Impossibility Results

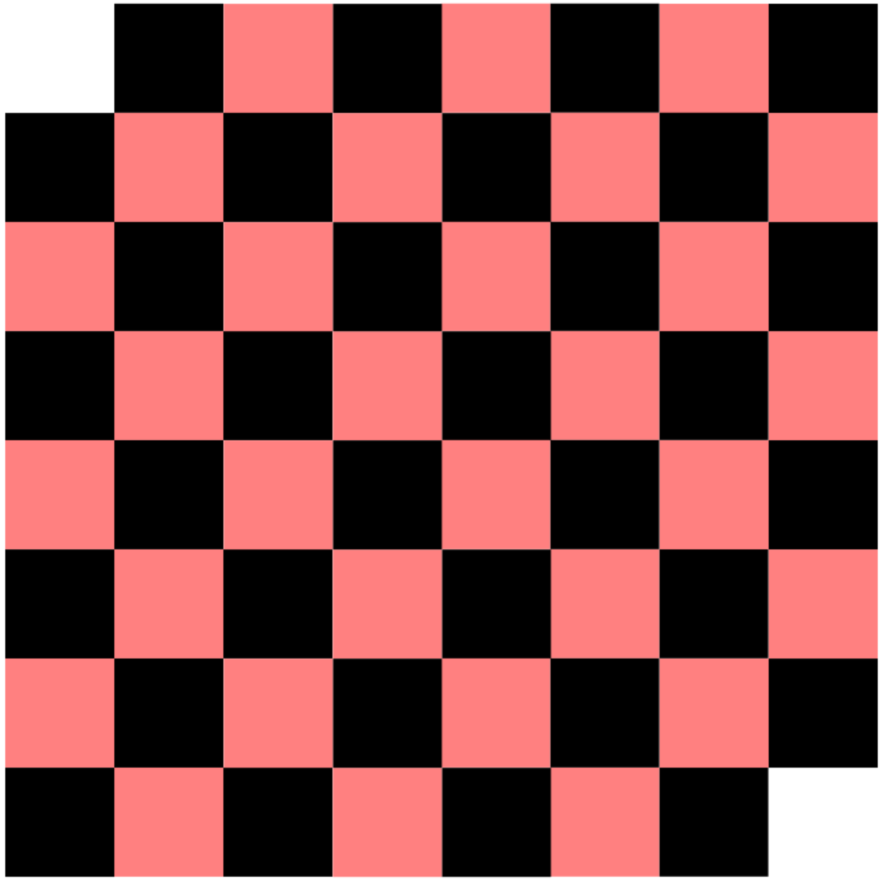
Berkeley Math Circle

Oct 9, 2024

Lecturer: Avishay Tal

Dominoes on a Mutilated Chessboard.

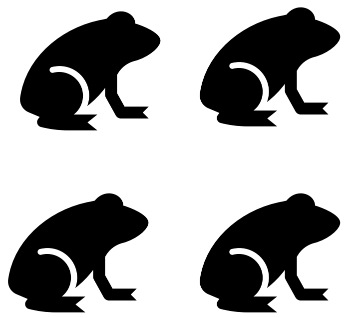
Is it possible to place 31 dominoes of size 2x1 to cover all the squares?



Source: Wikipedia

Leaping Frogs Puzzle – Double the Square

4 frogs are placed on the corners of a 1x1 size square: (0,0), (0,1), (1,0), (1,1)



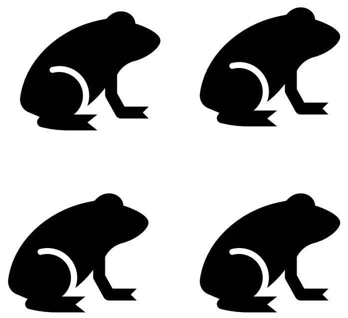
At each point, a frog can leap above another,
landing at the same distance on its other end.

Can we **double** the square?

i.e., can the frogs reach the corners of a 2x2 size square: (0,0), (0,2), (2,0), (2,2)?

Leaping Frogs Puzzle – Half the Square

4 frogs are placed on the corners of a 1x1 size square: (0,0), (0,1), (1,0), (1,1)



At each point, a frog can leap above another,
landing at the same distance on its other end.

Can we **half** the square?

i.e., reach the corners of a 0.5x0.5 size square: (0,0), (0,0.5), (0.5,0), (0.5,0.5)?

Connection between the two puzzles?

- Suppose you can double the square, then you can half the square.
  - Why?
  - Because the sequence of moves is reversible.
- But, we know we can't half the square, so we can't double it!

Two principles:

- **Invariants** –
  - Dominoes cover the same number of red and black squares.
  - Frogs can get only to integral points - (x,y) for integers x, y.
- **Reduction** – solving problem A implies solving problem B.
  But if problem B is impossible to solve, then so is A.

The Barber Paradox (Russell's Paradox)

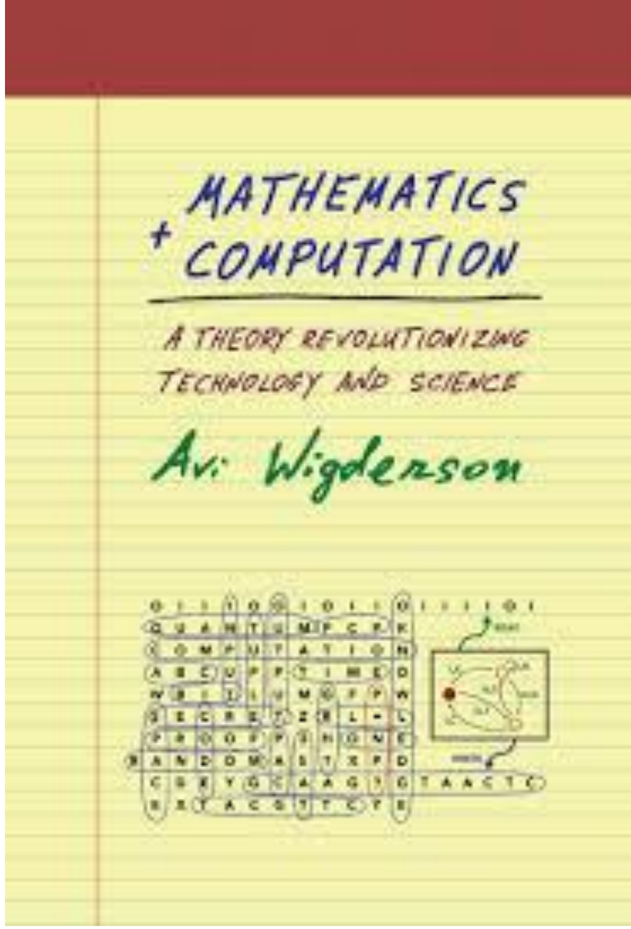
- In an island, some men shave themselves and others don't.
- A male barber, Tony, wants to shave all men that do not shave themselves.
- Is it possible?
- Does Tony shave himself?
    - If he does, then he shouldn't.
    - If he doesn't, then he should.

Theory of Computation

What is computation?

Computation is the evolution process of some environment, by a sequence of "simple, local" steps.

Avi Wigderson, "Mathematics and Computation"



- Bits in a computer.
- Computers in a network.
- Atoms in matter
- Neurons in the brain.
- Proteins in a cell.
- Cells in a tissue.
- Bacteria in a Petri dish.
- Deductions in proof systems.
- Prices in a market.

…

Computability Theory

What can be computed?

Programs -- some examples

```
def foo(n):
    for i from 2 to n-1:
        if i divides n:
            return False
    return True
```

What does this function check?

Programs -- some examples

```
def is_prime(n):
    for i from 2 to n-1:
        if i divides n:
            return False
    return True
```

What does this function check?

Programs – some examples

```
x = 1
while x != 101:
        x = x+2
```

What would the program do?

Programs – some examples

```
x = 1
while x != 101:
        x = x+3
```

What would the program do?

Program – some examples

- Is it easy to check if a program halts?
- In some cases, it does.
- Let’s see a more complicated one…

Programs – some examples

```
def is_prime(n): ... (as defined before)

n = 2
While True:
        n = n+2
        Flag = False
        for i from 2 to n-1:
                if (is_prime(i)  and  is_prime(n-i)):
                        Flag = True
        if (Flag==False):
                halt!
```

Goldbach's conjecture:
Every even natural number greater than 2 is the sum of two prime numbers.

Would this program halt?

It would halt if and only if the **Goldbach's conjecture** is false!

The halting problem

- We view programs both as text (the code of the program) and as algorithms.
- An input to a program can be the code of another program.
- **The Halting Problem:**
  Given (the code of) a program **P** and input **I**, does **P** halts on **I**?
- Why not just simulate the program?
- Well, we can do it.
  - If it halts, we can answer yes!
  - If it doesn't halt…

The halting problem is undecidable

[Turing] There's no program that can decide the halting problem.

Proof:
Suppose by contradiction there is a program that decides the halting problem – call it **Halt**.

Let's look at another program called **Turing**.

```
def Turing(P):
        if Halt(P,P):
                loop forever
        else:
                halt
```

Does **Turing** halt on **Turing**?

- If **Turing** halts on **Turing**, then it should loop forever.
- If **Turing** doesn't halt on **Turing**, then it should halt.

In either cases, we reach a contradiction! ➔ **Halt** cannot exist.

Another way to view the proof - Diagonalization

	P_1	P_2	P_3	P_4	…
P_1	**H**	L	H	L	
P_2	L	**H**	L	H	
P_3	L	H	**H**	L	
P_4	L	H	H	**L**	
…					

Entry (i,j) – does $\mathbf{P_i}$ halt on input $\mathbf{P_j}$?

Turing is the opposite of the diagonal!

- If $\mathbf{P_i}$ halts on $\mathbf{P_i}$, **Turing** loops on $\mathbf{P_i}$

- There's no program for Turing!
- There's no program for Halt!

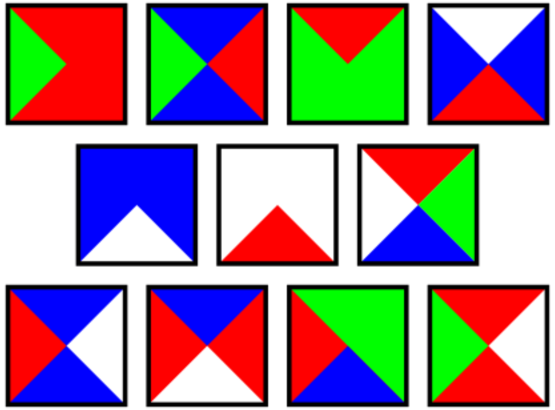
Computability Theory

- Computability theory studies which problems can be solved by computers, and which are undecidable.
- Using reductions, we can show that almost all “**Program Checking**” problems are undecidable.

Another undecidable problem:

Tiling: Given a collection of tiles as input,

can you tile the infinite plane with these tiles?



Source:Wikipedia

Computational Complexity

What can be computed efficiently?

Splitting a bar of chocolate to squares

How many times you need to cut the bar, to get all 24 1x1 pieces?

Can you do better?

Source: Wikipedia

Addition vs Multiplication

- How much time it take to add two **n** digit numbers?
- How much time it take to multiply two **n** digit numbers?

```
        1534568
      x 5714361
  -------------
        1534568
       92074080
      460370400
     6138272000
    15345680000
   107419760000
   767284000000
  -------------
  8769075531048
```

- It seems that multiplication is harder than addition.
  Can we prove it?

The story of Kolmogorov and Karatsuba

- In 1960, the famous mathematician, **Kolmogorov**, organized a seminar, where he stated a conjecture that multiplication cannot be done in less than n^2 time. The plan was to explore how to prove this conjecture
- After a week, a student, named **Karatsuba**, discovered a much faster algorithm – that takes roughly $n^{\log_2 3} \leq n^{1.59}$ time
- Kolmogorov was very excited about the discovery and terminated the seminar. He went on to give lectures on it in conferences around the world.
- Today we know of algorithms taking $n \log(n)$ time
- Is multiplication harder than addition? We still don't know

Sorting an Array

Sorting an array is quite useful.

[4,1,7,5,3,10] ➔ [1,3,4,5,7,10]

For example: It allows quick search.

There are several algorithms that sort n elements in $n \log n$ time.

Q: Can you do better?

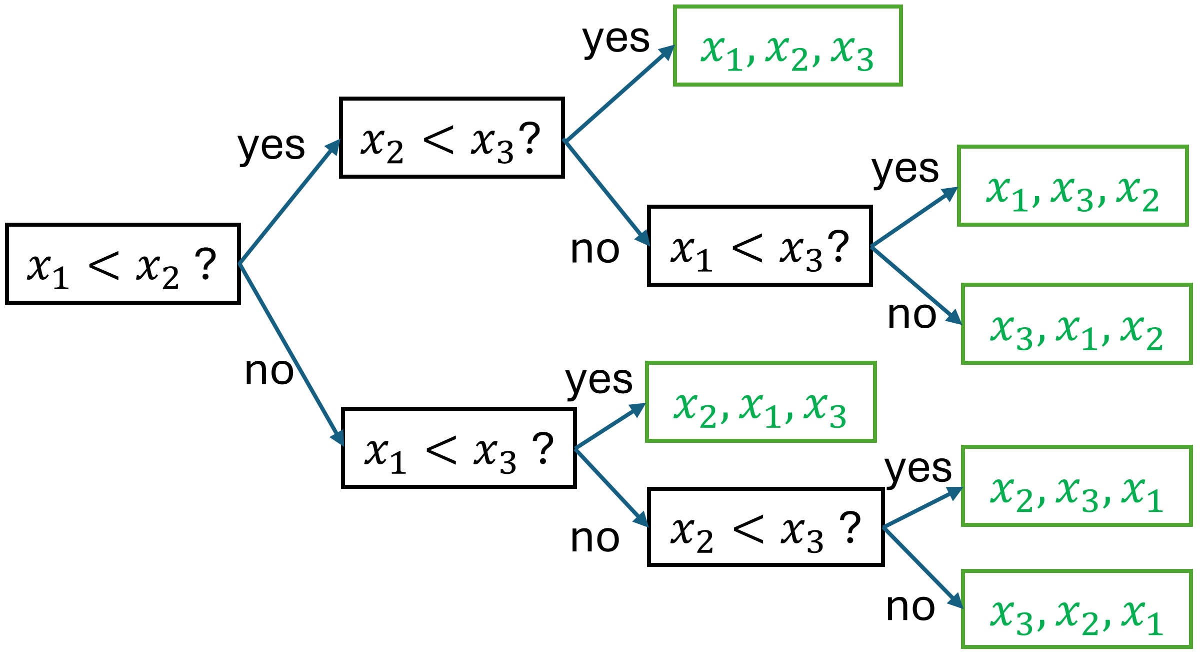
Lower Bound for Sorting

- Suppose $\mathbf{x_1}, \ldots, \mathbf{x_n}$ are the items in the array
- Suppose we can only compare items, e.g., is $\mathbf{x_3}<\mathbf{x_5}$? is $\mathbf{x_2}<\mathbf{x_7}$?
- How many comparisons do we need to do in the worst-case?

Theorem: Any comparison-based algorithm for sorting must take at least $\sim n \log n$ steps in the worst-case.

Proof Idea

View an algorithm as a decision tree



If the depth of tree is d, how many leaves it can have?

Every permutation must appear in at least one leaf.

➔ $n! \leq \#leaves \leq 2^d$

P, NP

P: We consider a problem to be **efficiently solvable**, if there's an algorithm solving it in polynomial time: on inputs of length n, the algorithm runs in at most n^c time.

NP: We consider a problem to be **efficiently verifiable**, if there's an algorithm that can check if a solution is valid in polynomial time.

The Million Dollar Question: Does **P=NP**?

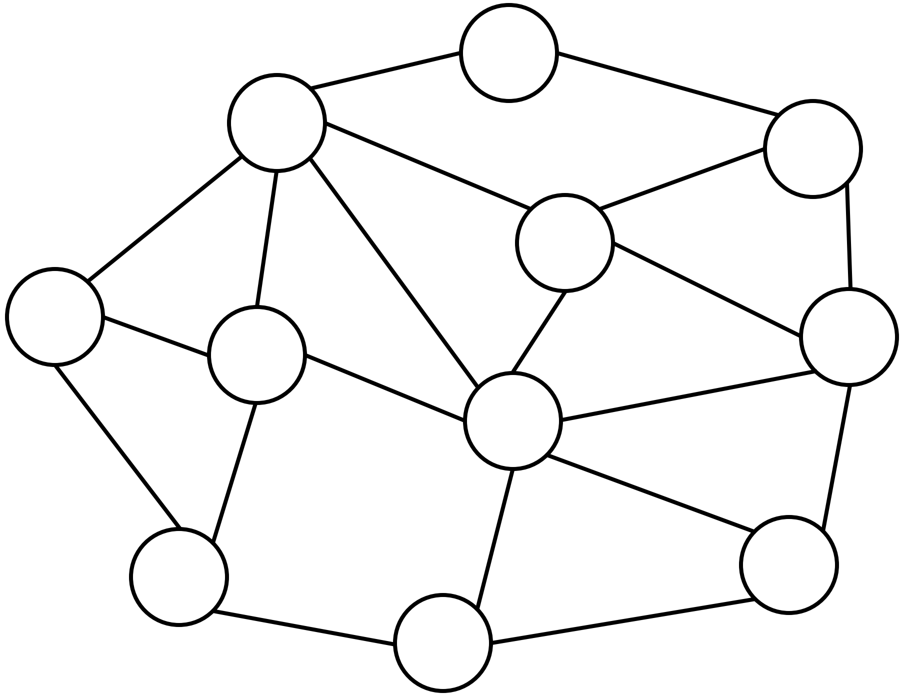
P, NP, NP-Complete

NP-complete: Problems in NP that are the “hardest to solve”.

Examples of NP-complete Problems:

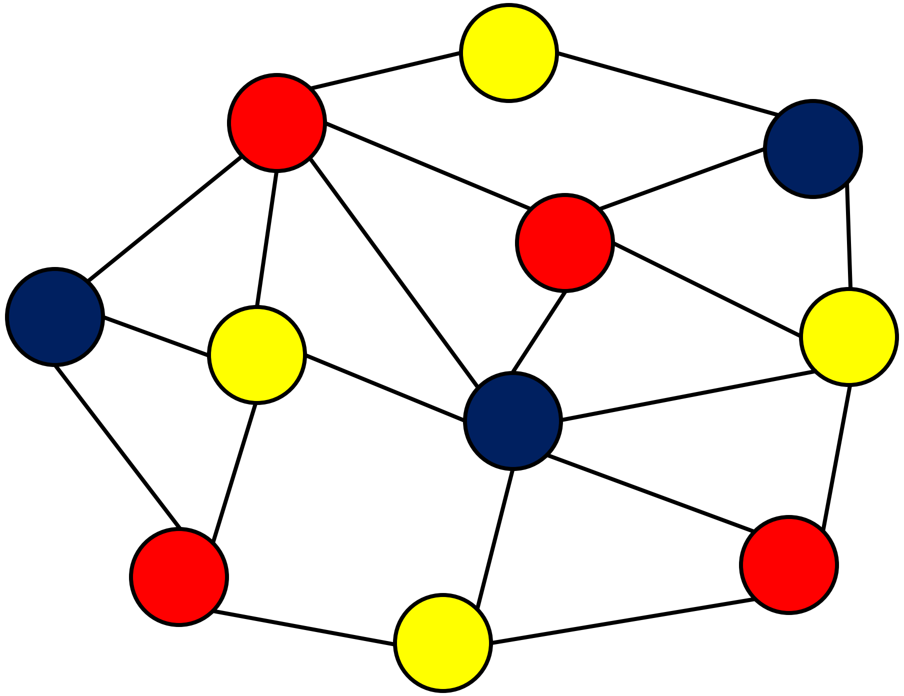
- Coloring a graph in 3 colors.
- Finding a large **clique** in a social network
- Satisfying multiple constraints (e.g., scheduling)
- Packing
- Solving a system of quadratic equations
- Generalized Sudoko
- Traveling Salesperson

Graph Coloring



Can you color the vertices of this graph with 3 colors so that every edge touches two different colors?

Graph Coloring



Can you color the vertices of this graph with 3 colors so that every edge touches two different colors?

Integer Factorization

Given a positive integer n find its prime factors.
For example: Given $n = 15$ output $3,5$.

We don't know how to do it quickly for numbers of 1000 digits.

Current technology allows only to factorize numbers with up to 250 digits and this took 2700 core-years.

Scaling: The current best algorithm runs in time exponential in $(\#\text{digits})^{1/3}$

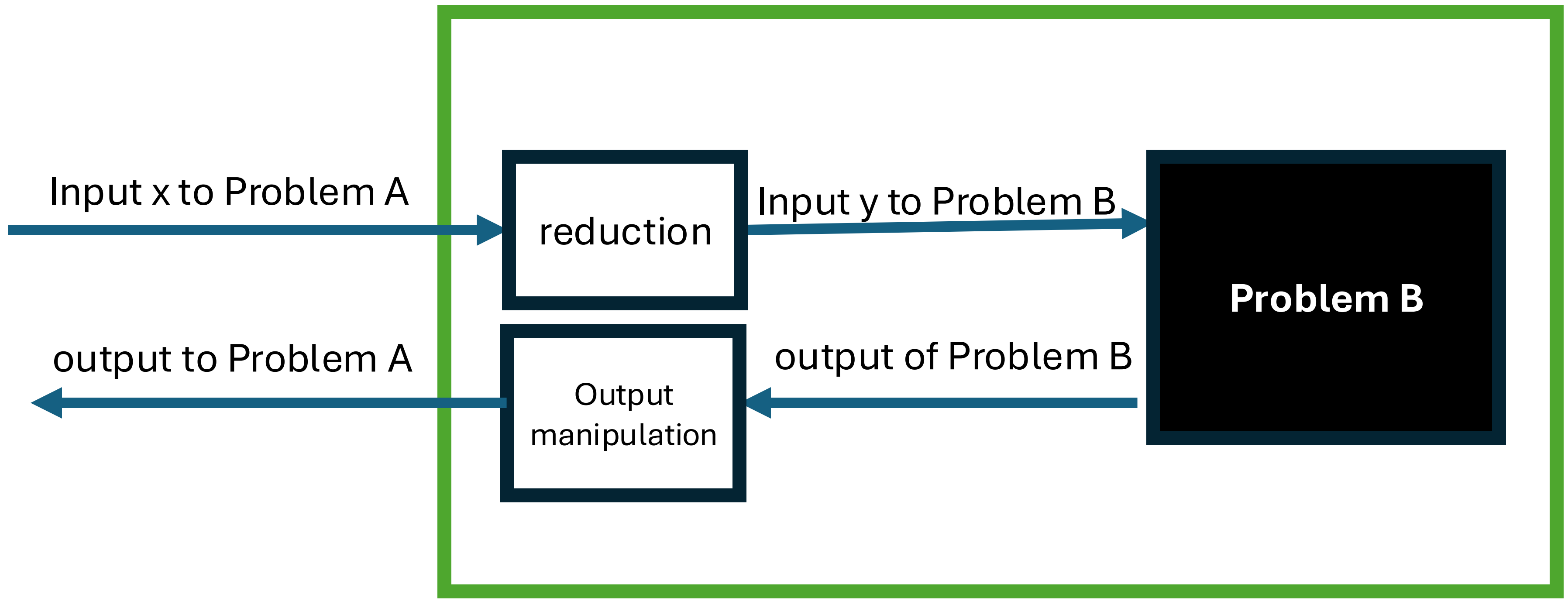
So same method would take

- 600 Billion core years on 500-digits numbers
- 56,000 Billion Billion core years on 1000-digits numbers
- 15 Billion Billion Billion Billion core years on 2000-digits numbers

Reduction between Problems

If we can solve Problem B efficiently,
then we can solve Problem A efficiently.

If we cannot solve A efficiently, then we cannot solve B efficiently:
“$A \leq B$”



NP-Complete

What does it mean: Problems in NP that are the "hardest to solve"?

A reduces to B, "$A \leq B$": If we can solve Problem B efficiently, then we can solve Problem A efficiently.

If we cannot solve A efficiently, then we cannot solve B efficiently:

Definition: A problem **A** is NP-complete if:

- **A** is a problem in **NP** – we can verify solutions in polynomial time.
- Any problem in **NP** reduces to **A**.
  - If we can solve **A** in polynomial time,
    then we can solve any other problem in **NP** in polynomial time.

Summary

- **Impossibility puzzles:** dominoes on a chessboard, frogs.
- **Impossibility by invariants** – every allowed configuration has a certain property that the end goal doesn't
- **Impossibility by reduction** – if we can solve B, we can solve A.
  If we cannot solve A, we cannot solve B.
- **The Barber Paradox**
- **Computability** – what can computers do?
- The Halting Problem is undecidable
- **Complexity** - what can computers do efficiently? (we don't know yet)
- **P**, **NP**, **NP**-complete
- **NP**-complete – one for all and all for one.
  (solve one of them, you solved all of them)