LECTURE 1

Introduction to Data Structures

CSC211 · Algorithms and Data Structures

COMSATS University Islamabad, Lahore Campus

AGENDA

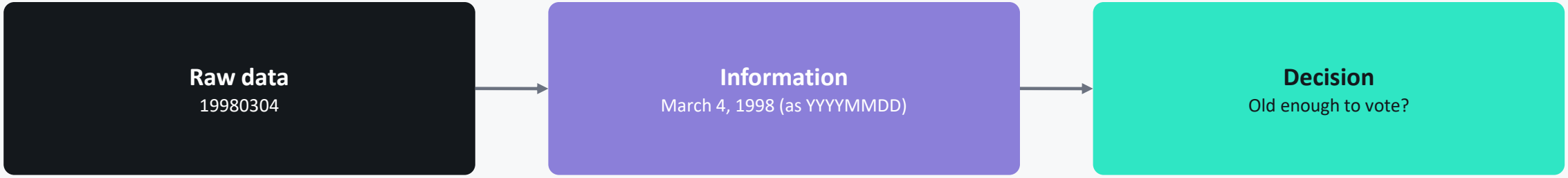
Learning Objectives

1. The difference between data and information
2. Data types, and how they relate to data structures
3. What a data structure actually is, formally
4. What an algorithm is, and how it depends on the data structure underneath it
5. Why data structures exist, and a timed example showing how much the choice matters
6. How to reason about picking the right structure for a real application

FOUNDATIONS

Data and Information

Data is raw, unprocessed facts with no meaning attached. **Information** is data that has been organized or interpreted so it becomes useful.



The same raw data means something different depending on context: 19980304 interpreted as a quantity is simply the number 19,980,304. A data structure's job is to hold the raw values AND preserve enough context (arrangement, field names, interface) that they can reliably become information again.

FOUNDATIONS

Data Types

A data type tells the compiler what KIND of data a value is, and therefore what operations are legal on it and how much memory it needs.

Primitive (built-in)

- int, double, char, bool
- Understood directly by the language
- Hold exactly one value

User-defined / composite

- struct, class, arrays
- Built out of primitive types
- Group values to represent something bigger

declaring primitive types

```
int studentCount = 320;
double gpa = 3.42;
char grade = 'A';
bool isEnrolled = true;

// wrong type choice can silently
// produce WRONG answers (overflow)
```

FOUNDATIONS

Choosing the Wrong Type Is a Bug, Not a Style Slip

type_choice_matters.cpp (excerpt)

```
int bigStudents = 500000;
int bigRevenue =
    bigStudents * revenuePerStudent;
  // overflows silently!

long long bigRevenueCorrect =
    (long long)bigStudents
    * revenuePerStudent;
  // does not overflow
```

1,025,163,520

int result — WRONG (silently overflowed)

22,500,000,000

long long result — correct

A 32-bit int can only hold ~2.1 billion — no crash, no warning, just a confidently wrong number.

CORE CONCEPT

What a Data Structure Actually Is

A data structure is a **data type** composed of the relationships between data elements, together with the **operations** that can be applied to that arrangement.

1

The Arrangement

HOW the pieces of data relate to each other — in a row, in a hierarchy, connected in a network.

2

The Operations

WHAT you're allowed to do with that arrangement — add, remove, look up, walk through.

Choosing a data structure is choosing WHICH operations you want to be fast — you can rarely make everything fast at once. That trade-off is the central theme of this entire course.

CORE CONCEPT

What an Algorithm Is

An algorithm is a finite, well-defined, step-by-step procedure for solving a problem — a recipe the computer can follow with no ambiguity.

F

Finite

Must terminate after a finite number of steps.

W

Well-defined

Every step is precise and unambiguous.

E

Effective

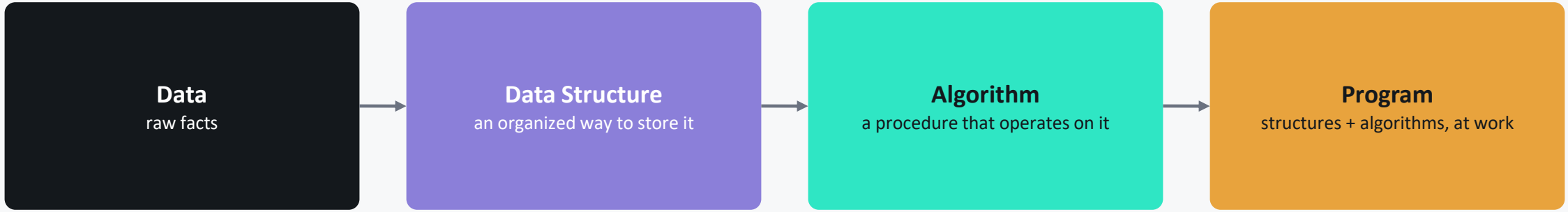
Every step is something the computer can actually do.

find_largest.cpp (excerpt)

```
int findLargest(const vector<int>& numbers) {
    int largest = numbers[0];
    for (int i = 1; i < numbers.size(); i++)
        if (numbers[i] > largest) largest = numbers[i];
    return largest;
}
```

THE BIG PICTURE

Data, Data Structures, and Algorithms



“Algorithms + Data Structures = Programs”

— Niklaus Wirth, creator of Pascal

You cannot design a good algorithm without knowing how its data is organized, and you cannot pick a good data structure without knowing what operations your algorithms need. The two are designed together, never in isolation.

WHY THEY MATTER

The Need for Data Structures

E

Efficiency

The right structure turns an operation taking seconds on millions of records into one taking microseconds.

R

Reusability

A structure implemented once (a stack, queue, tree) is reused across many unrelated programs.

A

Abstraction

You use it (push, pop, insert, search) without needing to know exactly how it stores things.

M

Manageable Complexity

Organizing related data together makes large programs easier to reason about.

PROOF, NOT JUST ASSERTION

Finding One Student Among 200,000

Same 200,000 IDs, stored two ways — a plain list scanned one entry at a time, vs. a hash-based lookup (unordered_set). Timed in the same run, same machine:

887 µs

Linear scan — checked up to 200,000 records

< 1 µs

Hashed lookup — too fast for the clock to register

Same data. Same question. The linear scan is reliably tens of times slower, every single run — purely because of HOW the same 200,000 numbers were organized in memory. That gap is, almost word for word, the entire argument for why this course exists.

PROOF, NOT JUST ASSERTION

The Gap Widens as Data Grows

Records (n)	Linear scan: worst-case comparisons	Hashed lookup: typical comparisons
1,000	1,000	~1
200,000	200,000	~1
10,000,000	10,000,000	~1

At a million records, “check every one” and “jump straight to it” aren't a little different — they're operating in entirely different universes of cost. Lecture 3 gives this precise mathematical language (O(n) vs. O(1)) — the intuition starts right here: the data structure you choose determines which of those two columns your application lives in.

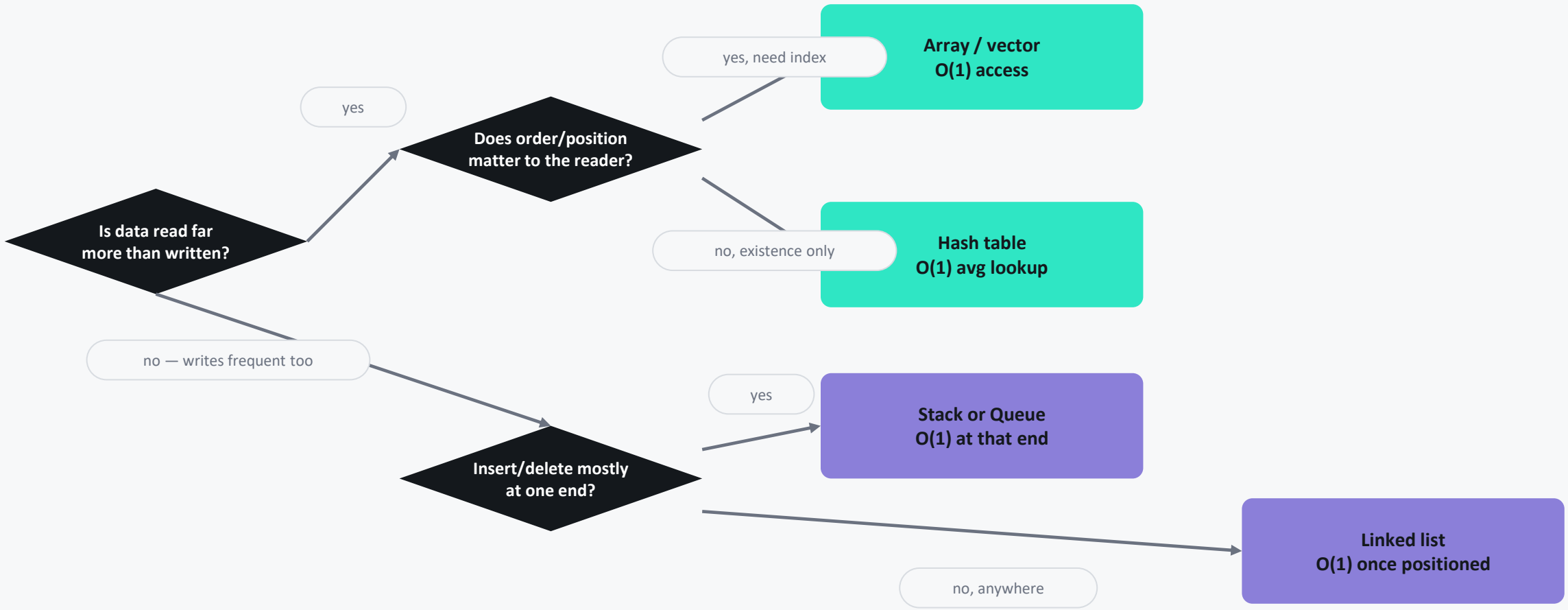
EVERYDAY EXAMPLES

Applications of Data Structures

Structure	Where you already use it
Array	A spreadsheet row, exam scores, pixels in an image
Stack	Browser “back” button, undo/redo, function calls
Queue	A printer job queue, a checkout line
Linked List	A playlist's “next song,” steppable browser history
Tree	A file system's folders, an org chart, site navigation
Graph	Google Maps road network, social-network friend links
Hash Table	A contacts app's instant name lookup, a spell-checker

PUTTING IT TOGETHER

Which Structure Fits First?



Simplified on purpose — real decisions also weigh memory limits and how often data changes shape. But it captures the first question this course keeps returning to: what operation does the app do most, and which structure makes exactly that operation cheap?

KEY TAKEAWAYS

Key Takeaways

- Data is raw and meaningless on its own; information is data given context and interpretation.
- A data structure is an arrangement of data plus the operations defined on it — the arrangement decides which operations are fast.
- An algorithm is finite, well-defined, and effective — and always designed together with the data structure it operates on.
- Data + Data Structure + Algorithm = Program: each layer builds on the one before it.
- A timed measurement showed a linear scan reliably tens of times slower than a hashed lookup over the same 200,000 records — proof the structure choice can be the entire performance story.