



COMPLETE  
SEARCH:  
BRUTE FORCE

## TABLE OF CONTENTS

01

### DEFINITION

Brute Force là gì ?

03

### PROS & CONS

Ưu và nhược điểm  
của Brute Force

02

### APPLICATIONS

Một số thuật toán  
tiêu biểu sử dụng  
Brute Force

04

### EXERCISE

Bài tập thực hành tại  
lớp

01

# DEFINITION

Brute Force là gì?





# Thuật ngữ cây trầu

## Brute Force là gì ?

- Là một thuật toán vét cạn, thuật toán này sẽ chạy tất cả các trường hợp có thể để giải quyết một vấn đề nào đó (Bao gồm cả trường hợp đúng và các trường hợp sai hay còn gọi là trường hợp dư thừa)
- Là cách tiếp cận đơn giản để giải quyết bài toán thường trực tiếp dựa trên đề bài và định nghĩa của các khái niệm liên quan

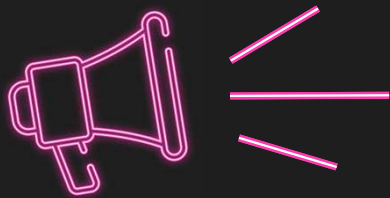


VÍ DỤ:

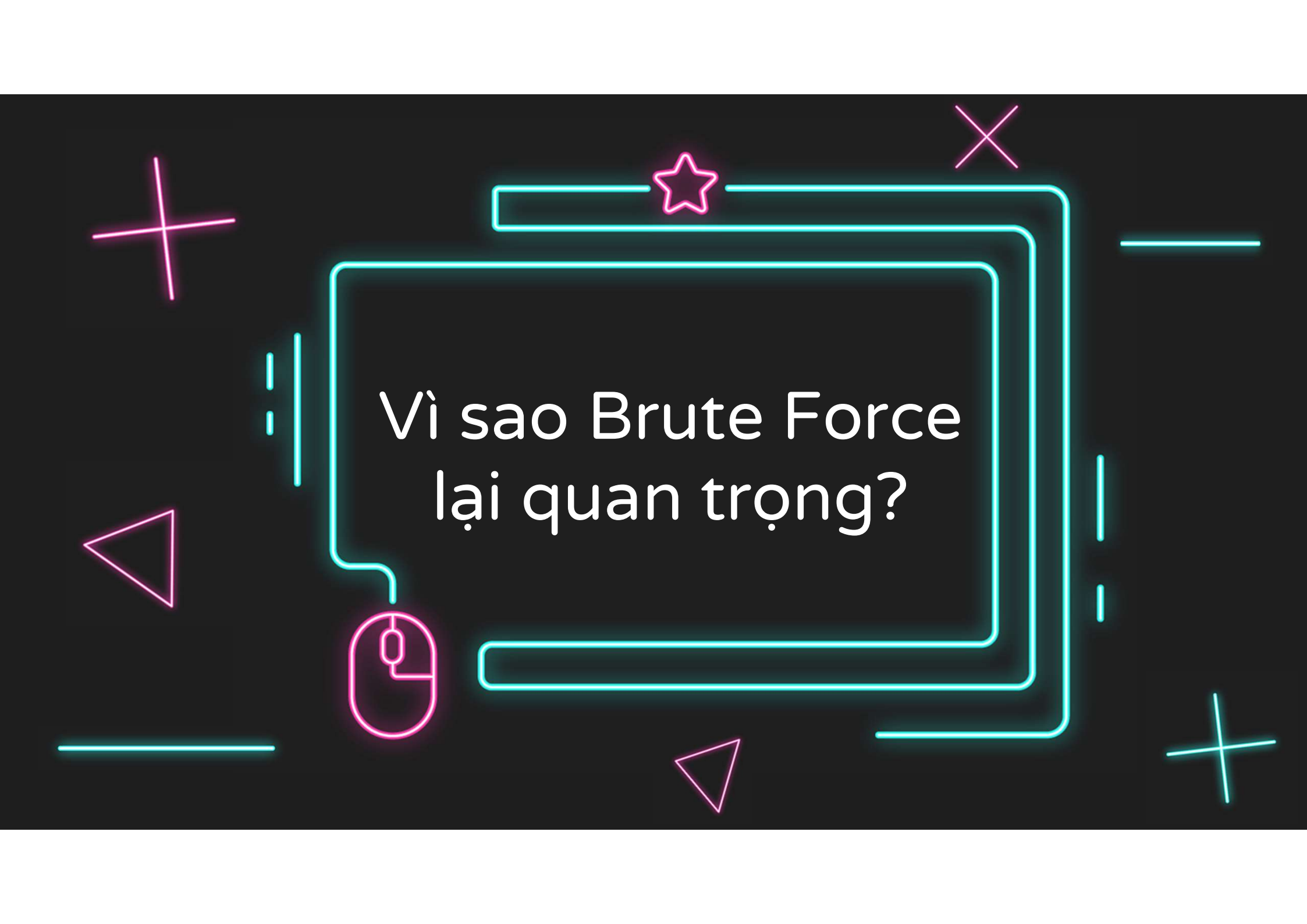
Phép tính lũy thừa

$$a^n = \underbrace{a * \dots * a}_{n \text{ times}}$$

# Độ phức tạp



$O(mn)$



Vì sao Brute Force  
lại quan trọng?



# LÝ DO BRUTE FORCE LẠI QUAN TRỌNG

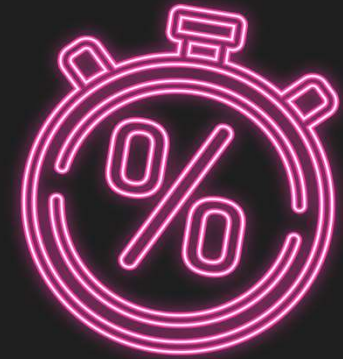


## Thứ hai

Trong một số bài toán như: sắp xếp, tìm kiếm, nhân ma trận,... Brute Force cho ra các thuật toán hợp lý có ít nhất vài giá trị thực tiễn mà không bị giới hạn về kích thước đối tượng.

## Đầu tiên

Được sử dụng trong một số mục đích nghiên cứu và giáo dục để đánh giá hiệu năng của các giải thuật khác



# LÝ DO BRUTE FORCE LẠI QUAN TRỌNG

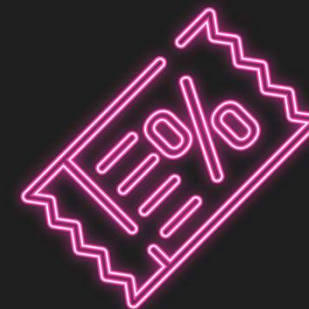


## Thứ 4

Dù hiệu năng không cao nhưng thuật toán vẫn hiệu quả với những bài toán có độ phức tạp nhỏ

## Thứ 3

Sự hao phí trong việc thiết kế ra thuật toán mới hiệu năng hơn là không cần thiết nếu bài toán có độ phức tạp không quá lớn, Brute Force có giải quyết các vấn đề đó với tốc độ phù hợp với bài toán



## Cuối cùng

Áp dụng rộng rãi để giải nhiều bài toán



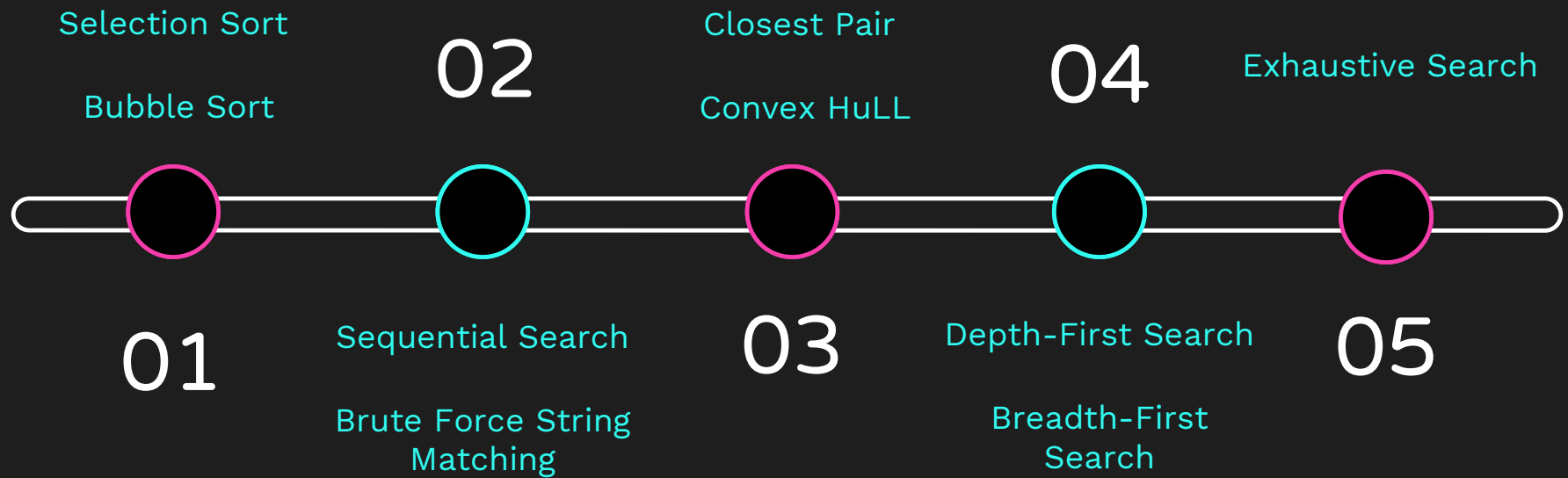
02

02

# APPLICATIONS

Một số thuật toán tiêu  
biểu ứng dụng Brute Force

# APPLICATIONS

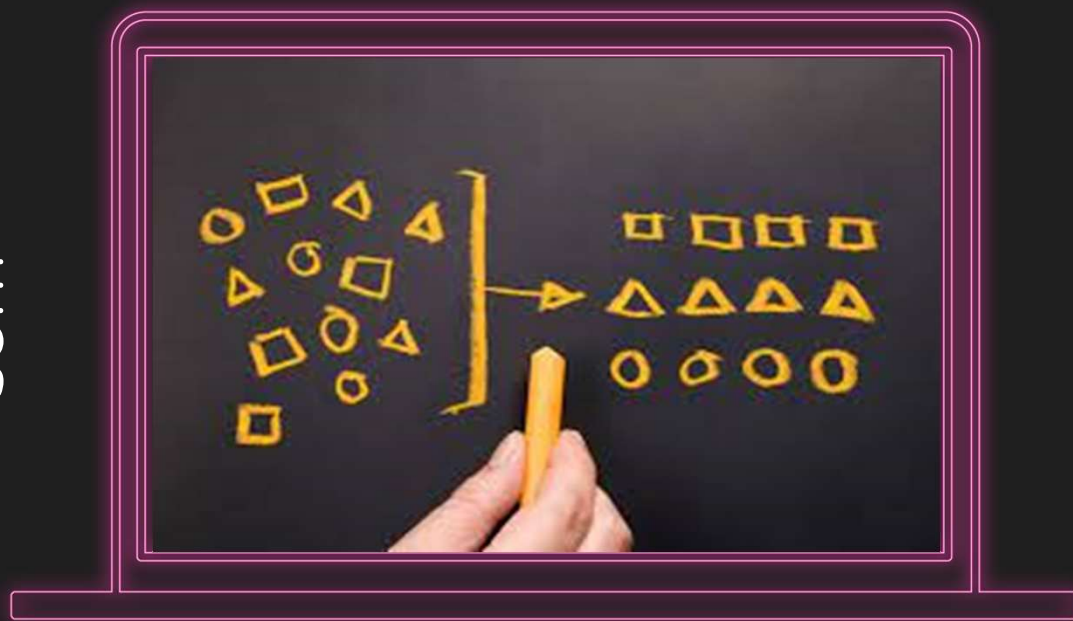




# Selection Sort & Bubble Sort



SORT



## Sắp xếp

Là quá trình bố trí lại các phần tử trong một tập hợp theo một **trình tự** nào đó giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn



## Phương pháp Chọn trực tiếp

(Mô phỏng một trong những cách sắp xếp tự nhiên nhất trong thực tế)

### Các bước

Chọn phần tử nhỏ nhất trong  $n$  phần tử ban đầu, đưa về đầu dãy

Xem dãy như chỉ còn  $n - 1$  phần tử kia, lặp lại bước trên cho đến khi chỉ còn 1 phần tử



## Thuật toán Selection Sort

```
// Sắp xếp một mảng đã cho theo Selection Sort  
// Đầu vào : Một mảng A[0..n-1], các phần tử có thể sắp xếp  
// Đầu ra : Mảng A[0..n-1] được sắp xếp theo thứ tự không giảm
```

```
for i ← 0 to n - 2 do  
    min ← i  
    for j ← i + 1 to n - 1 do  
        if A[i] > A[j]: min ← j  
    swap A[i] and A[min]
```

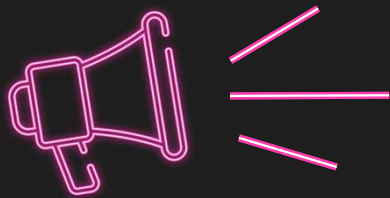


```
for i ← 0 to n - 2 do
  min ← i
  for j ← i + 1 to n - 1 do
    if A[i] > A[j]: min ← j
  swap A[i] and A[min]
```

## ĐÁNH GIÁ

Ở lượt thứ  $i$ , cần  $n - i$  lần so sánh để xác định phần tử nhỏ nhất hiện hành  
Số lượng phép so sánh không phụ thuộc vào tình trạng của dãy số ban đầu

# Độ phức tạp



$O(n^2)$



## Nhược điểm

Chỉ được áp dụng trong các trường hợp có số lượng phần tử so sánh ít  
Không nhận biết được mảng đã sắp xếp

## BUBBLE SORT

### Phương pháp nổi bọt

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn về đầu dãy, không xét nó trong các bước sau
- Trong vòng lặp  $i$ , vị trí đầu dãy là  $i$
- Lặp lại đến khi không còn cặp nào

## Thuật toán BubbleSort

```
// Sắp xếp một mảng đã cho theo Bubble Sort  
// Đầu vào : Một mảng A[0..n-1], các phần tử có thể sắp xếp  
// Đầu ra : Mảng A[0..n-1] được sắp xếp theo thứ tự không giảm
```

```
for i ← 0 to n - 2 do  
    for j ← 0 to n - 2 - i do  
        if A[j + 1] > A[j]  
            swap A[j] and A[j + 1]
```

## Thuật toán BubbleSort

### ĐÁNH GIÁ

- Số lượng phép so sánh không phụ thuộc vào tình trạng của dãy số ban đầu
- Số lượng phép hoán vị thực hiện tùy thuộc vào kết quả so sánh

```
for i ← 0 to n - 2 do
  for j ← 0 to n - 2 - i do
    if A[j + 1] > A[j]
      swap A[j] and A[j + 1]
```

# KHUYẾT ĐIỂM



- Không nhận diện được tình trạng dãy đã có thứ tự hay có thứ tự từng phần
- Các phần tử nhỏ được đưa về vị trí đúng rất nhanh, trong khi các phần tử lớn thì rất chậm

$O(n^2)$





## BÀI TẬP

Cho dãy chưa sắp  $\text{arr}[] = 64\ 25\ 12\ 22\ 11$

Hãy sắp xếp mảng này theo 2  
phương pháp Selection sort and  
Bubble Sort.

# LỜI GIẢI:

## Theo hướng Selection sort

```
A = [64, 25, 12, 22, 11]
for i in range(len(A)):
    # Tìm kiếm phần tử bé nhất trong dãy chưa được sắp xếp
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j
    # Đổi chỗ phần tử bé nhất với phần tử đầu tiên
    A[i], A[min_idx] = A[min_idx], A[i]
# Test Kết Quả
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i])
```

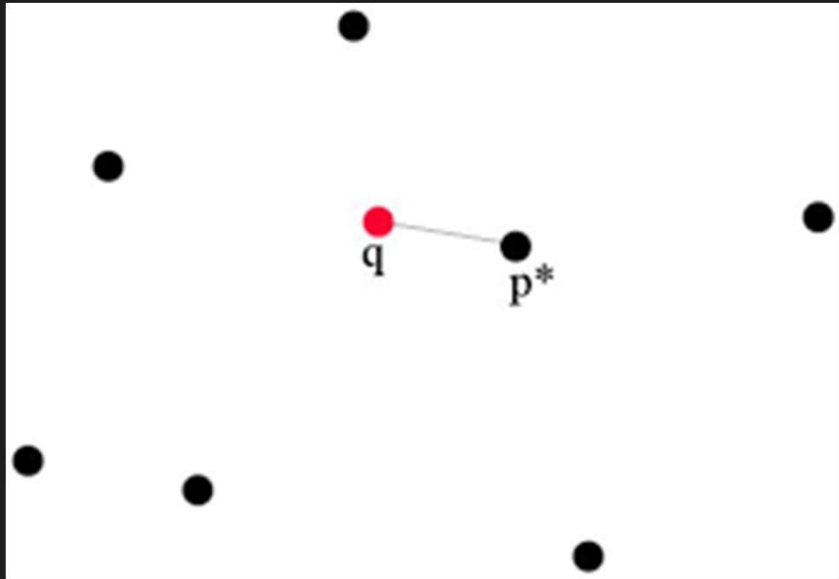
# LỜI GIẢI:

## Theo hướng Bubble sort

```
def bubbleSort(arr):  
    n = len(arr)  
    # Duyệt Tất Cả Phần Tử Của Mảng  
    for i in range(n):  
        for j in range(0, n-i-1):  
            # Duyệt phần tử trong mảng từ 0 to n-i-1  
            # Đổi chỗ khi tìm thấy phần tử lớn hơn  
            if arr[j] > arr[j+1]:  
                arr[j], arr[j+1] = arr[j+1], arr[j]  
# Chạy hàm vừa viết arr = [64, 25, 12, 22, 11]  
bubbleSort(arr)  
print ("Sorted array is:")  
for i in range(len(arr)):  
    print ("%d" %arr[i])
```

# Bài toán tìm cặp điểm gần nhất

Closest-Pair Problem



## Closest-Pair Problem

Yêu cầu tìm cặp điểm gần nhất trên tập  $n$  điểm

## MỘT ỨNG DỤNG CỦA BÀI TOÁN (Không phải của Brute Force)

Một ứng dụng quan trọng của bài toán này là phân tích cụm trong thống kê. Từ  $n$  điểm dữ liệu, công việc phân tích cụm phân cấp yêu cầu tổ chức dữ liệu trong một hệ phân cấp của các đám dựa theo các chỉ số tương tự nhau.

Đối với dữ liệu kiểu số, chỉ số này thường là khoảng cách Euclid. Đối với dữ liệu văn bản hay không phải số, người ta dùng các chỉ số như khoảng cách Hamming.



## Xét trường hợp 2 chiều

Mỗi điểm được xác định bởi tọa độ Đề-các  $(x, y)$

Khoảng cách giữa hai điểm là khoảng cách Euclid chuẩn.

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

## Closest-Pair Problem (theo hướng Brute Force)

```
// Tìm khoảng cách ngắn nhất giữa hai điểm trong tập n điểm  
// Đầu vào : Tập n điểm, mỗi điểm có tọa độ (x[i], y[i]), i=1..n  
// Đầu ra : Khoảng cách ngắn nhất giữa hai điểm trên tập
```

```
d ← infinity  
for i = 1 to n - 1 do  
    for j = i + 1 to n do  
        d ← min(d, sqrt((x[i] - x[j])^2  
                        + (y[i] - y[j])^2))  
return d
```

Có thể bỏ



## Closest-Pair Problem (theo hướng Brute Force)

```
// Tìm khoảng cách ngắn nhất giữa hai điểm trong tập n điểm  
// Đầu vào : Tập n điểm, mỗi điểm có tọa độ (x[i], y[i]), i=1..n  
// Đầu ra : Khoảng cách ngắn nhất giữa hai điểm trên tập
```


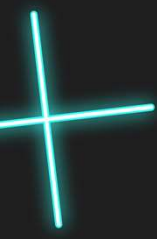
```
d ← infinity  
for i = 1 to n - 1 do  
    for j = i + 1 to n do  
        d ← min(d, (x[i]-x[j])^2 + (y[i]-y[j])^2)  
return d
```

Basic  
Operation


Dù đã  
cải thiện...

$O(n^2)$

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n 2 = 2 \sum_{i=1}^{n-1} (n-i) = (n-1)n$$
$$\approx n^2 \in \Theta(n^2)$$



Các nhóm sau sẽ giúp  
các bạn cải thiện  
thuật toán này



# BÀI TẬP

Bài toán tìm cặp điểm gần nhất có thể được biểu diễn trong không gian  $k$  chiều, trong đó khoảng cách Euclid giữa hai điểm  $p'(x'_1, \dots, x'_k)$  và  $p''(x''_1, \dots, x''_k)$  được định nghĩa như sau :

$$d(p', p'') = \sqrt{\sum_{s=1}^k (x'_s - x''_s)^2} .$$

Độ phức tạp thời gian của thuật toán Brute Force cho bài toán tìm cặp điểm gần nhất  $k$  chiều là gì ?

LỜI GIẢI

$$\Theta(n^2)$$



# EXHAUSTIVE SEARCH

Tìm kiếm toàn diện

# EXHAUSTIVE SEARCH

- Là tập con của thuật toán Brute Force.
- Dùng để tìm kiếm kết hợp và hoán vị trong một số bài toán đơn giản.
- Tập trung vào việc tìm ra giải pháp cho bài toán bằng cách chọn 1 trong số các phần tử thỏa mãn tất cả các ràng buộc, sau đó tìm ra phần tử thích hợp (phần tử tối ưu hóa hàm mục tiêu)

Traveling  
Salesman  
Problem




Knapsack  
Problem

Assignment  
Problem



The background is dark gray with several neon-colored geometric shapes. There are three triangles: one pink in the top-left, one pink in the top-right, and one cyan in the middle-left. There are also two plus signs: one pink in the bottom-left and one cyan in the bottom-right. A large, rounded rectangle with a pink border is centered on the slide.



# TRAVELING SALESMAN PROBLEM



Có một người giao hàng cần giao hàng tại  $n$  thành phố. Người đó xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu.

Người đó chỉ được đến mỗi thành phố một lần và khoảng cách từ mỗi thành phố đến các thành phố khác đã được xác định. Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, cũng có thể là cước phí hoặc thời gian di chuyển. Ta gọi chung là “độ dài”.

Hãy tìm một chu trình (đường đi khép kín) thỏa mãn điều kiện trên, sao cho tổng độ dài các cạnh là nhỏ nhất.



## Traveling Salesman Problem:



Bài toán có thể được biểu diễn bởi một đồ thị  $G(V, E)$  vô hướng có trọng số, trong đó mỗi thành phố được biểu diễn bởi một đỉnh, cạnh nối hai đỉnh biểu diễn đường đi giữa hai thành phố.



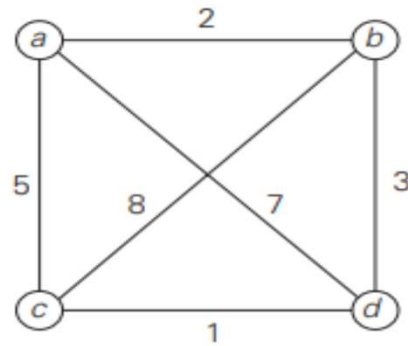
Một chu trình đi qua tất cả các đỉnh của  $G$ , mỗi đỉnh một lần duy nhất



Được gọi là chu trình Hamilton.



Bài toán trở thành bài toán tìm chu trình Hamilton mà tổng độ dài các cạnh là nhỏ nhất.



<u>Tour</u>	<u>Length</u>	
$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$	$l = 2 + 8 + 1 + 7 = 18$	
$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$	$l = 2 + 3 + 1 + 5 = 11$	optimal
$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$	$l = 5 + 8 + 3 + 7 = 23$	
$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$	$l = 5 + 1 + 3 + 2 = 11$	optimal
$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$	$l = 7 + 3 + 8 + 5 = 23$	
$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$	$l = 7 + 1 + 8 + 2 = 18$	

**FIGURE 3.7** Solution to a small instance of the traveling salesman problem by exhaustive search.

## LỜI GIẢI



Đường đi:  $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 1 = 6 + 3 + 4 + 3 + 3 + 1 = 20$

$$\frac{1}{2}(n-1)!$$

- Có thể tìm ra tất cả đường
- Tối ưu hóa và giảm chi phí trong quá trình giao hàng chặng cuối (last-mile deliveries)

ƯU ĐIỂM

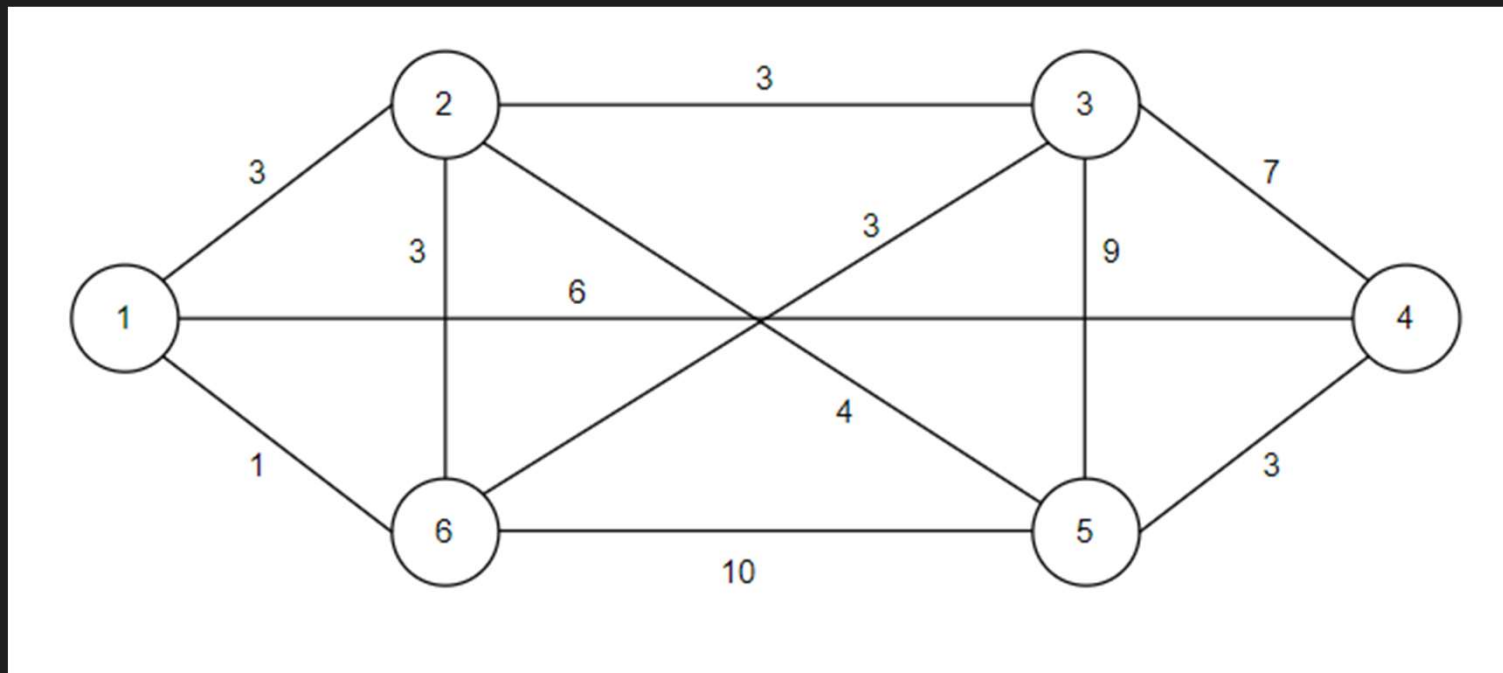


- Exhaustive Search tiêu tốn rất nhiều thời gian và tài nguyên
- Tập giải pháp có thể tăng theo cấp số nhân theo kích cỡ bài toán

HẠN CHẾ



Cho đồ thị vô hướng, liên thông, có trọng số sau:



Bắt đầu từ đỉnh 1, áp dụng thuật toán Exhaustive search vào đồ thị trên hãy chỉ ra chu trình Hamilton có có đường đi ngắn nhất.

A large, stylized number '03' rendered in a pink, double-outlined font. The '0' is a simple oval, and the '3' has a cursive, flowing design. The entire graphic is set against a dark, solid background.

03

ADVANTAGES AND DISADVANTAGES

## ADVANTAGES AND DISADVANTAGES

### Pros:

Có thể áp dụng không giới hạn trong 1 chủ đề nào

---

Phương pháp lý tưởng cho các bài toán nhỏ và đơn giản

---

Có thể so sánh với các thuật toán hiệu quả hơn, nhằm nhấn mạnh tính ưu việt của thuật toán đó

---

Tìm được tất cả các giải pháp khả thi để giải quyết bài toán

### Cons:

Không hiệu quả

---

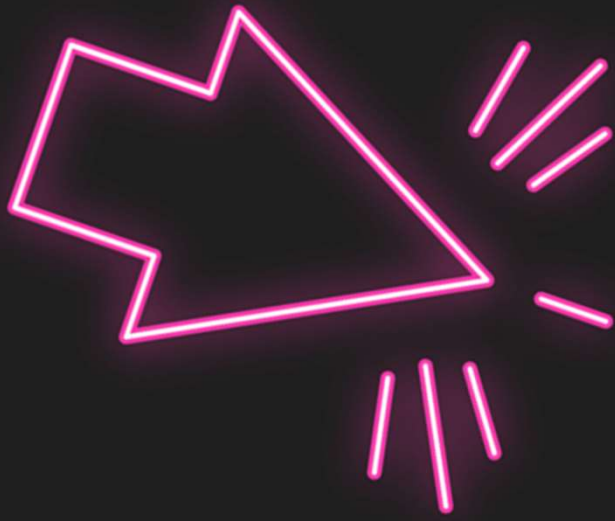
Chậm

---

Phụ thuộc vào hiệu suất của máy tính hơn là thiết kế thuật toán

---

Không sáng tạo so với nhiều thuật toán khác



# THANKS!

Huỳnh Nhân Thập - 21521457  
Đỗ Bá Huy - 21522137  
Nguyễn Tường Duy - 21520782

