**MỖI NGƯỜI 1 BẢN WORD, NGHIÊN CỨU HẾT TẤT CẢ 4 LOẠI NÀY - TỐI THỨ 2 HỌP VÀ NỘP LẠI**

* Các loại sắp xếp cần nghiên cứu:
  + Sắp xếp nổi bọt (bubbleSort)
  + Sắp xếp chọn (SelectionSort)
  + Sắp xếp chèn (insertionSort)
  + Sắp xếp nhanh (quickSort)
* Đánh giá hiệu quả về mặt thời gian (Không cần không gian) và giải thích
* Cụ thể là đưa ra hàm tăng trưởng cho mỗi thuật toán
* Phải cố kiếm cho đc code demo và có giải thích

Để đánh giá hiệu quả của 1 thuật toán tốt hay là Không, thì chúng ta Không thể dựa vào thời gian chạy của nó được, bởi vì thời gian chạy của 1 thuật toán nó Không chỉ dựa vào độ lớn của dữ liệu đầu vào này, ngôn ngữ lập trình, chương trình biên dịch, hệ điều hành, phần cứng của máy v.v Mặt khác, phải lập trình mới đo được thời gian thực hiện của giải thuật. Thế nên là, chúng ta Không nên dựa vào yếu tố thời gian để đánh giá hiệu quả của 1 thuật toán.

Vậy làm sao để đánh giá hiệu quả của 1 thuật toán?

Đầu tiên chúng ta đếm số lần thực hiện của câu lệnh giải thuật

Sau đó sẽ đánh giá hiệu quả của thuật toán bằng Hàm tăng trưởng thuật toán (Growth functions)

Xin chào cô và mọi người đã đến với phần thuyết trình của nhóm 1 chúng mình. Nhóm mình có 4 thành viên bao gồm: Nhất, Duy Minh, Hùng và Hữu Hoàng. (chuyển slide)

Trong nhiệm vụ lần này Minh sẽ thuyết trình về giải thuật sắp xếp nổi bọt, Nhất sẽ thuyết trình về sắp xếp chọn, Hoàng sẽ thuyết trình về sắp xếp nhanh, và Hùng là sắp xếp chèn. (chuyển slide)

Trong buổi thuyết trình lần trước, chúng ta đã nhau cùng tìm hiểu qua về khái niệm phân tích thuật toán, phân tích tiệm cận, cách mà nó hoạt động và một số kí hiệu về nó

Vâng buổi thuyết trình hôm nay nhóm chúng mình sẽ nói về cách đánh giá hiệu quả về mặt thời gian và đưa ra hàm tăng trưởng cho mỗi thuật toán. Mở đầu đó phần thuyết trình của bạn Duy Minh về sắp xếp nổi bọt, xin mời bạn.

----- **Duy Minh** -----

Đánh giá thuật toán sắp xếp bubble sort

Độ phức tạp thuật toán

Trường hợp tốt: O(n)

Trung bình: O(n^2)

Trường hợp xấu: O(n^2)

Không gian bộ nhớ sử dụng: O(1)

NDM

Ý tưởng của thuật toán bubble sort

Minh họa thuật toán sắp xếp bubble sort

Thuật toán sắp xếp bubble sort thứcj hiện sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự(số sau bé hơn số trước với trường hợp sắp xếp tăng dần) cho đến khi dãy số được sắp xếp. (next)

Làm thế nào nó hoạt động?

Loại bong bóng là một loại trao đổi nội bộ. Nó được coi là một trong những phương pháp đơn giản nhất để sắp xếp một mảng các đối tượng. Nó còn được gọi là sắp xếp chìm (vì các mục nhỏ nhất "chìm" xuống dưới cùng của mảng).

Thay vì tìm kiếm toàn bộ một mảng, sắp xếp bong bóng hoạt động bằng cách so sánh các cặp đối tượng liền kề trong mảng. Nếu các đối tượng không theo đúng thứ tự, chúng sẽ được đổi chỗ để đối tượng lớn nhất trong hai chuyển động lên trên. Quá trình này tiếp tục cho đến khi lớn nhất của các đối tượng, cuối cùng "bong bóng" lên đến vị trí cao nhất trong mảng. Sau khi điều này xảy ra, việc tìm kiếm đối tượng lớn nhất tiếp theo bắt đầu. Việc hoán đổi tiếp tục cho đến khi toàn bộ mảng theo đúng thứ tự.(next)

Lời giải

Sắp xếp nổi bọt (Bubble Sort) là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là Ο(n2) với n là số phần tử.

Giải thuật sắp xếp nổi bọt là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn giải thuật đổi chỗ trực tiếp mặc dù số lần so sánh bằng nhau, nhưng do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn.

----- **Lee Nhất** -----

**Bubble sort - Thuật toán sắp xếp nổi bọt**

Hiệu suất của bubble sort O(n):

- Tốt nhất: n

- Tệ nhất: n^2

Ưu điểm: Code ngắn gọn nhất

Nhược điểm: Hiệu suất thấp nhất

**2. Insert sort - Thuật toán sắp xếp chèn**

Hiệu suất của insert sort O(n):

- Tốt nhất: n

- Tệ nhất: n^2

Ưu điểm: Chạy nhanh khi mảng nhỏ hay được sắp xếp một phần

Nhược điểm: Hiệu suất thấp

**4. Selection sort - Thuật toán sắp xếp chọn**

**Giới thiệu giải thuật sắp xếp chọn (Selection Sort).**

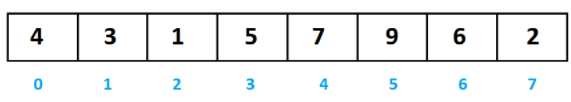
**Giải thuật sắp xếp chọn (Selection Sort)** là một giải thuật đơn giản. Giải thuật sắp xếp này là một giải thuật dựa trên việc so sánh **in-place**, trong đó danh sách được chia thành hai phần, phần được sắp xếp (**sorted list**) ở bên trái và phần chưa được sắp xếp (**unsorted list**) ở bên phải. Ban đầu, phần được sắp xếp là trống và phần chưa được sắp xếp là toàn bộ danh sách ban đầu.

Phần tử nhỏ nhất được lựa chọn từ mảng chưa được sắp xếp và được tráo đổi với phần bên trái nhất và phần tử đó trở thành phần tử của mảng được sắp xếp. Tiến trình này tiếp tục cho tới khi toàn bộ từng phần tử trong mảng chưa được sắp xếp đều được di chuyển sang mảng đã được sắp xếp.

Giải thuật này không phù hợp với tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là **O(n2)** với n là số phần tử.

**Cách hoạt động của Selection Sort:**

Ví dụ như bài toán sắp xếp dãy a thành dãy không giảm với:  
a = [4, 3, 1, 5, 7, 9, 6, 2]

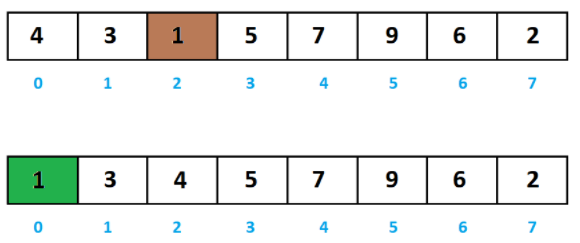


Để minh họa chúng ta sẽ dùng những phần tử màu xanh để chỉ những phần tử đã được sắp xếp đúng vị trí. phần tử màu nâu đề chỉ phần tử đang nhỏ nhất trong đoạn còn lại.

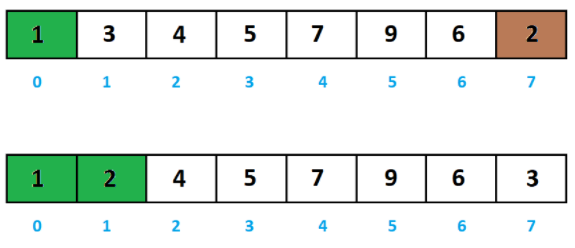
Ban đầu ta sẽ khởi tạo dãy con bằng a[0], đương nhiên với dãy con một phần tử thì nó đã được sắp xếp.

Ta bắt đầu xét từ phần tử có chỉ số 0 đến n-1.

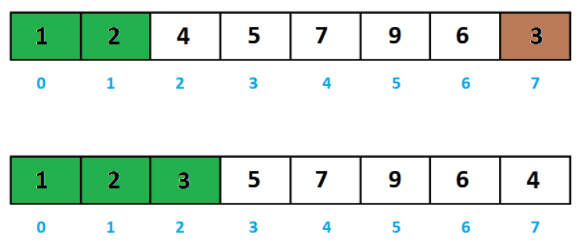
Đầu tiên ta tìm phần tử nhỏ nhất trong dãy trong khoảng từ 0 đến 7. Sau đó hoán đổi cho phần tử a[0].



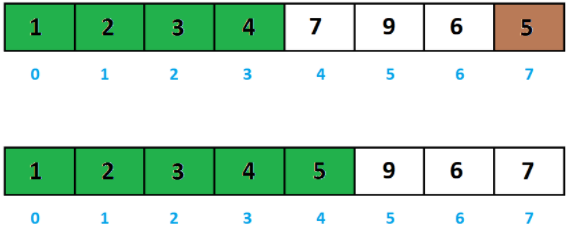
Tiếp theo tìm phần tử nhỏ nhất trong dãy trong khoảng từ 1 đến 7. Sau đó hoán đổi cho phần tử a[1].



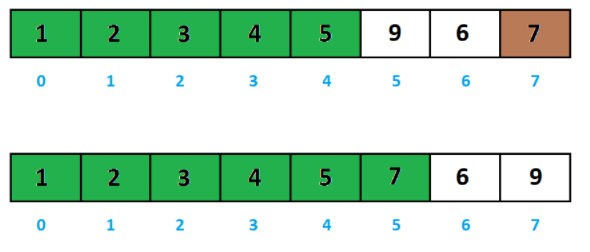
Tiếp theo tìm phần tử nhỏ nhất trong dãy trong khoảng từ 2 đến 7. Sau đó hoán đổi cho phần tử a[2].



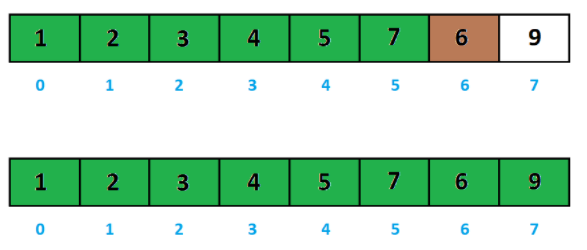
Tiếp theo tìm phần tử nhỏ nhất trong dãy trong khoảng từ 4 đến 7. Sau đó hoán đổi cho phần tử a[4].



Tiếp theo tìm phần tử nhỏ nhất trong dãy trong khoảng từ 5 đến 7. Sau đó hoán đổi cho phần tử a[5].



Tiếp theo tìm phần tử nhỏ nhất trong dãy trong khoảng từ 6 đến 7. Sau đó hoán đổi cho phần tử a[6].



Còn 1 phần tử cuối cùng chắc chắn đã ở vị trí đúng.

CODE DEMO

…….

GIẢI THÍCH CODE

* Cách chúng ta sắp xếp danh sách này là chia nó thành hai danh sách riêng biệt trong danh sách đó. Một danh sách phụ sẽ đại diện cho các số đã được sắp xếp của chúng tôi và danh sách kia sẽ đại diện cho các số chưa được sắp xếp của chúng tôi. Ban đầu, toàn bộ mảng của chúng tôi đại diện cho danh sách chưa được sắp xếp.
* Chúng ta sẽ tìm kiếm số nhỏ nhất trong mảng của chúng ta. Khi chúng tôi đã tìm thấy số nhỏ nhất, chúng tôi thêm nó vào danh sách đã sắp xếp của chúng tôi (tức là dành chỗ cho nó ở đầu mảng). Điều này xảy ra tại chỗ trong mảng đầu vào.
* Nếu chúng ta đi ngang qua mảng của mình, chúng ta bắt đầu ở chỉ số 0 và coi bất kỳ giá trị nào ở đó là giá trị nhỏ nhất của chúng ta.
* Tiếp tục duyệt qua mảng và nếu bạn tìm thấy bất kỳ giá trị nào nhỏ hơn giá trị đã lưu trước đó, hãy thay đổi giá trị nhỏ nhất thành giá trị mới. Vì vậy, nếu nhỏ nhất = 6 và chúng ta đi đến chỉ số tiếp theo, chúng ta thấy rằng 3 nhỏ hơn 6 vì vậy nhỏ nhất bây giờ bằng 3. Tiếp tục qua toàn bộ mảng cho đến khi bạn tìm thấy giá trị nhỏ nhất. Trong trường hợp của chúng tôi, chúng tôi thấy rằng 2 là giá trị nhỏ nhất.
* Giờ thì sao? Lấy giá trị ngoài cùng bên trái và hoán đổi nó với giá trị nhỏ nhất của bạn. Mảng của bạn bây giờ được chia làm hai: phần màu xanh lam là mảng đã được sắp xếp và phần còn lại là mảng chưa được sắp xếp. Hãy xem một lần lặp lại khác:
* Lần này mảng chưa được sắp xếp của chúng ta bắt đầu từ chỉ số 1 vì mảng đã sắp xếp của chúng ta có chỉ số dành riêng là 0. Chúng ta đặt nhỏ nhất thành chỉ số 1 (nhỏ nhất = 3). Ta sẽ so sánh giá trị nhỏ nhất với mọi giá trị khác trong mảng, nếu có giá trị nào nhỏ hơn thì ta đặt giá trị nhỏ nhất cho giá trị đó. Trong trường hợp này, chúng ta thấy rằng 3 là giá trị nhỏ nhất. Chúng ta không cần hoán đổi 3 với chính nó để 3 bây giờ trở thành một phần của mảng đã sắp xếp.
* Chúng ta bắt đầu từ chỉ số 2 với nhỏ nhất = 9. Sau khi xem qua toàn bộ mảng, chúng ta thấy rằng 4 là giá trị nhỏ nhất. 9 và 4 hoán đổi.
* Chúng tôi tiếp tục tăng kích thước của mảng đã sắp xếp của chúng tôi theo cách này cho đến khi chúng tôi nhận được mảng được sắp xếp đầy đủ:

**Thời gian và không gian phức tạp**

Để sắp xếp một mảng bằng Sắp xếp Lựa chọn, bạn phải lặp qua mảng một lần cho mọi giá trị bạn có trong mảng.

Nếu chúng ta có n giá trị trong mảng của mình, Sắp xếp Lựa chọn có độ phức tạp về thời gian là O (n²) trong trường hợp xấu nhất. Trong trường hợp tốt nhất, chúng ta đã có một mảng đã được sắp xếp nhưng chúng ta cần xem qua mảng O (n²) lần để chắc chắn! Do đó, độ phức tạp thời gian tốt nhất và trường hợp xấu nhất của Lựa chọn là như nhau.

Còn về độ phức tạp của không gian thì sao? Lựa chọn Sắp xếp sắp xếp tại chỗ, nghĩa là chúng ta không cần cấp phát bất kỳ bộ nhớ nào để sắp xếp xảy ra. Độ phức tạp không gian là O (1) -Số phép tính/thời gian chạy/dung lượng bộ nhớ không phụ thuộc vào độ lớn đầu vào

**Hiệu suất của selection sort O(n):**

- Tốt nhất: n^2

- Tệ nhất: n^2

Ưu điểm: Thuật toán chạy nhanh hơn khi mảng sắp xếp một phần

Nhược điểm: Hiệu suất không cao

----- **Hữu Hoàng** -----

**Về khái niệm của thuật toán sắp xếp nhanh**

Sắp xếp nhanh là một trong những Kỹ thuật sắp xếp khác nhau dựa trên khái niệm chia để trị, giống như sắp xếp trộn. Nhưng trong quicksort, tất cả các công việc chính được thực hiện trong khi chia mảng thành các mảng con, còn trong trường hợp của sắp xếp trộn, tất cả công việc lại này xảy ra trong quá trình hợp nhất các mảng con. Trong trường hợp nhanh chóng, bước kết hợp hoàn toàn không có tác dụng gì.

Nó cũng được gọi là phân vùng trao đổi. Thuật toán này đc chia thành ba phần chính

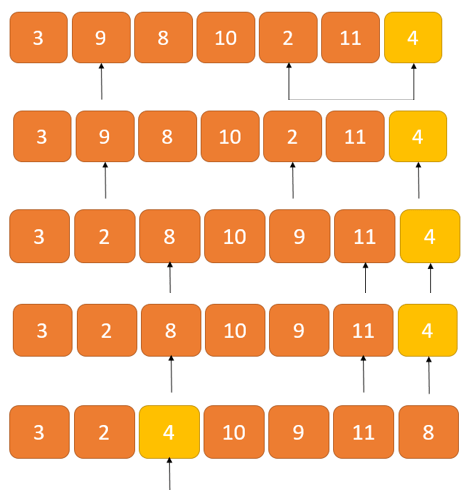
thứ nhất là Các phần tử nhỏ hơn phần tử Pivot

thứ 2 Phần tử tổng hợp (Phần tử trung tâm)

thứ 3 Các phần tử lớn hơn phần tử pivot **(chuyển slide)**

**Cách hoạt động của thuật toán này ntn, sau đây chúng ta cùng tìm hiểu**

* Bước 1: Một phần tử có tên là Pivot được chọn từ mảng. Ta có thể chọn bất kỳ phần tử nào trong mảng làm phần tử này. Ở đây, chúng ta đã sử dụng phần tử nằm ngoài cùng ở bên phải (tức là phần tử cuối cùng) của mảng làm phần tử Pivot.
* Bước 2: Các phần tử nhỏ hơn phần tử Pivot sẽ được đặt ở bên trái và các phần tử lớn hơn phần tử Pivot được đặt ở bên phải.
* Cách sắp xếp như sau:
  + Một con trỏ được cố định tại phần tử Pivot (4). Phần tử Pivot được so sánh với các phần tử khác bắt đầu từ chỉ số đầu tiên. Nếu đạt đến phần tử lớn hơn phần tử Pivot, con trỏ thứ hai được đặt cho phần tử đó.
  + Bây giờ, phần tử Pivot được so sánh với các phần tử khác (con trỏ thứ ba). Nếu đạt đến phần tử nhỏ hơn phần tử Pivot, phần tử nhỏ hơn sẽ được hoán đổi với phần tử lớn hơn được tìm thấy trước đó.



* Quá trình tiếp tục cho đến khi đạt được phần tử cuối cùng thứ hai.

**Phần này tự nói :v**

* Cuối cùng, phần tử pivot được hoán đổi với con trỏ thứ hai.

**Sau khi xong rồi thì chúng ta nhận thấy số 4 là phần tử trung tâm, hay còn gọi là trục trung tâm. Lúc này, ta sử dụng đệ quy để sắp xếp các phần con (chuyển slide)**

**Ta nói Quicksort sử dụng đệ quy để sắp xếp các phần con, Trên cơ sở của cách tiếp cận chia để trị, thuật toán Quick Sort có thể được giải thích như sau:**

1. Bước chia

Mảng được chia thành các phần con và lấy phần tử Pivot làm điểm phân chia. Các phần tử nhỏ hơn Pivot được đặt ở bên trái của nó và các phần tử lớn hơn được đặt ở bên phải.

2. Bước trị

Các phần con bên trái và bên phải lại được phân chia bằng cách chọn các phần tử Pivot cho chúng. Điều này có thể đạt được bằng cách chuyển đệ quy các phần con vào thuật toán.

3. Bước kết hợp

Như lúc đầu mình đã nói bước này không quan trọng cho lắm. Vì Mảng đã được sắp xếp ở cuối bước trị rồi. In ra là xong.

Tiếp theo là ý tưởng thuật toán của QuickSort **(chuyển slide)**

**Ý tưởng thuật toán**

Nếu phần tử cuối cùng bên trái < phần tử cuối cùng bên phải

* Thực hiện gán cho chỉ số của phần tử Pivot <- hàm phân chia(mảng,chỉ số bên trái cuối cùng, chỉ số bên phải cuối cùng)
* Hàm QuickSort(mảng, chỉ số bên trái cuối cùng, chỉ số của phần tử Pivot)
* Hàm QuickSort(mảng, chỉ số của phần tử Pivot + 1, chỉ số bên phải cuối cùng)

Cái mình vừa nói đó là hàm đệ quy, QuickSort sẽ dùng đệ quy để sắp xếp

**(chuyển slide)**

Được rồi, hàm tiếp theo đó là hàm **sort** để sắp xếp các phần tử

Ta có thêm 1 hàm sort gồm có (Mảng, phần tử cuối cùng bên trái , phần tử cuối cùng bên phải)

và ta thực hiện gán biến tên là l = lastLeft và biến r = lastRight - 1

Chúng ta sẽ triển khai chúng trong vòng lặp while:

Khi l <= r và mảng chứa phần tử l < mảng chứa lastRight thì l++

ngược lại, khi l < r và mảng chứa r > mảng chứa lastRight thì r--

Nếu mà l >= r rồi á thì dừng, break và ta cho 1 biến mới gọi là swap gán bằng arr[l]

arr[l] = arr[r], arr[r] = swap đồng thời l tăng và r giảm Kết thúc vòng lặp while

Ở bên ngoài vòng lặp while thì ta cho một biến mới gọi là xxx và gán nó = arr[l]

arr[l] này sẽ bằng arr chứa các phần tử cuối cùng bên phải

những phần tử này lại bằng biến xxx, sau đó trả về l **(chuyển slide)**

**Phân tích độ phức tạp của sắp xếp nhanh**

Đối với một mảng, trong đó việc phân vùng dẫn đến các mảng con không cân bằng, như ví dụ mà mình nêu ở trên thì mảng con bên trái chỉ được 2 phần tử, còn ở bên phải 4 phần tử

Và nếu tiếp tục nhận được các mảng con không cân bằng, thì thời gian chạy là trường hợp xấu nhất, đó là O(n2)

Còn khi việc phân vùng các mảng con gần như bằng nhau, thì thời gian chạy là tốt nhất, với độ phức tạp về thời gian là O (n \* log n).

Chung quy lại

Độ phức tạp về thời gian của trường hợp xấu nhất [Big-O]: O (n2)

Độ phức tạp về thời gian của trường hợp tốt nhất [Big-omega]: O (n \* log n)

Độ phức tạp thời gian trung bình [Big-theta]: O (n \* log n)

Như chúng ta biết bây giờ, nếu phân vùng con được tạo ra sau khi phân vùng không cân bằng, thì việc sắp xếp nhanh sẽ mất nhiều thời gian hơn để hoàn thành. Để tránh điều này, ta cũng có thể chọn phần tử xoay ngẫu nhiên. Nó sẽ không tạo ra bất kỳ sự khác biệt nào trong thuật toán, vì tất cả những gì chúng ta cần làm là chọn một phần tử ngẫu nhiên từ mảng, hoán đổi nó với phần tử ở chỉ mục cuối cùng, làm cho nó xoay vòng và tiếp tục với sắp xếp nhanh.

Không gian yêu cầu của sắp xếp nhanh là rất ít, chỉ cần thêm không gian O (n \* log n). Sắp xếp nhanh không phải là một kỹ thuật sắp xếp ổn định, vì vậy nó có thể thay đổi sự xuất hiện của hai phần tử giống nhau trong danh sách khi sắp xếp. **(END)**

<https://tek4.vn/thuat-toan-sap-xep-nhanh-quick-sort/>

**-----------HÙNG--------------**

**1. Dinh nghia**

Sắp xếp chèn là một thuật toán sắp xếp đơn giản cho phép sắp xếp hiệu quả, tại chỗ của mảng, một phần tử tại một thời điểm. Bằng cách sắp xếp tại chỗ, chúng tôi muốn nói rằng mảng ban đầu được sửa đổi và không cần cấu trúc tạm thời.

- Sắp xếp chèn là một thuật toán sắp xếp đặt một phần tử chưa được sắp xếp vào vị trí thích hợp của nó trong mỗi lần lặp.

- Sắp xếp chèn hoạt động tương tự như chúng ta sắp xếp các quân bài trên tay trong một trò chơi bài.

**Thuật toán**

Để sắp xếp một mảng có kích thước n theo thứ tự tăng dần:

1: Lặp lại từ arr [1] đến arr [n] trên mảng.

2: So sánh phần tử hiện tại (khóa) với phần tử tiền nhiệm của nó.

3: Nếu phần tử chính nhỏ hơn phần tử trước của nó, hãy so sánh nó với các phần tử trước đó. Di chuyển các phần tử lớn hơn lên một vị trí để tạo khoảng trống cho phần tử được hoán đổi.

**2. Code demo**

Sắp xếp chèn bắt đầu bằng cách xem xét hai phần tử đầu tiên của dữ liệu mảng, đó là dữ liệu [0] và dữ liệu [1]

- Tiếp theo, phần tử thứ ba, data [2], được xem xét và chèn vào vị trí thích hợp của nó (tăng dần)

- Độ phức tạp của nó là O (n2) và trường hợp tốt nhất là O (n))

**3. Time complex**

Độ phức tạp của trường hợp tồi tệ nhất: O (n2)

+ Giả sử, một mảng có thứ tự tăng dần, và bạn muốn sắp xếp nó theo thứ tự giảm dần. Trong trường hợp này, trường hợp xấu nhất phức tạp xảy ra.

+ Mỗi phần tử phải được so sánh với mỗi phần tử khác, do đó, với mỗi phần tử thứ n, (n-1) số phép so sánh được thực hiện.

- Độ phức tạp của trường hợp tốt nhất: O (n)

+ Khi mảng đã được sắp xếp, vòng lặp bên ngoài chạy n số lần trong khi vòng lặp bên trong hoàn toàn không chạy. + Như vậy chỉ có n số phép so sánh. Do đó, độ phức tạp là tuyến tính.

- Độ phức tạp của trường hợp trung bình: O (n2)

+ Nó xảy ra khi các phần tử của mảng có thứ tự lộn xộn (không tăng dần cũng không giảm dần).

- Không gian phức tạp

+ Độ phức tạp của không gian là O (1) vì một khóa biến phụ được sử dụng.

Độ phức tạp của trường hợp tồi tệ nhất: O (n2)

+ Giả sử, một mảng có thứ tự tăng dần, và bạn muốn sắp xếp nó theo thứ tự giảm dần. Trong trường hợp này, trường hợp xấu nhất phức tạp xảy ra.

+ Mỗi phần tử phải được so sánh với mỗi phần tử khác, do đó, với mỗi phần tử thứ n, (n-1) số phép so sánh được thực hiện.

- Độ phức tạp của trường hợp tốt nhất: O (n)

+ Khi mảng đã được sắp xếp, vòng lặp bên ngoài chạy n số lần trong khi vòng lặp bên trong hoàn toàn không chạy. + Như vậy chỉ có n số phép so sánh. Do đó, độ phức tạp là tuyến tính.

- Độ phức tạp của trường hợp trung bình: O (n2)

+ Nó xảy ra khi các phần tử của mảng có thứ tự lộn xộn (không tăng dần cũng không giảm dần).

- Không gian phức tạp

+ Độ phức tạp của không gian là O (1) vì một khóa biến phụ được sử dụng.)

4. Hàm tang trưởng

……….

**Tham khảo**

Link tham khảo 1: <https://viettuts.vn/bai-tap-java/cac-thuat-toan-sap-xep-trong-java#goto-h2-2>

Link tham khảo 2: <https://www.khanacademy.org/computing/computer-science/algorithms/sorting-algorithms/a/analysis-of-selection-sort>

<https://www.programiz.com/dsa/insertion-sort>

Link record 1: <https://drive.google.com/file/d/1aSnvexUkQhocK-J9ez8FEUUlRfudRfZ4/view?usp=sharing>

Link record 2: <https://drive.google.com/file/d/1X4F2Y01f19p_FjH4MunzYUrM_leeUIGR/view?usp=sharing>