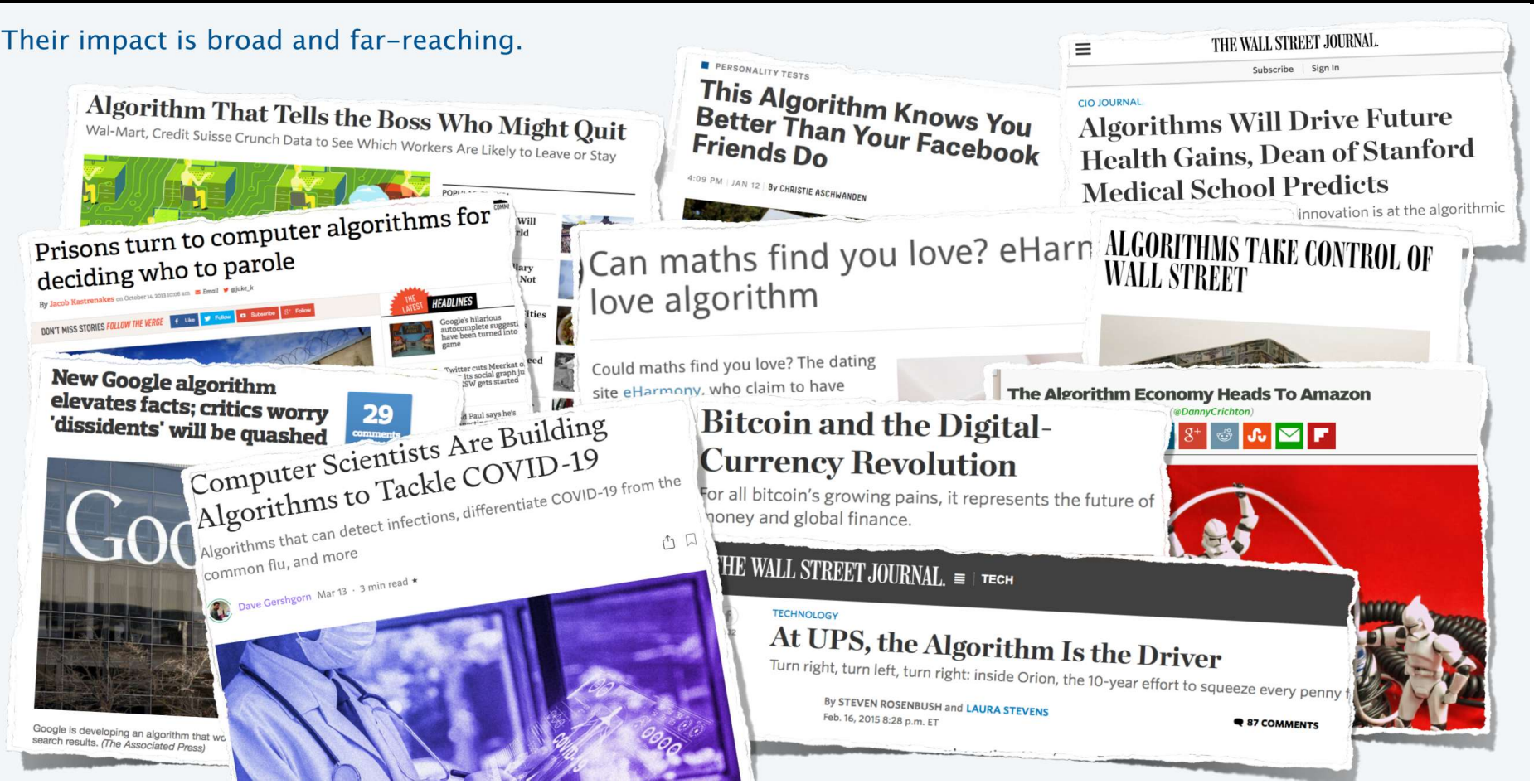
Algorithm Analysis and Complexity

Dr. Debasis Das
Department of CSE
IIT Jodhpur

Why study algorithms and data structures?

Their impact is broad and far-reaching.



Why study algorithms and data structures?

Why study algorithms and data structures?

- Their impact is broad and far-reaching.
- To become a proficient programmer.
- For intellectual stimulation.
- For fun and profit.

What is Data?

What is data?

Things are not data, but information about them is data.

Example 1.1

Age of people, height of trees, price of stocks, and number of likes.

Data is big!

We are living in the age of big data!

*Image is from the Internet.

Exercise 1.1

1. *Estimate the number of messages exchanged for status level in Whatsapp.*
2. *How much text data was used to train ChatGPT?*

We need to work on data

We need to work on data

We process data to solve our problems.

Example 1.2

1. *Predict the weather*
2. *Find a webpage*
3. *Recognize fingerprint*

Disorganized data will need a lot of time to process.

Exercise 1.2

How much time do we need to find an element in an array?

Problems

Problems

A problem is a pair of input specification and output specification

Example 1.3

The problem of search consists of the following specifications

- *Input specification: an array S of elements and an element e*
- *Output specification: position of e in S if exists. If not found, return -1.*

Output specifications refer to the variables in the input specifications

Algorithms

Algorithms

An algorithm solves a given problem.

- Input $\in$ Input specifications
- Output $\in$ Output specifications

Note: there can be many algorithms to solve a problem.

Exercise 1.3

1. *What is an algorithm?*
2. *How is it different from a program?*

Introduction to Algorithms

- *An algorithm is a set of instructions for accomplishing a task*.

Example: an algorithm for search

Example 1.4

```
int search( int* S, int n, int e) {
  // n is the length of the array S
  // We are looking for element e in S
  for( int i=0; i < n; i++ ) {
    if( S[i] == e ) {
      return i;
    }
  }
  return -1; // Not found
}
```

Exercise 1.4

How much time will it take to run the above algorithm if e is not in S?

Data Structure

Structured data helps us solve problems faster

We can exploit the structure to design efficient algorithms to solve our problems.

The goal of this course!

Example: search on well-structured data

Example 1.6

Let us consider the problem of search consisting of the following specifications

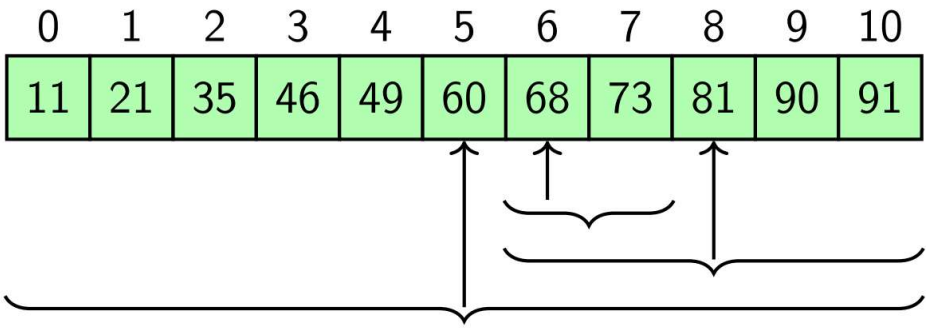
- *Input specification: a non-decreasing array S and an element e*
- *Output specification: Position of e in S. If not found, return -1.*

Example: search on well-structured data

Let us see how can we exploit the structured data!

Let us try to search 68 in the following array.

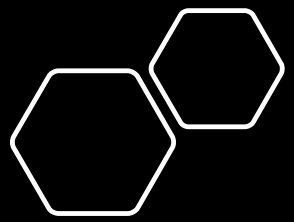
- Look at the middle point of the array.
- Since the value at the middle point is less than 68, we search only in the upper half of the array.
- We have halved our search space.



A better search

Example 1.7

```
int BinarySearch(int* S, int n, int e
  // S is a sorted array
  int first = 0, last = n;
  int mid = (first + last) / 2;
  while (first < last) {
    if (S[mid] == e) return mid;
    if (S[mid] > e) {
      last = mid;
    } else {
      first = mid + 1;
    }
    mid = (first + last) / 2;
  }
  return -1;
```



How much resource does an algorithm need?

There can be many algorithms to solve a problem.

Some are good and some are bad.

Good algorithms are efficient in

- time and
- space.

Our method of measuring time is cumbersome and machine-dependent.

We need approximate counting that is machine independent.

Input size

An algorithm may have different running times for different inputs.

How do we think about comparing algorithms?

We define the rough size of the input, usually in terms of important parameters of input.

Example 1.8

In the problem of search, we say that the number of elements in the array is the input size.

Please note that the size of individual elements is not considered. *(why?)*

Best/Average/Worst case

For a given size of inputs, we may further make the following distinction.

1. Best case: Shortest running time for some input.
2. Worst case: Worst running time for some input.
3. Average case: Average running time on all the inputs of the given size.

Exercise 1.6

How can we modify almost any algorithm to have a good best-case running time?

Example: Best/Average/Worst case

Example 1.9

```
int BinarySearch(int* S, int n, int e){
  // S is a sorted array
  int first = 0, last = n;
  int mid = (first + last) / 2;
  while (first < last) {
    if (S[mid] == e) return mid;
    if (S[mid] > e) {
      last = mid;
    } else {
      first = mid + 1;
    }
    mid = (first + last) / 2;
  }
  return -1;
}
```

In BinarySearch, let $n = 2^{k-1}$.

1. Best case: $e == S[n/2]$
   $T_{Read} + 6T_{Arith} + T_{return}$,
2. Worst case:$e \notin S$
   we have seen the worst case.
3. Average case: $\approx$ Worst case
   Most often loop will iterate k times.(why?)

Commentary: Analyzing the average case is hard. We will mostly focus on worst-case analysis. For some important algorithms, we will do an average time analysis.

Asymptotic behavior

For short inputs, an algorithm may use a shortcut for better running time.

To avoid such false comparisons, we look at the behavior of the algorithm in limit.

Ignore hardware-specific details

- Round numbers $100000000000001 \approx 100000000000000$
- Ignore coefficients $3kT_{Arith} \approx k$

Big-O notation: approximate measure

Definition 1.1

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in O(g(n))$ if there are c and n_0 such that

$$f(n) \leq cg(n) \qquad \text{for all } n \geq n_0.$$

- In limit, $cg(n)$ will dominate $f(n)$
- We say $f(n)$ is $O(g(n))$

Exercise 1.7

Which of the following are the true statements?

- $5n + 8 \in O(n)$
- $5n + 8 \in O(n^2)$
- $5n^2 + 8 \in O(n)$
- $n^2 + n \in O(n^2)$
- $500000000000000000000000n^2 \in O(n^2)$
- $50n^2 logn + 60n^2 \in O(n^2 logn)$

Example: Big-O of the worst case of BinarySearch

Example 1.10

In `BinarySearch`, let $n = 2^{k-1}$.

1. Worst case:$e \notin S$
   $kT_{Read} + (6k+5)T_{Arith} + (3k+1)T_{jump} + T_{return} \in O(k)$

   Since $k = log\ n + 1$, therefore $k \in O(log\ n)$

   We may also say *BinarySearch* is $O(log\ n)$.

   Therefore, the worst-case running time of *BinarySearch* is $O(log\ n)$.

Exercise 1.8

Prove that $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

What does Big-O says?

Expresses the approximate number of operations executed by the program as a function of input size

Hierarchy of algorithms

- $O(log\ n)$ algorithm is better than $O(n)$
- We say $O(log\ n) < O(n) < O(n^2) < O(2^n)$

May hide large constants!!

Θ-Notation

Definition 1.2 (Tight bound)

Let f and g be functions $\mathbb{N} \to \mathbb{N}$. We say $f(n) \in \Theta(g(n))$ if there are c_1, c_2, and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \qquad \text{for all } n \geq n_0.$$

There are more variations of the above definition. Please look at the end.

Names of complexity classes

- Constant: $O(1)$
- Logarithmic: $O(logn)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Polynomial : $O(n^k)$ for some given k
- Exponential : $O(2^n)$

Binary Search

- Data: List
- Requirement: List is alpha sorted
- Most efficient algorithm: Binary Search
- Analogy: Searching through an address/phone book

How to create programs

- Requirements
- Analysis: bottom-up vs. top-down
- Design: data objects and operations
- Refinement and Coding
- Verification
  - Program Proving
  - Testing
  - Debugging

Algorithm

- Definition
  An *algorithm* is a finite set of instructions that accomplishes a particular task.
- Criteria
  - input
  - output
  - definiteness: clear and unambiguous
  - finiteness: terminate after a finite number of steps
  - effectiveness: instruction is basic enough to be carried out

Data Type

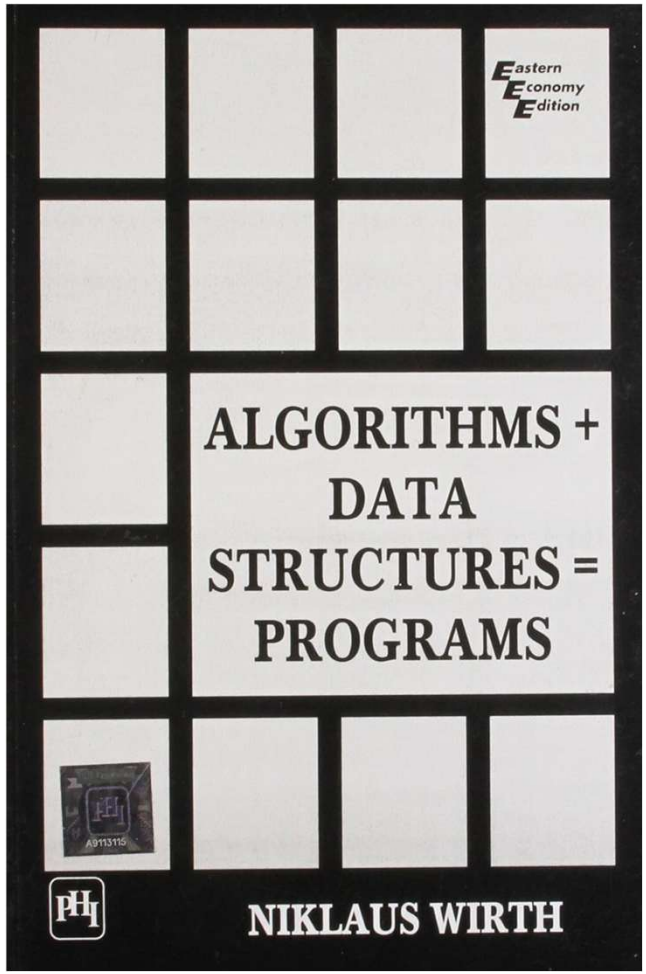
- Data Type
  A *data type* is a collection of *objects* and a set of *operations* that act on those objects.
- Abstract Data Type
  An *abstract data type(ADT)* is a data type that is organized in such a way that the specification of the objects and the operations on the objects is separated from the representation of the objects and the implementation of the operations.

Specification vs. Implementation

- Operation specification
  - function name
  - the types of arguments
  - the type of the results
- Implementation independent

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



Non-resources (generative AI)

Pedagogical rationale.

- Build a strong foundation in programming.
- Strengthen critical-thinking and problem-solving skills.
- Prepare to use GenAI effectively and responsibly in the future.
- Crush exams and technical job interviews.

Thank you!

Follow the work of VANET lab, at IIT Jodhpur

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

Email: debasis@iitj.ac.in