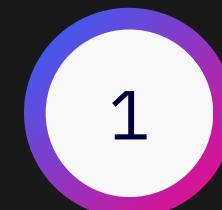


Algorithmic complexity

Presented by
Oleksandr Ukrainets

Зміст презентації



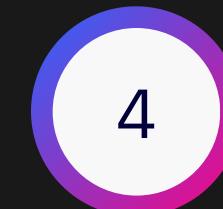
Теорія алгоритмів



Тимчасова складність
алгоритму



Асимптотична
складність



Класи складності
алгоритмів

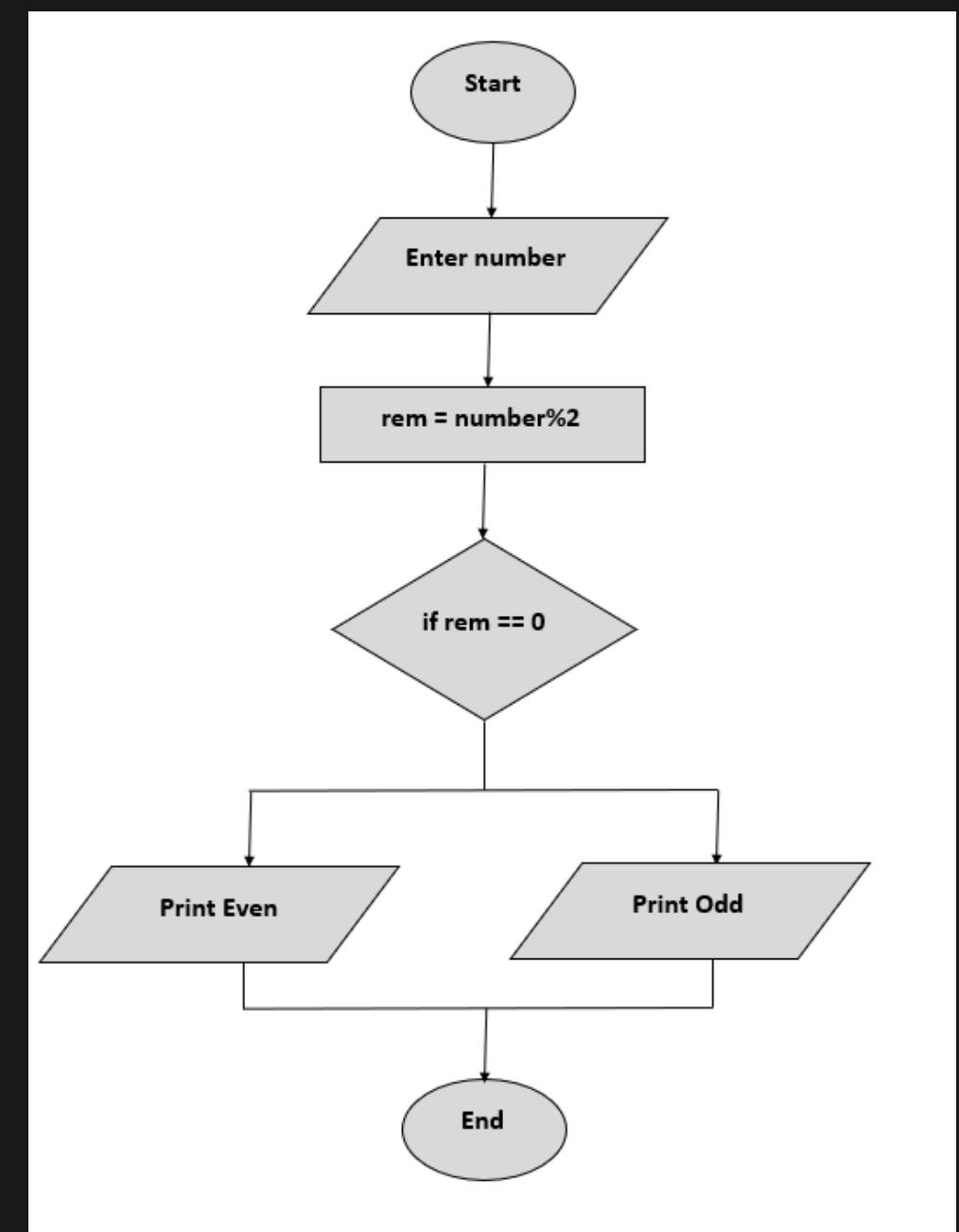


Практика

Теорія алгоритмів

Теорія алгоритмів є галуззю комп'ютерних наук, яка вивчає алгоритми — формальні процедури для вирішення певних завдань.

Метою теорії алгоритмів є розуміння можливостей та обмежень алгоритмів та розробка нових алгоритмів для різних задач.

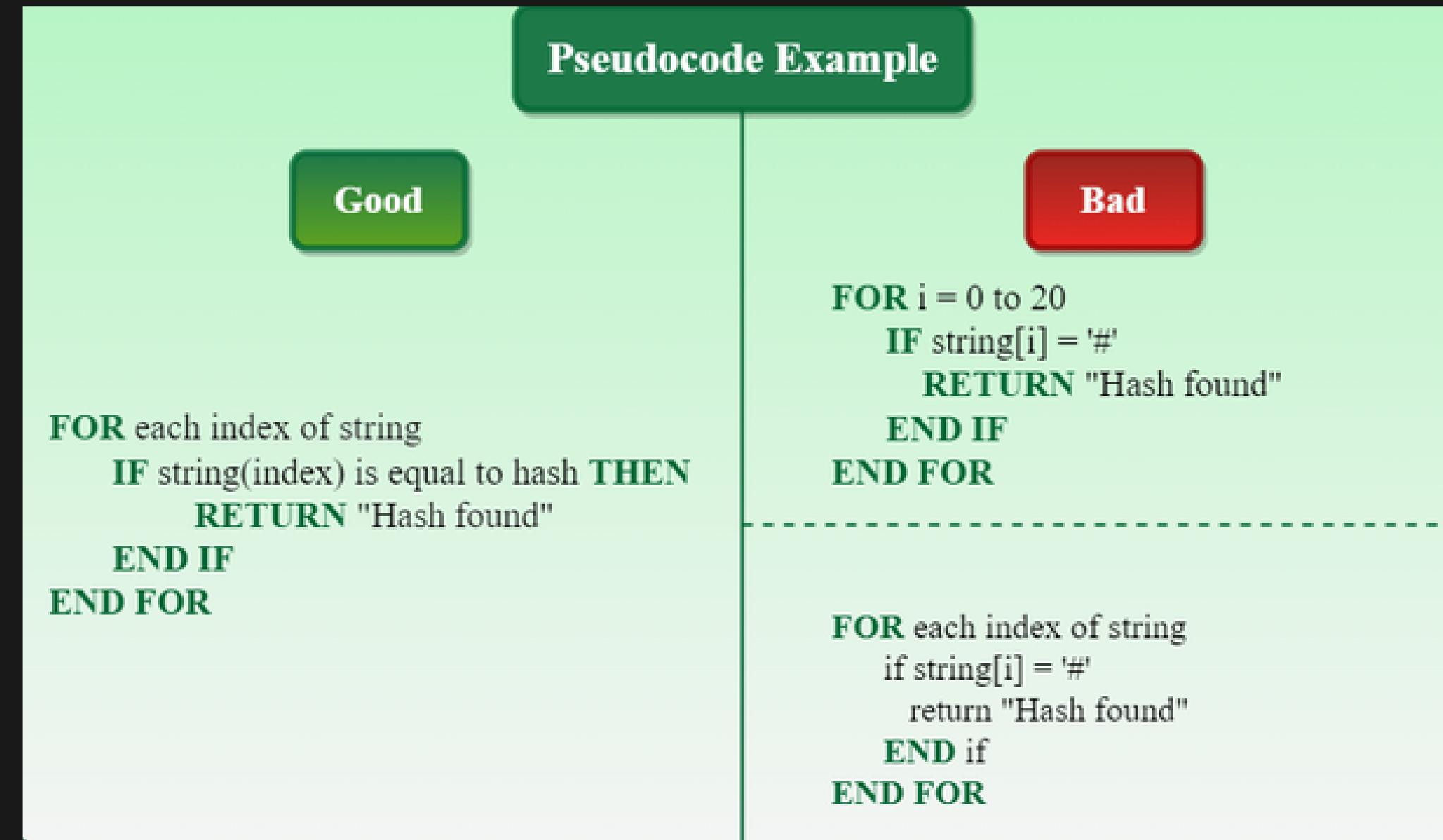


Алгоритм

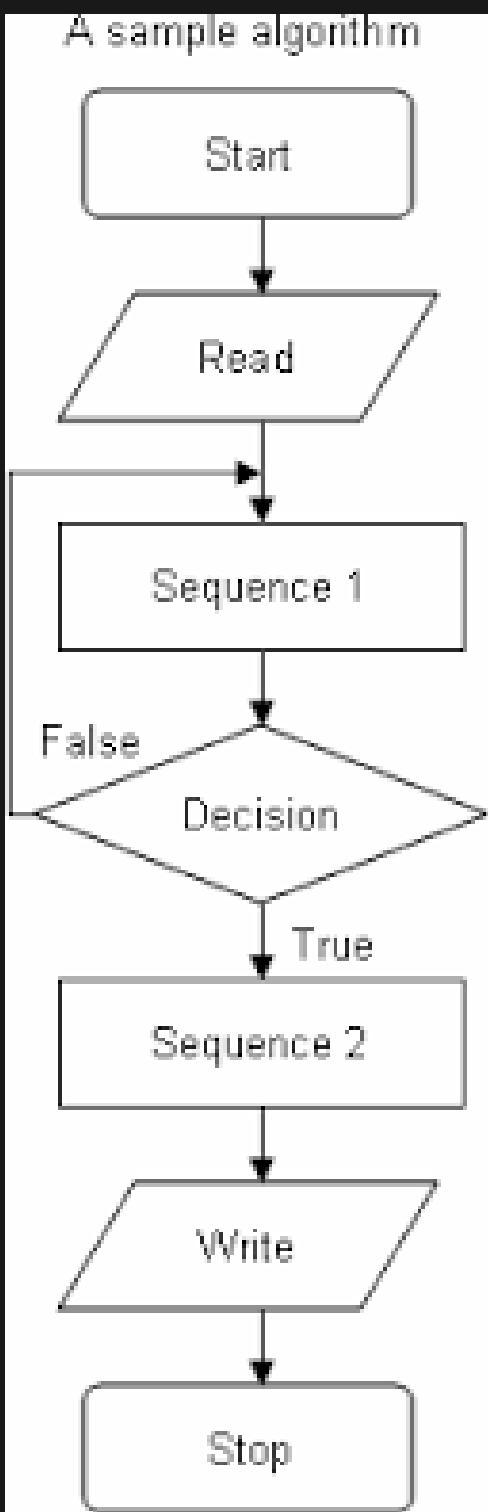
Алгоритм — це точно визначена послідовність інструкцій, призначена для виконання певної задачі або розв'язання певної проблеми. Ця послідовність інструкцій повинна бути чітко визначена та ефективна у виконанні задачі.

Алгоритм може бути виражений у вигляді блок-схеми, псевдокоду, мови програмування або будь-якого іншого способу, який зрозумілий для розробника програмного забезпечення. Його основна мета - розв'язати задачу за допомогою заздалегідь визначених дій.

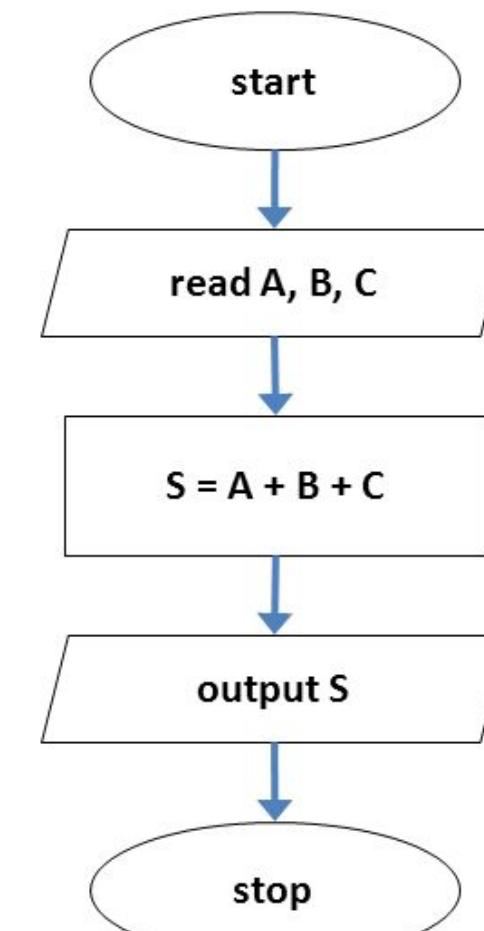
Псевдокод



БЛОК-СХЕМА



Adding the numbers



Завдання, що вирішує теорія алгоритмів

1

Формалізація та опис алгоритмів

Теорія алгоритмів допомагає математично описати та формалізувати алгоритми, що дозволяє їх легко аналізувати та порівнювати між собою.

2

Аналіз та оцінка складності алгоритмів

Теорія алгоритмів дозволяє визначити, наскільки швидко або ефективно працює алгоритм, враховуючи обсяг вхідних даних.

Завдання, що вирішує теорія алгоритмів

3

Розв'язання задач оптимізації

Теорія алгоритмів допомагає розв'язувати задачі оптимізації, які полягають у знаходженні найбільш оптимальних рішень для певної задачі.

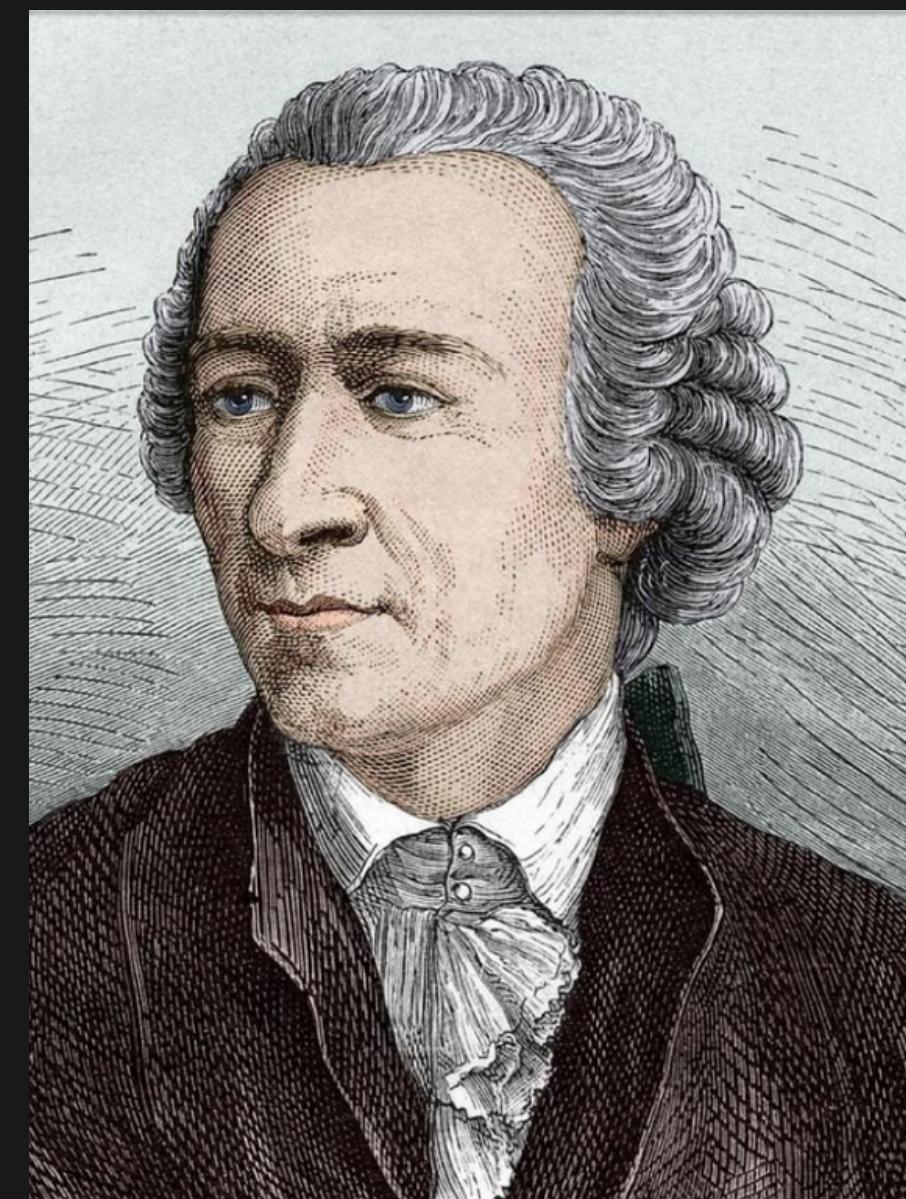
4

Розв'язання задач на графах

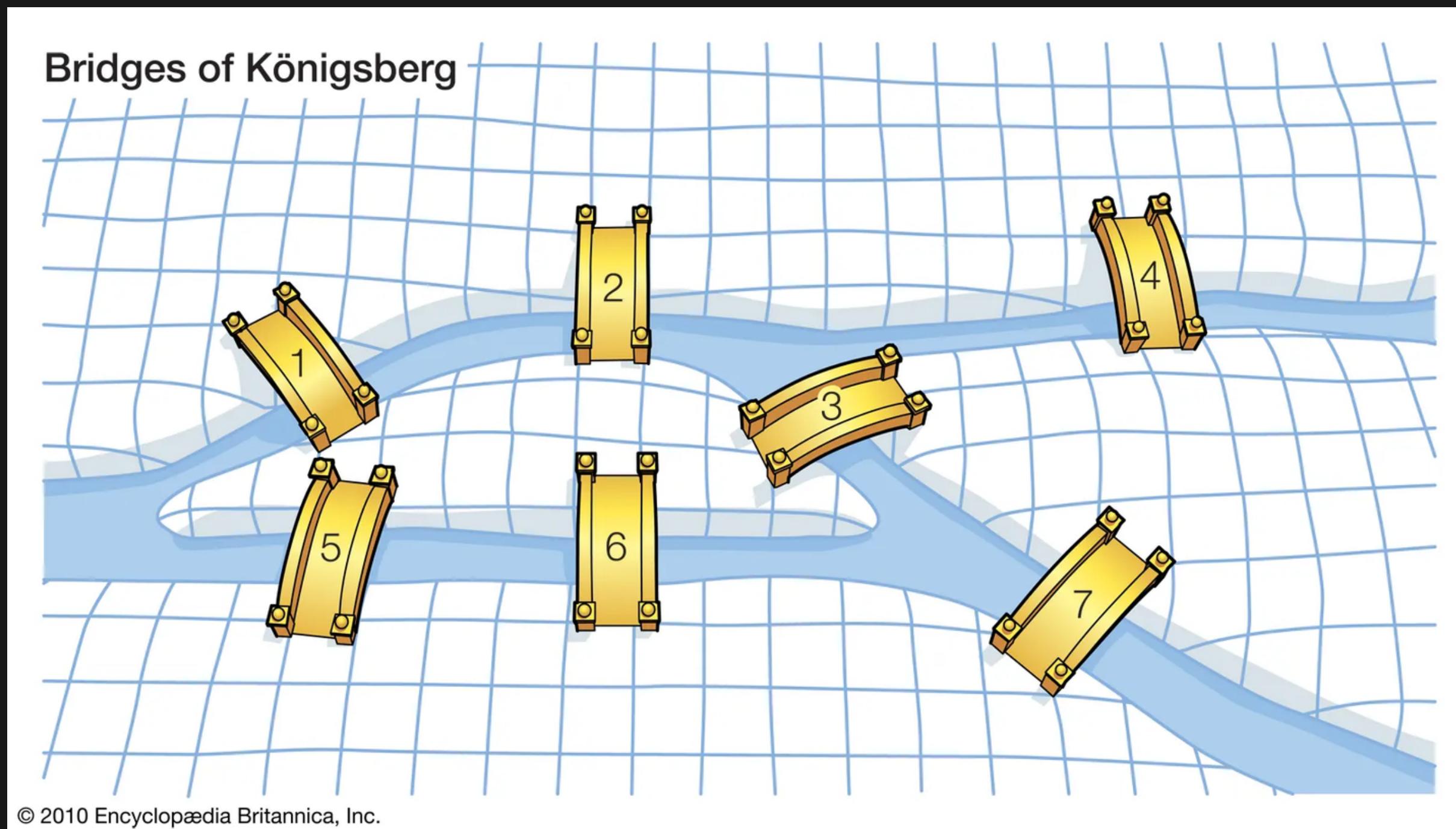
Теорія алгоритмів використовується для розв'язання задач на графах, таких як знаходження найкоротшого шляху, мінімального кістякового дерева, кольорування графу та інших.

Кеннісбергські мости

Задача про Кеннісбергські мости (також відома як задача про сім мостів Кенінгтону) — це математична головоломка, яка полягає в знаходженні маршруту, який проходить через кожний з семи мостів Кеннісберга тільки один раз і повертається в початкову точку маршруту.



Спробуй вирішити



Центральне завдання теорії алгоритмів

— з'ясувати, чи існує алгоритм розв'язання тієї чи іншої задачі.

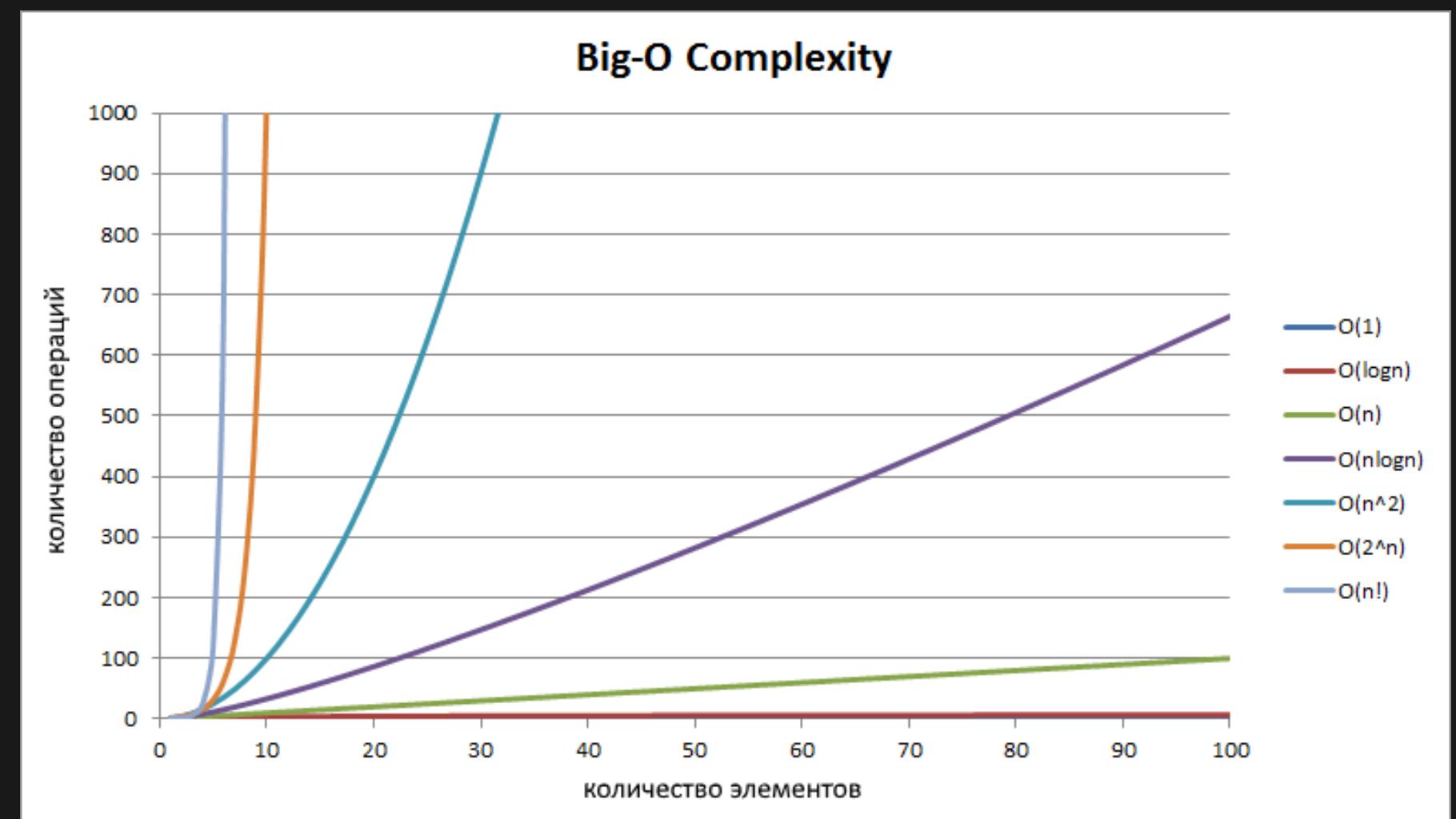
Якщо так, то чи можна ним скористатися на практиці, за сучасного рівня розвитку обчислювальної техніки?

Тобто чи здатен комп'ютер за прийнятний час отримати результат?

Тимчасова складність алгоритму

Виконання будь-якого алгоритму потребує:

1. Певного обсягу пам'яті комп'ютера для розміщення даних та програми
2. Часу процесора для обробки цих даних



Тимчасова складність алгоритму

Тимчасова складність алгоритму $T(n)$ — це функція, яка кожному розміру задачі n ставить у відповідність максимальну кількість елементарних операцій, виконаних під час виконання алгоритму.



Ефективність алгоритму

Ефективним називається алгоритм, що забезпечує найшвидше отримання результату в прийнятний час в обмеженому обсязі оперативної пам'яті.

Для порівняння алгоритмів за ефективністю необхідно вміти оцінювати складність алгоритмів.

Тимчасова складність — швидкодія

Просторова складність — обсяг необхідної пам'яті

Для визначення складності алгоритму оцінюють часову складність алгоритму

Ефективність алгоритму

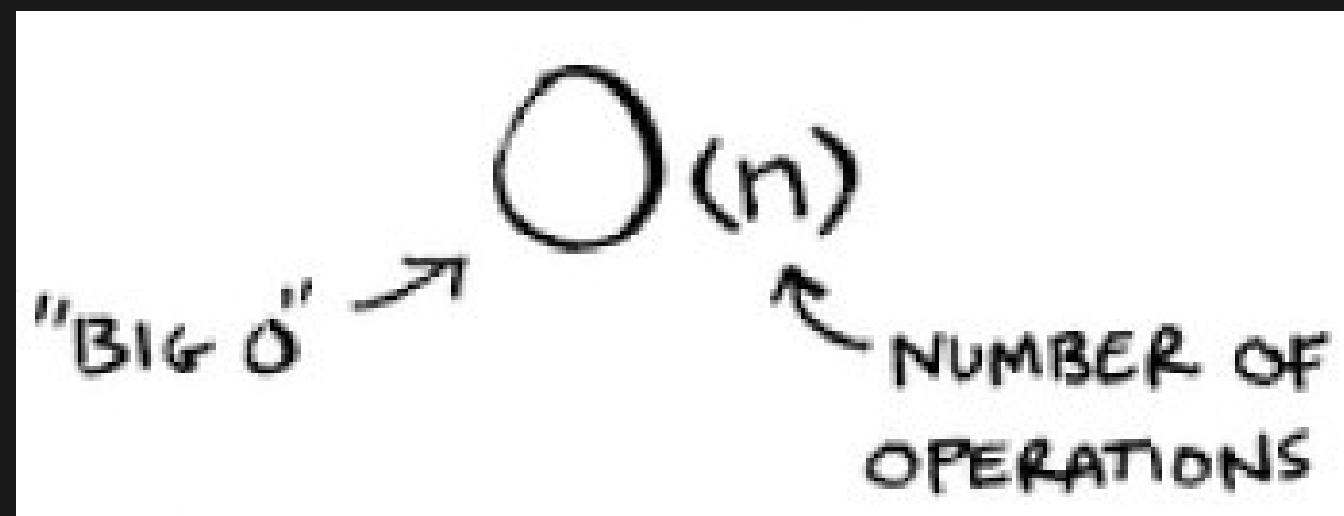
Час роботи алгоритму (T) – кількість виконаних ним елементарних операцій (не в секундах!)

Дозволяє оцінювати саме якість алгоритму, а не властивості виконавця (швидкодію комп'ютера)

Розмір задачі (n) – обсяг вхідних даних, необхідних для її вирішення.

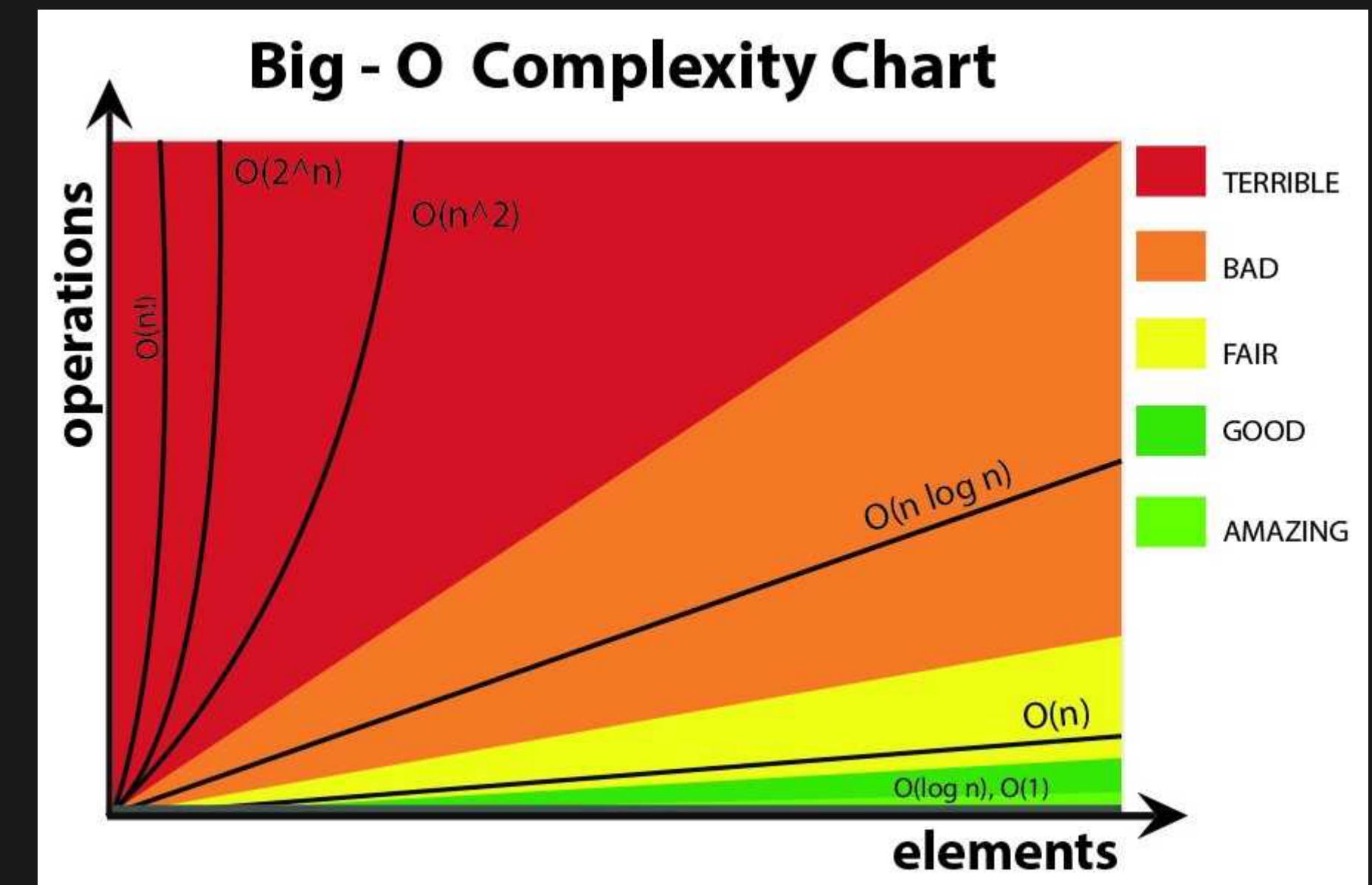
Асимптотична складність

У теоретичній інформатиці під час порівняння алгоритмів використовується їхня асимптотична складність $O(n)$, тобто швидкість зростання кількості операцій за великих значень n .



Класи складності алгоритмів

Безліч обчислювальних проблем, для яких існують алгоритми, схожі за часовою складністю, називається класом складності.



Клас складності $O(1)$

Алгоритми одничної складності

Характеристика: Алгоритми, що використовують частину вхідних даних та ігнорують усі інші дані. Такі алгоритми виконується за один прохід незалежно від значення n

Приклад: Дано масив A довжини n . Обчислити суму перших трьох елементів масиву

Розв'язання цієї задачі містить усього один оператор:

$$S := A[1] + A[2] + A[3], T(n) = 3$$

Задача

Операція доступу до елементу має константну складність $O(1)$, тому що швидкість її виконання не залежить від кількості елементів в масиві. Однак, все змінюється, коли ми намагаємося шукати, видаляти чи рухати елементи в масиві — тоді складність стає $O(n)$.

Чому?

Клас складності $O(n)$

Алгоритми лінійної складності

Характеристика: Для алгоритмів класу $O(n)$ для кожного вхідного елемента виконується тільки одна дія.

Насправді, може бути, звісно, не одна, а дві, три, тощо, але головне не більше $C \cdot N$, де C - константа.

Приклад: програми зі скінченим числом одновимірних циклів, що пробігають увесь набір вхідних даних, що йдуть один за одним (вкладених циклів бути не може) + деякий набір кроків до, після і між циклами.

Клас складності $O(\log n)$

Алгоритми логарифмічної складності

— більшість алгоритмів пошуку.

Algorithm	Best Case	Worst Case
Insertion sort	$O(n)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$
Bubble sort	$O(n^2)$	$O(n^2)$
Quick sort	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$
Bucket Sort	$O(n)$	$O(n \log n)$
Radix Sort	$O(n)$	$O(n \log n)$

Клас складності $O(n^2)$

Алгоритми квадратичної складності

Характеристика: Для алгоритмів класу $O(n^2)$ кожен вхідний елемент обробляється (або проходиться) $C \cdot n^2$ разів.

Це означає (для наочності, тільки як одну з можливостей) наявність вкладених (подвійних) циклів.

Приклад: в основному, всі найпростіші алгоритми сортування

Клас складності $O(n^3)$

Алгоритми кубічної складності

Характеристика: В алгоритмах класу $O(n^3)$ кожен елемент обробляється $C \cdot n^3$ разів (цикл-в-цикли-в-цикли)

Приклад: множення матриць розміру $n \times n$

Thank
you!

Another beautiful day
without using recursion,
binary tree, graphs,
bubble sort, linked list,
memoization, hash map

