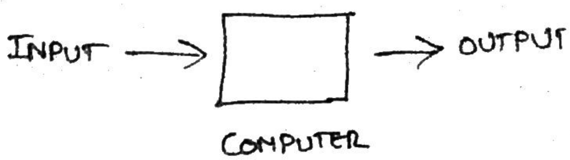
Complexity theory - How hard is a certain problem?

Computability - Problems that are impossible to solve



Sometimes problems are so hard to solve, that no matter how long you wait you don't get an answer.

Complexity theory : HOW MUCH? (time, space etc.)

Computability : IS SOMETHING POSSIBLE?
CAN A PROBLEM BE SOLVED?

INITIAL GOALS

- Recognize challenging problems (3 hrs)
- Understand the challege (1.5 hrs)
- Navigate around such challenges (1.5 hrs)
- LEARN ABOUT UNSOLABLE PROBLEMS (1 hr)

How fast can a comp. find a way between two nodes in a graph? $\boxed{\text{COMPLEXITY THEORY}}$

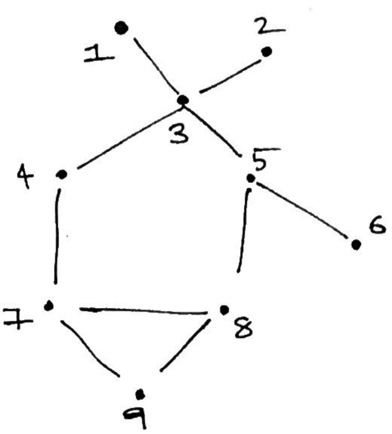
Can a program decide if another program is malware? $\boxed{\text{COMPUTABILITY}}$

How much memory is needed to sort a sequence of data? $\boxed{\text{COMPLEXITY THEORY}}$

We will now discuss 3 problems that have practical relevance.

1. Consider a graph $G(V, E)$. Find the minimum set of vertices such that every edge is connected to at least one vertex in the set.

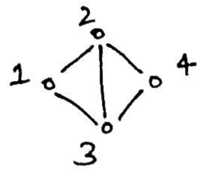
eg:



Answer : $\{5, 8, 7, 3\}$

idea : try all possibilities!

consider



$\Rightarrow \{2, 3\}$

in this case, we have 16 different possibilities (2^4)

```
min_vertices = # vertices
for each assignment of (0,1) to the vertices:
        if assignment is valid
                number_vertices = # 1's in assignment
                min_vertices = min( min_vertices, number_vertices)
```

---

n vertices

2^n assigments $\Rightarrow 2^n$ iterations

if $n = 500 \Rightarrow$ way more than a trillion trillion iterations!

$$2^{500} > 10^{150}$$

Problem vs Algorithm

Name — Program description

Input

Output

Hard / Easy

Running time of an algorithm depends on

- → input size
- → content / structure of input
- → computer spec.
- → implementation of algo
- → programming language

Need simplifications

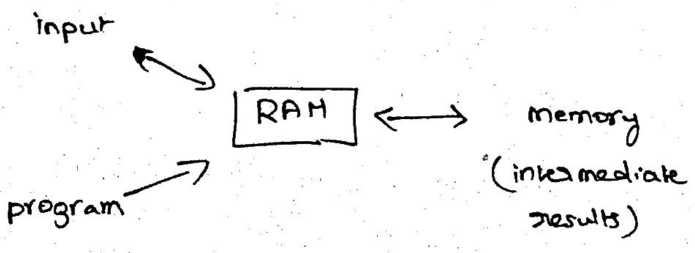
- → Analyze w/o implementing (RAM model)
- → Worst input cases (worst case running time)
- → Ignore details (Big O)

RAM model

idea: to have an abstract model that only has essentials

Working memory?	✓
graphics card?	✗
monitor?	✗
input / output?	✓
os?	✗
cd-rom?	✗
programming capability?	✓

In RAM model (random access machine),



⇒ simple ops. take 1 timestep (addn, if, assignment)

⇒ complex ops take as many as they # runs (for, while)

⇒ memory access is free (assignments and reads)

```
a = 1          0
b = 2 * a      1
              ---
               1 timestep
```

```
s = 5          0
while s > 0:   6
    s = s - 1  5
              ---
              11 timesteps
```

Looks like we need further simplification. Can't analyse step by step for each input.

Can't analyze every input, so let's just consider WORST CASE!

for a given input size, some take longer to solve than the others

---

```
count = 0
for ch in S:
    if ch == 'a':
        count = count + 1
```

---

timesteps = $2n + x$ | $n = len(S)$, x = # a's in S

$2n$ → size indicator

x → structure indicator

best case ⇒ $x \leftarrow 0$

worst case ⇒ $x \leftarrow n$

avg case ⇒ hard to define, not worth it

not very useful

offers guarantees

interesting, not relevant

By opting worst case analysis, we have eliminated the detail of analysing input structure to calculate running time.

The process to calculate running time is still tedious as you need to go over each line of the program.

Big O

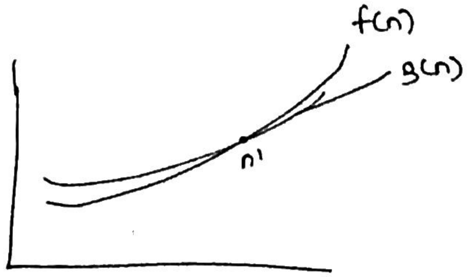
Worst case

Algorithm A: $2n^2 + 23n - 5$
Algorithm B: $2^n - 50n + 256$

Which one would you prefer given you donot know about the input?

Obviously, A.

Let's consider running times where input size is arbitrarily large.



for all $n > n'$
$f(n') \geq g(n')$

We only focus on the lines of program that get affected as the input grows...

$f(n) \in O(g(n))$

$g(n)$ grows atleast as fast as n

$\Rightarrow \forall\, n \geq n' \quad c \cdot g(n) \geq f(n)$ where c is a constant

Basically, $g(n)$ eventually out grows $f(n)$.

$3n+1 \in O(n)$

$18n^2 - 50 \in O(n^2)$

$2^n + 30n^6 + 123 \in O(2^n)$

$2^n \cdot n^2 + 30n^6 \in O(2^n \cdot n^2)$

I have grossly underestimated time to finish units of information.

< End of hour 1/ >