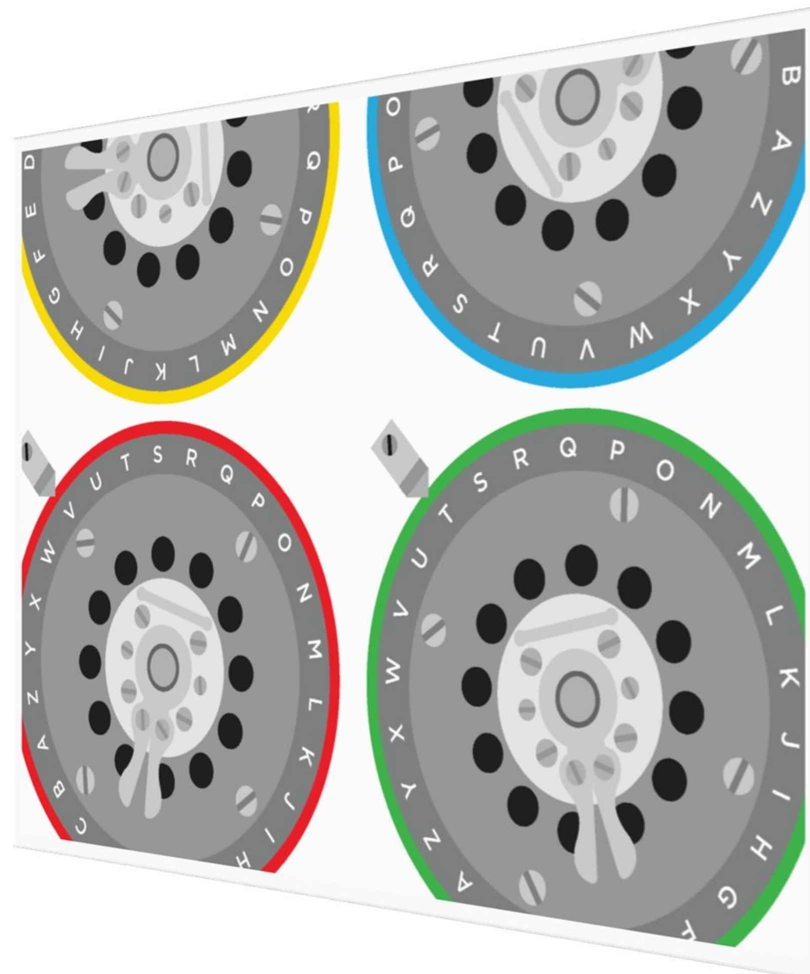
بسم الله الرحمن الرحیم

Computability Theory

Ali Shakiba
Vali-e-Asr University of Rafsanjan
ali.shakiba@vru.ac.ir

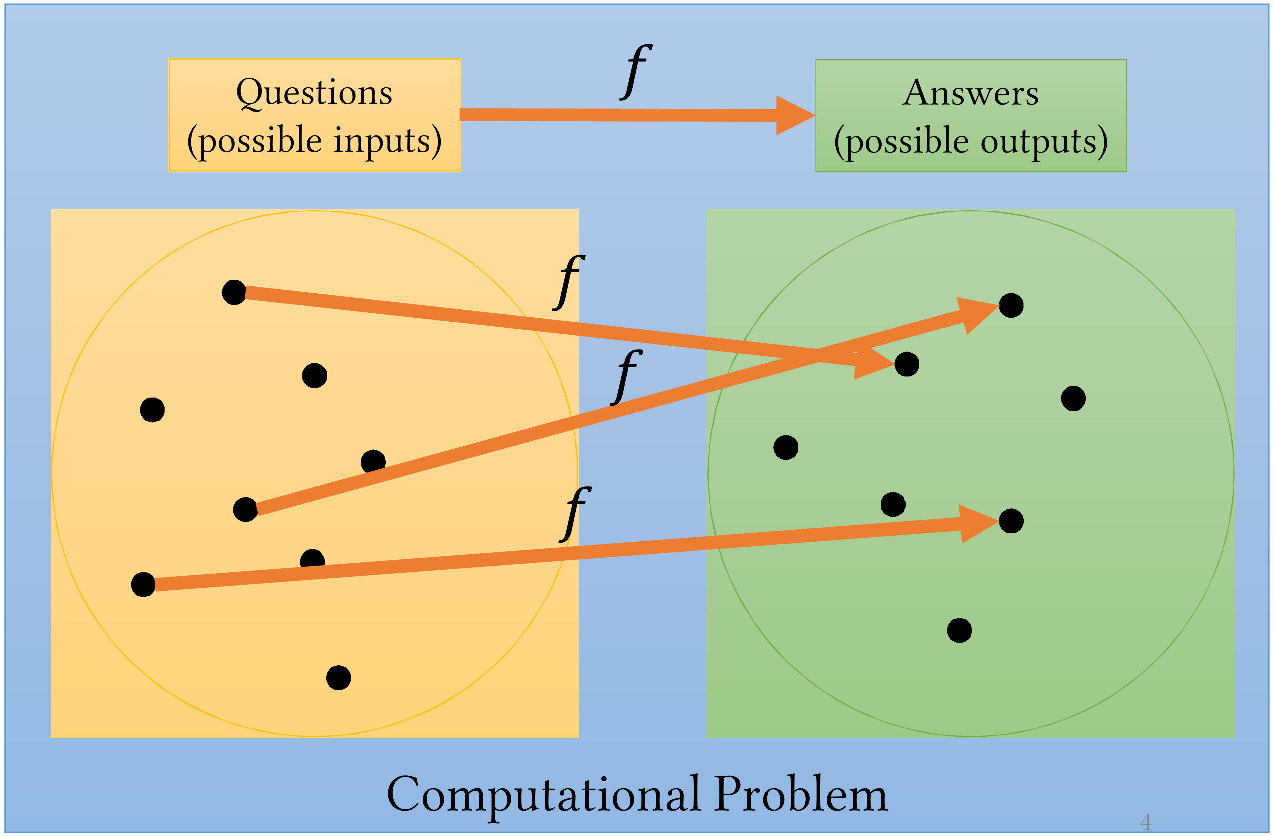
What is Computation?



The Functional Model

Assumes Computations:

1. Read an input, think for a while, write an output, and halt
2. Just the "relation between input and output" is important



The Imperative Model

- What about computations that do not "compute a function"?
  - deleting a file, anti-lock break system, …

sequences of imperatives which manipulate representations

More inclusive than functional model

advantageous

dis-advantageous

Hard to reason about programs at this level

Computability

- A function is called *computable* if there is a computer program which executes it.

- What are the limits of computational power?
  - We need to abstract the essentials.
  - The answer needs to be independent of hardware advances.
  - No limits on time and memory use in advance.

Essential components

- Memory
  - with read/write capability
- Arithmetic
- if ... then
- looping (for, while)

- Extra tools make it easier for humans to use

A program in PASCAL

```
Program Sample_Program;
Var
    Num1, Num2, Sum : Integer;
Begin
      Write('Input number 1:');
      Readln(Num1);
      Writeln('Input number 2:');
      Readln(Num2);
      Sum := Num1 + Num2; // addition
      Writeln(Sum);
      Readln;
End.
```

A program in QBASIC

```
CLS
INPUT "Enter a number: ", Number
IF Number < 100 THEN
  PRINT "Your number was less than 100"
ELSE
  PRINT "Your number was greater than or equal to 100"
END IF
```

A program in C++

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  int num;
  double sq_root;
  for(num=1; num < 10; num++) {
    sq_root = sqrt((double) num);
    cout << num << "  " << sq_root << '\n';
  }
  return 0;
}
```

Commonalities

- Finite sequence of symbols out of a finite alphabet.

- Could be ordered in some way and assign a number to each of them.

Enumeration (Listing)

- Sequences on alphabet $\{a, b\}$ may be enumerated as follows:

1. a	6. bb	11. baa
2. b	7. aaa	12. bab
3. aa	8. aab	13. bba
4. ab	9. aba	14. bbb
5. ba	10. abb	...

- Some of them are not valid programs, however that is fine. We just consider invalid programs as the ones which do nothing.

A counting argument

- There are as many programs in a given language as there are natural numbers (contably many)

- There are as many functions on the natural numbers as there are real numbers (uncountably infinite)

The latter is strictly larger!

- So, there are uncountably many non-computable functions.

The Halting Problem

- Fix an enumeration of programs $P_0, P_1, \ldots$. The following function is not computable by any program on the list.

$$f(x) = \begin{cases} 1, & \text{if } P_x(x) \text{ halts} \\ 0, & \text{if } P_x(x) \text{ goes into an infinite loop} \end{cases}$$

We are not just limited to functions, we can use sets ...

Sets, Sequences, Functions

- The function $f: \mathbb{N} \to \{0,1\}$ is associated with the sequence with entries $f(0), f(1), f(2), \ldots,$ in order.
- A binary sequence S is associated with the set A where $n \in A$ if the n^{th} entry of S is 1 and $n \notin A$ otherwise.

$f(n) \mapsto n \bmod 2$	0101010101 ...	The set of odd numbers

Comparing Non-computability

- If we choose some set A and allow our programs to include statements of the form

$$\text{"if } n \in A\text{, then ..."},$$

we are working with *oracle programs*.

If B can be computed by a program with oracle A, we say that B is Turing reducible to A and write $B \leq_T A$.

- If A is ...
  - non-computable, then we can compute more sets than we could do before.
  - computable, then nothing is added.

If I allow "halting problem H" as an oracle …

- Let's add H as an oracle to every program in our enumeration, i.e. $P_0^H, P_1^H, \ldots$.
- Consider the following function f:

$$f(x) = \begin{cases} 1, & \text{if } P_x^H(x) \text{ halts} \\ 0, & \text{if } P_x^H(x) \text{ goes into an infinite loop} \end{cases}$$

Halting problem *relativized* to H

it is non-computable by any P_e^H

No matter what oracle A is chosen, there are always sets, uncountably many, B which $B \not\leq_T A$.

Turing Degrees

- Let H' be the set associated with the Halting problem relativized to H, then we have $H \nleq_T H'$.
  - This can be iteratively continued as $H \nleq_T H' \nleq_T H'' \nleq_T \ldots$, since we have uncountably many left at each iteration.

The relation $A \equiv_T B$ defined as $(A \leq_T B) \wedge (B \leq_T A)$ partitions the subsets of $\mathbb{N}$ into equivalence classes called *Turing degrees*.

Each Turing degree contains countably many sets.

There are uncountably many Turing degrees.

Computable Enumerable Sets

- The collection of “Computable Enumerable Sets” is the next-larger set than “Computable Sets”

Set A is *computable enumerable* if its elements may be listed out computably, but not necessarily in order.

Some Basic Facts on Computable Enumerable Sets

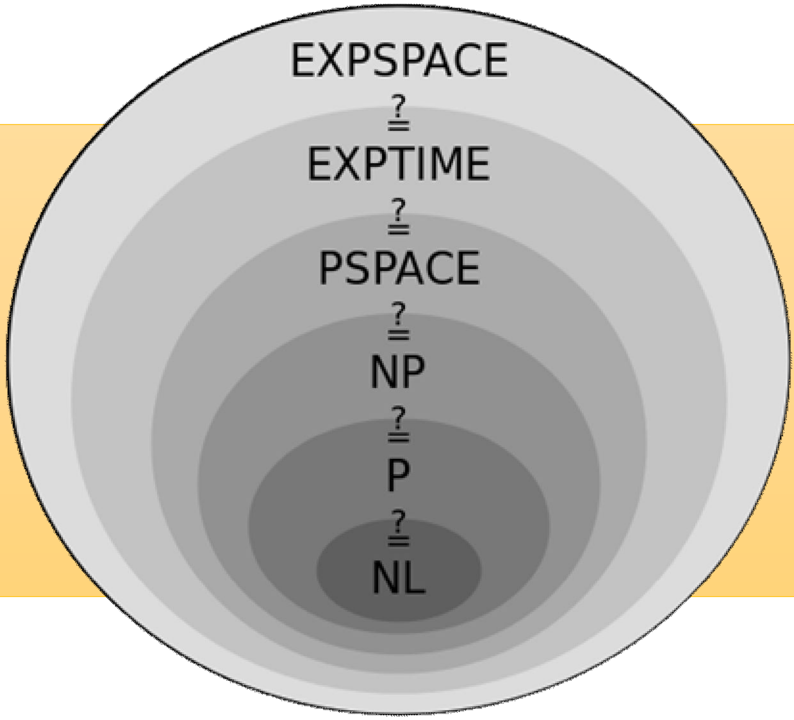
- A set is "computable" if and only if its elements may be enumerated in order.
- A set B is computable if and only if both B and B^c are computable enumerable.
- All the computable enumerable sets are Turing reducible to the Halting problem.

Some Basic Facts on Computable Enumerable Sets

- A set is "computable" if and only if its elements may be enumerated in order.
- A set B is computable if and only if both B and B^c are computable enumerable.
- All the computable enumerable sets are Turing reducible to the Halting problem.
- The halting problem is itself computably enumerable.
- The are non-computable enumerable sets B such that $B \leq_T H$.

Computational Complexity

general study of what can be achieved within limited time and/or other limitations on natural computational resources

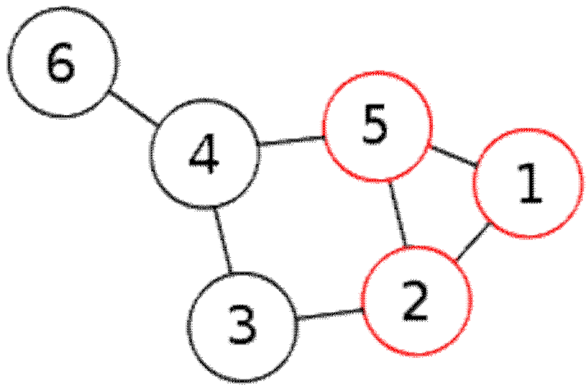


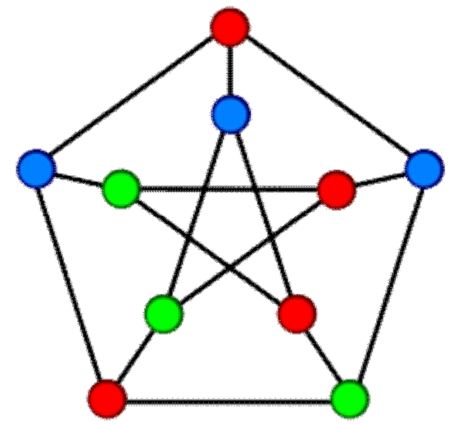
Two Concerns of Complexity

1. determination of the complexity of any well-defined task
2. obtaining an understanding of the relations between various computational phenomena

P, NP, and NP-completeness

$(x \lor x \lor y) \land$
$(\neg x \lor \neg y \lor \neg y) \land$
$(\neg x \lor y \lor y)$





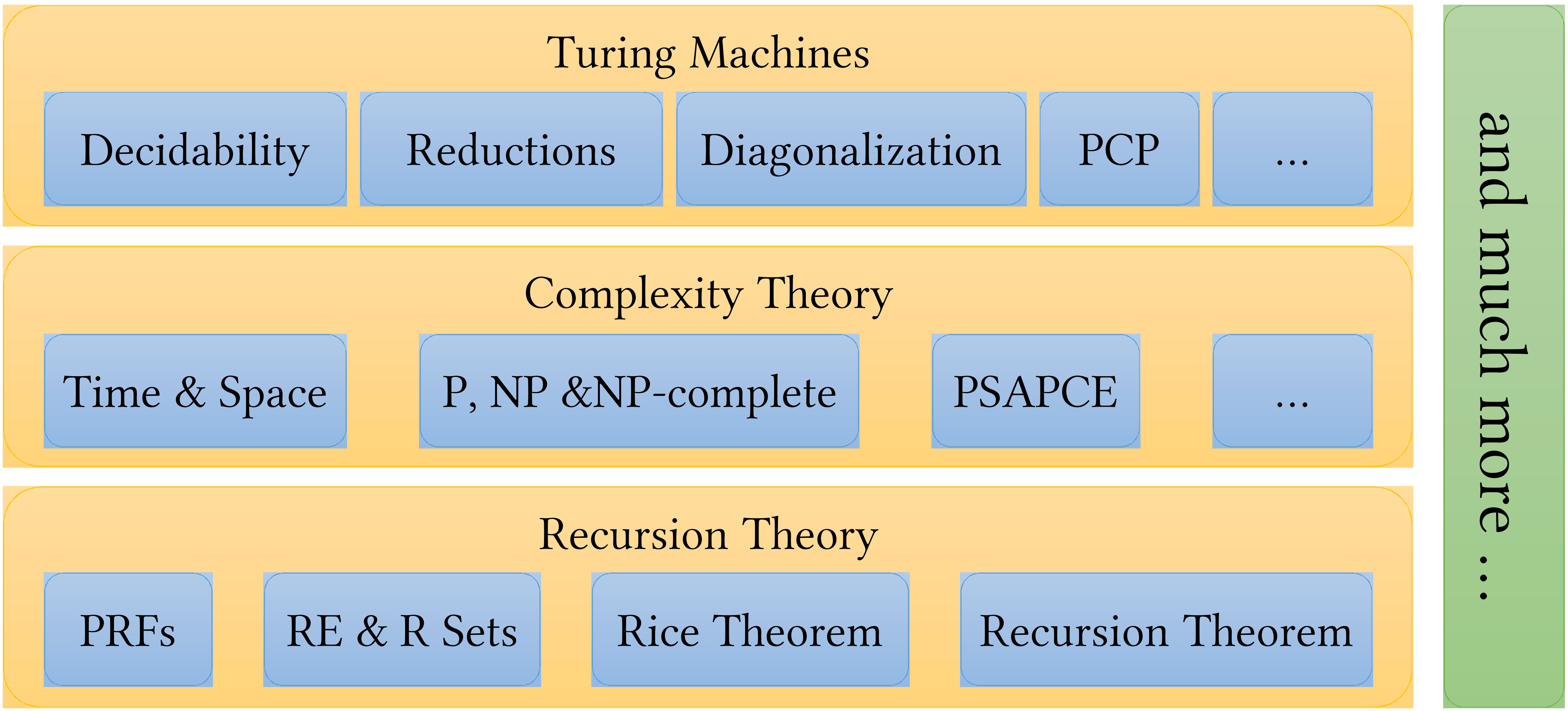
These two seemingly different computational tasks are computationally equivalent.

Precise Mathematical Models of Computing

- Turing machines
- Kleene's partial recursive functions
- Church's lambda calculus

Any computable function is computable by a Turing machine.

Our Plan

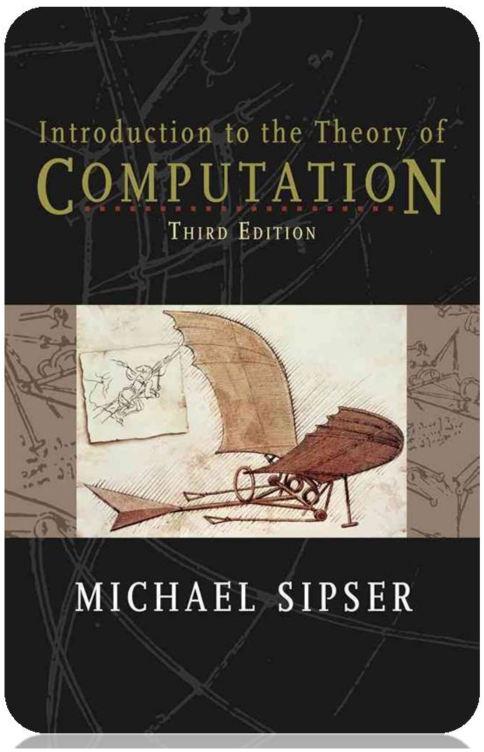

Turing Machines
Decidability
Reductions
Diagonalization
PCP
...
Complexity Theory
Time & Space
P, NP &NP-complete
PSAPCE
...
Recursion Theory
PRFs
RE & R Sets
Rice Theorem
Recursion Theorem
and much more ...

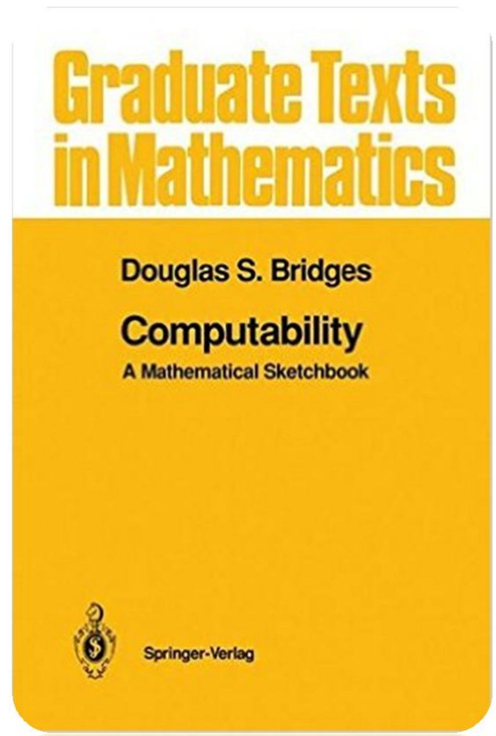
Our References

[S12] Sipser, Michael. Introduction to the Theory of Computation, 3rd edition. Cengage Learning, 2012.

[B94] Bridges, Douglas S. Computability: a mathematical sketchbook. Vol. 146. Springer Science & Business Media, 1994.

Chapters 5 to 8





Chapters 2 to 5

Evaluation

Title	Grade	Description
Exercises	5	At least 12 series
First Mid-term	3	Sunday, Aban 2nd, 1395 Chapters 3-5 of **[S12]**
Second Mid-term	3	Tuesday, Azar 16th, 1395 Chapters 7 and 8 of **[S12]**
Final	9	All of the topics
Excellence	+2	
Total	20+2	

10% penalty for every late day.
100% penalty after 72 hours.

We have a great emphasis on PROOFS.

A ***good mathematical proof*** should be

Clear – easy to understand
Correct

In proofs, we will provide **three** levels of detail

- a short phrase/sentence giving a hint for the proof
  - e.g. "The proof is by contradiction", "The proof is by induction", "The proof uses the Pigeonhole principle", "The proof is by construction", etc.
- a short paragraph describing the main ideas
- the full proof

Please write your solutions in this way.

An example

Proposition: Suppose $A \subseteq \{1,2,\ldots,2n\}$ with $|A| = n+1$. There are **always two** numbers in A such that one number divides the other number.

Level 1:

odd part

- We use the Pigeonhole principle.
- We also use the fact that "Every integer a can be written as $a = 2^k m$ where m is an odd integer and k is also an integer".

Level 2:

The proof idea is as follows. We will show using the Pigeonhole principle that there are $a_1 \neq a_2$ of A such that $a_1 = 2^i m$ and $a_2 = 2^k m$ for some odd integer k and integers i and k.

An example

Proposition: Suppose $A \subseteq \{1,2,\ldots,2n\}$ with $|A| = n+1$. There are **always two** numbers in A such that one number divides the other number.

TRY! ☺

Level 2:
The proof idea is as follows. We will show using the Pigeonhole principle that there are $a_1 \neq a_2$ of A such that $a_1 = 2^i m$ and $a_2 = 2^k m$ for some odd integer k and integers i and k.

Proofs in class …

- During the lectures, I generally provide proofs of the first two levels and only some parts of the third level.

- The reasons for this:
  - In this course, usually, the second level is more important than the third,
  - You can master the material by thinking and trying to fill the missing parts,
  - It is a matter of time! We have a limited time.

Now, let's begin our journey!