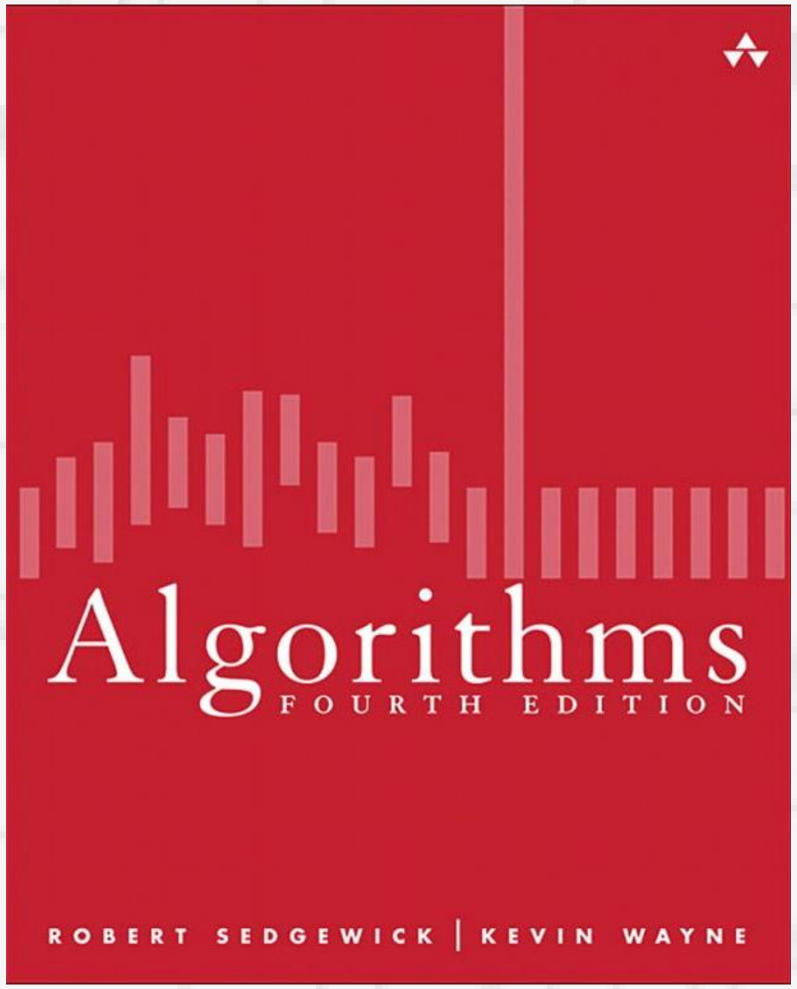
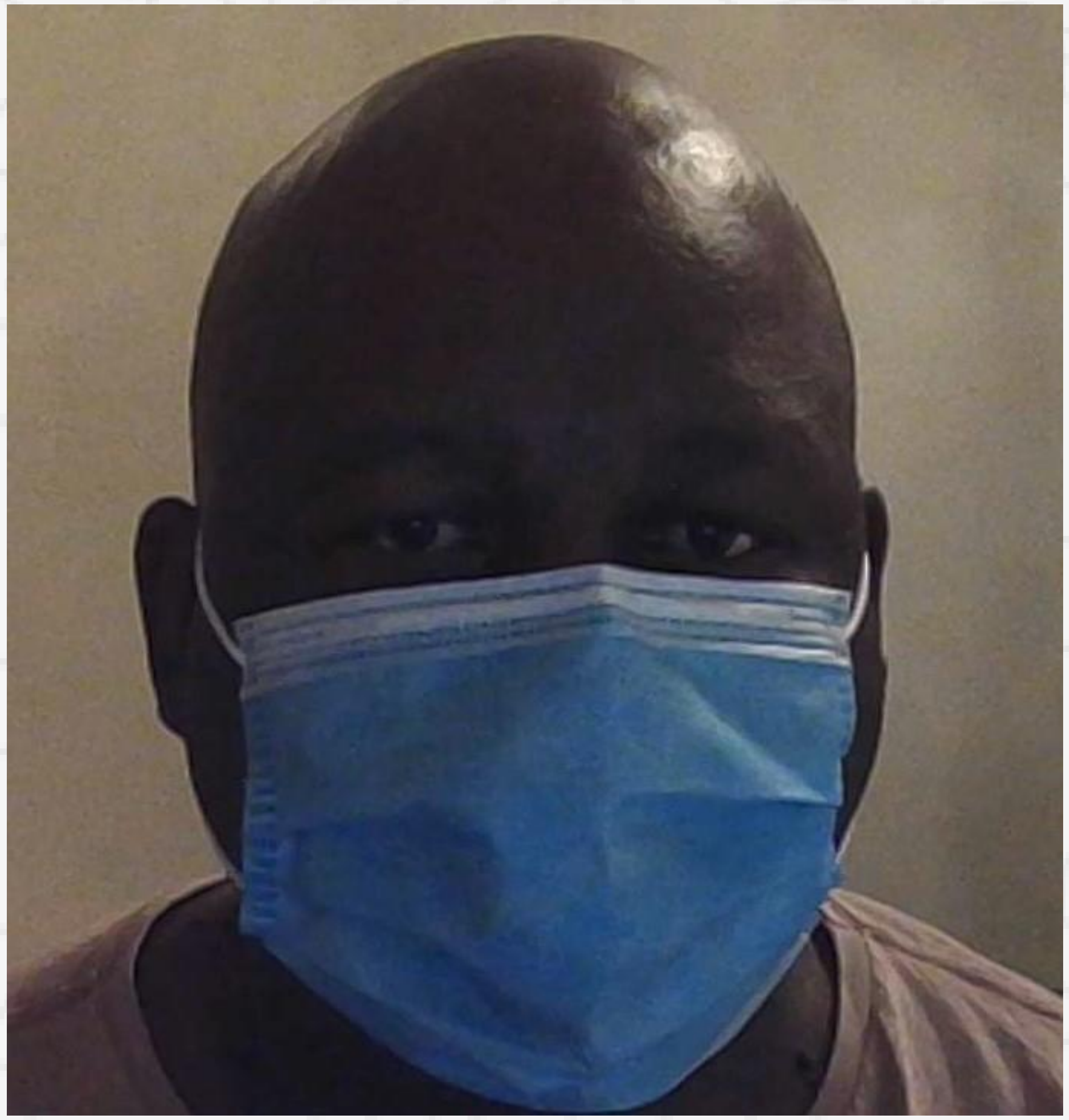
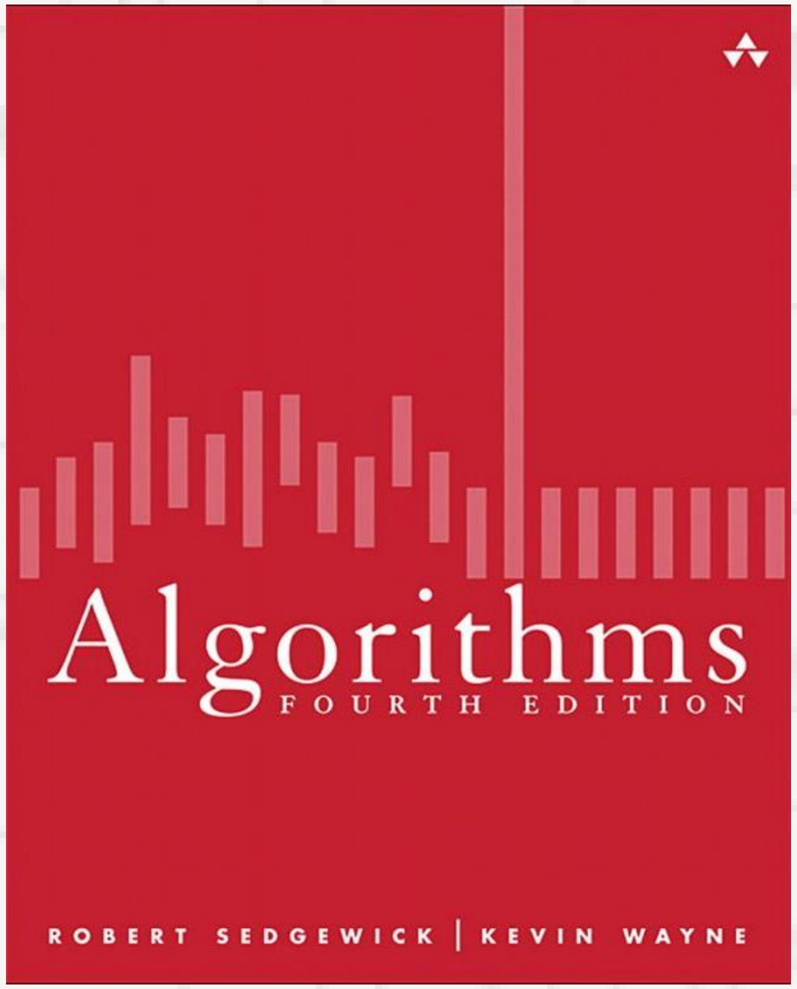
Stay safe

Wash hands with soap & water
Sanitize
Social distance



https://algs4.cs.princeton.edu



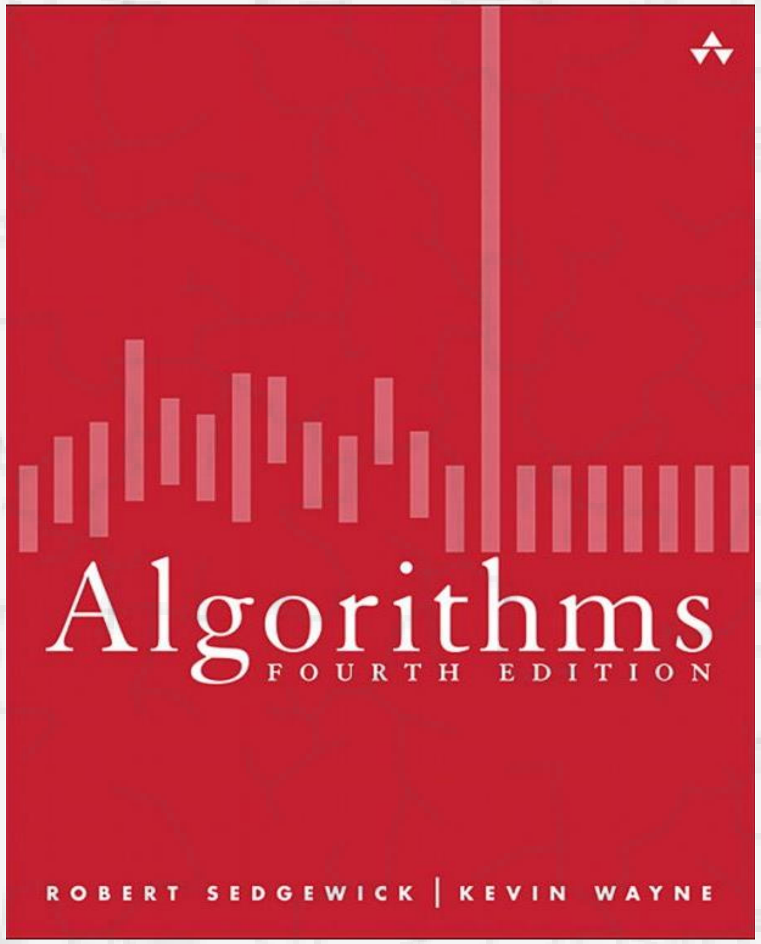


https://algs4.cs.princeton.edu

COMPUTER SCIENCE

CSI323
ALGORITHMS

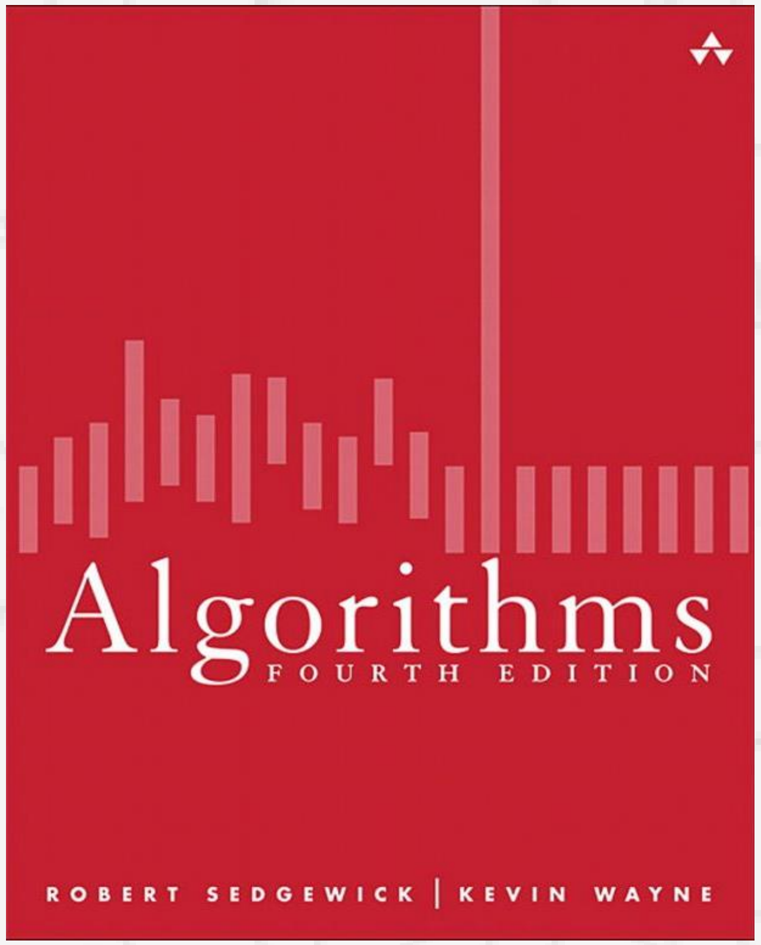
CSI323 - Algorithms TALLMAN NKGAU



https://algs4.cs.princeton.edu

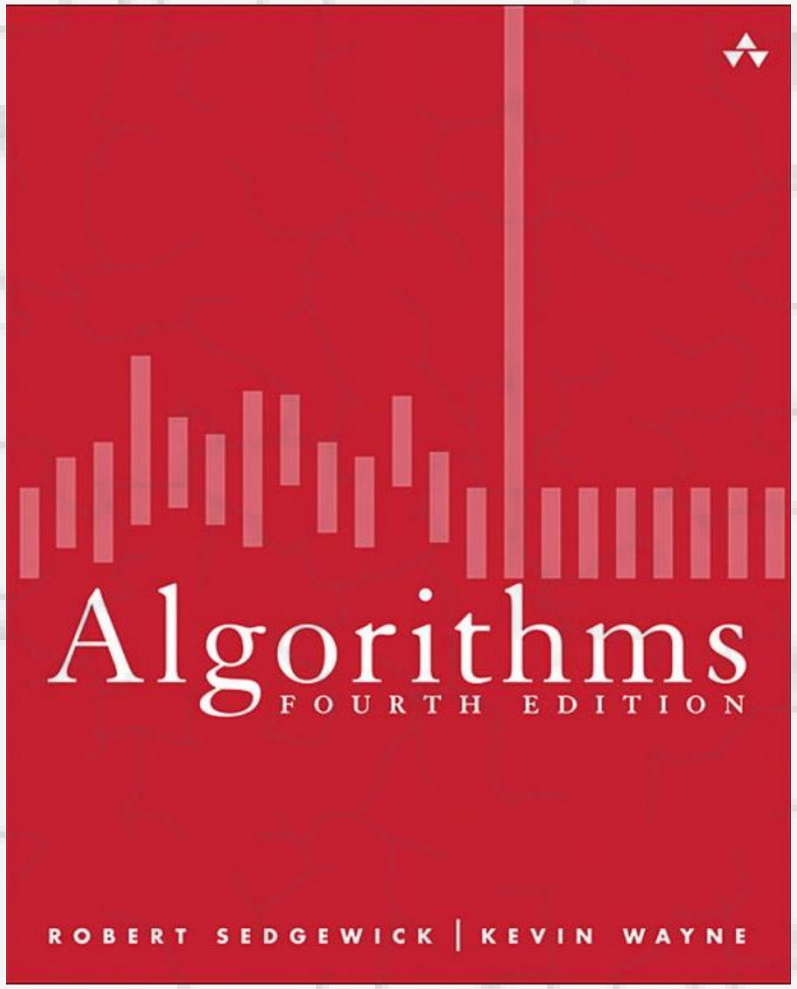
LECTURE 1

- *Course info*
- *Course learning outcomes*
- *Course assessment criteria*
- *Course rules*
- *What is an algorithm?*
- *Describing algorithms*



https://algs4.cs.princeton.edu

ACKNOWLEDGEMENT:
SLIDES' THEME & CONTENT FROM
ROBERT SEDGEWICK | KEVIN WAYNE



https://algs4.cs.princeton.edu

Lecture 1

- ▸ *Course info*
- ▸ *Course learning outcomes*
- ▸ *Course assessment criteria*
- ▸ *Course rules*
- ▸ *What is an algorithm?*
- ▸ *Describing algorithms*

Course Info

Course Lecturer

Tallman Nkgau
Office: 247/285
Phone: 3552260
Email: nkgautz@ub.ac.bw

Lectures

- **Monday: 1000 – 1050, 232-005**
- **Tuesday: 0900 – 0950, 235-027**

Lab

- **Thursdays: 0700 – 0900**
- **Location: 247-293, 247-294, 247-295, 247-296, 247-297**

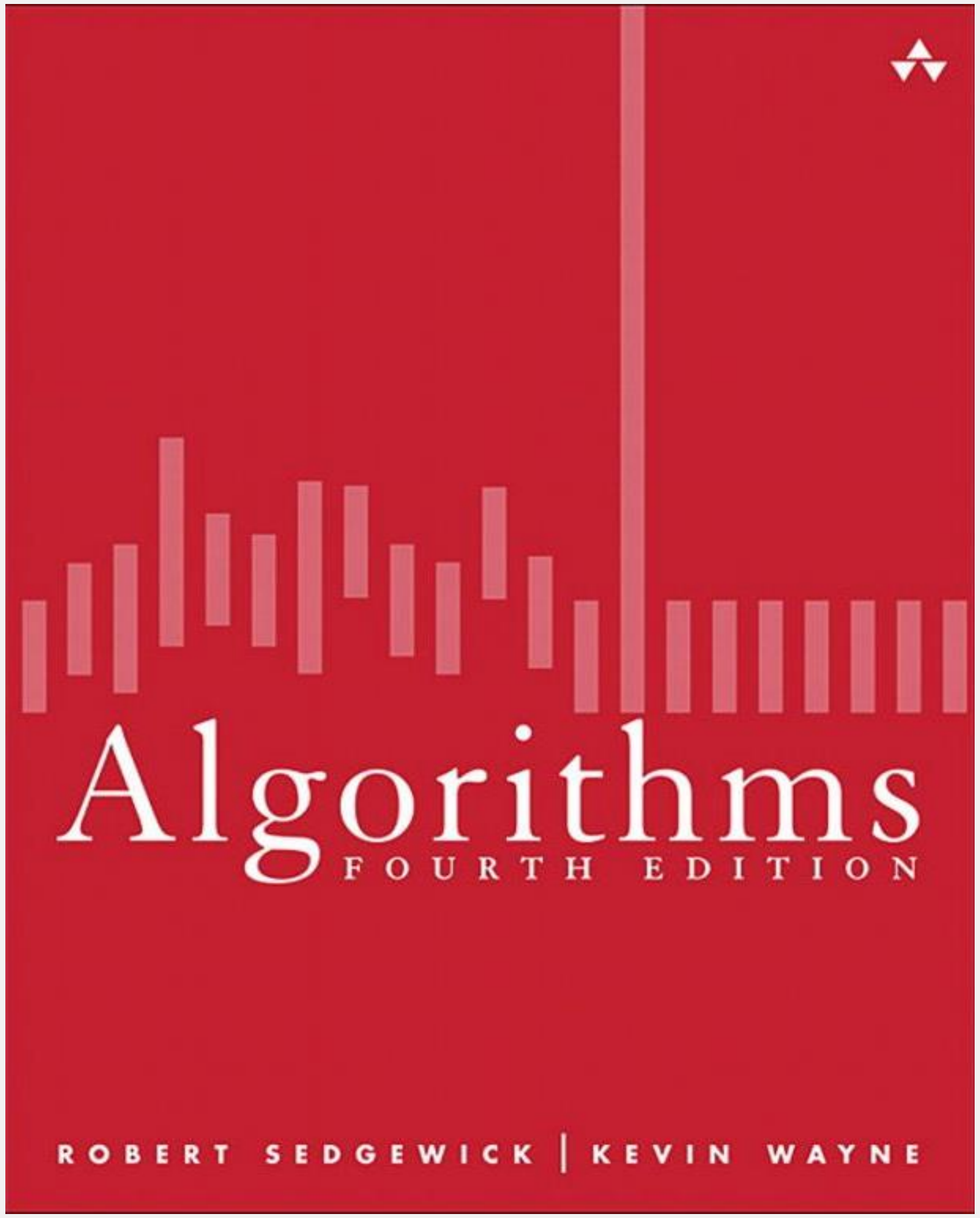
Course title: Algorithms

Course code: CSI323

Pre-requisite: CSI247 Data Structures ($\geq 50\%$)

Credits: 3

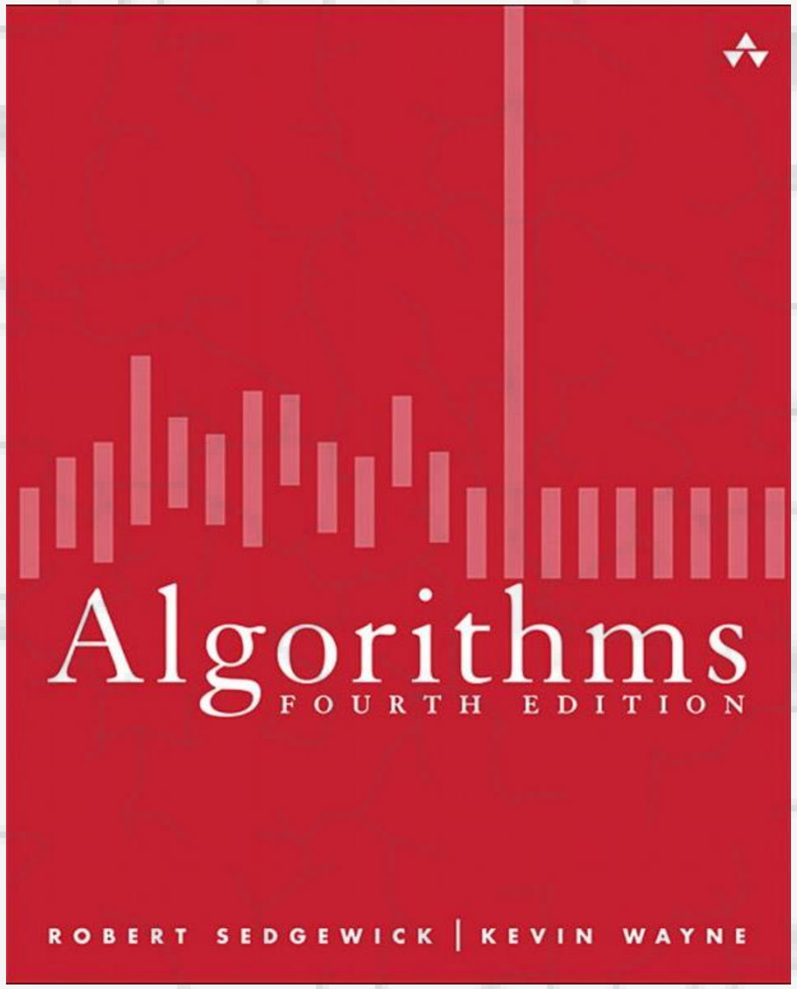
Course textbook



- Algorithms, 4th Edition,
- Authors: R Sedgewick & K Wayne
- Publisher: Addison Wesley
- Book website:

  https://algs4.cs.princeton.edu

https://algs4.cs.princeton.edu



https://algs4.cs.princeton.edu

Lecture 1

- *Course info*
- *Course learning outcomes*
- *Course assessment criteria*
- *Course rules*
- *What is an algorithm?*
- *Describing algorithms*

Course learning outcomes

Course description

Efficiency is currency. Efficient algorithms are behind all successful technology driven organizations. This course introduces students to the analysis and design of efficient computer algorithms. It will also discuss techniques for dealing with problems whose solutions are difficult to compute efficiently.

.

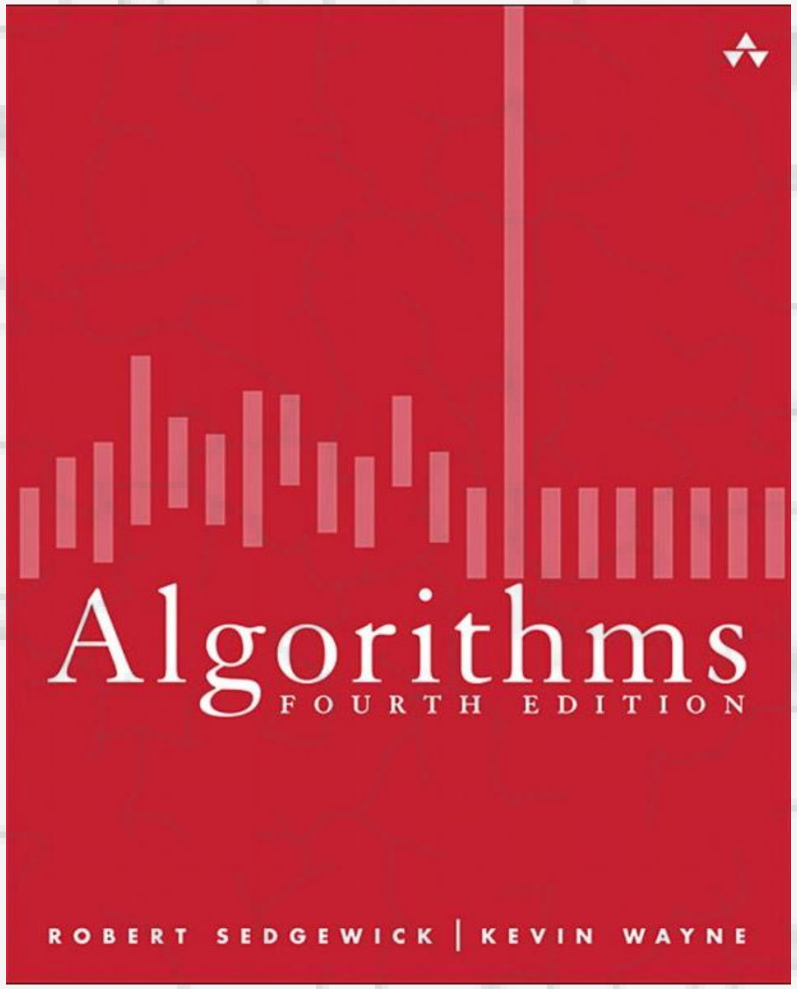
Learning outcomes

At the end of the course, you should be able to

1. Describe models of computation used in algorithm analysis.
2. Use big-O notation to show asymptotic upper bounds on time and space complexity of algorithms.
3. Use recurrence relations to determine the time complexity of recursively defined algorithms by solve elementary recurrence relations.
4. Describe the greedy approach, divide and conquer, dynamic programming, and recursive backtracking algorithm design patterns.
5. Use elementary data structures in designing algorithms

Course learning outcomes

6. Use appropriate techniques (e.g., greedy approach, divide-and-conquer algorithm, recursive backtracking, or dynamic programming) that considers the trade-offs between the brute-force to solve a problem.
7. Describe how graphs are represented.
8. Recite elementary graph algorithms: depth-first search, breadth-first search.
9. Compute minimum weight spanning trees of graphs.
10. Compute shortest paths in graphs.
11. Argue the correctness of algorithms.
12. Describe the notion of NP-completeness and its importance. Recite several NP-complete problems and their proofs.
13. Prove that a problem is NP-complete.
14. Implement algorithms to meet desired requirements



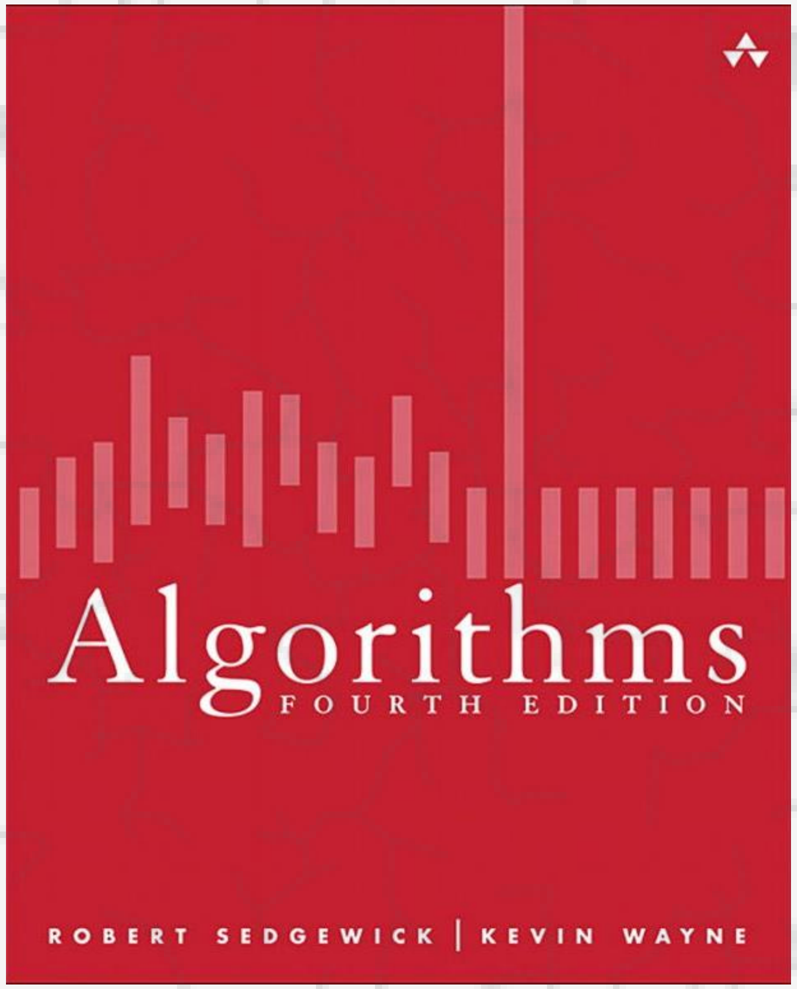
https://algs4.cs.princeton.edu

LECTURE 1

- *Course info*
- *Course learning outcomes*
- *Course assessment criteria*
- *Course rules*
- *What is an algorithm?*
- *Describing algorithms*

Course evaluation

▪ Final Examination:	50%
▪ Midterm Test:	10%
▪ Quizzes (every 3rd week, in lab):	15%
▪ Assignments & scribing	25%



https://algs4.cs.princeton.edu

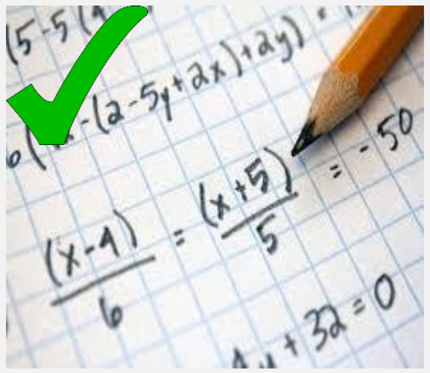
LECTURE 1

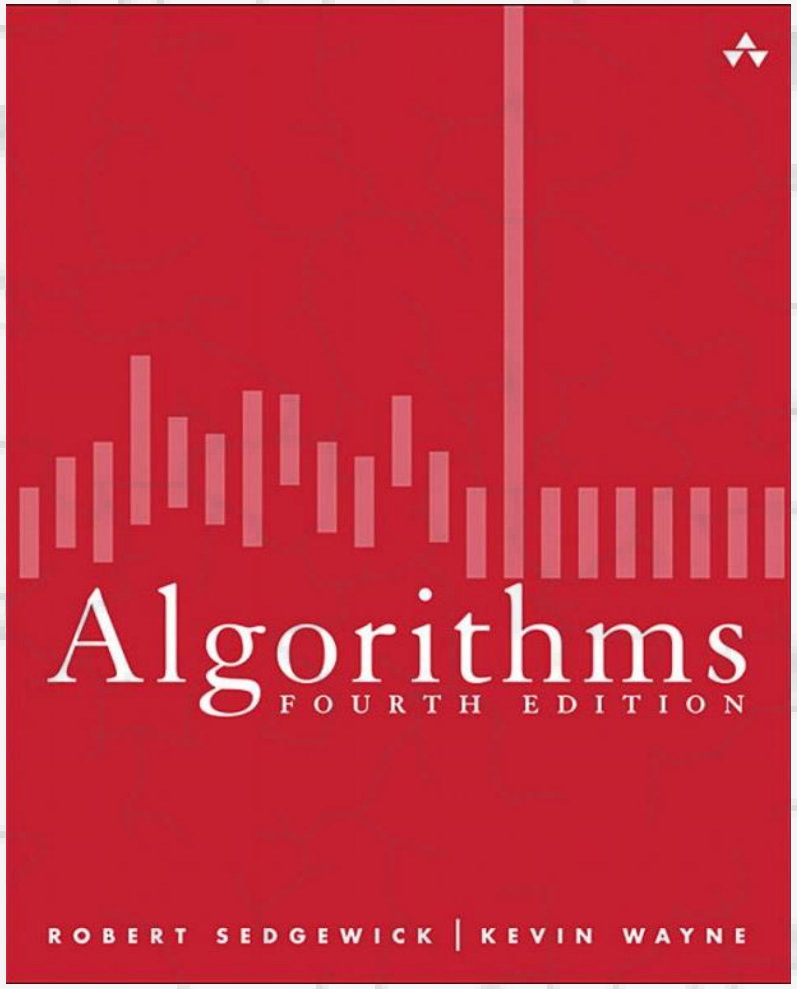
- ▸ *Course info*
- ▸ *Course learning outcomes*
- ▸ *Course assessment criteria*
- ▸ *Course rules*
- ▸ *What is an algorithm?*
- ▸ *Describing algorithms*

Course rules

Course rules – Do's and Dont's

- Use of cellphones & laptops during lectures is discouraged.
- All students should have a **math notebook.**
- Ensure your **CS Labs** & **CS Moodle** accounts are active.
- **Cheating** & **plagiarism** will be dealt with according to university policy.
- RESPECT others





https://algs4.cs.princeton.edu

LECTURE 1

- ‣ *Course info*
- ‣ *Course learning outcomes*
- ‣ *Course assessment criteria*
- ‣ *Course rules*
- ‣ *What is an algorithm?*
- ‣ *Describing algorithms*

What is an algorithm?

An **algorithm** is a finite sequence of rigorous instructions, typically used to solve a class of specific **problems** or to perform a computation.” - Wikipedia

An **algorithm** is a set of well-defined instructions to solve a particular **problem**. - https://www.programiz.com/dsa/algorithm

What is a problem?

What is a problem?

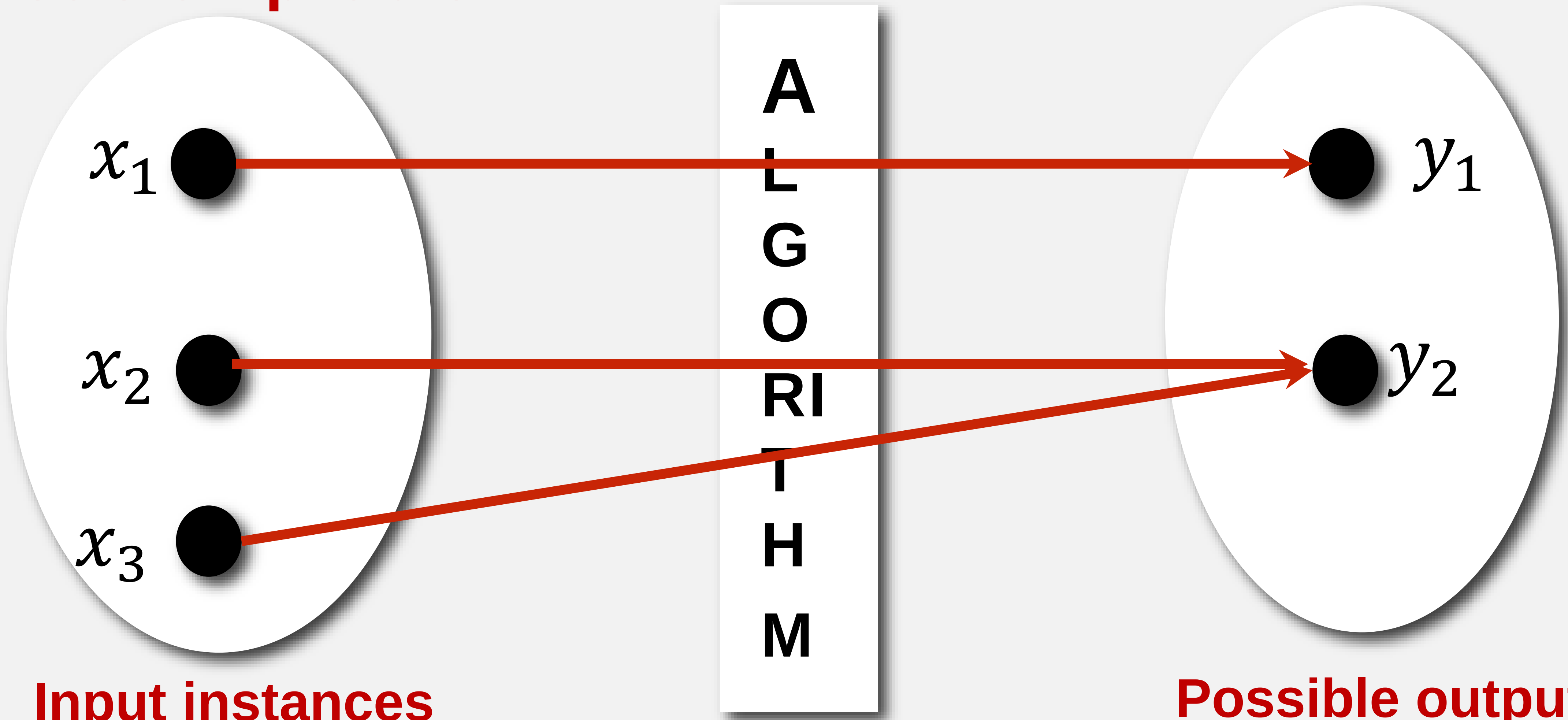
A **problem** is a set of input instances and a task to be performed on the input instances.

Example 1
Problem: ADDITION
Input instance: Two positive integers x, y
Task: Add the two integers
problem instances: 5, 8; -3, 7

Example 2
Problem: SORTING
Input instance: A list of numbers $x_1, x_2, \ldots, x_n$
Task: permute the numbers in the list so that they appear in nondecreasing order in the list
problem instances: [1, 4, 4], [-3, -5, 1, 7]

What is a problem?

Example 3
Problem: SEARCHING
Input instance: A list of numbers $[x_1, x_2, \dots, x_n]$, and y
Task: Answer the question: Is y in the list?
problem instances: [1, 4, 4] and 10, [-3, -5, 1, 7] and 1
A problem whose answer is either YES or NO, is called a **decision problem**.



Input instances

Possible output

What is an algorithm?

An **algorithm** is a set of well-defined instructions to solve a particular **problem**.
There are many algorithms to solve a problem – and each algorithm can be implemented in many different ways!

Required properties of an algorithm:

- **Finiteness** – must terminate after a finite number of steps
- **Definiteness** – each step is precisely defined
- **Effectiveness** – operations require less space and time resources (may need to strike a balance)
- **Input/output** – at least 0 inputs and at least 1 output
- **Correctness** – Output should be what is desired

What is an algorithm?

A **problem instance** consists of input (satisfying problem constraints) needed to compute a solution to a problem.

An algorithm that halts with the **correct output** for every valid problem instance is said to be **correct** (solves the problem)

Why study algorithms?

- **[Profit]** Algorithms are a technology: efficient algorithms → efficient programs → riches! (with good marketing!)
- **[Impact]** The Fast Fourier Transform (FFT) revolutionized digital signal processing.
- **[Scale problem solving]** Helps us understand scalability: how algorithm behaves as input gets large
- **[Improved knowledge]** To better understand and talk about program behavior.
- **[Applicable in other fields]** Knowledge gained can be applied to other related fields.

Is algorithm performance the only important aspect of an algorithm?

Specifying algorithms

- We use **pseudocode** – Math, English, a bit of programming language
- Our style is based on that of CLRS – clrscode4e to be specific

- A document will be uploaded on course website describing the style

Example 1

Maximum subarray problem: MAXSUB ARRAY

Problem: MAXSUB ARRAY

Input instance: An array of small integers $A[0..n-1]$

Task: Compute $max_{\{i \le j\}} \sum_{k=i}^{j} A[k]$

Problem solving Steps:

1. **Understand the problem**
2. **Devise a plan**
3. **Carry out the plan**
4. **Look back and reflect**

https://ccmit.mit.edu/problem-solving/

Solution 1 – Brute-force

Problem: MAXSUB ARRAY

Input instance: An array of small integers $A[0{:}n-1]$

Task: Compute $max_{\{i \le j\}} \sum_{k=i}^{j} A[k]$

```
MAXSUB_ARRAY(A)
1. Maxarray_sum = $-\infty$
2. n = A.length - 1
3. for i = 0 to n
4.   for j = i to n
5.     subarray_sum = 0
6.     for k = i to j
7.       subarray_sum = subarray_sum + A[k]
8.     if subarray_sum > maxarray_sum
9.       maxarray_sum = subarray_sum
10.return maxarray_sum
```

Solution 2 - a bit clever

Problem: MAXSUB ARRAY

Input instance: An array of small integers $A[0..n-1]$

Task: Compute $max_{\{i \le j\}} \sum_{k=i}^{j} A[k]$

```
MAXSUB_ARRAY(A)
1.maxarray-sum = $-\infty$
2.n = A.length - 1
3.for i = 0 to n
4.   sum = 0
5.   for j = i to n
6.      sum = sum + A[j]
7.      if sum > maxarray_sum
8.         maxarray_sum = sum
9.return maxarray_sum
```

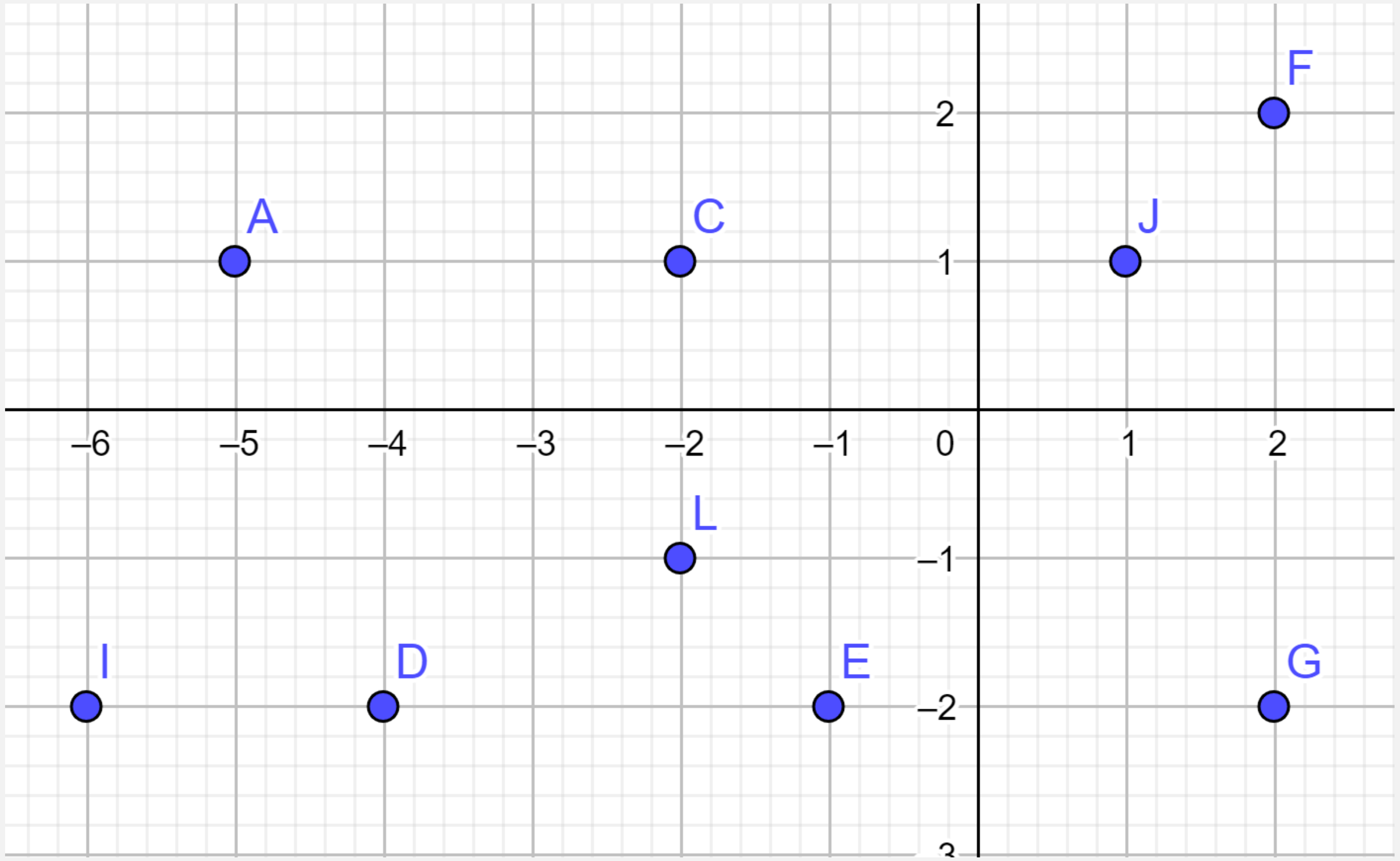
Example 2

Robotic arm tour optimization problem

Problem: ROBOT TOUR OPTIMIZATION

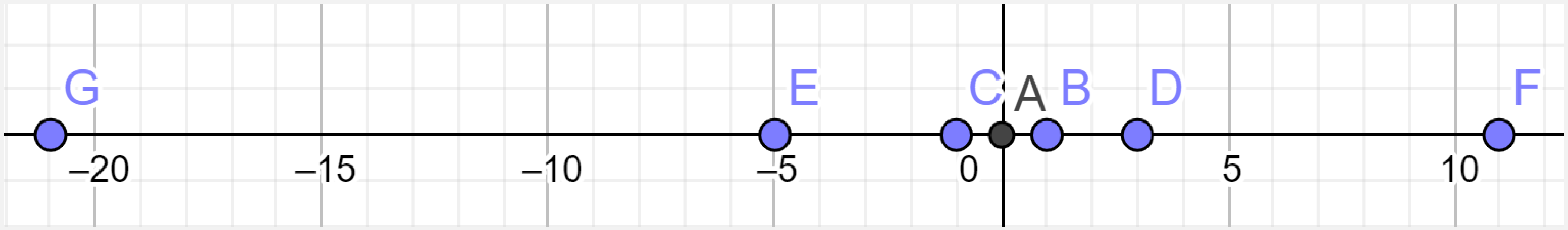
Input instance: A set S of points in the plane

Task: Compute the shortest cycle tour that visit each point in the set S



Solution 1 - Nearest neighbor

```
ROBOT_TOUR(S)
1. pick a point $p_0$ from S
2. visit $p_0$
3. i = 0
4. While there are still unvisited points in S
5.    i = i + 1
6.    select $p_i$ to be the closest unvisited point to $p_{i-1}$
7.    visit $p_i$
8. return to $p_0$ from $p_{n-1}$
```



Solution 2 - Optimal (Brute-force)

```
ROBOT_TOUR(S)
1. $d = \infty$
2. for each of the $\frac{(n-1)!}{2}$ permutations $P_i$ of the points
   in S
3.     if $cost(P_i) \leq d$
4.         $d = cost(P_i)$
5.         $P_{min} = P_i$
6. return $P_i$
```

Pop quiz

- **Are all the four algorithms correct?**
- **How efficient (time and space) are they?**

Reflection

- MAXSUB ARRAY problem is solvable.
  - There are **efficient** algorithms to solve it
  - Which one is better? Stay tuned.
- Robot Tour Optimization is hard, really hard!
  - Need to recognize this! Stay tuned.

Summary

- ❑Know course meeting times. Know the course rules.
- ❑Be active in class and labs.
- ❑Do your own work! Write your own programs.
- ❑Know why you are taking CSI323
- ❑Know how your grade will be calculated at the end
- ❑Know how to describe problems
- ❑Know what an algorithm is
- ❑Know how to specify algorithms