Module 1 – Regular Languages & Finite Automata

Lecture 1 :

Basics of Theory of Computation

Join **GO+ GO Classes Combined Test Series** for BEST quality tests, matching GATE CSE Level:

Visit www.gateoverflow.in website to join Test Series.

1. **Quality Questions:** No Ambiguity in Questions, All Well-framed questions.
2. Correct, **Detailed Explanation**, Covering Variations of questions.
3. **Video Solutions.**

Join GO Classes Resources, Notes, Content, information Telegram Channel:

Public Username: goclasses_cse

Join GO Classes **Doubt Discussion** Telegram Group :

Username: GATECSE_Goclasses

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: gateoverflow_cse

Module 1 – Regular Languages & Finite Automata

Next Topic :

GATE weightage, books, complexity

Theory of Computation

Different names of the same course:

Theory of computation
Automata Theory
Theory of Formal Languages and Computation
Formal languages, Automata, Grammars

GATE : 8 - 10 marks (~ 10 %.)

Scoring ↘ Can Easily get 100 %. marks.

Books : Easy : Peter Linz ↘

Good : Sipser ↘

Stanford Toc Course ↘

Practice
- Peter Linz
- John Martin

How to Study this Course:

Lectures ✓✓✓

- Every Concept ✓
- No Rote-learning
- A LOT of Questions (PYQs: GATE, NIET, TIFR)
- Quizzes
- Test series (GO + GO classes)

(After our Course) → Ullman

Not easy to understand for beginner.

TOC — 1930

Computers — 1950-60

Programming languages, Compilers — 1970s

Module 1 – Regular Languages & Finite Automata

Next Topic :

WHAT IS this subject about???

– (Computational) Problems Solving

Computer

Calculator

:

Computational Devices

(Mechanical Devices)

Computer Solves a Problem means?

Computer Solves a Problem P means ?

Problem P $\Rightarrow$ $\exists$ program M

"Program" — A tool to Express a Algorithm

(In some prog. Language)

A Program Expresses Algorithm.

Computer Solves a Problem P

$\exists$ program M

$\exists$ Algorithm A

Problem P1 : Given a List of integers, find maximum?

Is there any Algorithm? Yes.

(Algo A_1) → O(n)

Scan list and find max

Problem P1:

I_1 : 60, 65, 50 | $f(I_1) = 65$

I_2 : 2, 3, 4, 5, 6, 1, 0 | $f(I_2) = 6$

I_3 : −1, −2, 1, 2 | $f(I_3) = 2$

Problem $\equiv$ function

Algo A_1 Solves P_1

means

Algo A_1 Computs function f_1

Every Algo Solves a problem.

" " Computes a function.

Problem P2 : Given a list of integers, Sort it.

$\exists$ Algorithm ? — Yes

Algo A_2 → O(nlogn)

merge sort

Problem P2:

I_1: 3, 2, 4, 1 $\quad f(I_1) = 1, 2, 3, 4$

I_2: 4, 3, 1, 2 $\quad f(I_2) = 1, 2, 3, 4$

I_3: 4, 3, 1, 2, 5, 4 $\quad f(I_3) = 1, 2, 3, 4, 4, 5$

Problem P_2 : ≡ function f_2

Inputs f_2 o/ps

I_1 : 3, 2, 4, 1 $\quad f(I_1) = 1, 2, 3, 4$

I_2 : 4, 3, 1, 2 $\quad f(I_2) = 1, 2, 3, 4$

I_3 : 4, 3, 1, 2, 5, 4 $\quad f(I_3) = 1, 2, 3, 4, 4, 5$

Algo A_2 solves Sorting problem P_2

means

Algo A_2 Computes function f_2

(Computational) problem ≡ function

Solving a problem ≡ Computing function.

Prob P3: Given two Graphs, are they Isomorphic ?

G, H — #Vertices = n

∃ Algo ? — Yes.

Algo A_3 → check all bijections

n!

How V(G), V(H) for Adjacency Preservation

O(n!)

	Input size	time
Problem P_1	64 integers	$O(n) \approx$ 64 sec
" P_2	64 "	$O(n \log n) \approx$ 400 sec
" P_3	64 vertices	$O(n!) \approx$ 5×10^{11} years
		$O(64!) \Rightarrow$

Very Hard

↳ NP Hard

Yes or No

$P_1, P_2, P_3 \Rightarrow$ Computable / Solvable by Algorithm (Computer)

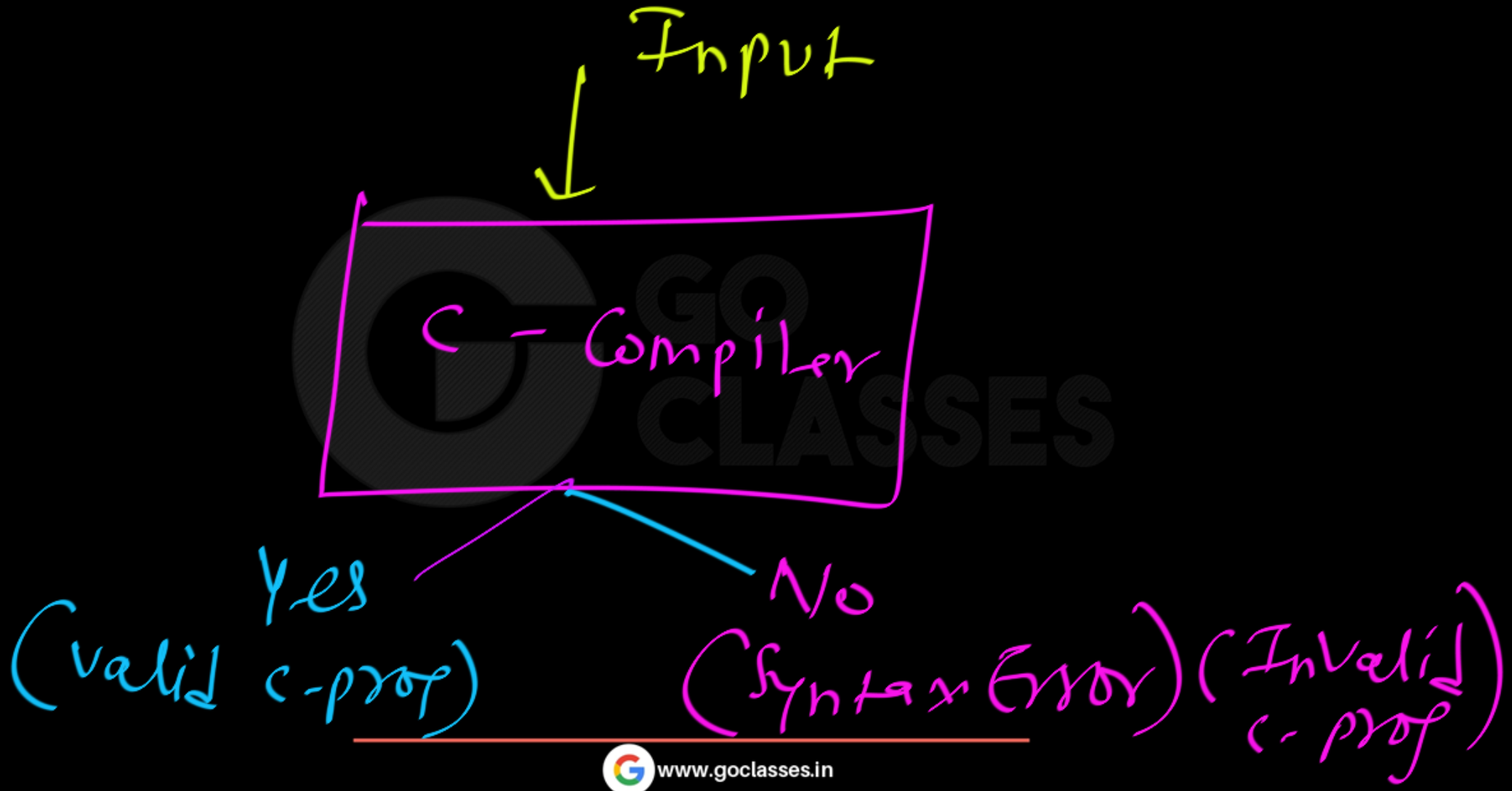
Different Amount of Resources
- Time
- Space

Problem Py : Given a program,

Is it Valid C-program?

$\exists$ Algorithm? — Yes

Algo A_y ⟶ C-Compiler


Input
C - Compiler
Yes
(valid C-prog)
No
(Syntax Error)
(Invalid C-prog)

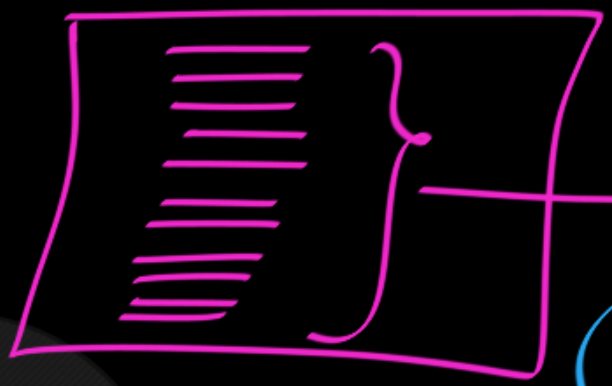
Prob P5 : Given a C-prog,

Will it Ever Halt when Executes?

∃ Algorithm? — No

NEVER will Exist.

Algorithm:

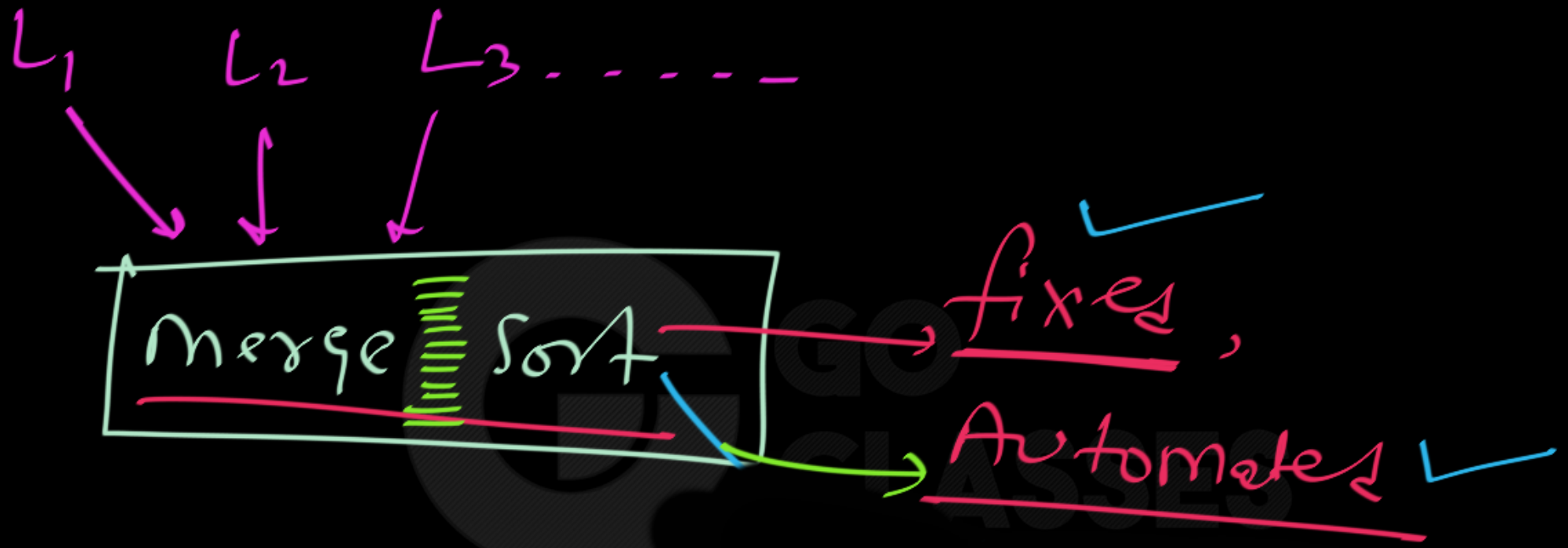


"finite" set of instructions (However easy or Complicated)

(2) Must Halt on Every Input (can take VERRYY long time)

(3) Automated procedure

Algo ⟶ Automated procedure for all inputs ⟹ Same Procedure


L1
L2
L3 - - - - -
Merge Sort
fixed,
Automated

Prob P_5 :

Halting Prob of programs

↓

No Algorithm

(Does not exist)

NEVER will Exist.

Prob P5 : ⟹ Any Computational Devices ⟹ Can NEVER solve,

Not Computable

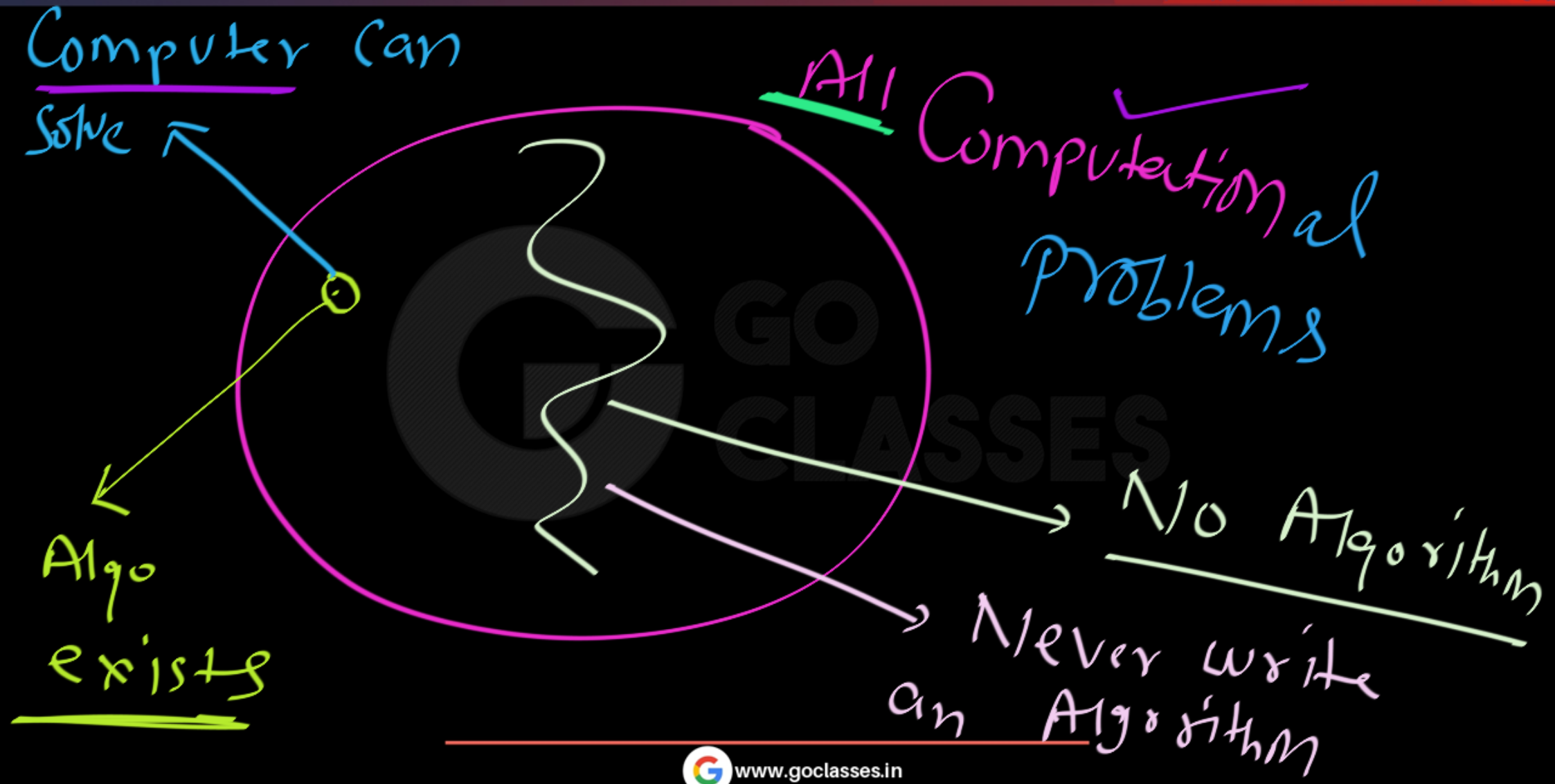
Problem / function

Algorithm exists

Ex: Sorting, Sortest path, Searching, longest path, MST, Coloring, Isomorphism

No Algo (NEVER)

Ex: Halting problem of programs.


Computer Can
Solve
All Computational Problems
Algo
exists
No Algorithm
Never write
an Algorithm

IISc ⟶ 1st sem : PAV

Prof "Analysis" & "Verification"

Study many properties of programs

- Will given prog Halt ?
- Does prog has null pointer ?

Project →  Program

Automates
Algo

one of the program property

T

Goal of Toc :

(1) Which problems can "Never" be solved using any Computational Device ?

Goal of ToC :

(1) Which functions can "Never" be Computed using any algorithm.

Goal of Toc ?

(2) Among the problems that are solvable, How much Resources they take

- Time
- Space

Problems

∃ Algorithm

No Algorithm

Problems

Depending on time, Space they need.

Module 1 – Regular Languages & Finite Automata

Next Topic :

WHAT IS

Theory of Computation???

What do you understand by Computation?

How do you "Compute" something?

writing a program

Execution of program

Theory of Computation

↓ ↓ ↓

" " Programs

What do you understand by Program?

What do you understand by Program?

Programs Express Algorithms (in some programming language)

What do you understand by Algorithm?

WHAT IS A RECIPE / Algorithm

1) sequence of simple **steps**

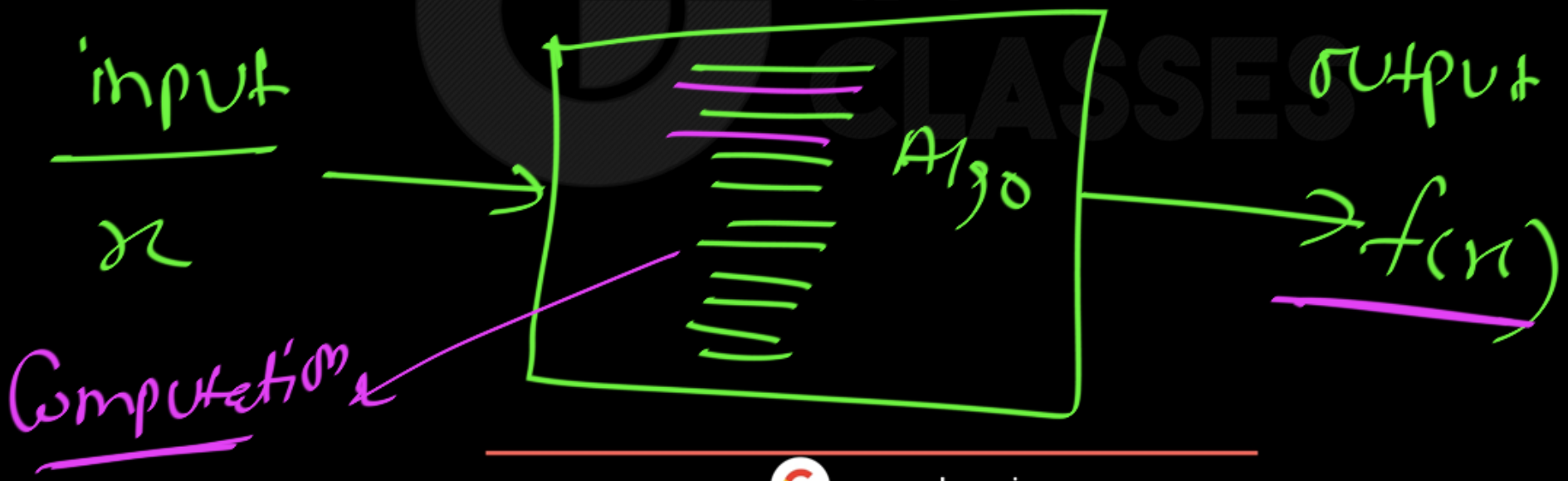
2) **flow of control** process that specifies when each step is executed

3) a means of determining **when to stop**

1+2+3 = an **algorithm**!

What do you understand by Algorithm?

An algorithm is a Recipe for carrying out input to output transformation.



Problem ≡ function

Algorithm Computes function.

" Solves Problem.

Algorithms Compute Functions?

What do you understand by Algorithm?

An algorithm is a Recipe for carrying out input to output transformation.

So, Algorithm computes a function.

An Algorithm tells us HOW TO Compute a function.

GOAL of Our Subject TOC?

Finding Class of Functions/Problems for which Algorithm exists that computes that function(solves that problem)

GOAL of Our Subject TOC?

To find out, for which Functions/Problems, There exists an Algorithm that computes it.

Every Algorithm computes a function.

BUT can every function/problem be computed by some algorithm?

GOAL of Our Subject TOC?

Every Algorithm computes a function.

BUT can every function/problem be computed by some algorithm?

GOAL of Our Subject TOC?

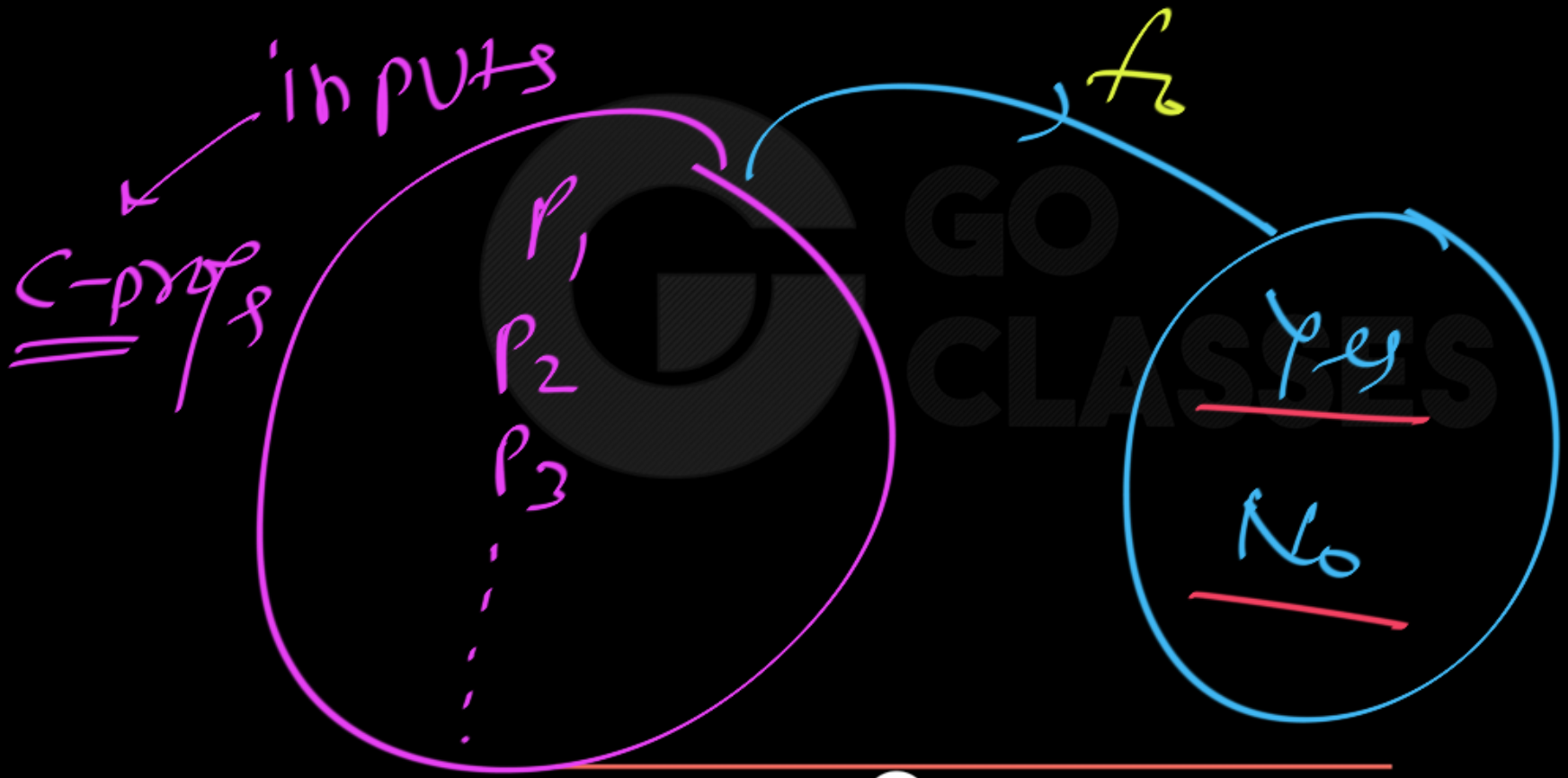
If you can define a function/problem, does it mean you can compute it?

GOAL of Our Subject TOC?

If you can define a function/problem, does it mean you can compute it? No

Problem: Given a C-prof, Does it Ever Halt? (P_6) ⇒ NOT Solvable

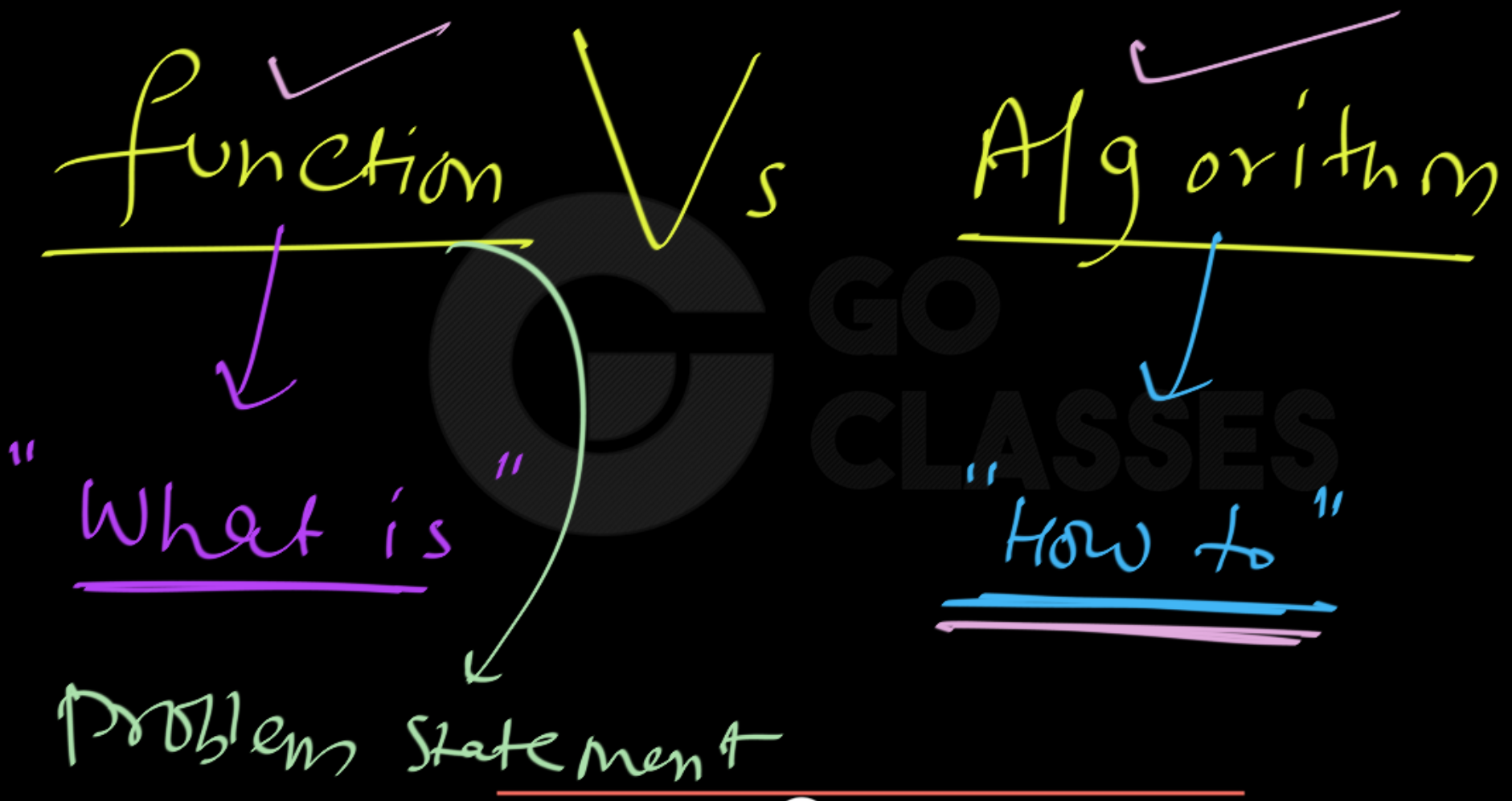
Prob P_i : ≡ function f_i



fun fc ⟶ Defined

↳ Not Computable

↳ Non-Computable function ≡ Unsolvable Problem



Ex: function: Is_prime(x)

Inputs = Natural Numbers

o/p = {Yes, No}

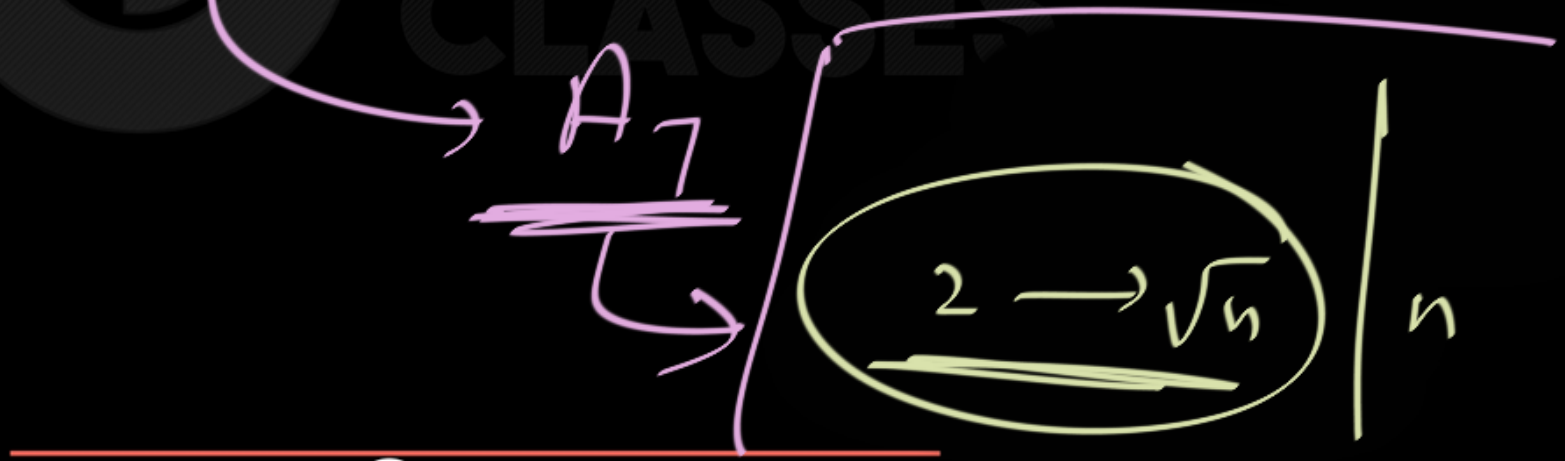
Definition of function:

$$Is\_prime(x) = \begin{cases} Yes & ; \text{If } x \text{ is prime} \\ No & ; \text{If } x \text{ is not prime} \end{cases}$$

fun → Is_prime(n)

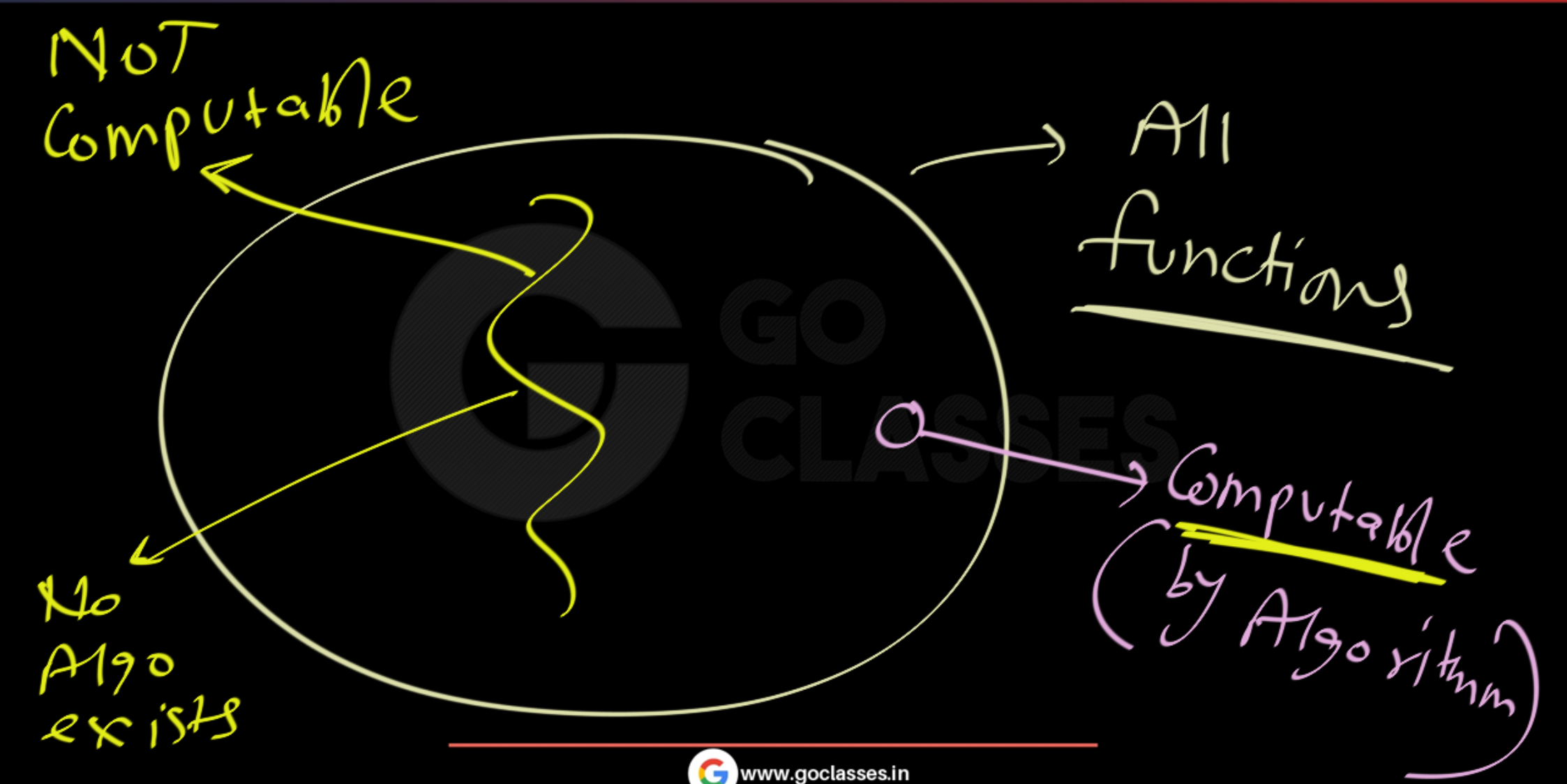
∃ Algo ? ✓ yes.

A1 → A7



Algorithm Computes function.

II Solves Problem.


NOT Computable
All functions
No Algo exists
Computable (by Algorithm)

Computable functions :

→ Algo exists

Non-Computable functions :

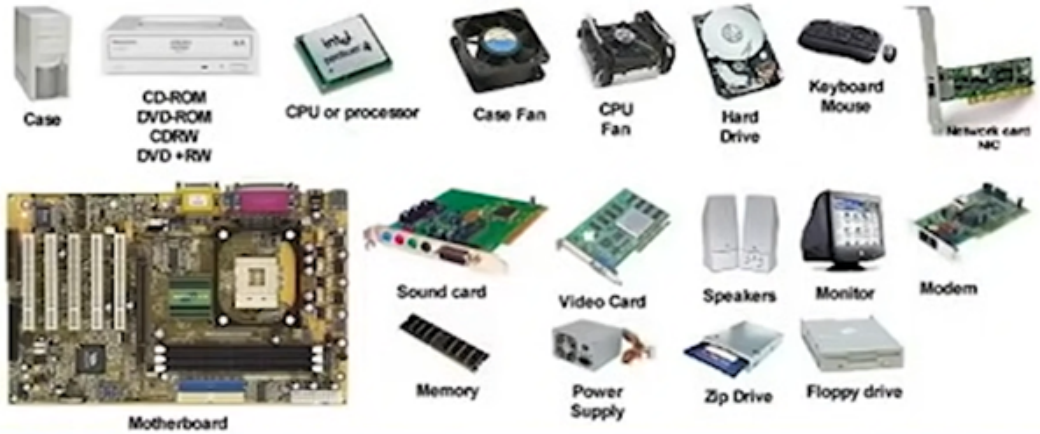
→ No Algo exists

(NEVER Algo will exist)

COMPUTERS ARE MACHINES

- how to capture a recipe in a mechanical process
- **fixed program** computer
  - calculator

- **stored program** computer
  - machine stores and executes instructions



Alan Turing — Mathematician

↳ founder / father of Computer Science

~1930-40 : Studies Limitations of any Computational Devices

Alan Turing ⟹ to study Limitations of Computational Device

↓ ||

created a Theoretical / mathematical Device ⟹ Turing Machine

Turing m/c

Claimed : Turing m/c is "the model of Computation"

↳ "Not Computable by TM" ⟹ Not Computable by any mechanical Device

Proves ⇒

Many Problems have No Algorithm (will never have any Algo)

Alan Turing

Nothing to do with TM.

movie: "The Imitation Game"

Machine

NOT the TM

TM ≡ modern Computation Device

TM ≡ Program ≡ Proj. languages

Problem → Algo → Program → Quantum / Super Computer

TM

1930 - 40 —— "Turing machine"

Very Powerful machine

Most Powerful model of Computation

1930-40 — TM — most Powerful model of Computation

1950-60 — "Simple models" — with limited space, time

FA

PDA

LBA ≡ Modern Computer

TM — Infinite space

No Limit on time

finite amount of time

Models of Computation :

- FA
- PDA

Theoretical / mathematical Devices to compute functions (to solve Problems)

FA = finite Automata — finite memory

PDA = Push Down Automata — infinite memory

LBA = linear bounded Automata

LBA ⊏ modern Computer

TM — infinite memory

Analogy :

FA ≡ Calculator

PDA ≡ mobile

LBA ≡ Computer

TM ≡ Most Powerful Device

Module 1 – Regular Languages & Finite Automata

Next Topic :

Roadmap of this Subject


FA
PDA
LBA
TM
No Algorithm
Undecidability

Module 1 – Regular Languages & Finite Automata

Next Topic :

Syllabus

Theory of Computation???

Section 6: Theory of Computation

Regular expressions and finite automata. Context-free grammars and push-down automata. Regular and contex-free languages, pumping lemma. Turing machines and undecidability.

Theory of Computation Syllabus :

Formal languages, machine (mathematical) models, Grammars.

- Regular languages, finite automata, Type 3 grammar(regular grammar), regular expression, Myhill-Nerode theorem
- Context-free languages, Context-free grammars, pushdown automata
- Unrestricted grammars / Turing machines
- Non-determinism
- Closure properties
- Pumping lemma
- Undecidable problems
- Countability etc...

Course Organization

- Very broadly, the course will contain three parts:
  - Part I) Regular languages (FA)
  - Part II) Context-free languages (PDA)
  - Part III) Turing machines & decidability

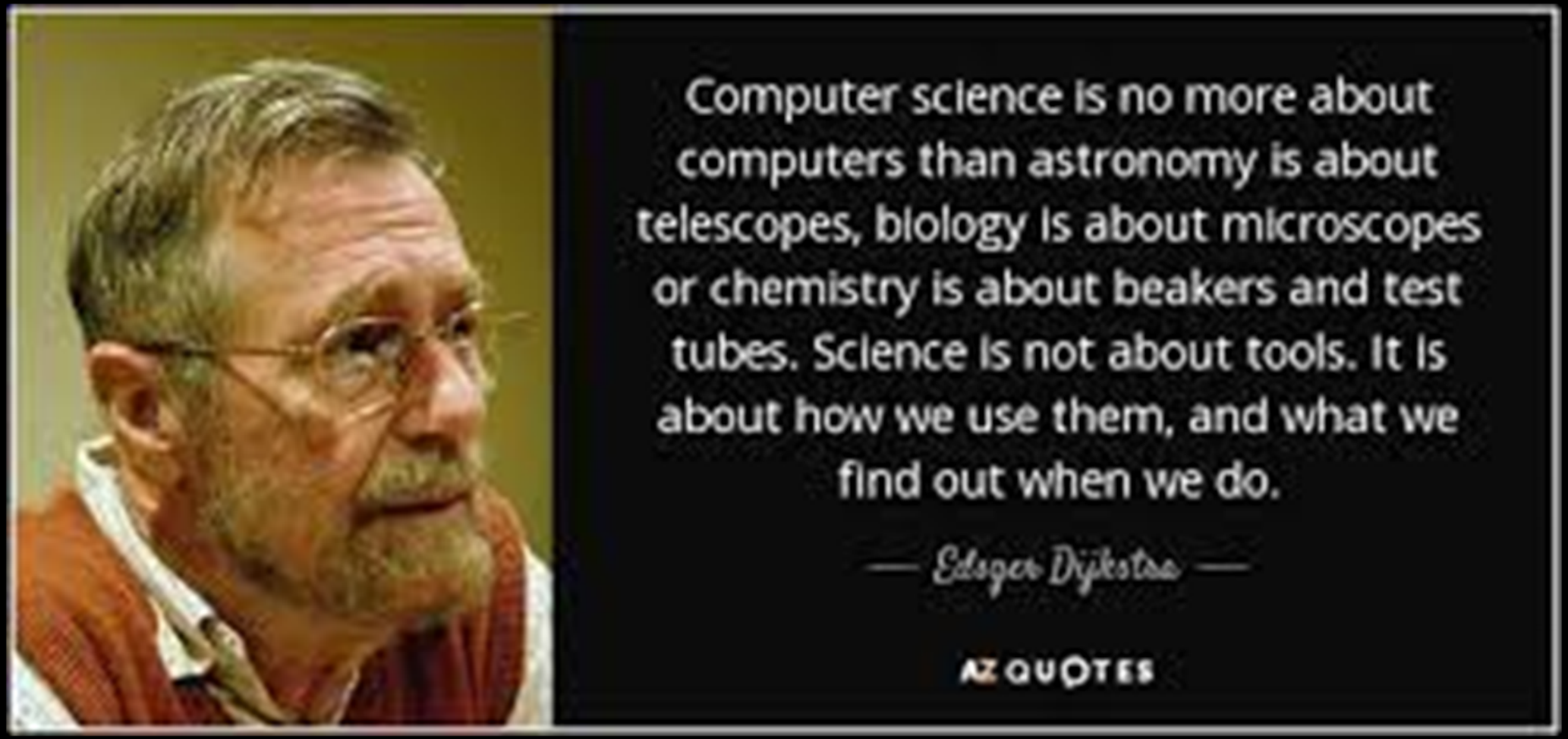
Module 1 – Regular Languages & Finite Automata

Next Topic :

WHY Study Theory of Computation???

Theory of Computation tells you why Computer Science is NOT only about computers, but that it is actually a science.

"Computer science is no more about computers than astronomy is about telescopes." — Dijkastra


Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes. Science is not about tools. It is about how we use them, and what we find out when we do.
— Edsger Dijkstra —
AZ QUOTES

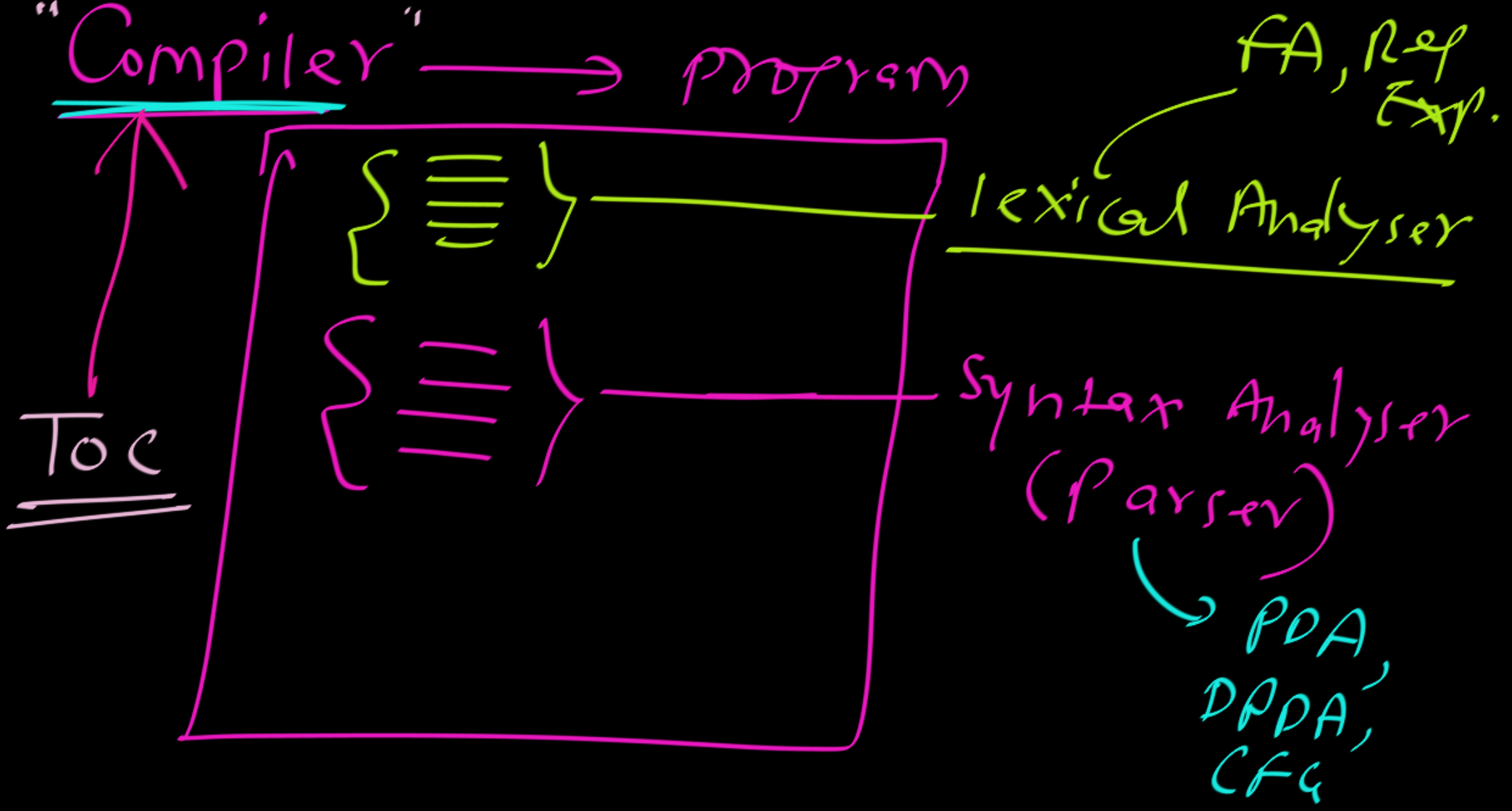
Why Finite Automata and Regular Expressions?

Used in Compilers. (Compiler Design)

Regular expressions (REs) are used in many systems.

- E.g., UNIX, Linux, OS X,… a.*b.

Finite automata model electronic circuits. (sequential circuits (Mealy, Moore))


"Compiler"
program
FA, Reg Exp.
Lexical Analyser
Syntax Analyser
(Parser)
PDA
DPDA,
CFG
ToC

Why Context-Free Grammars?

- Context-free grammars (CFGs) are used to describe the syntax of essentially every modern programming language.
- Every modern complier uses CFG concepts to parse programs

Why Turing Machines?

- When developing solutions to real problems, we often confront the limitations of what software can do.
- Undecidable things – no program can do it 100% of the time with 100% accuracy.
- Intractable things – there are programs, but no fast programs.
- A course on Automata Theory and Formal Languages tells you about which problems can NEVER be solved.

What is Automata Theory?

- *Study of abstract computing devices, or "machines"*
- Automaton = an abstract computing device
  - Note: A "device" need not even be a physical hardware!
- A fundamental question in computer science:
  - Find out what different models of machines can do and cannot do
  - The *theory of computation*
- Computability vs. Complexity

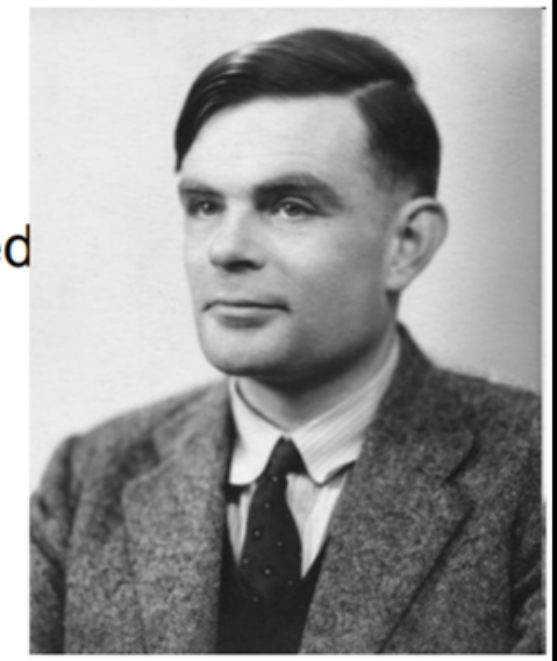
Introduction to Automata Theory

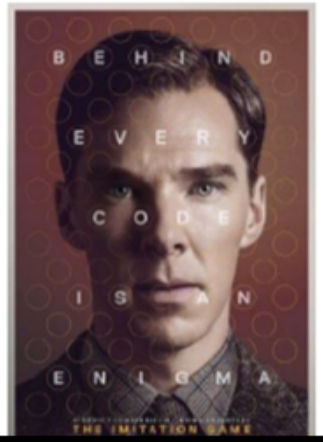
- Automata theory : **the study of abstract computing devices, or "machines"**
- Before computers (1930), **A. Turing** studied an abstract machine (*Turing machine*) that had all the capabilities of today' s computers (concerning what they could compute). His goal was to describe precisely the boundary between what a computing machine could do and what it could not do.
- Simpler kinds of machines (**finite automata**) were studied by a number of researchers and **useful for a variety of purposes**.
- Theoretical developments bear directly on what computer scientists do today
  - Finite automata, formal grammars: design/ construction of software
  - Turing machines: help us understand what we can expect from a software
  - Theory of intractable problems: are we likely to be able to write a program to solve a given problem? Or we should try an approximation, a heuristic...

(A pioneer of automata theory)

Alan Turing (1912-1954)

- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called ***Turing machines*** even before computers existed
- Heard of the Turing test?





Historical Perspectives

Alan Turing (1912-1954)

- Mathematician, logician, cryptanalyst, and founder of computer science
- First to formally define computation / algorithm
- Invented the Turing machine model
  - theoretical basis of all modern computers
- Investigated computational "universality"
- Introduced "definable" real numbers
- Proved undecidability of halting problem
- Originated oracles and the "Turing test"
- Pioneered artificial intelligence
- Anticipated neural networks
- Designed the Manchester Mark 1 (1948)
- Helped break the German Enigma cypher
- Turing Award was created in his honor

The Imitation Game movie

Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolve