CSL 2020: Data Structure and Algorithm(DSA)

Dr. Debasis Das
Department of CSE
IIT Jodhpur

Objectives

1. To introduce and practice the implementation of various data structures used for **indexing, searching, and sorting operations.**

2. To introduce basic mathematical techniques for algorithm analysis and design.

Learning Outcomes

1. Ability to design and implement appropriate data structures for indexing, searching, and sorting operations for real-world problems.
2. Designing of new algorithms using standard data structures.
3. Analyzing the time and space complexities of standard data structures and basic algorithms.

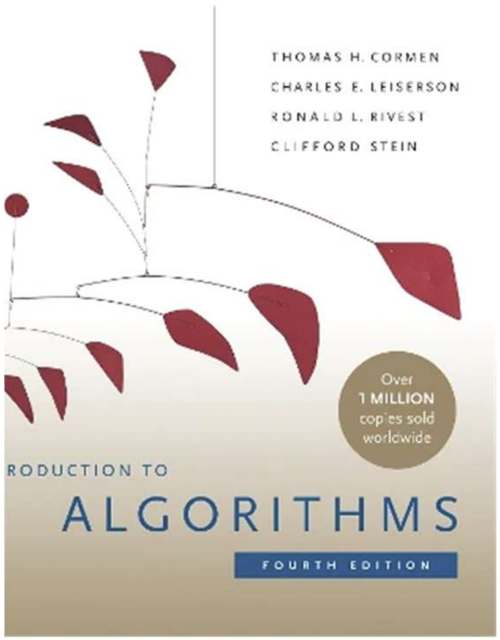
Contents

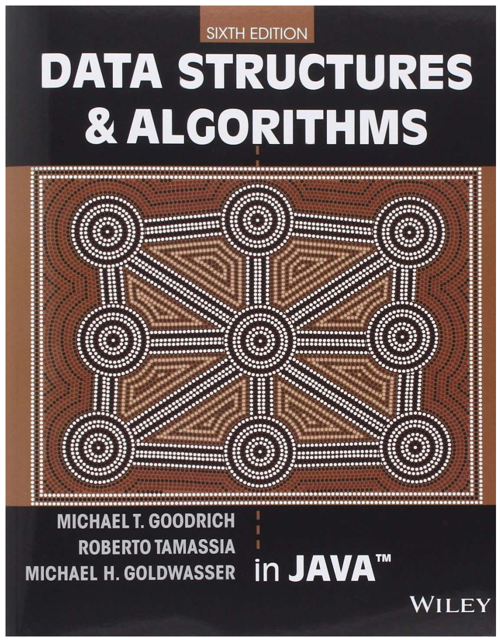
- ***Algorithm analysis and complexity:*** Big/little -Oh, Omega, Theta notation, Recurrence equations. (2 Lectures)
- ***Abstract data types:*** Linear data structures, Tree, Binary trees, Tree traversal, Applications. (7 Lectures)
- ***Search trees:*** Binary search trees, Balanced search trees, AVL trees, B-Trees. (5 Lectures)
- ***Heaps:*** Binary Heap, Heap order property and min/max heaps. (3 Lectures)
- ***Sets:*** Disjoint set ADT, Basic operations on Sets, Union/Find algorithm. (2 Lectures)
- ***Sorting algorithms:*** Bubble sort, Selection sort, Bucket sort, Insertion sort, Overview of Divide-andconquer, Quick sort, Merge sort. (6 Lectures)
- ***Hashing:*** Hash tables and operations, Hash function, Open and closed hashing, External and internal hashing, Collision resolving methods, Rehashing. (5 Lectures)
- ***Graph algorithms:*** Definitions, Branch and bound, Backtracking, Representation, Traversal, Shortest-path algorithms, Minimum Spanning Tree algorithm, Topological sorting. (8 Lectures)
- ***Greedy techniques and Dynamic programming*** (4 Lectures)

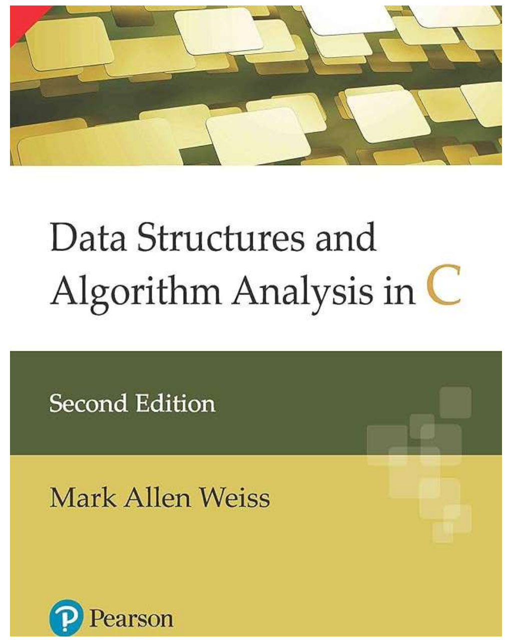
Laboratory

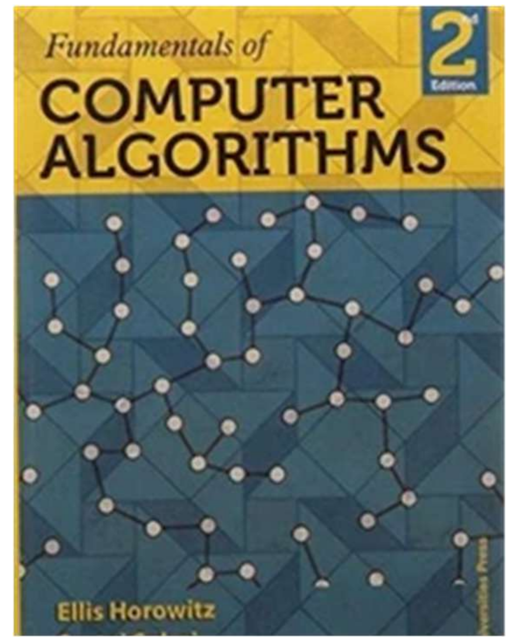
- 1. Implementation of data structures using C programming language.
- 2. Practically verifying and comparing run-time performance and asymptotic behavior of various data structures and related algorithms.
- 3. Applications of data structures from real-life scenarios.

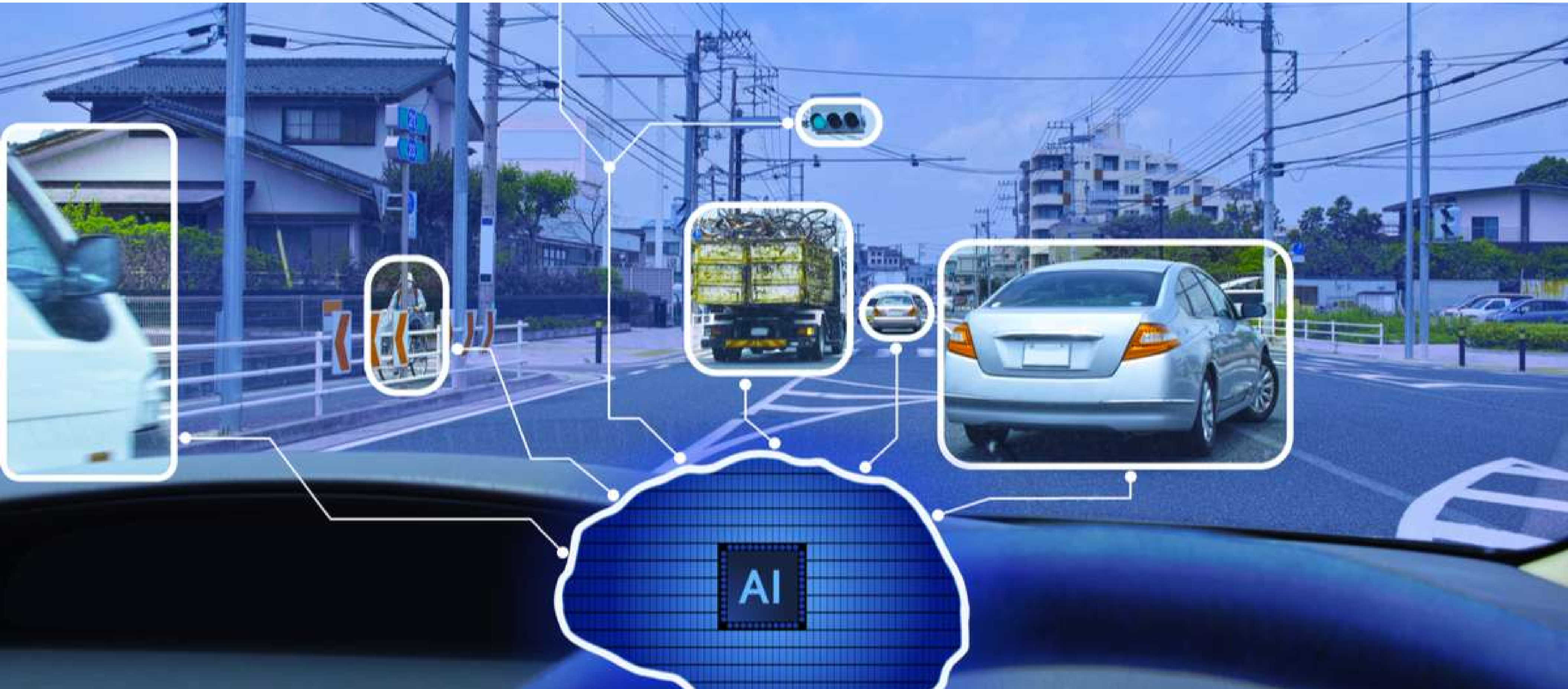
Text and Reference Books









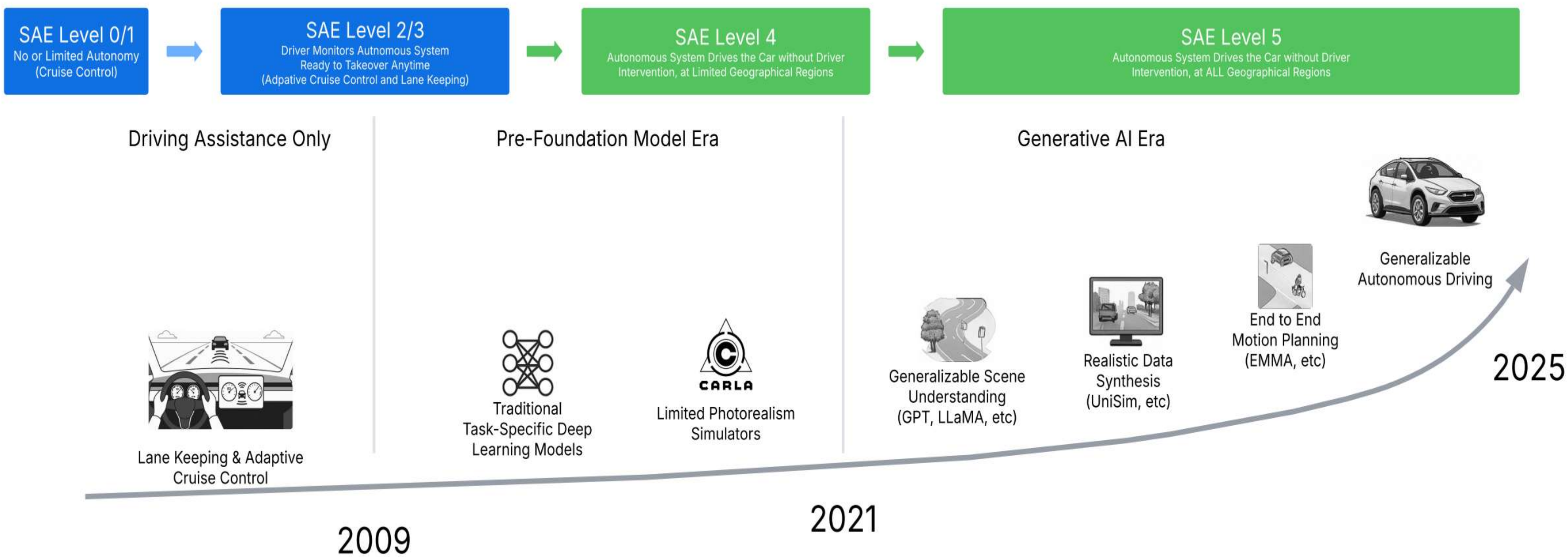


AI for Autonomous Driving

AI for Autonomous Driving

- Autonomous driving has long been envisioned as a transformative technology that promises to revolutionize transportation by significantly impacting **road safety, mobility, and logistics efficiency.**

Historical overview of autonomous driving development



Companies that are Developing L2/3 Driver-Assistance Systems

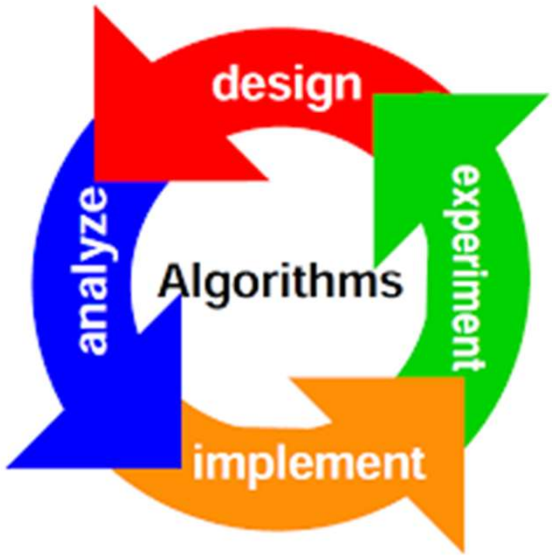
mobileye	BOSCH	Mercedes-Benz	gm	TESLA	HONDA	NVIDIA	TOYOTA	woven planet	And More
1993	1999	1999	2000	2014	2015	2015	2015	2021	

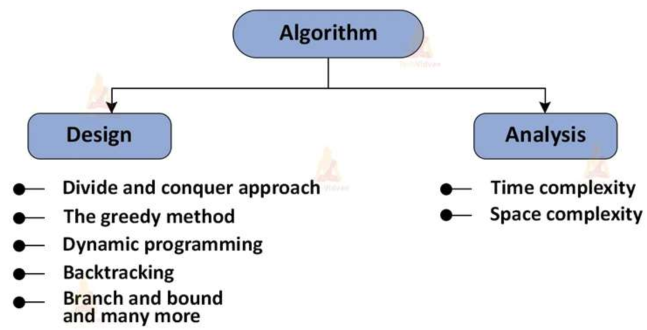
Companies that are Developing L2/3 Driver-Assistance Systems

Algorithm

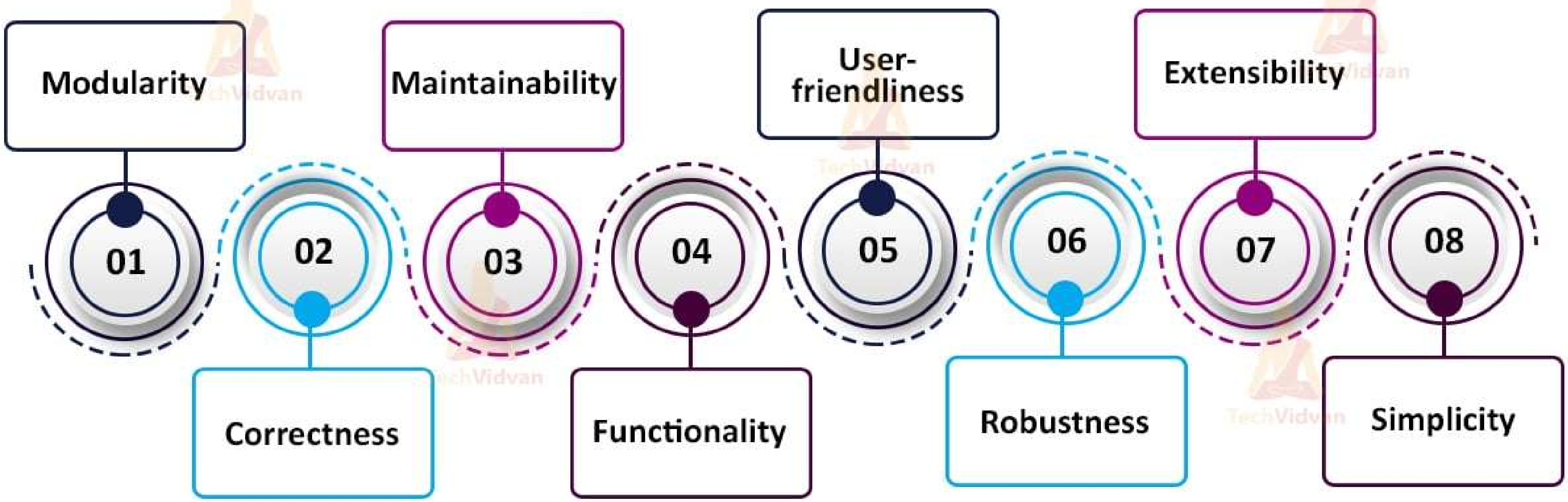
- Algorithms are step-by-step procedures used everywhere, from daily recipes to complex AI, to solve problems, automate tasks, and find the best solutions

Example: Powering search engines (Google), social media feeds (YouTube), GPS navigation (Waze), online recommendations, fraud detection, and digital assistants (Siri, Alexa), optimizing everything from data sorting and image processing to financial trading and robotics.

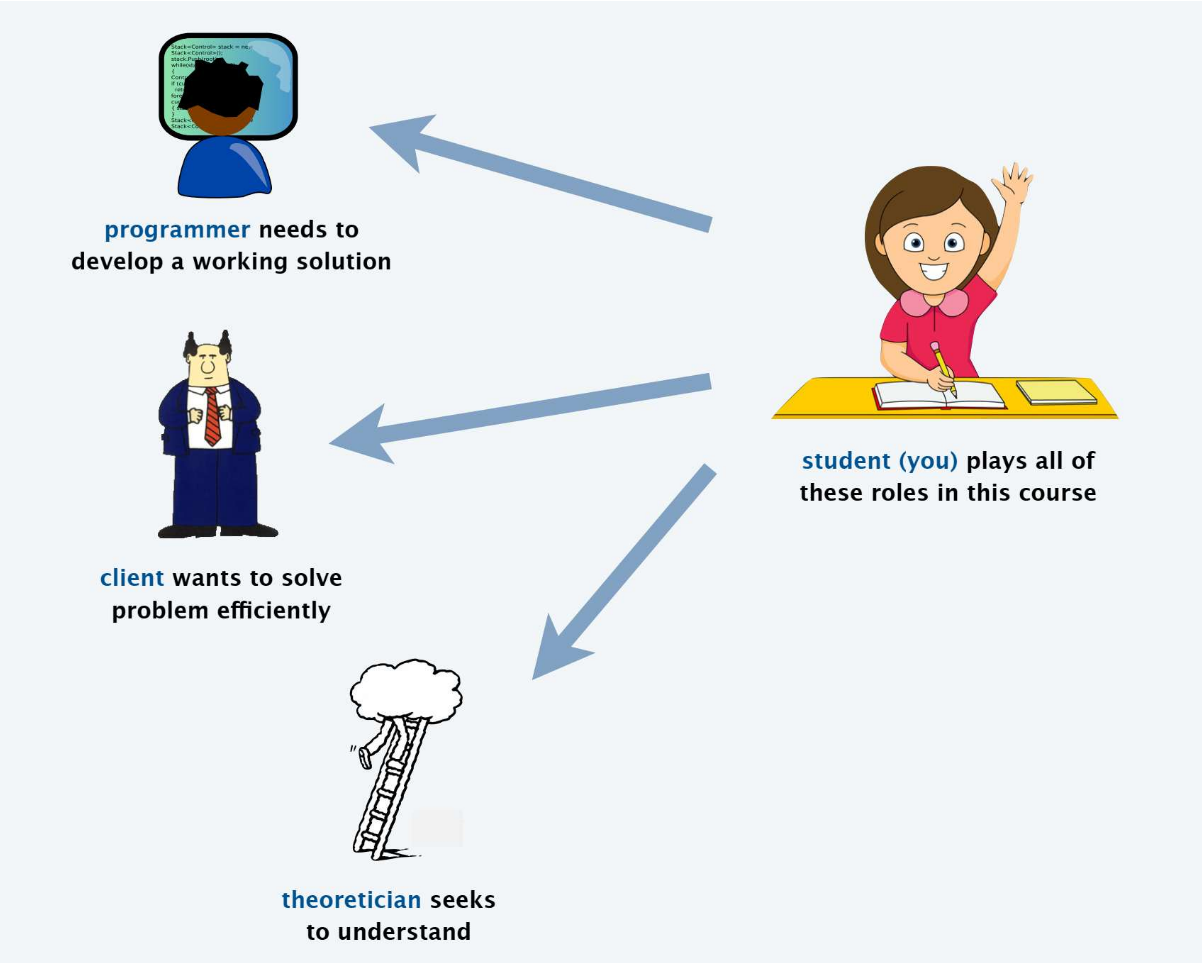




Factors of an Algorithm



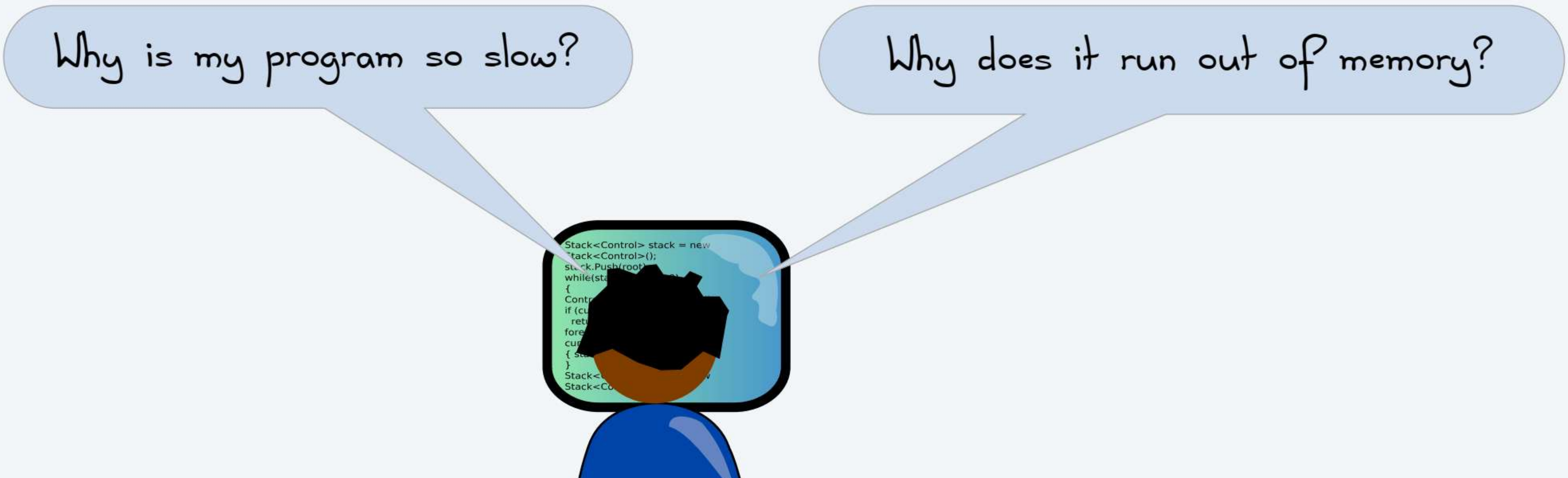
Different viewpoints



The challenge

Q1. Will my program be able to solve a large practical input?

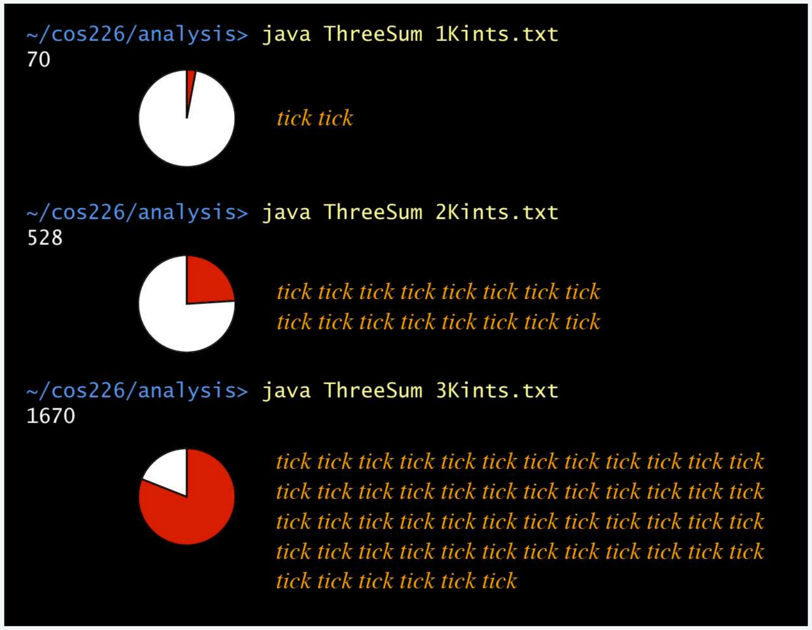
Q2. If not, how might I understand its performance characteristics so as to improve it?



Measuring the running time

Running time. Run the program for inputs of varying size; measure running time.

Observation. The running time $T(n)$ increases as a function of the input size n.



Measuring the running time

Running time. Run the program for inputs of varying size; measure running time.

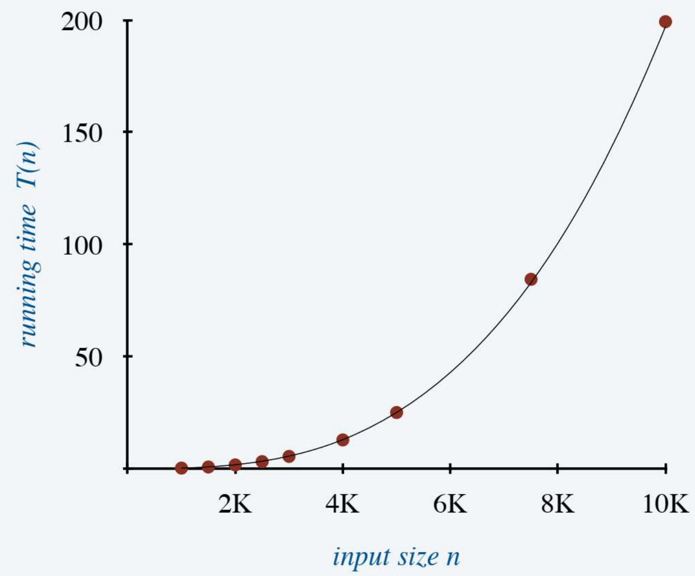
n	time (seconds) †
1,000	0.21
1,500	0.71
2,000	1.63
2,500	3.11
3,000	5.43
4,000	12.8
5,000	25.0
7,500	84.4
10,000	199.3

† *Apple M2 Pro with 32 GB memory running OpenJDK 11 on MacOS Ventura*

Data analysis: standard plot

Standard plot. Plot running time $T(n)$ vs. input size n.

n	time (seconds) †
1,000	0.21
1,500	0.71
2,000	1.63
2,500	3.11
3,000	5.43
4,000	12.8
5,000	25.0
7,500	84.4
10,000	199.3



Hypothesis. The running time obeys a power law: $T(n) = a \times n^b$ seconds.

Questions. How to validate hypothesis? How to estimate constants a and b ?

Order of growth

Hypothesis. Running times on different computers differ by a constant factor.

Note. That factor can be several orders of magnitude.

1970s
(VAX-11/780)

10,000× *faster*

2020s
(Macbook Pro M2)

Experimental algorithmics

System independent effects.

- Algorithm.
- Input data.

determines exponent b in power law $T(n) = a \times n^b$

System dependent effects.

- Hardware: CPU, memory, cache, …
- Software: compiler, interpreter, garbage collector, …
- System: operating system, network, other apps, …

determines leading coefficient a in power law $T(n) = a \times n^b$

M4

OpenJDK

22.04 LTS

Bad news. Sometimes difficult to get accurate measurements.

Context: the scientific method

Experimental algorithmics is an example of the scientific method.

Chemistry
(1 experiment)

Biology
(1 experiment)

Computer Science
(1 million experiments)

Physics
(1 experiment)

Good news. Experiments are easier and cheaper than other sciences.

Common order-of-growth classifications

order of growth	emoji	name	typical code framework	description	example	$T(2n) / T(n)$
$\Theta(1)$	😍	**constant**	`a = b + c;`	statement	*add two numbers*	1
$\Theta(\log n)$	😎	**logarithmic**	`for (int i = n; i > 0; i /= 2)` `{ ... }`	divide in half	*binary search*	~ 1
$\Theta(n)$	😁	**linear**	`for (int i = 0; i < n; i++)` `{ ... }`	single loop	*find the maximum*	2
$\Theta(n \log n)$	😀	**linearithmic**	*mergesort*	divide and conquer	*mergesort*	~ 2

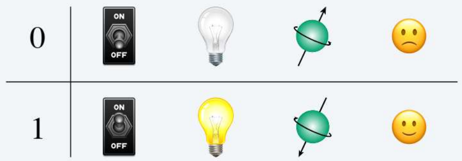
$\Theta(n^2)$	😕	**quadratic**	`for (int i = 0; i < n; i++)`
			`for (int j = 0; j < n; j++)`
			`{ ... }`
				double loop	*check all pairs*	4

$\Theta(n^3)$	☹️	**cubic**	`for (int i = 0; i < n; i++)`
			`for (int j = 0; j < n; j++)`
			`for (int k = 0; k < n; k++)`
			`{ ... }`
				triple loop	*check all triples*	8

$\Theta(2^n)$	👿	**exponential**	*towers of Hanoi*	exhaustive search	*check all subsets*	2^n

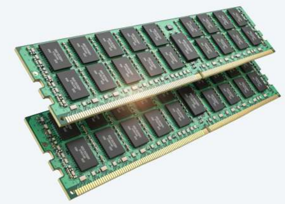
Memory basics

Bit. 0 or 1.



term	symbol	quantity
byte	B	8 bits
kilobyte	KB	1000 bytes
megabyte	MB	1000^2 bytes
gigabyte	GB	1000^3 bytes
terabyte	TB	1000^4 bytes

some define using powers of 2
(e.g., MB = 2^{20} *bytes)*



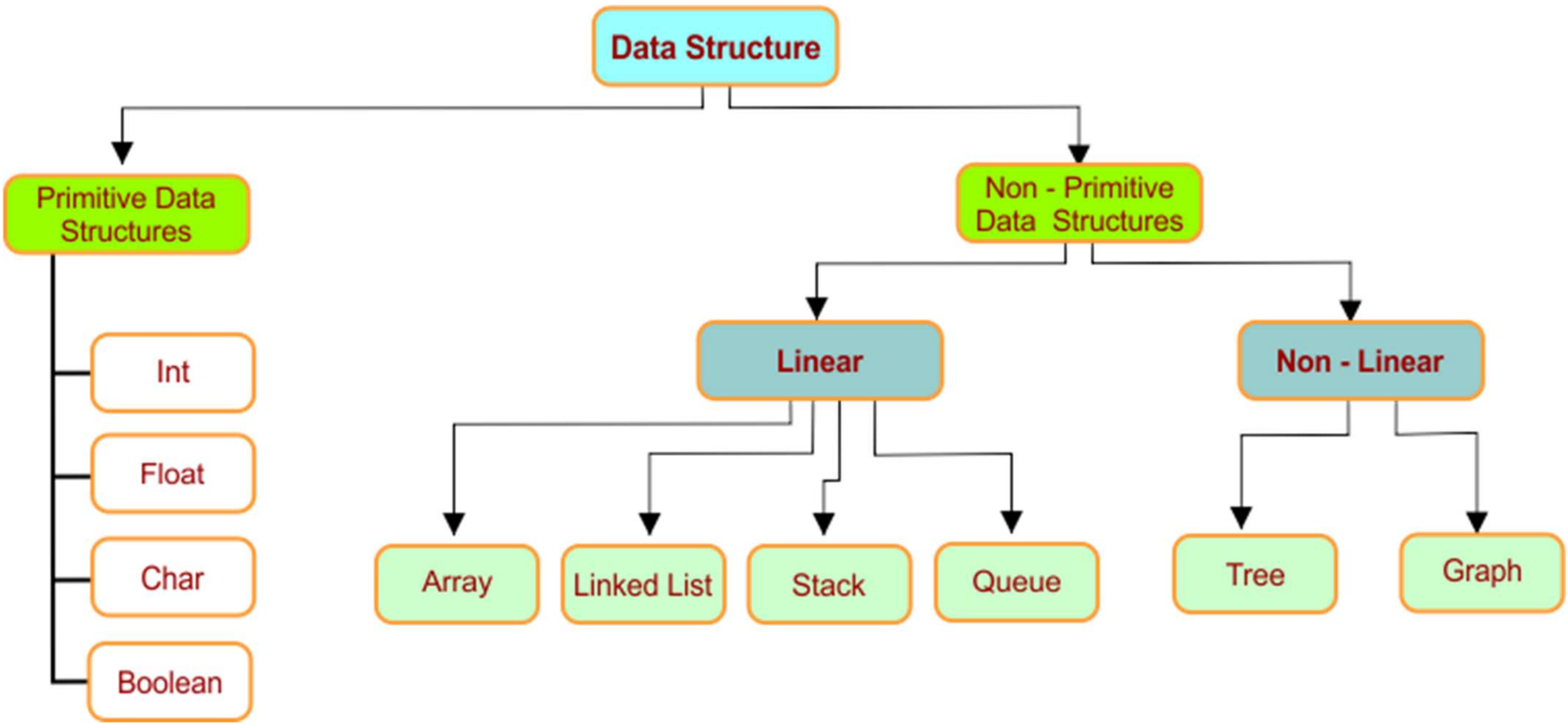
64-bit machine. We assume a 64-bit machine with 8-byte pointers.

some JVMs "compress" pointers
to 4 bytes to avoid this cost

Data Structure

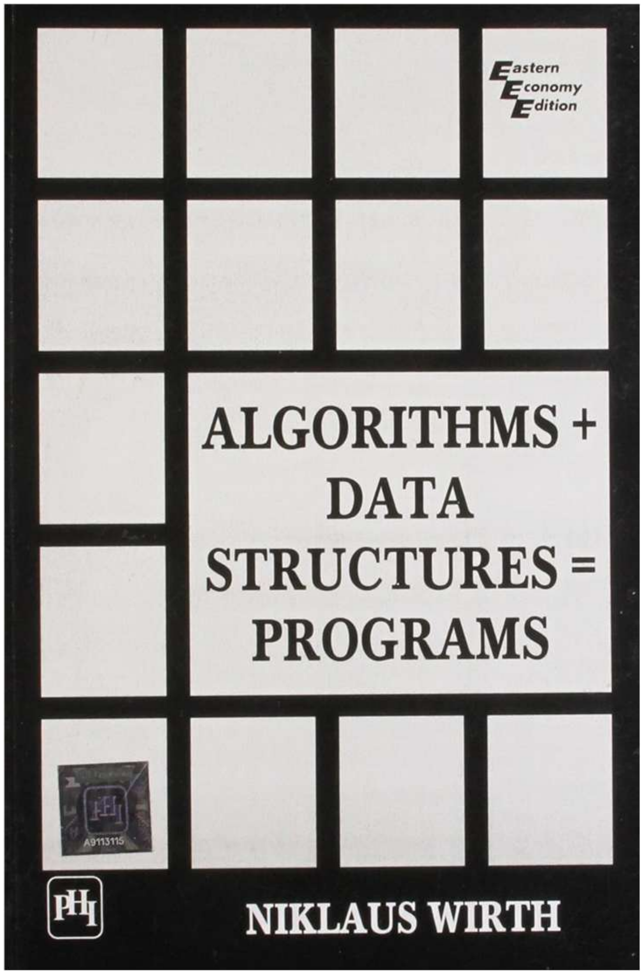
- A data structure is a specific way to organize, store, and manage data in a computer for efficient access and modification, acting as a framework for data elements and their relationships, crucial for building efficient algorithms and software.
- Key types include **linear** (arrays, linked lists, stacks, queues) for sequential data and **non-linear** (trees, graphs, hash tables) for hierarchical or complex relationships, chosen based on application needs like databases or compilers.
- Key Concepts
  - Organization: Arranges data in a systematic format, like a library's shelves, to make it understandable and usable.
  - Efficiency: Reduces time and space needed for tasks like searching, insertion, and deletion, vital for large datasets.
  - Abstract vs. Concrete: An Abstract Data Type (ADT) defines what data is and its operations (e.g., a list), while the data structure defines how it's physically stored (e.g., an array or linked list

Types of Data Structure



Program

- The relationship between an **algorithm**, a **data structure**, and a **program** can be summarized by the equation popularized by computer scientist Niklaus Wirth:
- **Algorithms + Data Structures = Programs**.
- In short, data structures organize the data that algorithms manipulate, and together, they are implemented in a programming language to create a functional program



Thank you!

Follow the work of VANET lab, at IIT Jodhpur

https://vanets-iitj.github.io/

Email: debasis@iitj.ac.in