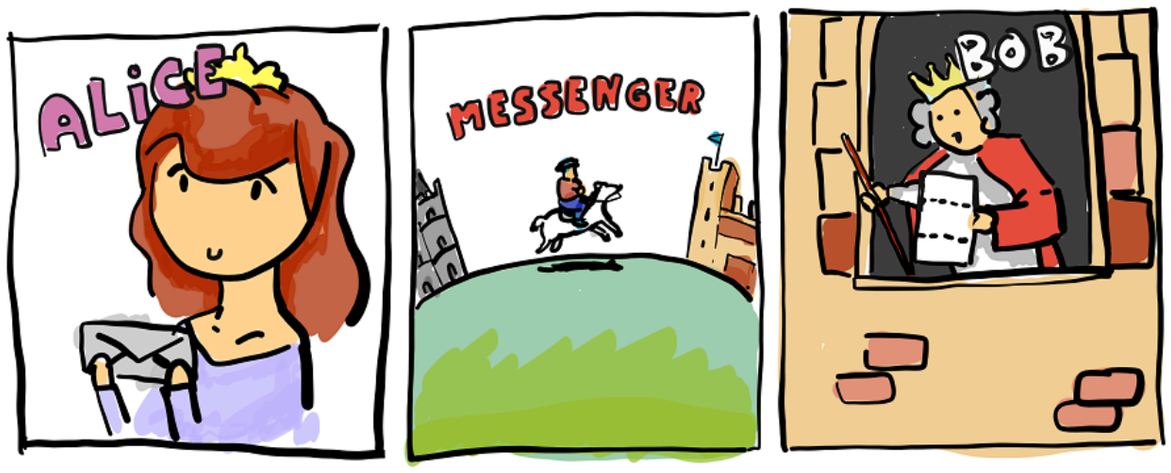
- They want the messenger to learn nothing useful.

- How can they establish a secret when every message they send is seen?



Using Impossibility as Feature: Cryptography

Solution: Alice encrypts her message. To decrypt it, an attacker must solve an “infeasible” problem. But Bob and Alice being legitimate parties have the right “certificate”, which will make this infeasible task easy for them.



Learning Outcomes

By the end of CSC4010, I want you to look at a problem and ask not only “how would I program this” but also,

- Is an algorithm possible?
- Is an efficient one possible?
- If not, what evidence supports this conclusion?
- Can the difficulty be useful?

Is this a CS Class?

- **Question**: Why does this feel like a Math class instead of a CS class?
- **Answer**: Its both! The only way to answer questions we raise is to **define mathematical models** and apply a **rigorous proof based** methodology to the questions.

- **Question**: Why do we prove things?
- **Answer**: a) To understand. b) to be confident in correctness of our "intuition".

Why study CS Foundations?

1. Theoretical ideas underlie a lot of software (e.g. hardware design, Internet routing, cryptography, many more).

   ("Everyone knows Moore's Law — a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years...in many areas, performance gains due to **improvements in algorithms** have vastly exceeded even the dramatic performance gains due to increased processor speed." -Report to the US President and Congress: Designing a Digital Future (2010)).

2. It's very easy to write code that is either incorrect or will run for millions of years.

   (“For every complex problem there is an answer that is clear, simple, and wrong." -H. L. Mencken)

Why study CS Foundations?

3. In your job you might be asked you to solve a problem that's provably impossible. Save yourself the trouble!

4. Foundational reasoning applies to any computer: past, present, or future!

(Mathematical models of computing predated computers. Present-day example: we know a lot about quantum computing, but no large scale computers have been built yet!)

Why study CS Foundations?

5. Computation is everywhere, not just in computers. For example, your brain, economics, or animal behavior.
6. A chance to learn some interesting math. You will be asked to think in a different way, it will make you a better problem solver and computer scientist.

Art from Simons Institute Program on Online and Matching-Based Market Design

Struggling is Good

This course asks you to comprehend deep ideas and solve challenging problems. You are supposed to struggle!

It is important to have wrong ideas. The discoverers of every idea in this course had many wrong ideas before having the right idea.

We are here to help! You will likely do better in this course if you ask questions during lecture, in tutorial, and in office hours.

An example of a good question to ask in class: "Can you repeat what you just said?"

AI

What is the point of learning, designing, and analyzing algorithms anymore?

- You need subject matter expertise to interact productively with AI: ask the right questions and accurately interpret its output.

- The real world is messy and you need to be able to distill real-world problems into well-defined computational problems.

- You should give yourself a chance to struggle with new ideas, develop your own understanding of them and grow!

- It is more fun to brainstorm with peers, TAs, USTFs or, the instructor!

A Bit More About This Course…

Mathematical Proofs

- This class will emphasize mathematical proofs.
- A good proof:
  - Is correct.
  - We should also strive to make it easy to understand/clear.

Mathematical Proofs

Proofs are a rigorous method to formalize our intuition regarding a mathematical statement. It can be helpful to provide 3 levels of details.

- **Level 1:** A short sentence conveying the intuition. It can be a phrase such as “proof by induction” or, “proof by contradiction”.

- **Level 2:** A short one paragraph description of the main ideas.

- **Level 3:** The full proof (and nothing but the proof).

Example

Claim: Suppose $A \subseteq \{1,2,\cdots,2n\}$ and $|A| = n+1$. Then there are two distinct numbers $x, y \in A$ that divide each other.

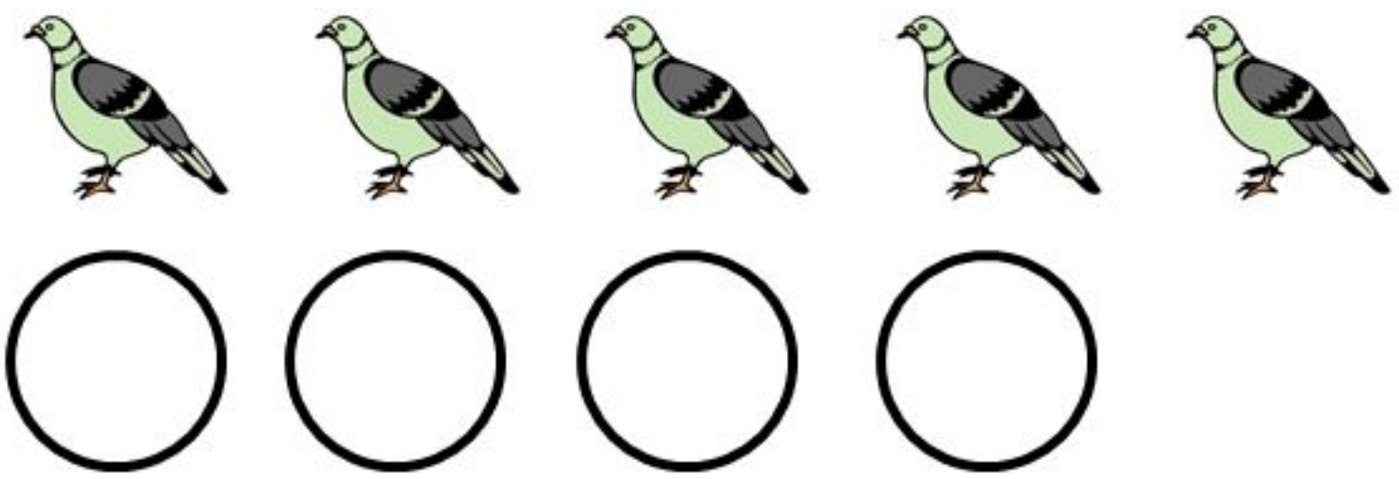
Sanity Check: Suppose $A \subseteq \{1,2,3,4\}$ and $|A| = 3$.

- If $1 \in A$, then 1 divides every number.

- If $1 \notin A$, then $A = \{2,3,4\}$. And 2 divides 4.

Level 1

Intuition: If you drop 6 pigeons in 5 holes then one hole will have at least two pigeons.



Intuition: Every positive integer can be written as $a = 2^k m$, where m is odd (called **odd part**), and k is an integer.

Level 2

Given $A \subseteq \{1,2,\cdots,2n\}$ and $|A| = n + 1$,

Applying the pigeonhole principle, we will show there are $x, y \in A$ such that $x = 2^i m$ and $y = 2^j m$ for odd m, and integers $i < j$.
Then x divides y.

Level 3: The Proof

Given $A \subseteq \{1,2,\cdots,2n\}$ and $|A| = n+1$,

Write each $x \in A$ as $2^k m$, where m is an odd number in $\{1,2,\cdots,2n\}$.

There are n distinct odd numbers in $\{1,2,\cdots,2n\}$.

Since $|A| = n+1$, there are two distinct numbers in A with the same odd part. By Pigeonhole Principle.

Let x, y have the same odd part m, where $x < y$. Then $x = 2^i m$ and $y = 2^j m$, where $i < j$. Thus x divides y.

What is the right level of detail in the proof?

During lectures, my proofs will contain Levels 1 and 2. But only parts of Level 3. We will guide you through Level 3 in tutorials, office hours.

Think about how to fill in the details!

As we progress in this course, it will turn out that the big ideas and concepts are more important than the gritty details.

Come to the office hours or email us, if you are worried about this.

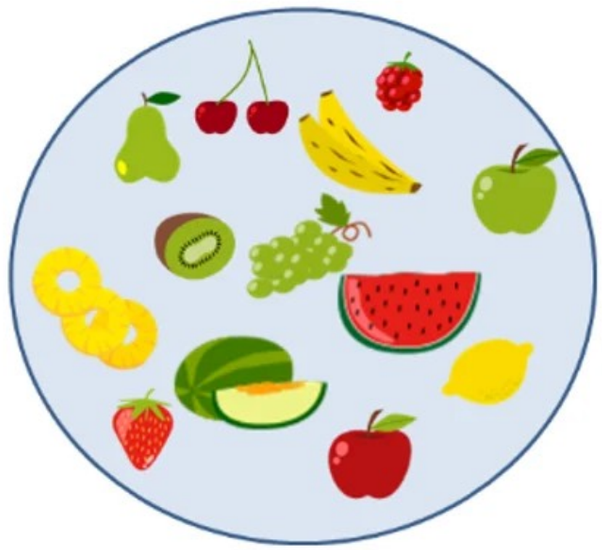
Let’s Get Started

Learning Outcomes

- Formally define what it means for two sets have the same cardinality.
- Prove two sets have the same cardinality by showing a bijection.

Recall

- Sets are a collection of distinct objects.



- They can be finite. For example: “All positive even numbers less than 100.”
- They can be infinite. For example: $\mathbb{N} = \{0,1,2,\cdots,\}$ or $\mathbb{Z}^+ = \{1,2,\cdots\}$.

Cardinality

Definition: The cardinality of a finite set S, denoted $|S|$, is the number of distinct elements contained in it.

Example: There are 5 positive even numbers less than 10.

There are 390,900 plant species in the world (upper bounds the number of fruits).

- What about infinite sets?
- ∞ is not a number.
- Do all infinite sets have the same size?
- How do we generalize the notion of cardinality to incorporate infinite sets?

A New Definition of Cardinality

Use Case of Cardinality: Comparing sizes of finite sets. Is $|A| \stackrel{?}{=} |B|$?

Intuition: $|A| = |B|$ if their elements can be paired up.

$A = \{2,4,6,8,10\}$ and $B = \{1,3,5,7,9\}$.

Formal Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if there is a bijection from A to B.

A New Definition of Cardinality

Use Case of Cardinality: Comparing sizes of finite sets. Is $|A| \stackrel{?}{=} |B|$?

Intuition: $|A| = |B|$ if their elements can be paired up.

$A = \{2,4,6,8,10\}$ and $B = \{1,3,5,7,9\}$.

Formal Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if there is a bijection from A to B.

$A = \{2,4,6,8,10\}$ and $B = \{1,3,5,7,9\}$.

Consider: $f : A \to B$ defined as $f(x) = x - 1$

x	f(x)
2	1
4	3
6	5
8	7
10	9

Infinite Sets Example: $\mathbb{Z}^+$ vs $\mathbb{N}$

For Finite Sets: If $A \subsetneq B$, then $|A| < |B|$.

We will show: For $\mathbb{Z}^+ = \{1,2,\cdots\}$ and $\mathbb{N} = \{0,1,2,\cdots\}$, $|\mathbb{Z}^+| = |\mathbb{N}|$.

Infinite Sets Example: $\mathbb{Z}^+$ vs $\mathbb{N}$

We will show: For $\mathbb{Z}^+ = \{1,2,\cdots\}$ and $\mathbb{N} = \{0,1,2,\cdots\}$, $|\mathbb{Z}^+| = |\mathbb{N}|$.

Formal Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if there is a bijection from A to B.

Equivalent Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if

a) $|A| \leq |B|$, and

b) there is a one-one function from B to A.

Infinite Sets Example: $\mathbb{Z}^+$ vs $\mathbb{N}$

We will show: For $\mathbb{Z}^+ = \{1,2,\cdots\}$ and $\mathbb{N} = \{0,1,2,\cdots\}$, $|\mathbb{Z}^+| = |\mathbb{N}|$.

Formal Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if there is a bijection from A to B.

Equivalent Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if

a) $|A| \leq |B|$, and

b) there is a one-one function from B to A.

Equivalent Definition: Two sets A, B have the same cardinality $|A| = |B|$ if and only if

a) There is a one-one function $f : A \to B$, and

b) there is a one-one function from $g : B \to A$.

Next Class (Tomorrow): More Examples