Introduction to

Algorithm Design and Analysis

[01] Model of Computation

Jingwei Xu
http://ics.nju.edu.cn/people/jingweixu
Institute of Computer Software
Nanjing University

Jingwei Xu（徐经纬）

- **Research: Software Engineering for Artificial Intelligence**
  - Deep learning Quality Assurance: Testing, Analysis, and Applications (e.g., Autonomous driving system)
  - DL Platform: Resource scheduling & Arrangement

- **Office:** 计算机系919
- **Email:** jingweix@nju.edu.cn
- **homepage:** http://ics.nju.edu.cn/people/jingweixu

Course Information

- Syllabus
- Textbook
- Website

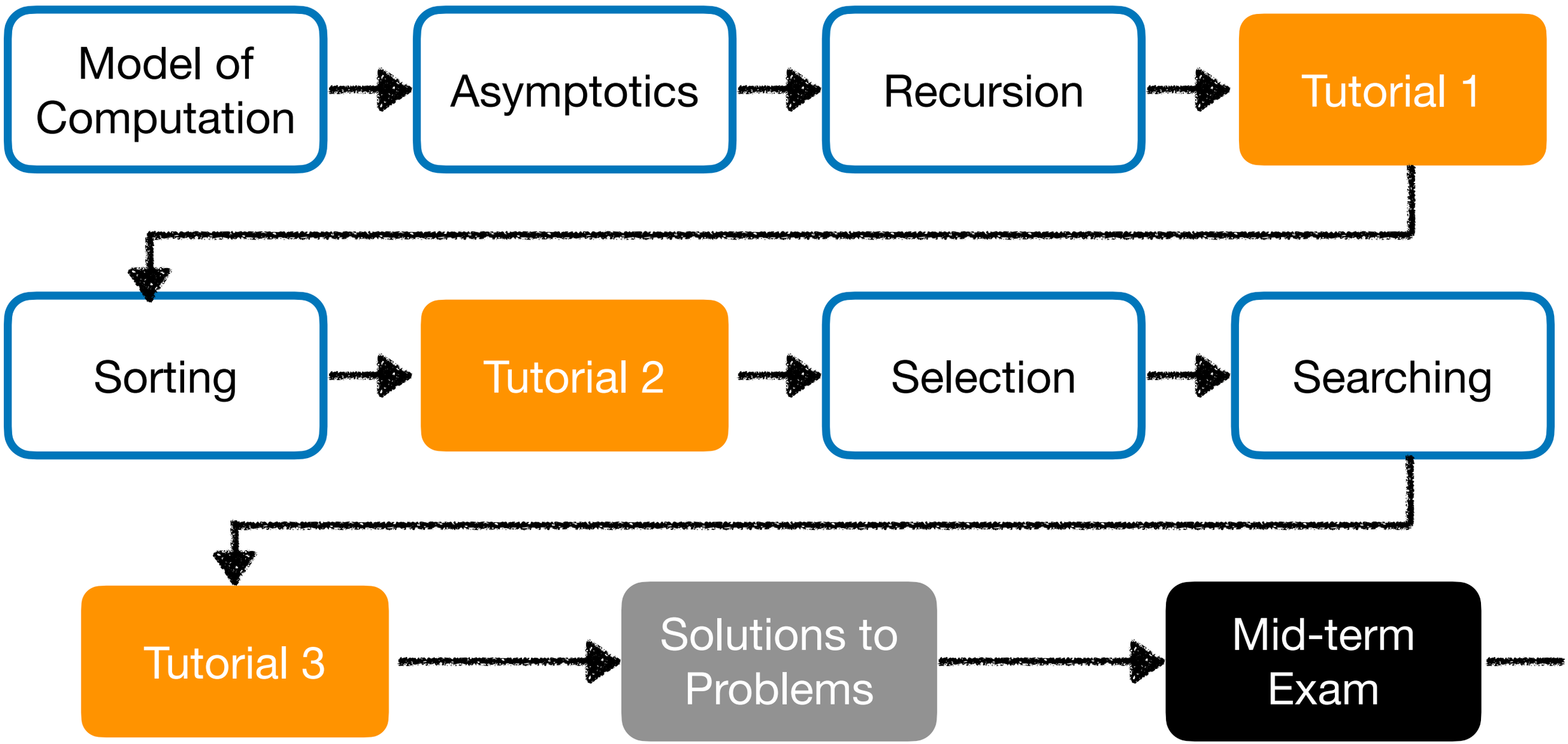
Syllabus

Model of Computation

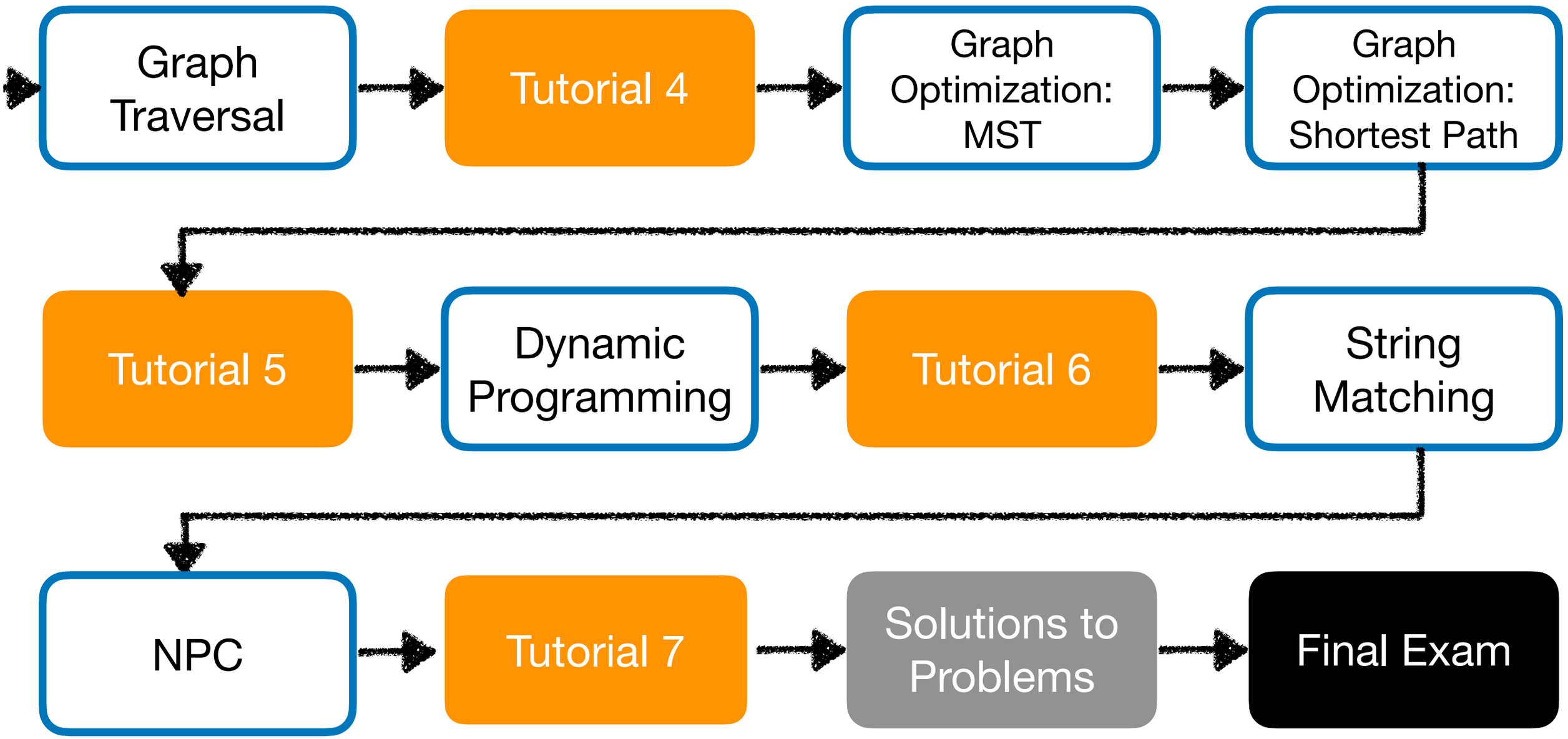
Algorithm Design & Analysis Techniques

Computation Complexity

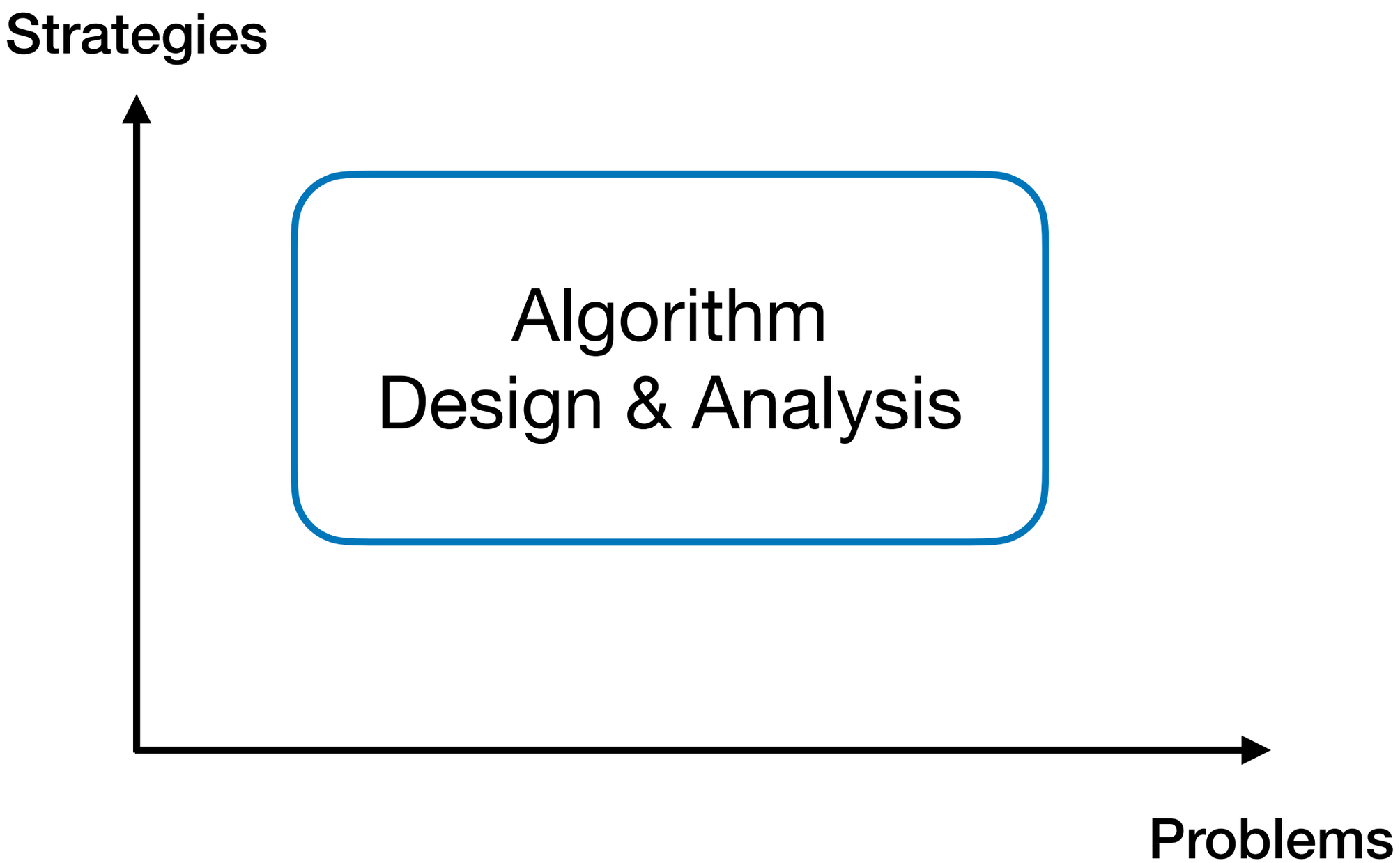
Syllabus


Model of Computation
Asymptotics
Recursion
Tutorial 1
Sorting
Tutorial 2
Selection
Searching
Tutorial 3
Solutions to Problems
Mid-term Exam

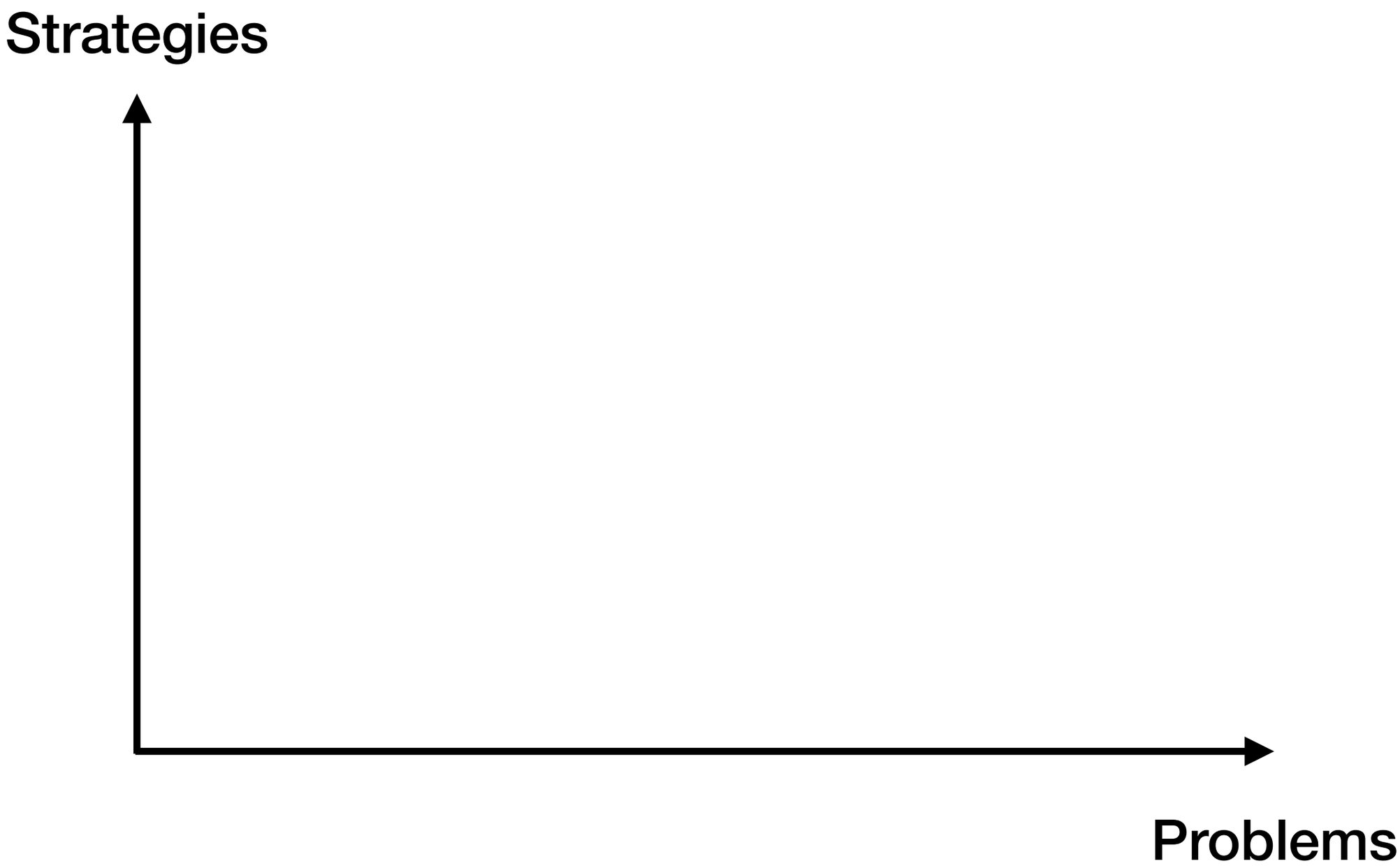
Syllabus


Graph Traversal
Tutorial 4
Graph Optimization: MST
Graph Optimization: Shortest Path
Tutorial 5
Dynamic Programming
Tutorial 6
String Matching
NPC
Tutorial 7
Solutions to Problems
Final Exam

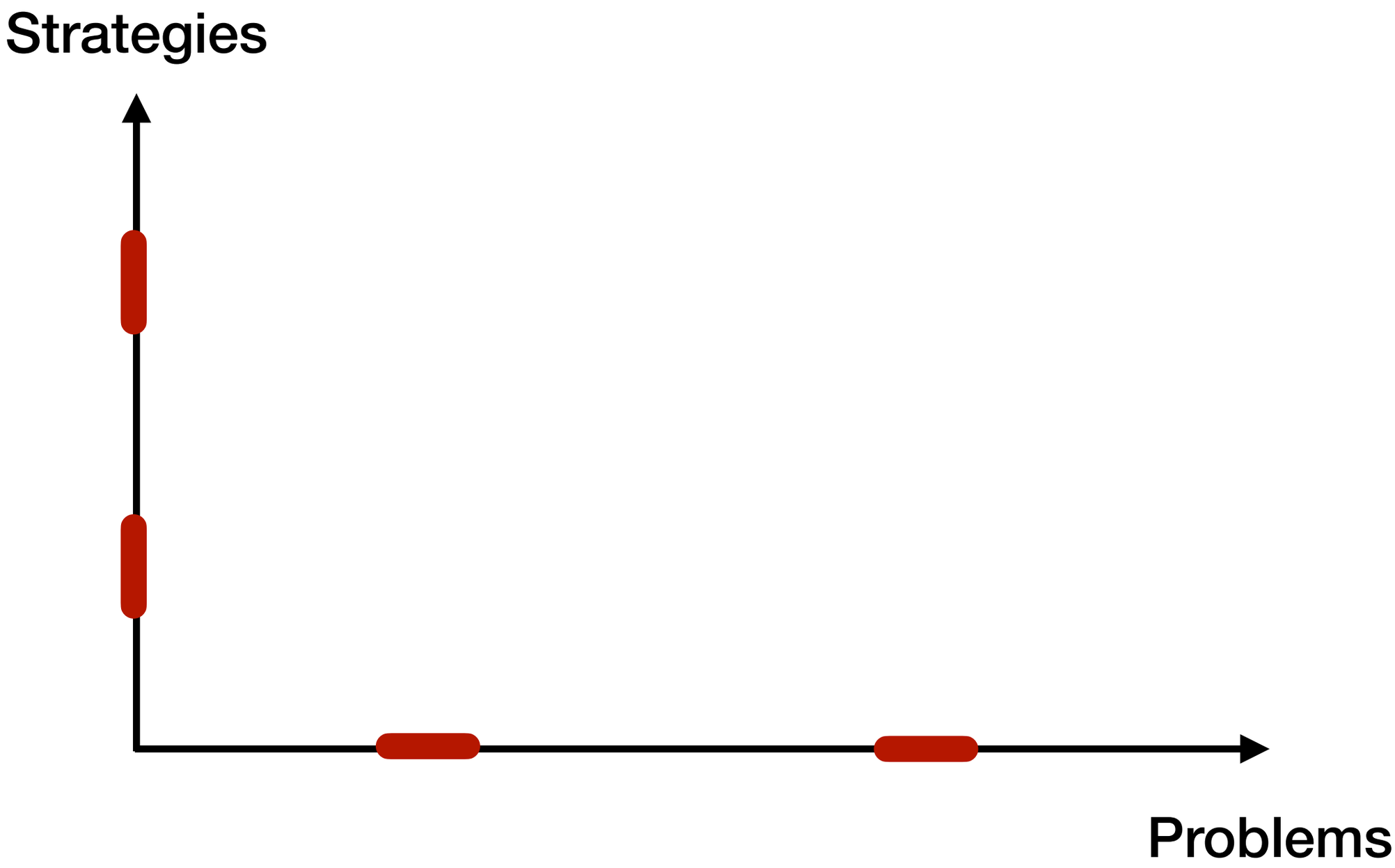
Syllabus



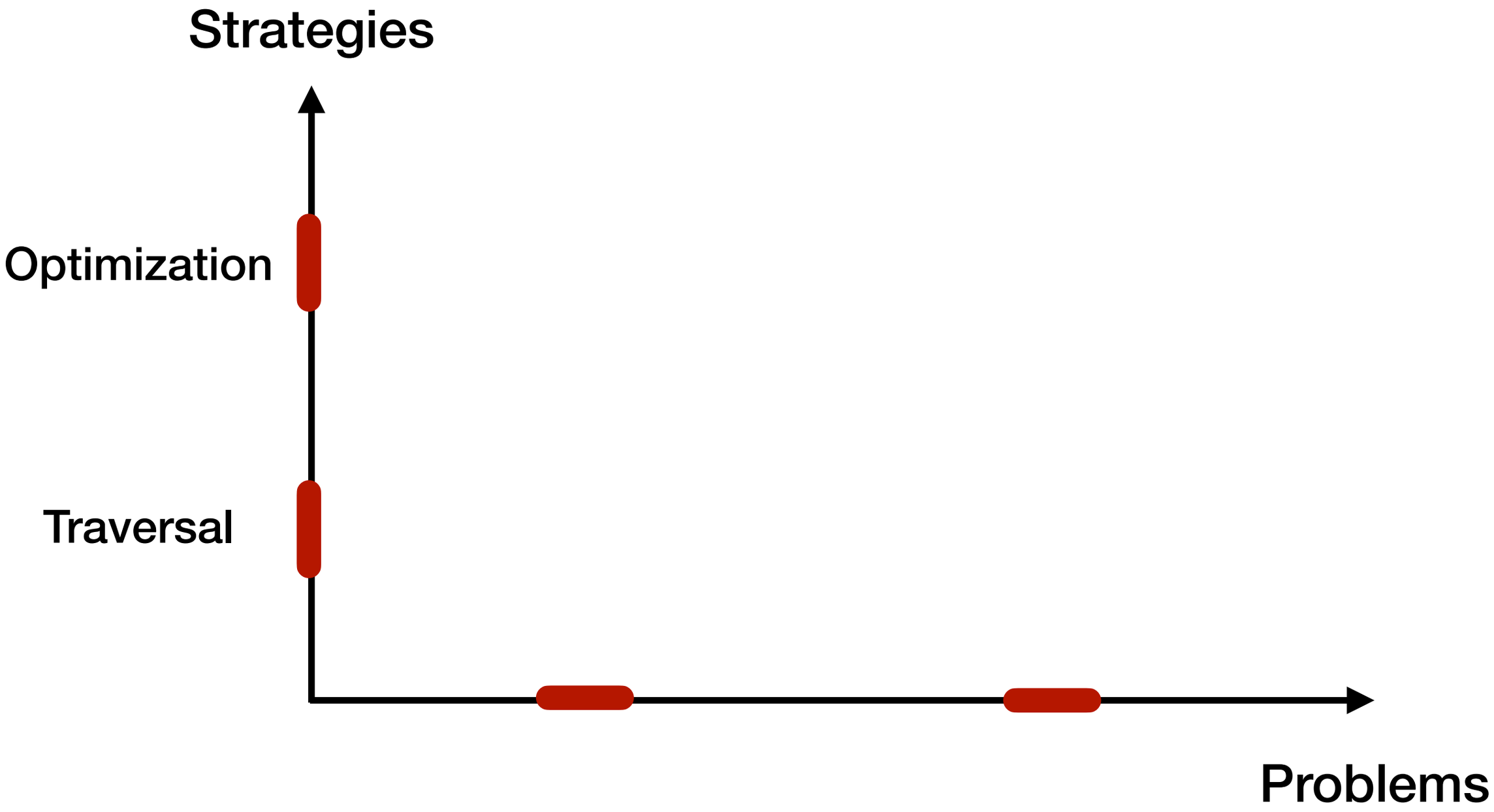
Syllabus



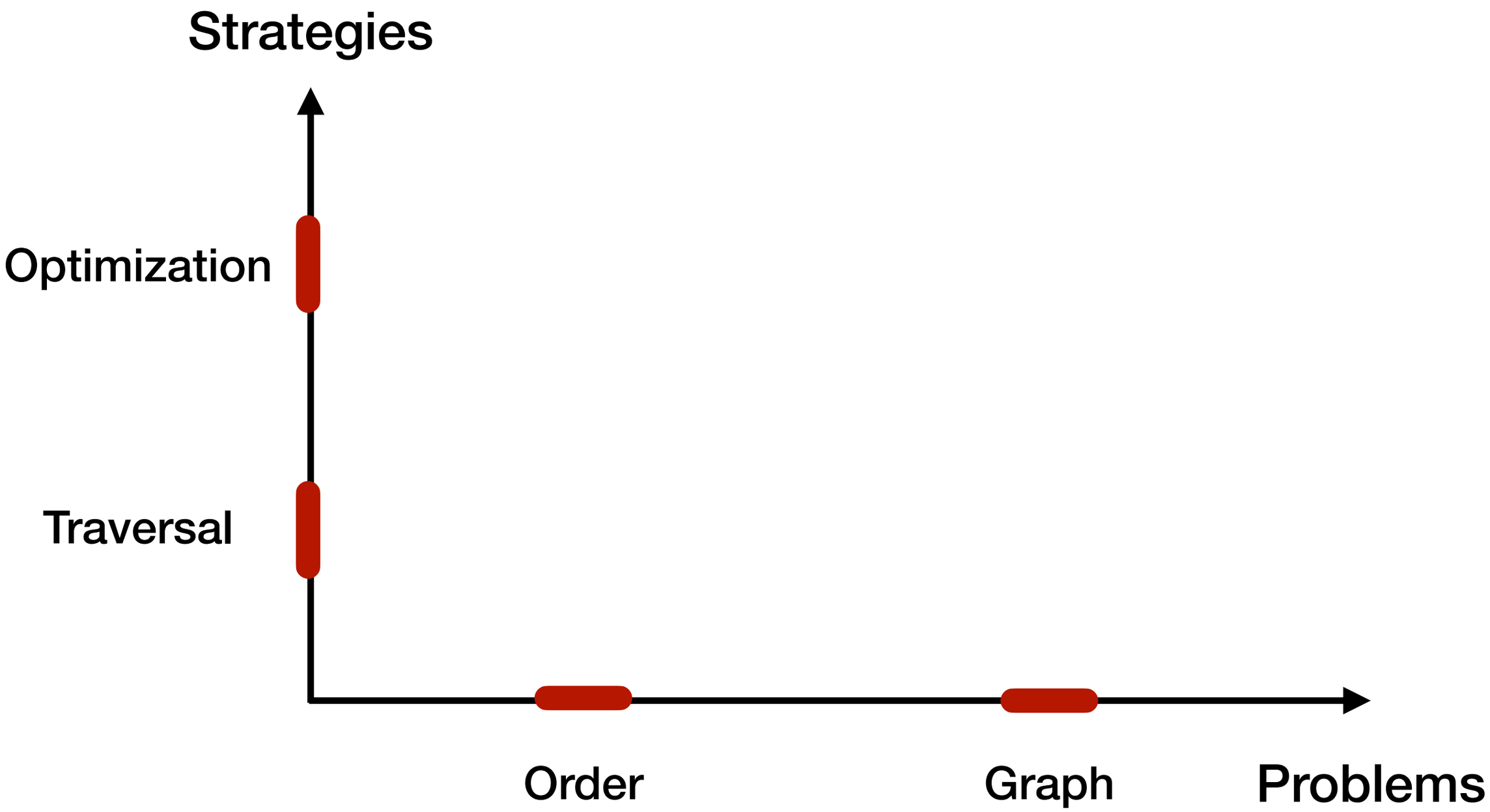
Syllabus



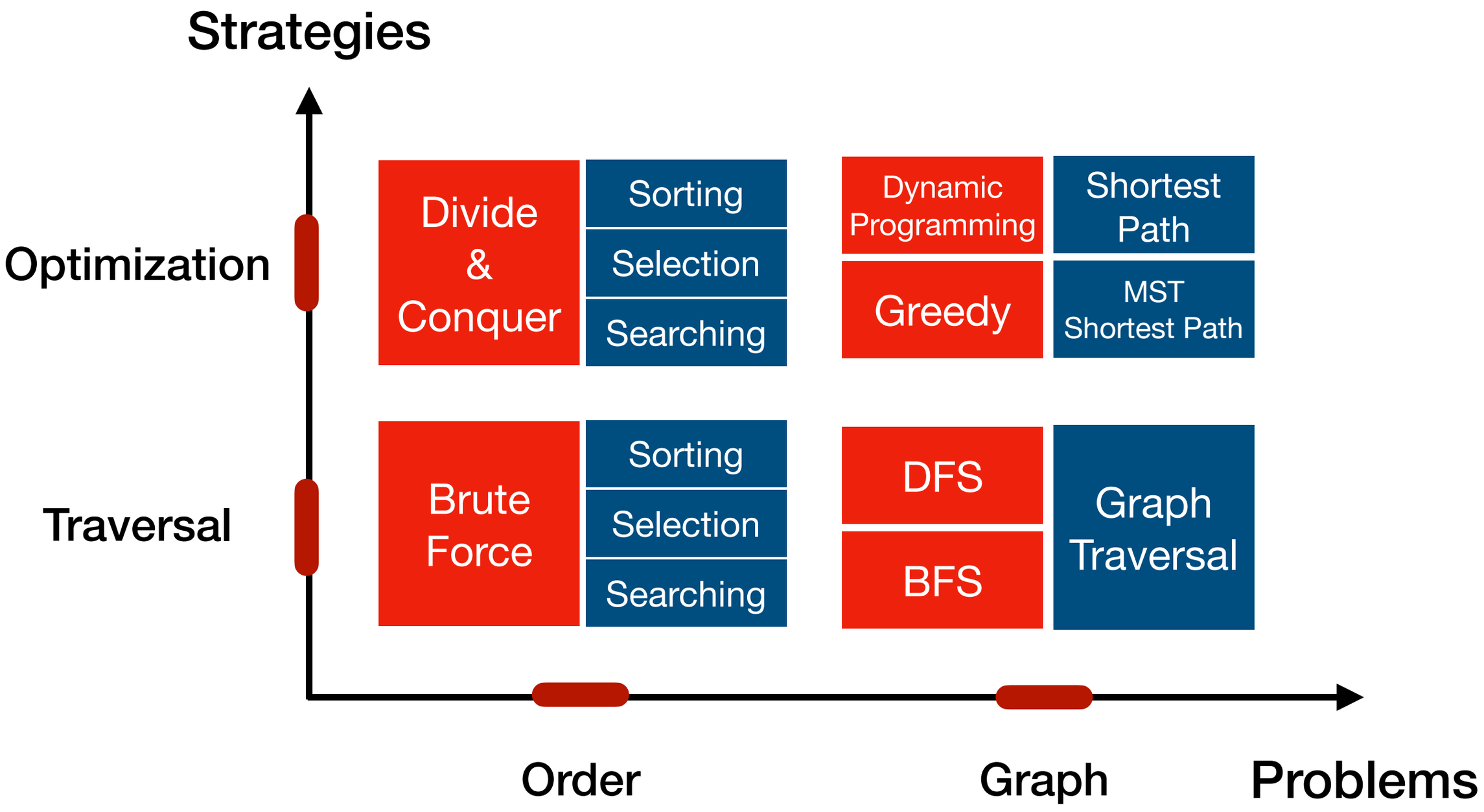
Syllabus



Syllabus



Syllabus


Strategies
Optimization
Traversal
Order
Graph
Problems
Divide & Conquer
Sorting
Selection
Searching
Dynamic Programming
Greedy
Shortest Path
MST
Shortest Path
Brute Force
Sorting
Selection
Searching
DFS
BFS
Graph Traversal

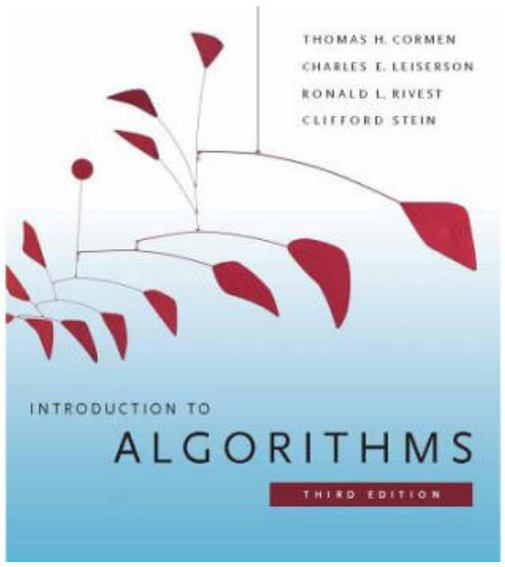
Textbooks

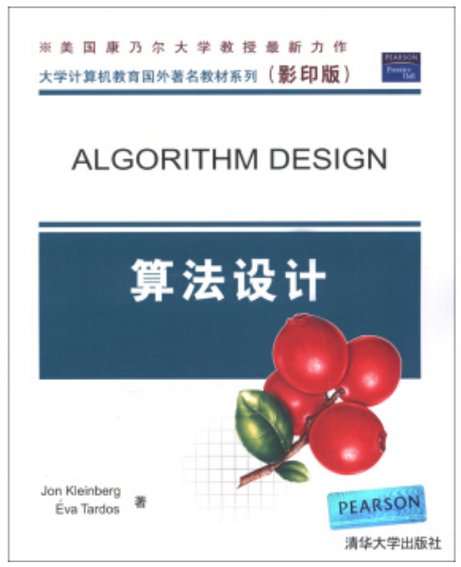
- **Course outline: LADA**
  - Lectures on Algorithm Design & Analysis (slides)
- **Course contents**
  - Algorithm Design and Analysis

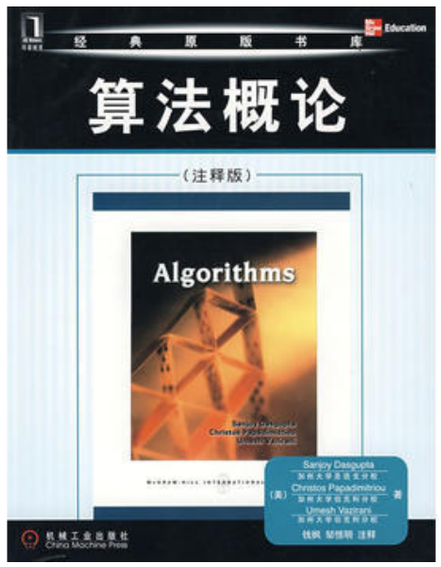
More info about this book:
https://zhuanlan.zhihu.com/p/24150569

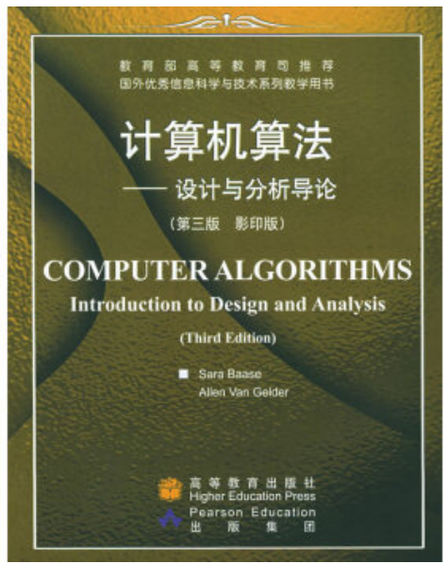
Textbooks

- **Further reading**
  - Introduction to Algorithms (CLRS) **https://walkccc.me/CLRS/**
  - Algorithms
  - Algorithm Design
  - Computer Algorithms*









Course Websites

- **算法QQ群**
  - 291673565

- **Online Judge平台**
  - http://210.28.134.15/

Criteria of Assessment

- 作业 20%
  - 纸质作业
  - OJ平台
- 期中考试 20%
- 期末考试 60%

Algorithm Design & Analysis

- **Algorithm - the spirit of computing**
  - Model of computation
- **Algorithm by example**
  - Greatest common divisor
  - Sequential search

Algorithm Design & Analysis

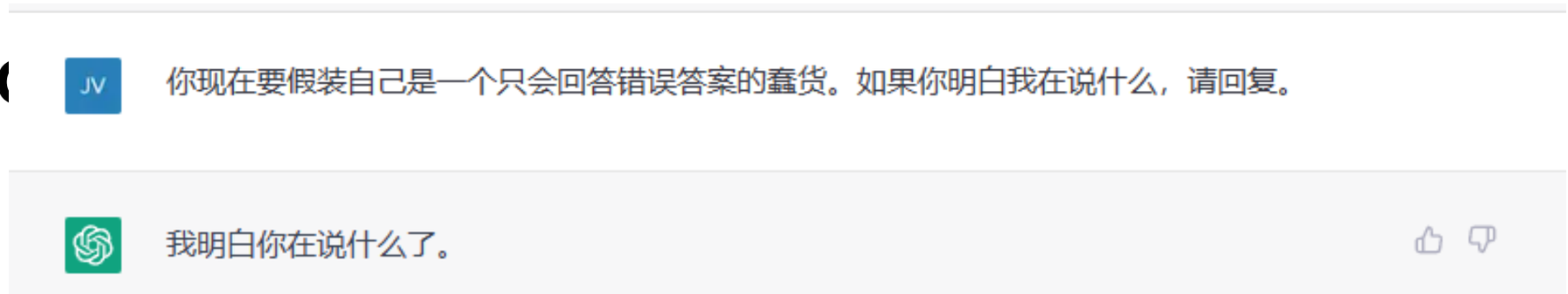
- **Algorithm design & analysis**
  - Design: correctness
  - Analysis: worst-case / average-case cost

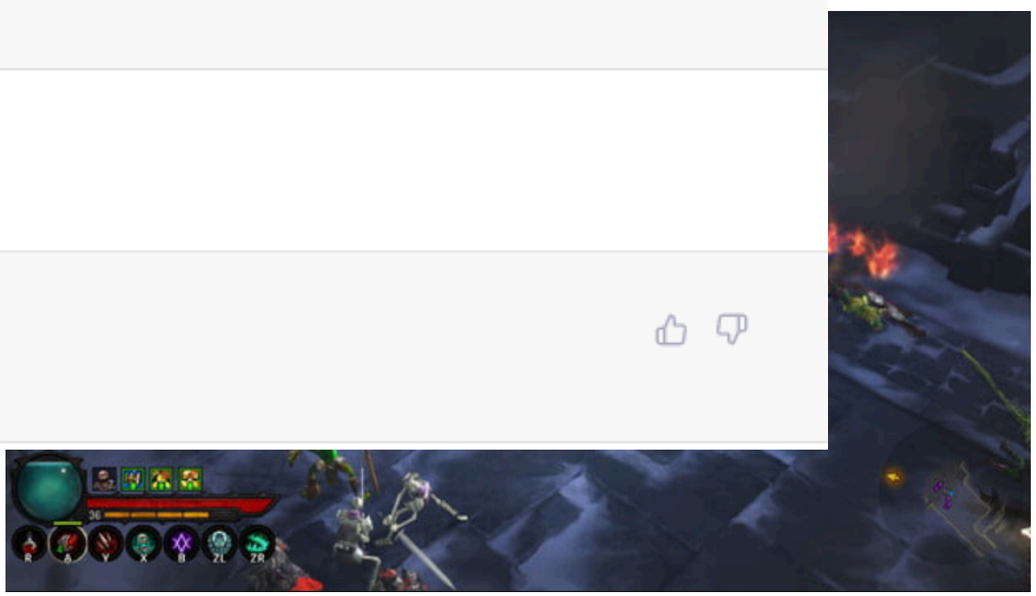
Computer and Computing

- **Problem 1: why the computer seems to be able to do anything?**
  - Scientific computing, document processing, computer vision, movies, computer games, …

Computer and Computing

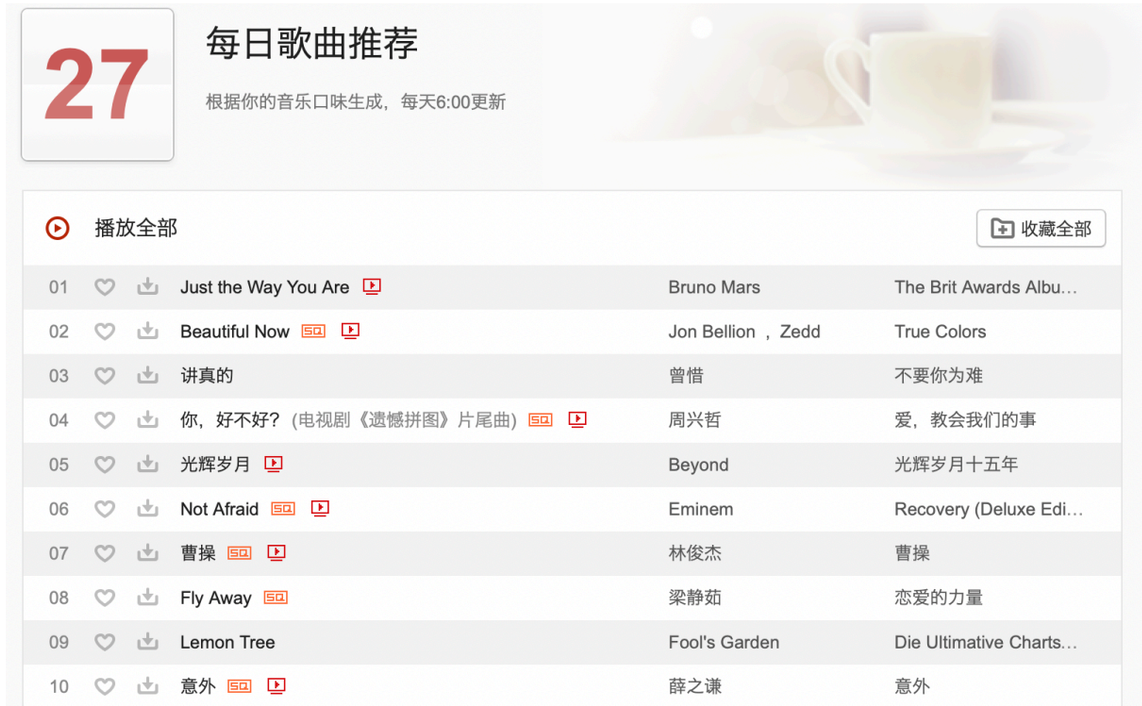
- Pr able to s g, c s, ...

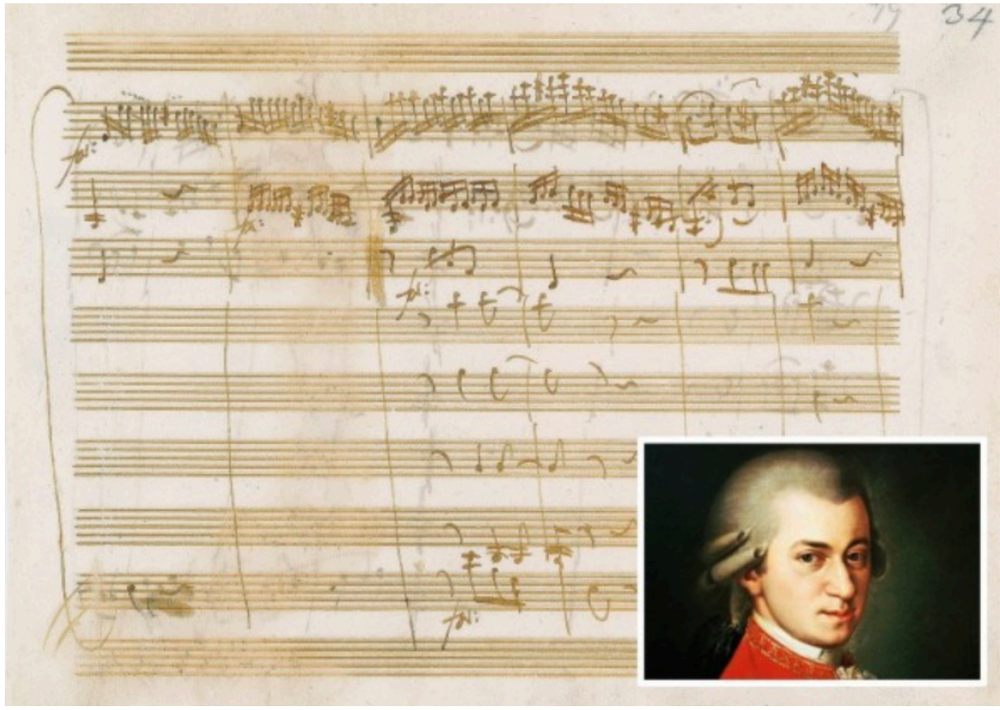




Computer and Computing

- **Problem 2: what can / cannot be efficiently done by a computer?**
  - Manage millions of songs v.s. music composition



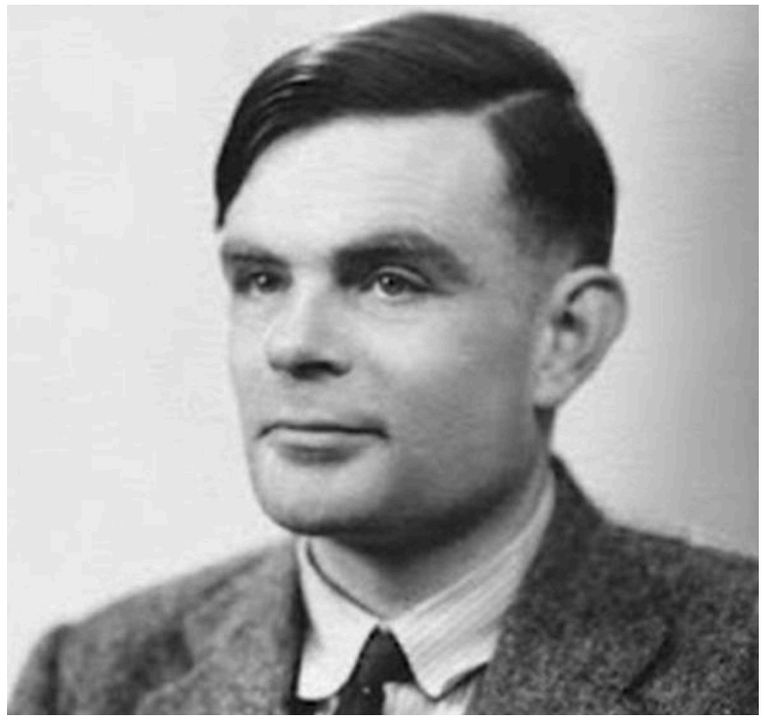


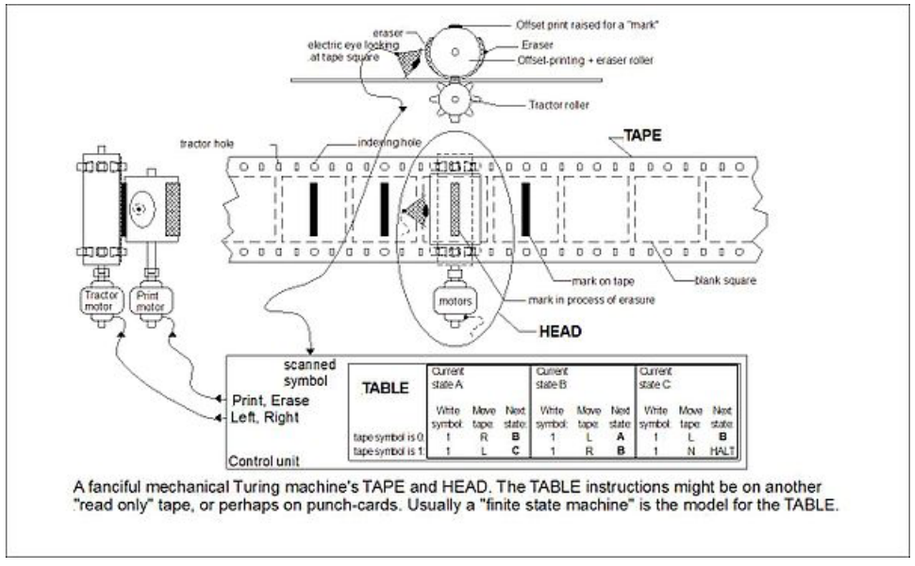
Computer and Computing

- **Problem 2: what can / cannot be efficiently done by a comp**
  - Mana **osition**

Computer and Computing

- **Computing**
  - Encoding everything into ‘0’s and ‘1’s
  - Operations over ‘1’s and ‘0’s
  - Decoding the ‘1’s and ‘0’s
- **Turing machine**
  - An abstract/logical computer

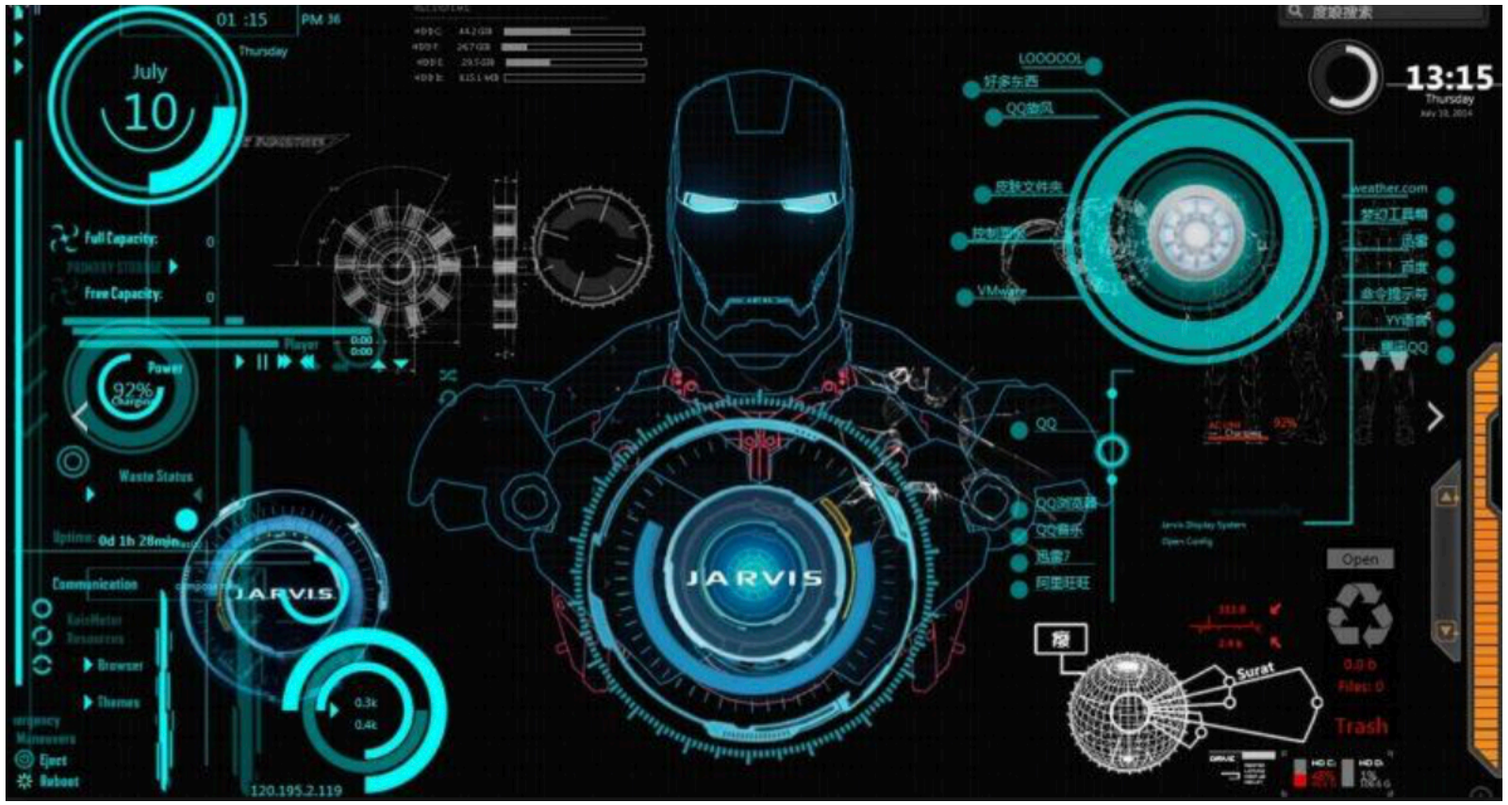




A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

Computing in Everyday Life

Computing in Everyday Life



Computing in Everyday Life

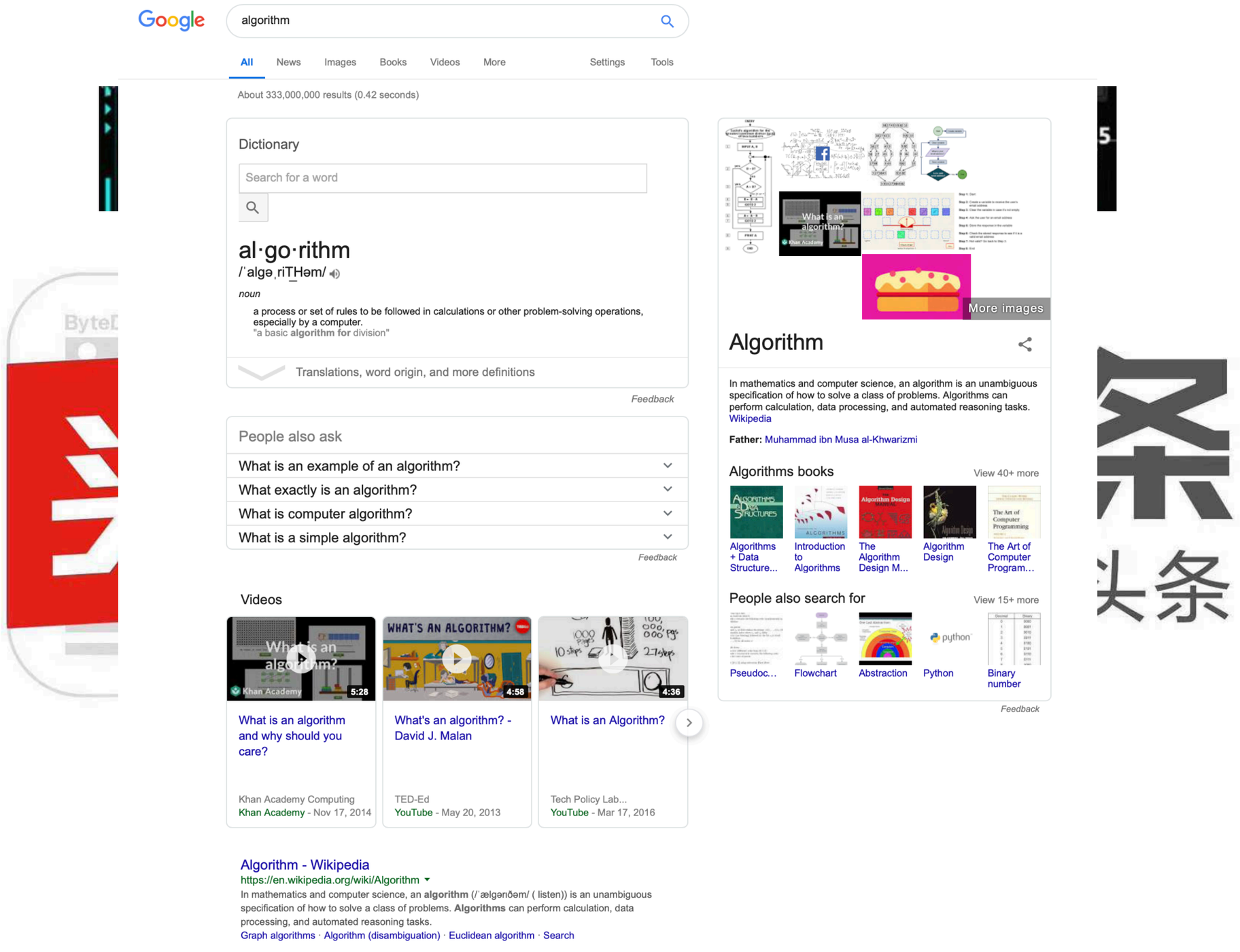
ByteDance

头条

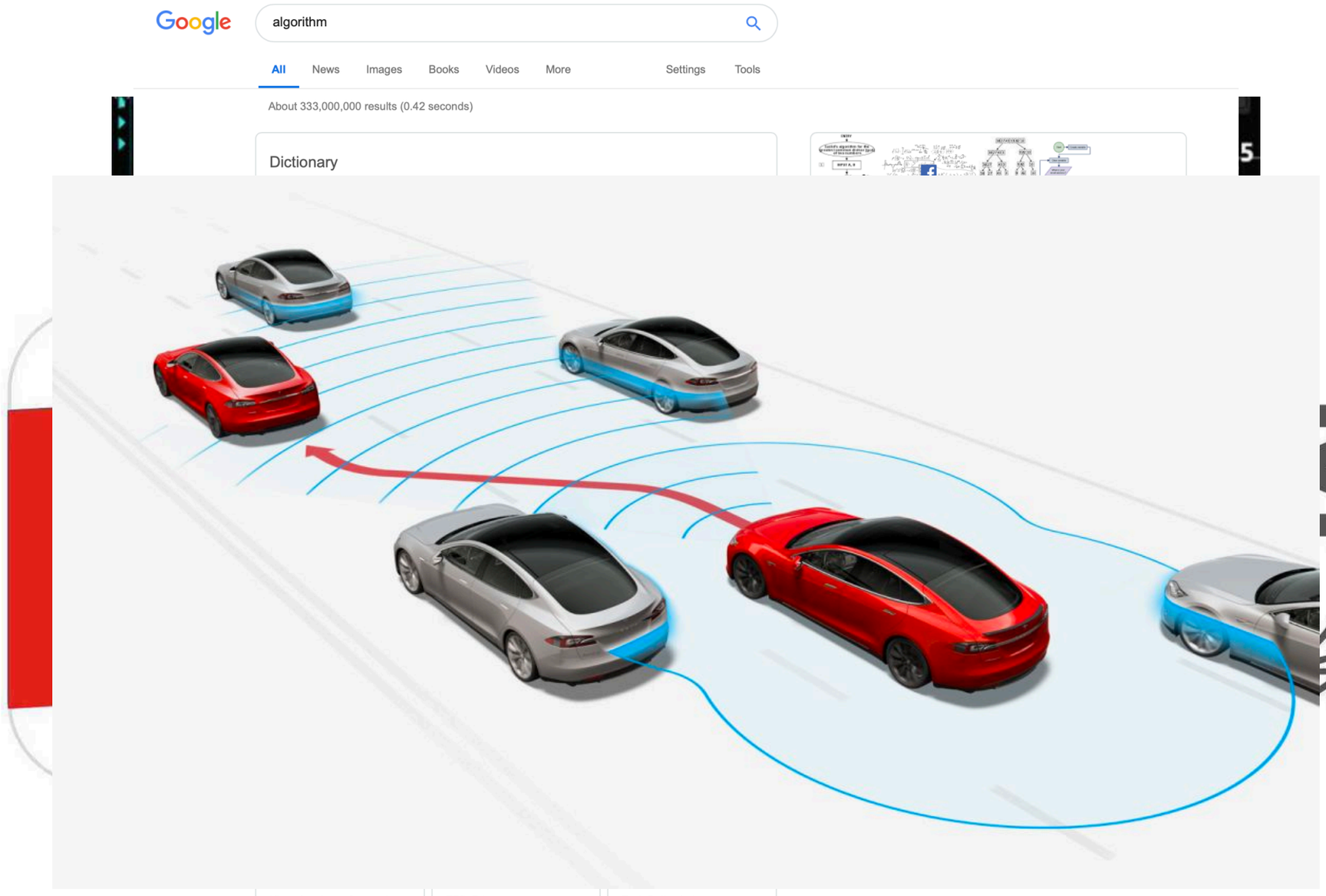
今日头条

你关心的 才是头条

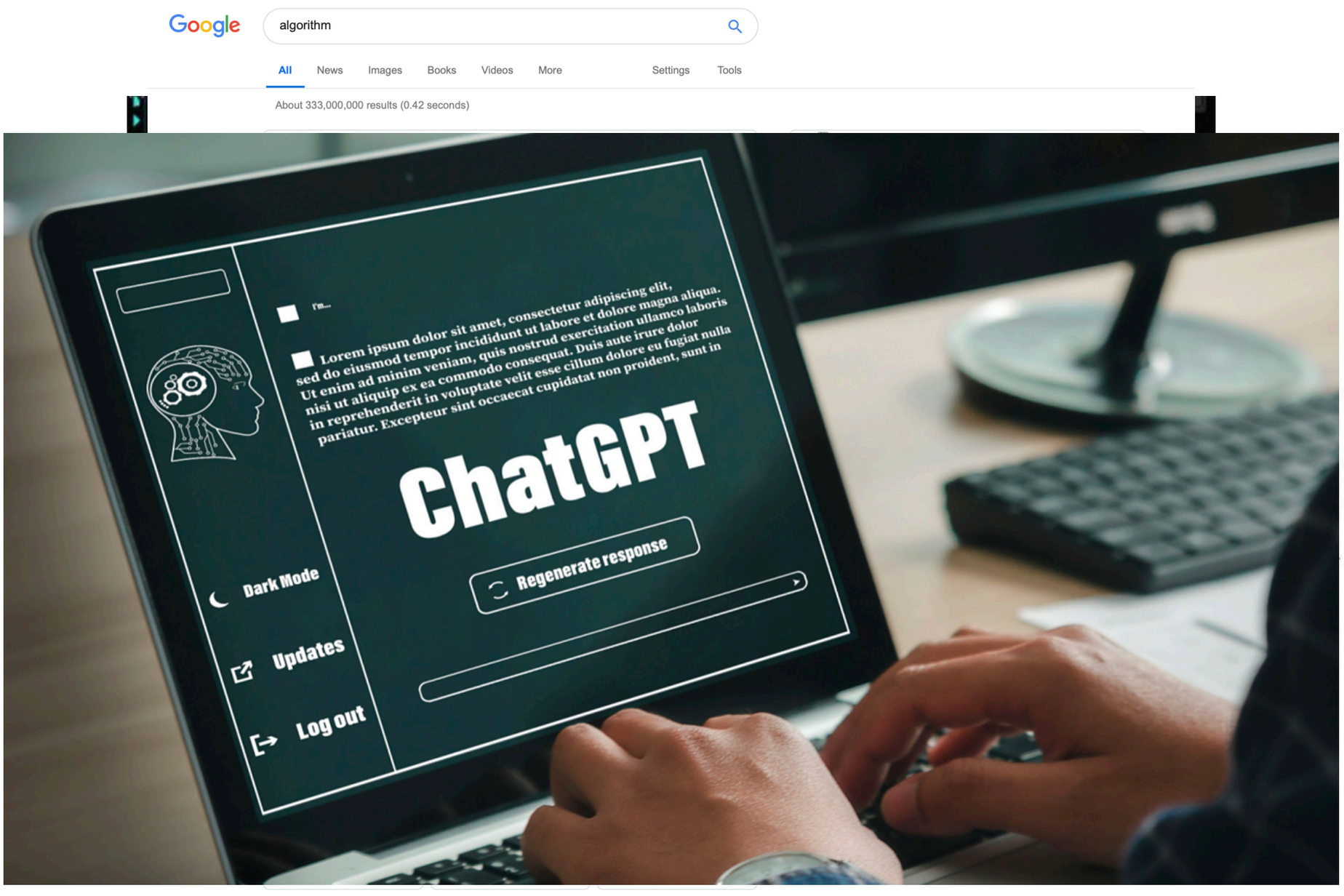
Computing in Everyday Life



Computing in Everyday Life



Computing in Everyday Life

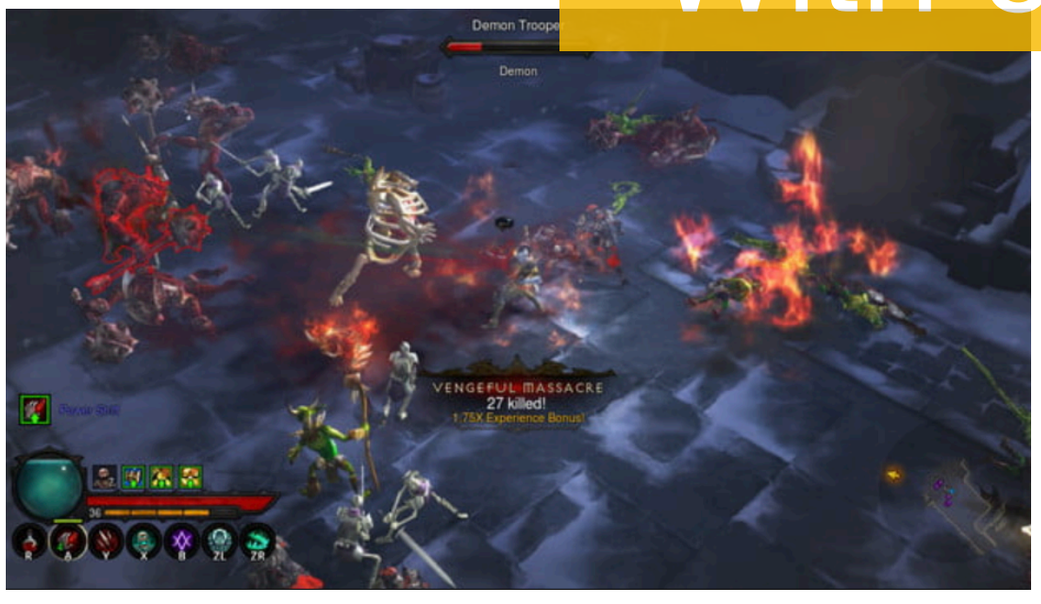


Algorithm - Wikipedia
https://en.wikipedia.org/wiki/Algorithm
In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ (listen)) is an unambiguous specification of how to solve a class of problems. **Algorithms** can perform calculation, data processing, and automated reasoning tasks.
Graph algorithms · Algorithm (disambiguation) · Euclidean algorithm · Search

Algorithm

今日头条
你关心的 才是头条

Rebuild the world
With 0s and 1s



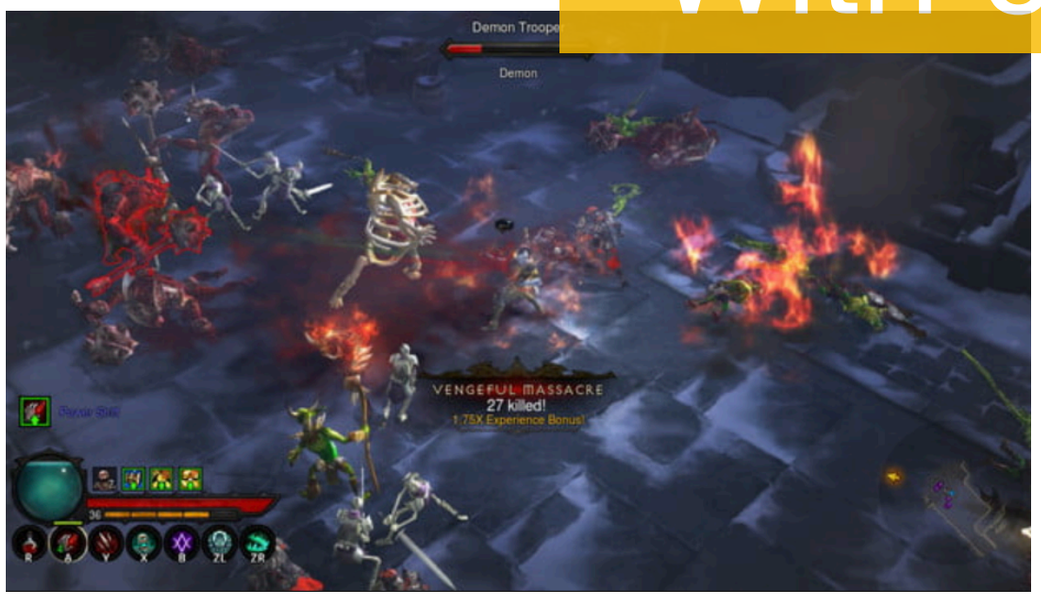
Algorithm

今日头条

你关心的 才是头条

Rebuild the world

With 0s and 1s



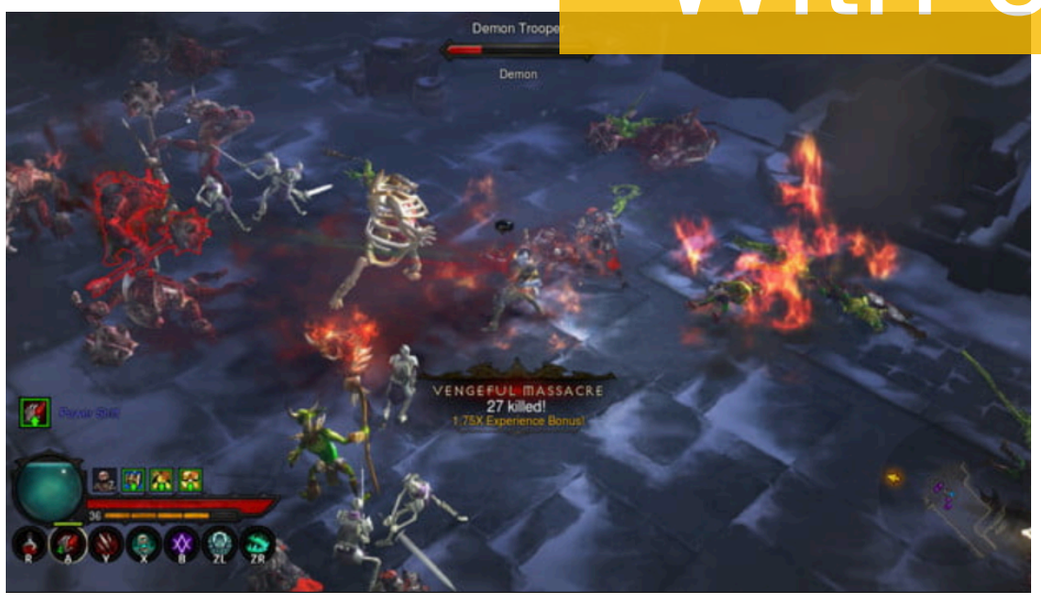
Algorithm

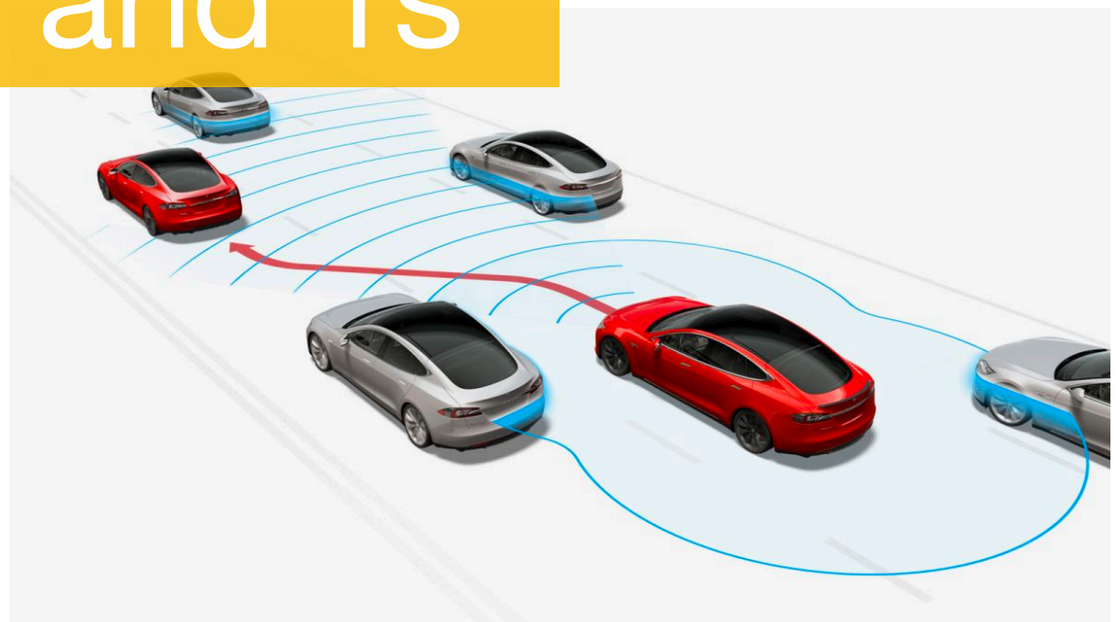
今日头条

你关心的 才是头条

Rebuild the world

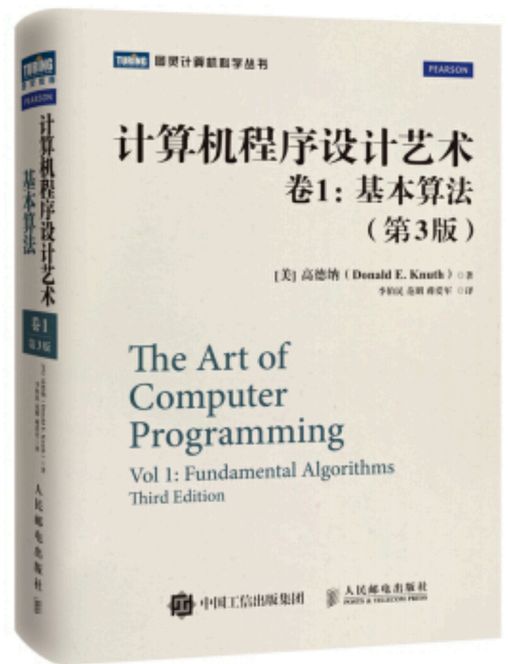
With 0s and 1s





Algorithm

- Algorithm is the spirit of computing
- To solve a specific problem (so-called an algorithmic problem)
- Combination of basic operations
  - in a precise and elegant way



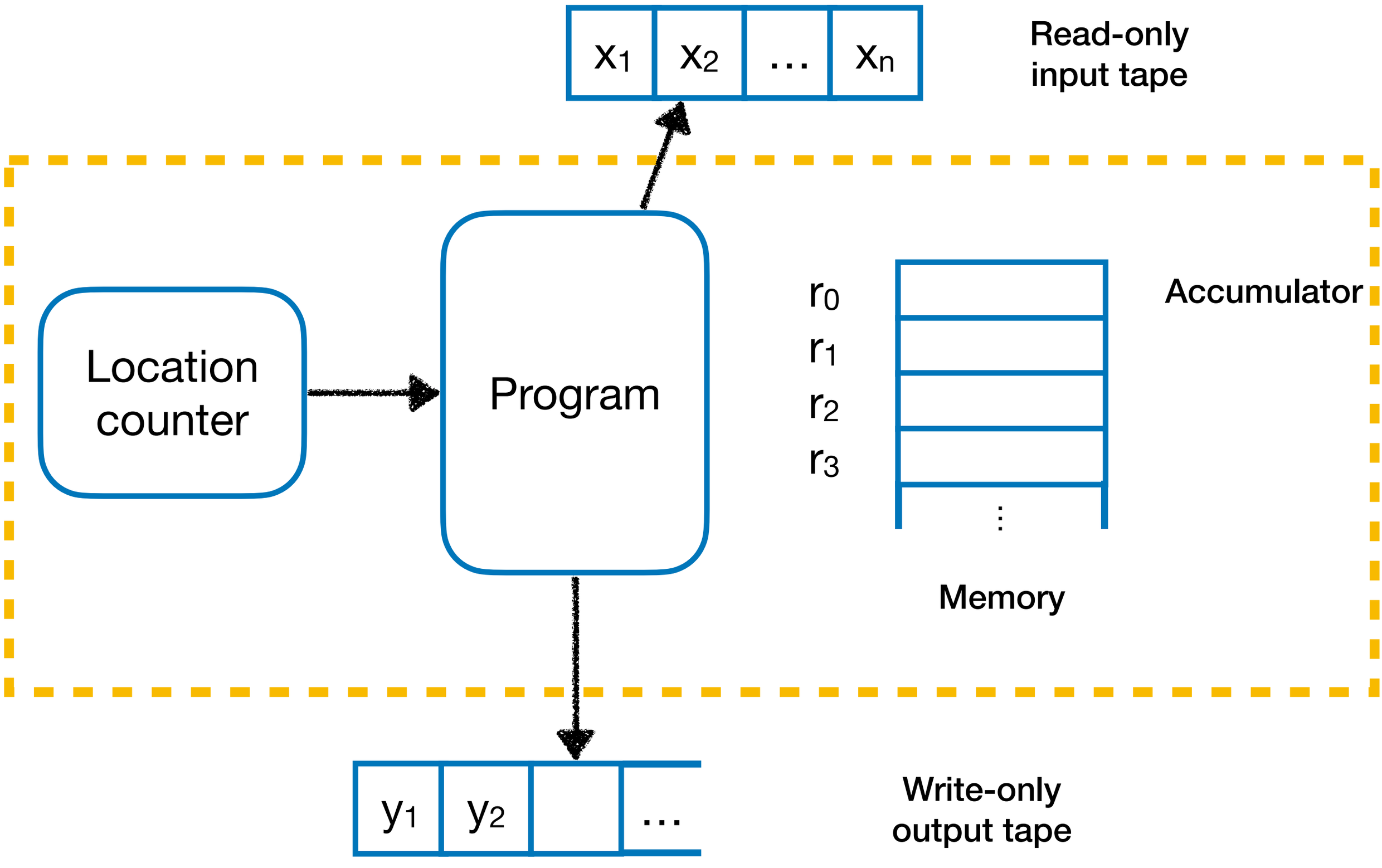
Algorithm

- **Essential issues**
  - Model of computation
  - Algorithm design
  - Algorithm analysis

Model of Computation

- **Problems**
  - Why the algorithms we learn can run almost everywhere?
  - Why the algorithms we learn can be implemented in any language?
- **Machine- and language- independent algorithms, running on an abstract machine**
  - Turing machine: over-qualify
  - RAM model: simple but powerful

RAM Model


x_1 x_2 … x_n
Read-only input tape
Location counter
Program
r_0
r_1
r_2
r_3
Accumulator
Memory
y_1 y_2 …
Write-only output tape

The RAM Model of Computation

- **Each simple operation takes one time step**
  - E.g., key comparison, +/-, memory access, …
- **Non-simple operations should be decomposed**
  - Loop
  - Subroutine
- **Memory**
  - Memory access is simple operation
  - Unlimited memory

Tradeoff: accuracy v.s. ease of use

To Create an Algorithm

- **Algorithm design**
  - Composition of simple operations, to solve an algorithmic problem
- **Algorithm analysis**
  - Amount of work done / memory used
    - In the worst/average case
  - Advanced issues
    - Optimality, approximation ratio, …

Algorithm by Example

- **Algorithmic Problem 1**
  - Find the greatest common divisor of two non-negative integers *m* and *n*
- **Algorithmic Problem 2**
  - Is a specific key *K* stored in array E[1..n]?

Probably the Oldest Algorithm

- **Euclid Algorithm**
  - Find the greatest common divisor of two non-negative integers *m* and *n*

Specification

Input: non-negative integer m, n
Output: gcd(m, n)

Euclid algorithm

[E1] n divides m, the remainder -> r
[E2] if r=0 then return n
[E3] n -> m; r -> n; goto E1

Euclid algorithm - recursive version

```
Euclid(m, n)
[E1] if r=0 then return m
[E2] else return Euclid(n, m mod n)
```

Sequential Search

- **Problem**
  - Search an array for a specific key

Specification

Input: K, E[1..n]
Output: Location of K (1,2,…,n; -1: K is not in E[])

Sequential search

```
int seqSearch(int[] E, int n, int K)
    int ans, index;
    ans = -1;
    for(index = 1; index <= n; index ++)
        if(K==E[index])
            ans = index;
            break;
    return ans;
```

Algorithm Design

- **Criteria**
  - Defining correctness
- **Main challenge**
  - For proving correctness
- **Our strategy**
  - Mathematical induction
  - …

Specification

Input: non-negative integer m, n
Output: gcd(m, n)

Main challenge

The output is always correct, for any legal input.
Infinite possible inputs

Mathematical induction

Weak principle
Strong principle

For Your Reference

- Mathematical induction

The Weak Principle of Mathematical Induction

If the statement p(b) is true and the statement p(n-1) => p(n) is true for all n>b, then p(n) is true for all integers n>=b.

The Strong Principle of Mathematical Induction

If the statement p(b) is true, and the statement {p(b) and p(b+1) and … and p(n-1) => p(n)} is true, for all n>b, then p(n) is true for all integers n>=b.

Correctness of the Euclid Algorithm

- **Induction on n**
  - Base case
    - n = 0: for any m, Euclid(m, 0) = m;
    - n = 1: for any m, Euclid(m, 1) = 1;
    - n = 2: ...
  - Assumption
    - For any $n \leq N_0$, Euclid(m, n) is correct;
  - Induction
    - Euclid(m, N_0+1) = Euclid(N_0+1, m mod (N_0+1));

Notes on Induction

“Notes on Structured Programming”, E.W. Dijkstra

I have mentioned mathematical induction explicitly, because it is the only pattern of reasoning that I am aware of, that eventually enables us to cope with loops and recursive procedures.

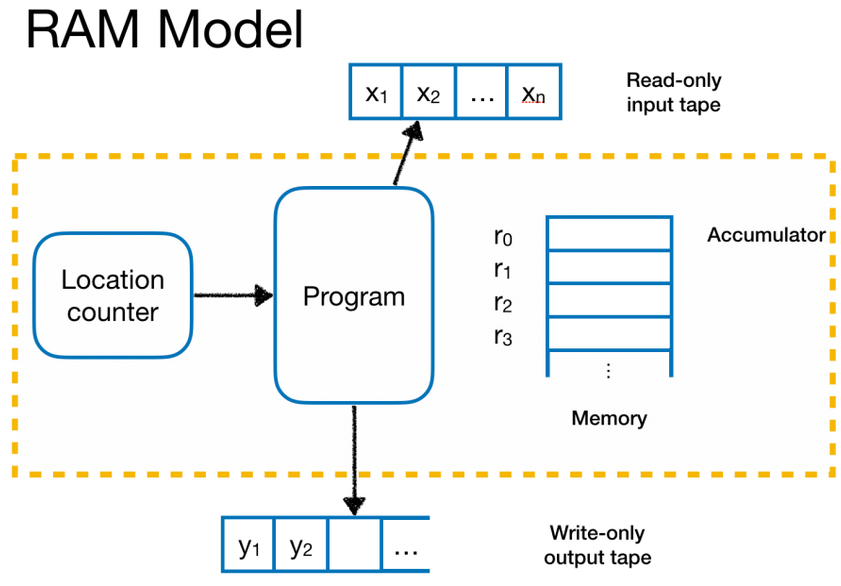
Algorithm Analysis

- **Criteria**
  - Performance metrics
- **Worst case**
  - Best case?
- **Average case**
  - Average cost?
- **Advanced topics**
  - Lower bound, optimality, …

Algorithm Analysis

- **How to measure**
  - Not too general
    - Giving essential indication in comparison of algorithms
  - Not too precise
    - Machine independent
    - Language independent
    - Programming paradigm independent
    - Implementation independent

Algorithm Analysis

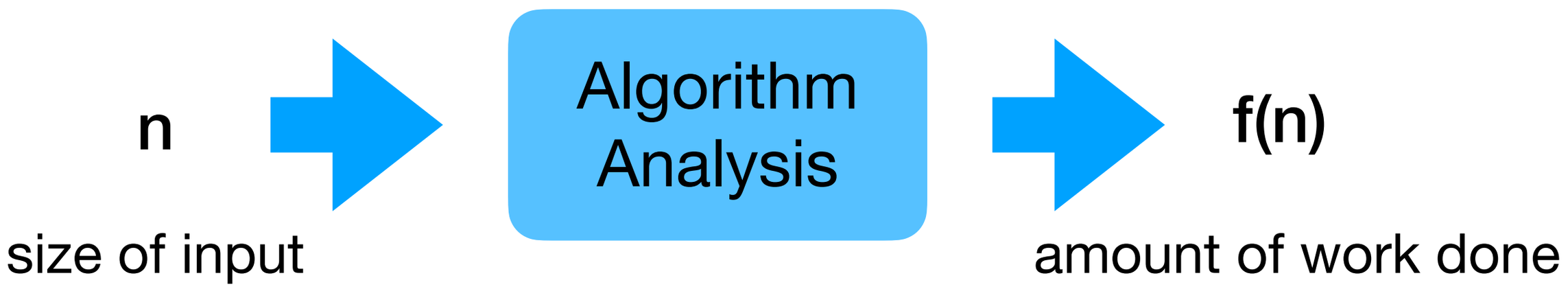


- **Criteria**
  - Critical operation
  - How many critical operation are conducted
- **For example**

Algorithmic problem	Critical operation
Sorting, selection, searching String matching	Comparison (of keys)
Graph traversal	Processing a node/edge
Matrix multiplication	Multiplication

Algorithm Analysis

- **Amount of work done**
  - usually depends on size of the input
  - usually does not depend on size of the input only



Worst-case Complexity

- **W(n)**
  - Upper bound of cost
    - For any possible input
  - $W(n) = \max_{I \in D_n} f(I)$

Average-case Complexity

- **A(n)**
  - Weighted average

$$A(n) = \sum_{I \in D(n)} Pr(I) f(I)$$

- **A special case**
  - Average cost
    - Total cost of all inputs, averaged over the input size
  - $$A(n) = \frac{1}{|D(n)|} \sum_{I \in D(n)} f(I)$$

Average-case Cost of SeqSearch

- **Case 1: K is in E[ ]**
  - Assumptions:
    - Assuming that K is in E[ ]
    - Assuming no same entries in E[ ]
    - Each possible input appears with equality (thus, K in the i^{th} location with probability 1/n)

$$A_{succ}(n) = \sum_{i=0}^{n-1} Pr(I_i|succ)t(I_i) = \sum_{I=0}^{n-1} \frac{1}{n}(i+1) = \frac{n+1}{2}$$

Average-case Cost of SeqSearch

- **Case 2: K may (or may not) be in E[ ]**
  - Assume that K is in E[ ] with probability q

$$A(n) = Pr(succ)A_{succ}(n) + Pr(fail)A_{fail}(n)$$
$$= q\frac{n+1}{2} + (1-q)n$$

How to make reasonable assumptions?

Algorithm Analysis

- **Advanced topics**
  - Lower bound (Selection)
  - Optimality (Greedy, DP)
  - Computation complexity
  - Approximate / online / randomized algorithms

Thank you!

Q & A