TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI GIỮA KÌ MÔN TOÁN TỔ HỢP VÀ ĐỒ THỊ**

**XÂY DỰNG CÂU ĐỐ SUDOKU BẰNG PHƯƠNG PHÁP HOÁN VỊ**

*Người hướng dẫn*: **NGUYỄN CHÍ THIỆN**

*Người thực hiện*: **NGUYỄN QUỐC CƯỜNG**

Lớp **: 18H50203**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI GIỮA KÌ MÔN TOÁN TỔ HỢP VÀ ĐỒ THỊ**

**XÂY DỰNG CÂU ĐỐ SUDOKU BẰNG PHƯƠNG PHÁP HOÁN VỊ**

*Người hướng dẫn*: **NGUYỄN CHÍ THIỆN**

*Người thực hiện*: **NGUYỄN QUỐC CƯỜNG**

Lớp **: 18H50203**

Khoá  **: 22**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2020**

# LỜI CẢM ƠN

Xin gửi lời cảm ơn chân thành đến thầy Nguyễn Chí Thiện, dù trong thời điểm dịch bệnh không thể đến trường mà thầy vẫn đã hổ trợ tôi rất tốt để có thể hoàn thành được bài tiểu luận này.

**ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy Nguyễn Chí Thiện;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 19 tháng 11 năm 2020*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Quốc Cường*

# PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

## Phần xác nhận của GV hướng dẫn

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

## Phần đánh giá của GV chấm bài

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

MỤC LỤC

[LỜI CẢM ƠN 3](#_Toc56275483)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN 5](#_Toc56275484)

[1. Phần xác nhận của GV hướng dẫn 5](#_Toc56275485)

[2. Phần đánh giá của GV chấm bài 5](#_Toc56275486)

[CHƯƠNG I: THUẬT TOÁN BRUTE – FORCE 7](#_Toc56275487)

[1. Giới thiệu về brute - force: 7](#_Toc56275488)

[2. Ý tưởng của thuật toán: 7](#_Toc56275489)

[3. Các ví dụ cho thuật toán : 7](#_Toc56275490)

[a. Thuật toán sắp xếp chọn: 7](#_Toc56275491)

[b. Thuật toán sắp xếp bong bóng: 9](#_Toc56275492)

[c. Thuật toán tìm kiếm tuần tự: 11](#_Toc56275493)

[CHƯƠNG II: THUẬT TOÁN DIVIDE AND CONQUER 13](#_Toc56275494)

[1. Giới thiệu về divide and conquer: 13](#_Toc56275495)

[2. Ý tưởng của thuật toán: 13](#_Toc56275496)

[3. Các ví dụ cho thuật toán : 13](#_Toc56275497)

[a. Thuật toán sắp xếp chọn: 13](#_Toc56275498)

[b. Thuật toán sắp xếp bong bóng: 15](#_Toc56275499)

[c. Thuật toán tìm kiếm tuần tự: 17](#_Toc56275500)

# CHƯƠNG I: THUẬT TOÁN BRUTE – FORCE

## Giới thiệu về brute - force:

* Thuật toán vét cạn (brute-force) là một dạng giải quyết vấn đề một cách chân thật nhất, thông thường là dựa vào vấn đề được đặt ra và định nghĩa của các khái niệm có liên quan.

## Ý tưởng của thuật toán:

* “Đẩy tới” (force) bao hàm bởi định nghĩa của chiến lược là định nghĩa của máy tính chứ không phải là định nghĩa của trí tuệ.
* “Hãy làm đi” (just do it) có thể là một cách khác để mô tả qui tắc của các tiếp cận vét cạn. Và thông thường, chiến thuật vét cạn là một trong những cách đơn giản nhất được áp dụng.

## Các ví dụ cho thuật toán :

### Thuật toán sắp xếp chọn:

#### Ý tưởng giải thuật :

* Chúng ta bắt đầu thực hiện phương pháp sắp xếp chọn bằng cách quét toàn bộ danh sách được cho, sau đó tiếp tục tìm ra phần tử nhỏ nhất trong danh sách và đổi chỗ với phần tử đầu tiên trong danh sách. Kế đến, chúng ta tiếp tục quét danh sách và bắt đầu với phần tử thứ hai để tìm ra phần tử nhỏ nhất trong số n-1 phần tử và đổi chỗ với phần tử thứ 2 của danh sách. Nói tóm lại, phần tử thứ i mỗi lần quét trong danh sách, chúng ta sẽ đánh số từ 0 tới n-2, giải thuật tìm kiếm phần tử nhỏ nhất trong số n-1 phần tử cuối và đổi với Ai :

Trong vị trí cuối cùng của danh sách Phần tử n – i cuối cùng

* Sau khi quét qua n – 1 lần, danh sách đã được sắp xếp.

#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | SelectionSort(A[**0.**.n−**1**])  # Sorts a given array by selection sort  # Input: An array A[0..n − 1] of orderable elements  # Output: Array A[0..n − 1] sorted in nondecreasing order  #import time  **for** i ← **0** to n − **2** do  min ← i  **for** j ← i + **1** to n − **1** do  **if** A[j] < A[min] min ← j  swap A[i] **and** A[min] | |

#### Phân tích giải thuật:

* Kích thước đầu vào được cho bởi tổng số các phần tử n; phép toán cơ bản là só sánh ở dòng thứ 8 “A[j] < A[min]”. Tổng số lần thực thi chỉ phụ thuộc vào kích thước của mảng được tính bằng công thức sau:

|  |
| --- |
|  |

#### Triển khai với python3:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # Sắp xếp mảng cho được cho bằng phương pháp sắp xếp chọn  # Input: Một mảng A[0..n−1] chưa được sắp xếp  # Output: Một mảng A[0..n−1] được sắp xếp theo thứ tự tăng dần  **def** **SelectionSort**(A):  **for** i **in** range(len(A)):  min = i  **for** j **in** range(i+**1**,len(A)):  **if** A[j] < A[min]:  min = j  A[i],A[min] = A[min],A[i]  A = [**64**, **25**, **12**, **22**, **11**]  SelectionSort(A)  print(A)  #Output : [11, 12, 22, 25, 64] | |

#### Demo:

### Thuật toán sắp xếp bong bóng:

#### Ý tưởng giải thuật :

* Một ứng dụng khác của thuật toán brute-force trong việc sắp xếp là so sánh các phần tử liền kề của danh sách và đổi chỗ chúng nếu chúng không theo thứ tự. Bằng việc lập đi lập lại, chúng ta sẽ đẩy phần tử lớn nhất lên vị trí cuối cùng trong danh sách. Kế đến là thực hiện tương tự với phần tử lớn thứ 2, và cứ tiếp tục như vậy, cho đến khi sau lần thứ n - 1, danh sách được sắp xếp. Vượt qua i (0 ≤ i ≤ n - 2) của sắp xếp bong bóng có thể được biểu diễn bằng sơ đồ sau:

Trong vị trí cuối cùng của danh sách

#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7 | BubbleSort(A[**0.**.n − **1**])  # Sorts a given array by bubble sort  # Input: An array A[0..n − 1] of orderable elements  # Output: Array A[0..n − 1] sorted in nondecreasing order import time  **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**] | |

#### Phân tích giải thuật:

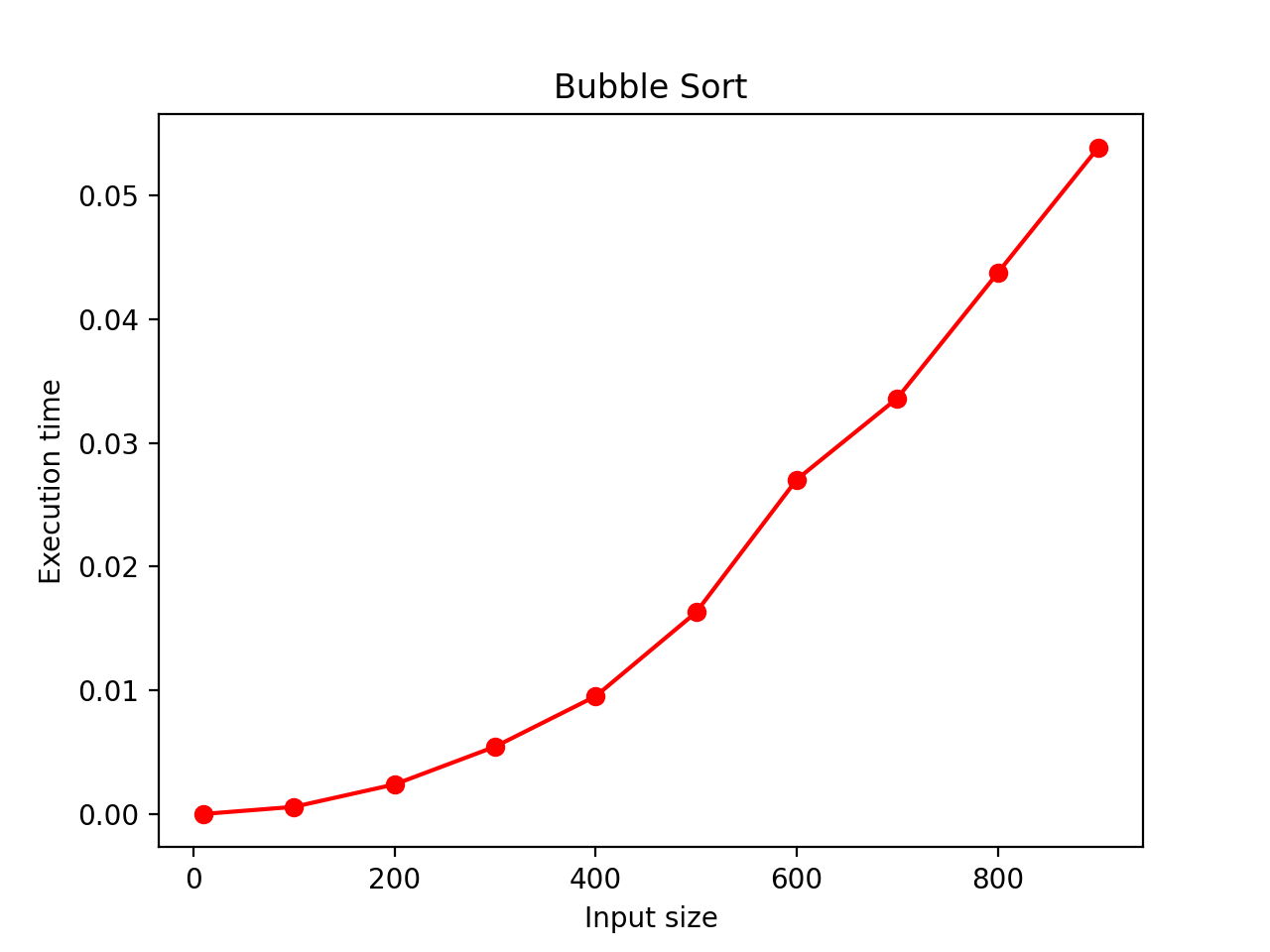
* Kích thước đầu vào được cho bởi tổng số các phần tử n; phép toán cơ bản là só sánh ở dòng thứ 7 “A[j+1] < A[j]”. Tổng số lần thực thi chỉ phụ thuộc vào kích thước của mảng được tính bằng công thức sau:

|  |
| --- |
|  |

#### Triển khai với python3:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12 | # Sorts a given array by bubble sort  # Input: An array A[0..n − 1] of orderable elements  # Output: Array A[0..n − 1] sorted in nondecreasing order import time  **def** **BubbleSort**(A):  **for** i **in** range(len(A)-**1**):  **for** j **in** range(len(A)-**1**-i):  **if** A[j+**1**] < A[j]:  A[j],A[j+**1**] = A[j+**1**],A[j]  A = [**64**, **25**, **12**, **22**, **11**]  BubbleSort(A)  print(A)  #Output : [11, 12, 22, 25, 64] | |

#### Demo:



### Thuật toán tìm kiếm tuần tự:

#### Ý tưởng giải thuật :

* Thuật toán chỉ cần so sánh các phần tử liên tiếp của một danh sách nhất định với một khóa tìm kiếm nhất định cho đến khi gặp một kết quả phù hợp (tìm kiếm thành công) hoặc danh sách hết mà không tìm thấy kết quả phù hợp (tìm kiếm không thành công). Một thủ thuật bổ sung đơn giản thường được sử dụng trong việc triển khai tìm kiếm tuần tự: nếu chúng ta nối khóa tìm kim vào cuối danh sách, thì việc tìm kiếm khóa sẽ phải thành công và do đó chúng ta có thể loại bỏ hoàn toàn việc kiểm tra cuối danh sách.

#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | SequentialSearch2(A[**0.**.n], K)  //Implements sequential search with a search key as a sentinel  //Input: An array A of n elements and a search key K  //Output: The index of the first element in A[0..n − 1] whose value  //is equal to K or −1 if no such element is found  A[n]← K  i ← **0**  **while** A[i]!= K do  i ← i + **1**  **if** i < n **return** i  **else** **return** −**1** | |

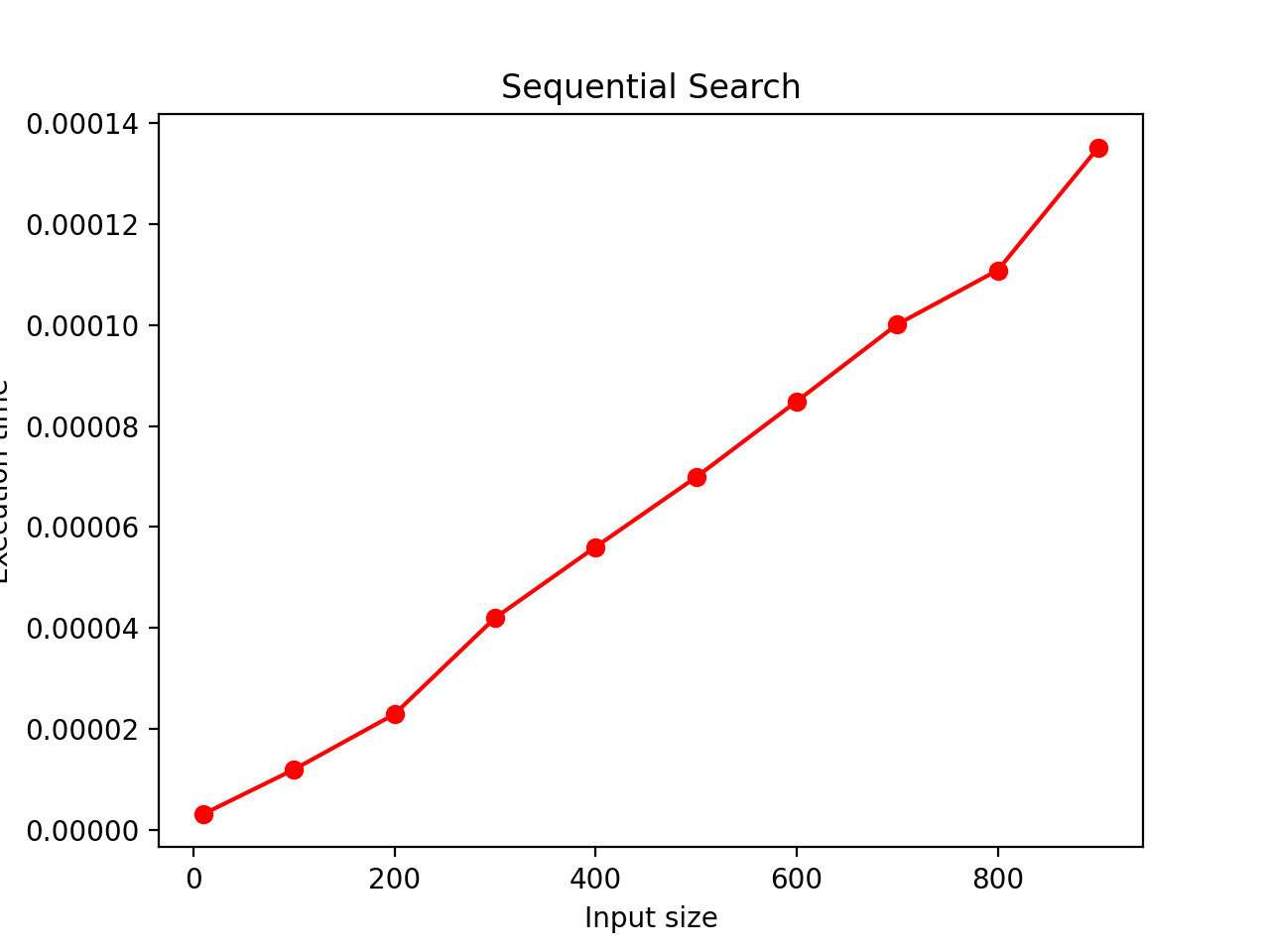
#### Phân tích giải thuật:

* Tìm kiếm trong một danh sách như vậy có thể bị dừng ngay khi gặp phần tử lớn hơn hoặc bằng khóa tìm kiếm. Tìm kiếm tuần tự cung cấp một minh họa tuyệt vời về cách tiếp cận vét cạn, với điểm mạnh đặc trưng (tính đơn giản) và điểm yếu (hiệu quả kém hơn).

#### Triển khai với python3:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | # Implements sequential search with a search key as a sentinel  # Input: An array A of n elements and a search key K  # Output: The index of the first element in A[0..n − 1] whose value is equal to K or −1  #if no such element is found  **import** **time**  **import** **pylab**  **import** **random**  **def** **SequentialSearch**(A,k):  i= **0**  **while** (i <= len(A)-**1**):  **if** A[i] == k:  **return** i  i+=**1**  **return** -**1**  A0 = ranGen(**10**)  print(A0)  print(SequentialSearch(A0, **100**))  #[24, 2, 32, 10, 90, 15, 69, 99, 58, 56]  #-1 | |

#### Demo:



# CHƯƠNG II: THUẬT TOÁN DIVIDE AND CONQUER

## Giới thiệu về divide and conquer:

* Thuật toán phân chia và chinh phục (device and conquer) là kỹ thuật thiết kế thuật toán chung được biết đến nhiều nhất. Mặc dù sự nổi tiếng của nó có thể liên quan đến cái tên hấp dẫn của nó, nhưng nó rất xứng đáng: một vài thuật toán khá hiệu quả là những triển khai cụ thể của chiến lược chung này.

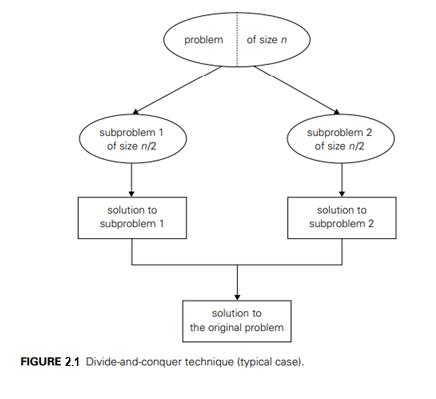
## Ý tưởng của thuật toán:

* Các thuật toán phân chia và chinh phục hoạt động theo kế hoạch chung sau:

1. Một bài toán được chia thành nhiều bài toán con cùng loại, lý tưởng nhất là về kích thước bằng nhau.

2. Các bài toán con được giải quyết (thường là đệ quy, mặc dù đôi khi sử dụng một thuật toán khác, đặc biệt khi các bài toán con trở nên nhỏ đủ).

3. Nếu cần thiết, các giải pháp cho các bài toán con được kết hợp để có được một giải pháp đến vấn đề ban đầu.



## Các ví dụ cho thuật toán :

### Thuật toán sắp xếp hợp nhất:

#### Ý tưởng giải thuật :

* Sắp xếp hợp nhất (Merge Sort) là một ví dụ hoàn hảo cho ứng dụng thành công của kỉ thuật divide-and-conquer. Thuật toán sắp xếp một mãng A[0..n-1] cho trước bằng cách chia thành 2 mãng nhỏ A[0..n/2-1] và A[n/2 … n-1], sắp xếp 2 mãng bằng các đệ quy sau đó nối thành một mãng .Kết quả thu được một mãng được sấp xếp có thứ tự.

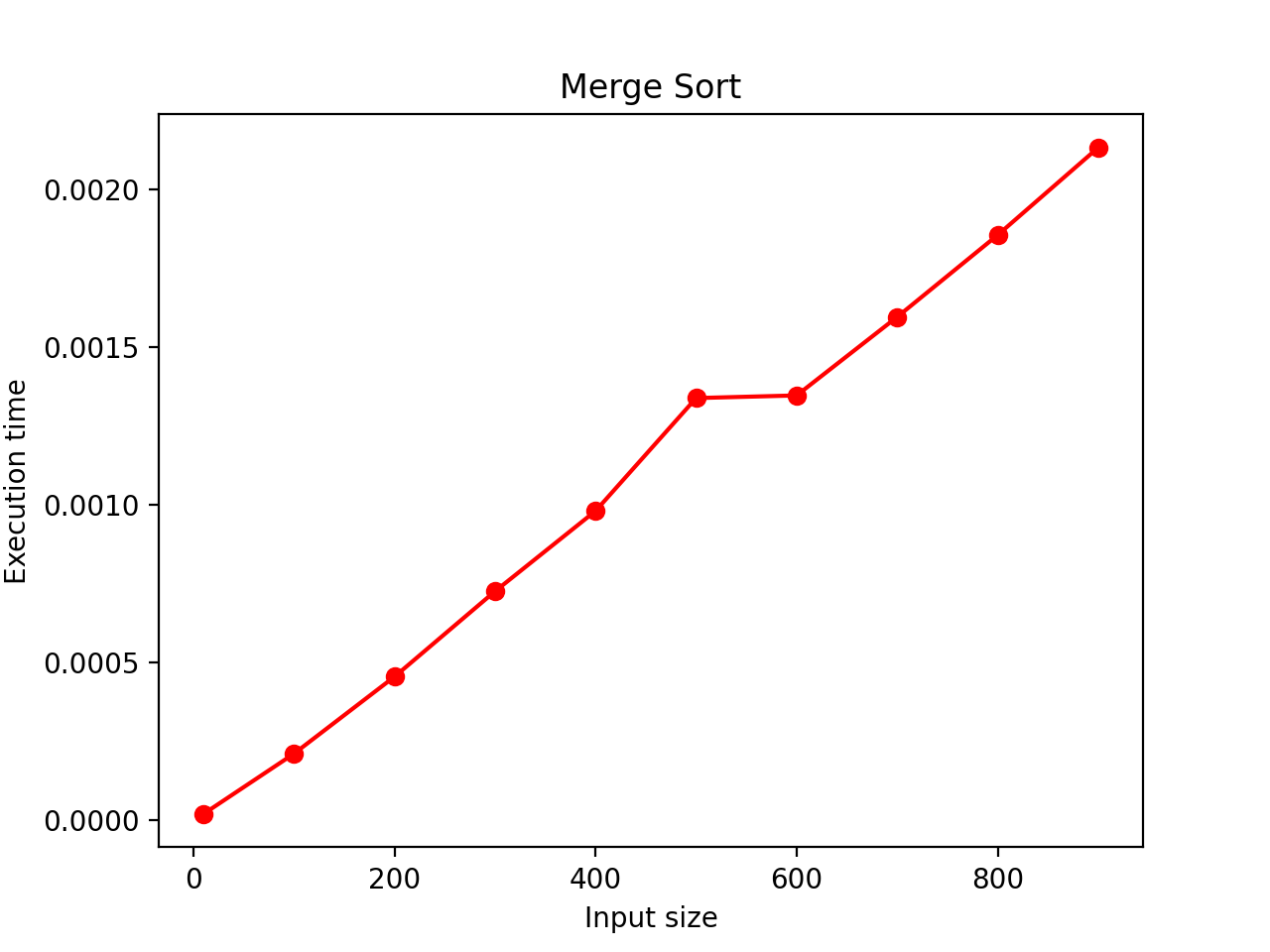
#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | Mergesort(A[**0.**.n − **1**])  //Sorts array A[**0.**.n − **1**] by recursive mergesort  //Input: An array A[**0.**.n − **1**] of orderable elements  //Output: Array A[**0.**.n − **1**] sorted **in** nondecreasing order  **if** n > **1**  copy A[**0.**.n/**2** − **1**] to B[**0.**.n/**2** − **1**]  copy A[n/**2**..n − **1**] to C[**0.**.n/**2** − **1**]  Mergesort(B[**0.**.n/**2** − **1**])  Mergesort(C[**0.**.n/**2** − **1**])  Merge(B, C, A)  Merge(B[**0.**.p − **1**], C[**0.**.q − **1**], A[**0.**.p + q − **1**])  //Merges two sorted arrays into one sorted array  //Input: Arrays B[**0.**.p − **1**] **and** C[**0.**.q − **1**] both sorted  //Output: Sorted array A[**0.**.p + q − **1**] of the elements of B **and** C  i ← **0**; j ← **0**; k ← **0**  **while** i < p **and** j < q do  **if** B[i]≤ C[j]  A[k]← B[i]; i ← i + **1**  **else** A[k]← C[j]; j ← j + **1**  k ← k + **1**  **if** i = p  copy C[j..q − **1**] to A[k..p + q − **1**]  **else** copy B[i..p − **1**] to A[k..p + q − **1**] | |

#### Triển khai với python3:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | # Sorts array A[0..n − 1] by recursive mergesort  # Input: An array A[0..n − 1] of orderable elements  # Output: Array A[0..n − 1] sorted in nondecreasing order  **def** **MergeSort**(A):  **if**(len(A)>**1**):  mid = len(A)//**2**  B = A[:mid]  C = A[mid:]  MergeSort(B)  MergeSort(C)  Merge(B,C,A)  # Merges two sorted arrays into one sorted array  # Input: Arrays B[0..p − 1] and C[0..q − 1] both sorted  # Output: Sorted array A[0..p + q − 1] of the elements of B and C  **def** **Merge**(B,C,A):  i=**0**  j=**0**  k=**0**  **while** i < len(B) **and** j < len(C):  **if** B[i] < C[j]:  A[k]=B[i]  i=i+**1**  **else**:  A[k]=C[j]  j=j+**1**  k=k+**1**    **while** i < len(B):  A[k]=B[i]  i=i+**1**  k=k+**1**  **while** j < len(C):  A[k]=C[j]  j=j+**1**  k=k+**1** | |

#### Demo:



### Thuật toán sắp xếp nhanh:

#### Ý tưởng giải thuật :

Sắp xếp nhanh là một thuật toán sắp xếp quan trọng khác dựa trên cách tiếp cận chia theo tỷ lệ. Không giống như sắp xếp hợp nhất, phân chia các phần tử đầu vào của nó theo vị trí của chúng trong mảng, sắp xếp nhanh phân chia chúng theo giá trị của chúng. Phân hoạch là sự sắp xếp các phần tử của mảng sao cho tất cả các phần tử ở bên trái của một số phần tử A [s] nhỏ hơn hoặc bằng A [s] và tất cả các phần tử ở bên phải của A [s] đều lớn hơn hoặc bằng với nó. Sau khi đạt được phân vùng, A [s] sẽ ở vị trí cuối cùng của nó trong mảng đã sắp xếp và chúng ta có thể tiếp tục sắp xếp hai mảng con bên trái và bên phải của A [s] một cách độc lập (ví dụ: theo cùng một phương pháp).

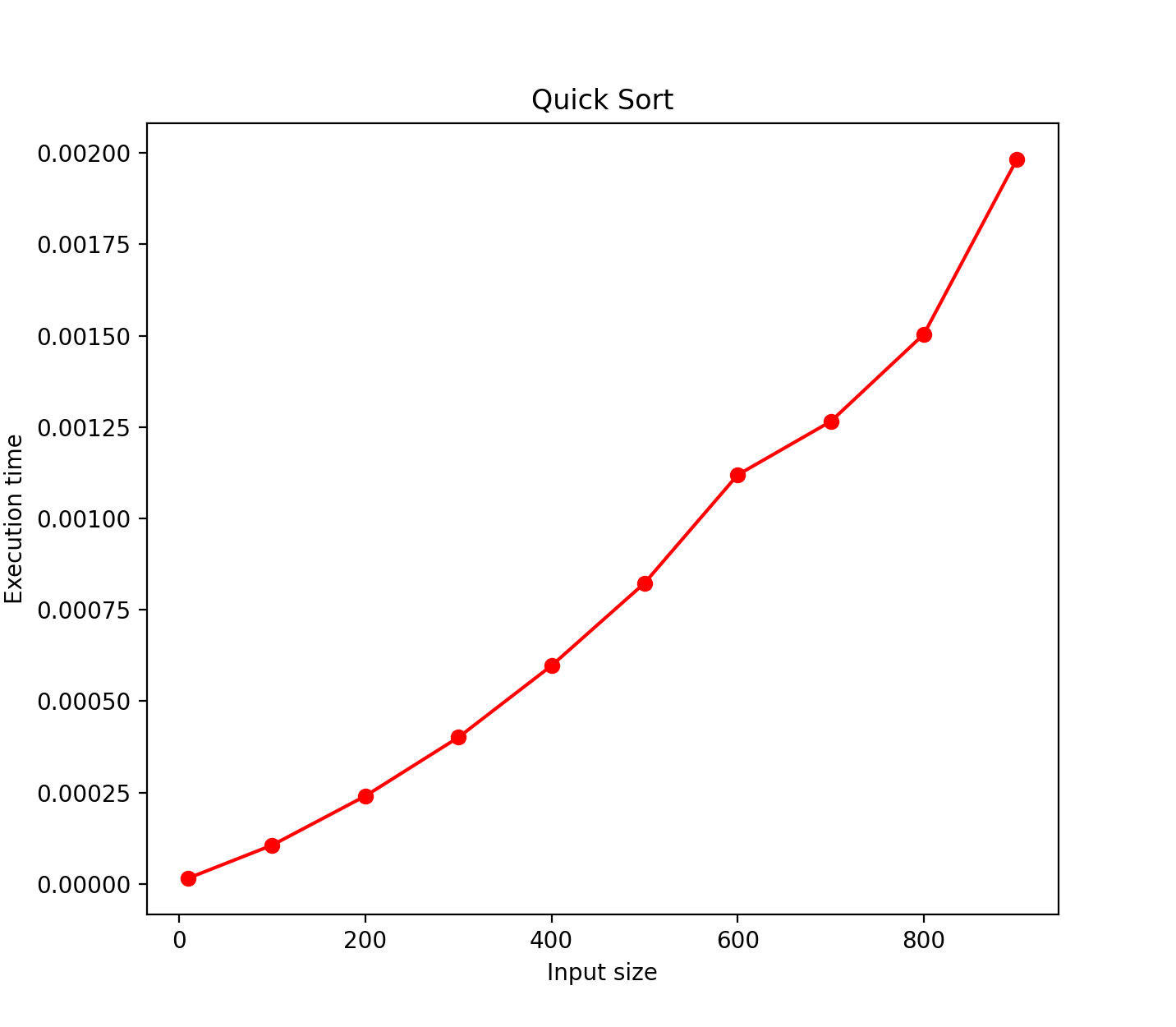
#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | Quicksort(A[l..r])  //Sorts a subarray by quicksort  //Input: Subarray of array A[**0.**.n − **1**], defined by its left  //**and** right  // indices l **and** r  //Output: Subarray A[l..r] sorted **in** nondecreasing order  **if** l < r  s ←Partition(A[l..r]) //s **is** a split position  Quicksort(A[l..s − **1**])  Quicksort(A[s + **1.**.r])  HoarePartition(A[l..r])  //Partitions a subarray by Hoare’s algorithm, using the  //first element  // **as** a pivot  //Input: Subarray of array A[**0.**.n − **1**], defined by its left  //**and** right  // indices l **and** r (l < r)  //Output: Partition of A[l..r], **with** the split position  //returned **as**  // this function’s value  p ← A[l]  i ← l; j ← r + **1**  repeat  repeat i ← i + **1** until A[i]≥ p  repeat j ← j − **1** until A[j]≤ p  swap(A[i], A[j])  until i ≥ j  swap(A[i], A[j]) //undo last swap when i ≥ j  swap(A[l], A[j])  **return** j | |

#### Triển khai với python3:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | Sorts a subarray by quicksort  # Input: Subarray of array A[0..n − 1], defined by its left and right  # indices l and r  # Output: Subarray A[l..r] sorted in nondecreasing order  **def** **QuickSort**(A,l,r):  **if** l < r:  s = Partition(A,l,r)  QuickSort(A,l,s-**1**)  QuickSort(A,s+**1**,r)  # Partitions a, using the last element as a pivot  # Input: Subarray of array A[0..n − 1], defined by its left and right  # indices l and r (l < r)  # Output: Partition of A[l..r], with the split position returned as  # this function’s value  **def** **Partition**(A,l,r):  i = (l-**1**)  p = A[r]  **for** j **in** range(l, r):  **if** A[j] <= p:  i = i+**1**  A[i], A[j] = A[j], A[i]    A[i+**1**], A[r] = A[r], A[i+**1**]  **return** (i+**1**) | |

#### Demo:



### Thuật toán tìm kiếm tuần tự:

#### Ý tưởng giải thuật :

* Thuật toán chỉ cần so sánh các phần tử liên tiếp của một danh sách nhất định với một khóa tìm kiếm nhất định cho đến khi gặp một kết quả phù hợp (tìm kiếm thành công) hoặc danh sách hết mà không tìm thấy kết quả phù hợp (tìm kiếm không thành công). Một thủ thuật bổ sung đơn giản thường được sử dụng trong việc triển khai tìm kiếm tuần tự: nếu chúng ta nối khóa tìm kim vào cuối danh sách, thì việc tìm kiếm khóa sẽ phải thành công và do đó chúng ta có thể loại bỏ hoàn toàn việc kiểm tra cuối danh sách.

#### Mã giả:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11 | SequentialSearch2(A[**0.**.n], K)  //Implements sequential search with a search key as a sentinel  //Input: An array A of n elements and a search key K  //Output: The index of the first element in A[0..n − 1] whose value  //is equal to K or −1 if no such element is found  A[n]← K  i ← **0**  **while** A[i]!= K do  i ← i + **1**  **if** i < n **return** i  **else** **return** −**1** | |

#### Phân tích giải thuật:

* Tìm kiếm trong một danh sách như vậy có thể bị dừng ngay khi gặp phần tử lớn hơn hoặc bằng khóa tìm kiếm. Tìm kiếm tuần tự cung cấp một minh họa tuyệt vời về cách tiếp cận vét cạn, với điểm mạnh đặc trưng (tính đơn giản) và điểm yếu (hiệu quả kém hơn).

#### Triển khai với python3: (Nhớ sửa laị )

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | # Implements sequential search with a search key as a sentinel  # Input: An array A of n elements and a search key K  # Output: The index of the first element in A[0..n − 1] whose value is equal to K or −1  #if no such element is found  **import** **time**  **import** **pylab**  **import** **random**  **def** **SequentialSearch**(A,k):  i= **0**  **while** (i <= len(A)-**1**):  **if** A[i] == k:  **return** i  i+=**1**  **return** -**1**  A0 = ranGen(**10**)  print(A0)  print(SequentialSearch(A0, **100**))  #[24, 2, 32, 10, 90, 15, 69, 99, 58, 56]  #-1 | |

#### Demo:

