**Sắp xếp và tìm kiếm là gì?**

Thuật toán sắp xếp và tìm kiếm là hai khái niệm căn bản trong tin học nói chung và trong chuyên ngành lập trình nói riêng. Ngay cả trong cuộc sống hằng ngày bạn cũng gặp rất nhiều người sử dụng hai hành động này. Nhớ ngày xưa lúc học lớp 1 các cô giáo hay bắt học sinh sắp xếp một dãy số theo thứ tự tăng dần hoặc giảm dần, tìm số lớn nhất và nhỏ nhất trong một dãy số, ... đó cũng là những bài toán sắp xếp nhưng ở khía cạnh khác.

Bây giờ chúng ta sẽ tìm hiểu hai khái niệm thuật toán sắp xếp và thuật toán tìm kiếm.

**Mục lục**

* [1. Thuật toán sắp xếp là gì?](https://freetuts.net/sap-xep-va-tim-kiem-la-gi-545.html#goto-h2-0)
* [2. Thuật toán tìm kiếm là gì?](https://freetuts.net/sap-xep-va-tim-kiem-la-gi-545.html#goto-h2-1)
* [3. Lời kết](https://freetuts.net/sap-xep-va-tim-kiem-la-gi-545.html#goto-h2-2)

**1. Thuật toán sắp xếp là gì?**

Sắp xếp là quá trình bố trí lại các phần tử trong một tập hợp theo một trình tự nào đó nhằm mục đích giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn. Điển hình nhất trong thực tế là các ứng dụng quản lý danh bạ điện thoại thì có sắp xếp theo số, theo tên. Quản lý học sinh thì có sắp xếp theo điểm, theo lớp, theo trường, ... Như vậy có rất nhiều mục đích cần phải sắp xếp các phần tử theo một trình tự. Điển hình nhất lúc bạn học lập trình đó là sắp xếp dãy số tăng dần, giảm dần bởi các kỹ thuật sắp xếp được nghiên cứu và phân tích kỹ lưỡng.

Trong khoa học máy tính và trong toán học, thuật toán sắp xếp là một thuật toán sắp xếp các phần tử của một danh sách (*hoặc một mảng*) theo thứ tự (*tăng hoặc giảm*). Và để dễ dàng cho việc nghiên cứu và học tập thì người ta thường gán các phần tử được sắp xếp là các chữ số.

*Bài viết này được đăng tại [free tuts .net]*

Chúng ta có khá nhiều thuật toán như:

* Sắp xếp chọn
* Sắp xếp chèn
* Sắp xếp nổi bọt
* Sắp xếp Quick sort
* Sắp xếp Help sort
* Sắp xếp Merge sort

**2. Thuật toán tìm kiếm là gì?**

Tìm kiếm là quá trình tìm một phần tử nằm trong một tập hợp nhiều phần tử dựa vào một miêu tả nào đó. Ví dụ bạn cần tìm đồng 10k trong một đống tiền từ 10k đến 100k thì quá trình đó ta gọi là tìm kiếm.

Nếu một tập hợp đã được sắp xếp thì quá trình tìm kiếm trong tập hợp đó sẽ nhanh hơn. Ở ví dụ trên nếu đống tiền từ 10k đến 100k đã được sắp xếp tăng dần hoặc giảm dần thì ta rất dễ dàng tìm kiếm tờ 10k. Nhưng nếu đó là một đống tiền lộn xộn thì bạn sẽ mất nhiều thời gian để xử lý chúng.

Vậy thuật toán tìm kiếm là thuật toán dùng để tìm kiếm phần tử trong một danh sách cho trước theo một tiêu chí nhất định. Chúng ta thường sử dụng hai thuật toán đó là thuật toán tìm kiếm tuyến tính và thuật toán tìm kiếm nhị phần.

Thuật toán tìm kiếm tuyến tính là quá trình tìm kiếm trong một tập hợp chưa sắp xếp, giống với đống tiền chưa được sắp xếp ở ví dụ trên. Còn thuật toán tìm kiếm nhị phân là quá trình tìm kiếm trong một dãy đã được sắp xếp. Cả hai thuật toán đều có những ưu và nhược điểm khác nhau.

**3. Lời kết**

Như vậy trong series này chúng ta sẽ học tổng cộng 6 thuật toán sắp xếp và 2 thuật toán tìm kiếm. Đây là một chương rất quan trọng trong trong học phần cấu trúc dữ liệu của các trường đại học công nghệ thông tin. Đây là bài mở đầu cho chương này nên cũng chỉ là giới thiệu cho các bạn, cái hay sẽ còn ở các bài tiếp theo, mời các bạn theo dõi.

**Thuật toán tìm kiếm tuyến tính (Linear search)**

Trong bài viết này chúng ta sẽ tìm hiểu về thuật toán tìm kiếm tuyến tính( linear search), hay còn được gọi là tìm kiếm tuần tự (sequential search).



**Đề bài:** Cho một mảng arr[] gồm n phần từ. Viết một hàm đưa ra vị trí của phần từ x trong mảng.

**Ví dụ:**

Input : arr[] = {10, 20, 80, 30, 60, 50,

110, 100, 130, 170}

x = 110;

Output : 6

Với đề bài này chúng ta có thể giải bằng cách sử dụng *thuật toán tìm kiếm tuyết tính* được trình bày sau đây.

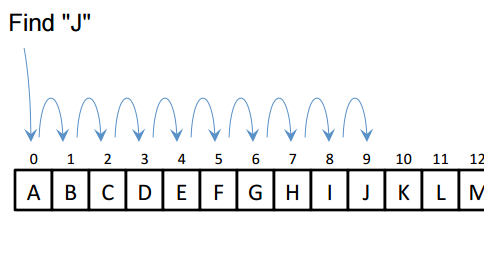
*Bài viết này được đăng tại [free tuts .net]*

**Mục lục**

* [1. Tìm kiếm tuyến tính là gì?](https://freetuts.net/thuat-toan-tim-kiem-tuyen-tinh-2630.html#goto-h2-0)
* [2. Ý tưởng](https://freetuts.net/thuat-toan-tim-kiem-tuyen-tinh-2630.html#goto-h2-1)
* [3. Triển khai thuật toán](https://freetuts.net/thuat-toan-tim-kiem-tuyen-tinh-2630.html#goto-h2-2)

**1. Tìm kiếm tuyến tính là gì?**

**Thuật toán tìm kiếm tuyến tính** là phương pháp tìm kiếm một phần tử cho trước trong một danh sách bằng cách duyệt lần lượt từng phần từ của danh sách đó đến khi nào tìm được giá trị mong muốn hay đã duyệt hết qua hết danh sách.



Đây là một giải thuật khá đơn giản để thực hiện, nó rất phù hợp khi cần tìm kiếm trên một danh sách vửa đủ và chưa được sắp xếp. Trong trường hợp cần tìm kiếm với một danh sách lớn hoặc nhiều lần chúng ta nên tìm một giải thuật khác hiệu quả hơn.

Thuật toán tìm kiếm tuyến tính rất hiếm khi được sử dụng bởi các thuật toán tìm kiếm khác như tìm kiếm nhị phân, bảng băm,..cho phép tìm kiếm nhanh hơn rất nhiều so với tìm kiếm tuyến tính.

**2. Ý tưởng**

**Thuật toán tìm kiêm tuyến tính** là một thuật toán khá đơn giản. Sau đây là ý tưởng triển khai thuật toán.

* Bắt đầu từ bản ghi đầu tiên của mảng, duyệt từ đầu mảng đến cuối mảng với x.
* Nếu phần tử đang duyệt bằng x thì trả về vị trí.
* Nếu không tìm thấy bất cứ phần từ nào khi đã duyệt hết thì trả về -1.

Trong trường hợp tốt nhất độ phức tạp của thuật toán này là O(1), trường hơp xấu nhất là O(n), trung bình cũng là O(n).

**3. Triển khai thuật toán**

Ở đây chúng ta sẽ triển khai thuật bằng ngôn ngữ C++, các bước triển khai sẽ được đề cập bên dưới.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | #include <iostream>  using namespace std;  **int** linearSearch(**int** arr[], **int** n, **int** x){          //Lặp từng phần tử của mảng và kiểm tra.      for(**int** i = 0; i < n; i++)          if (arr[i] == x)              return i;          // Trả về -1 nếu đã duyệt hết mà ko tìm thấy.      return -1;  }  **int** main() {  **int** arr[] = {1, 5, 12, -10, 5, 11};  **int** x = -10;          // Số phần có trong mảng.  **int** n = sizeof(arr) / sizeof(arr[0]);  **int** result = linearSearch(arr, n, x);      if (result == -1)          cout << "Khong tim thay " << x               << " trong mang";      else          cout << "Vi tri: " << result;  } |

Output: Vi tri 3

Bên trên là cách triển khai đơn giản nhất của thuật toán tìm kiếm tuyến tính. Đây cũng là một thuật toán hay được sử dụng cũng như rât hữu ích trong quá trình giải các bài toán tìm kiếm. Rất mong bài viết sẽ hữu ích cho bạn !

**Thụât toán tìm kiếm nhị phân (Binary Search)**

Trong bài viết này chúng ta sẽ tìm hiểu về thuật toán tìm kiếm nhị phân (Binary search). Đây là thụât toán phổ biến để tìm kiếm vị trí một phần tử trong một mảng đã sắp xếp.



**Để bài:**Cho một danh sách arr[] đã được sắp xếp gồm n phần từ , viết một hàm đưa ra vị trí của phần từ x trong mảng.

**Ví dụ:**

Input : arr[] = {15, 20, 25, 30, 31, 44, 66}

x = 25;

Output : 2

Một cách tìm kiếm đơn giản hơn rất nhiều đó là thụât toán tìm kiếm tuần tự (linear search) nhưng độ phức tạp khá lớn lên tới O(n), không khả thi để thực hiện tìm kiếm trên một mảng lớn. Để giải quyết bài toán này chúng ta sẽ sử dụng thụât toán tìm kiếm nhị phân (binary search) được giới thiệu bên dưới.

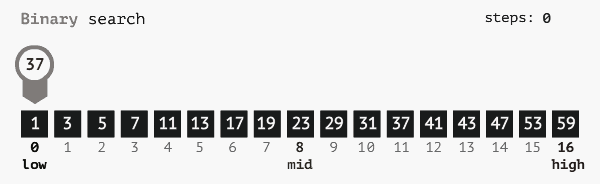
*Bài viết này được đăng tại [free tuts .net]*

**Mục lục**

* [1. Tìm kiếm nhị phân (Binary Search)](https://freetuts.net/thuat-toan-tim-kiem-nhi-phan-2634.html#goto-h2-0)
* [2. Ý tưởng triển khai thụât toán](https://freetuts.net/thuat-toan-tim-kiem-nhi-phan-2634.html#goto-h2-1)
* [3. Triển khai thụât toán](https://freetuts.net/thuat-toan-tim-kiem-nhi-phan-2634.html#goto-h2-2)

**1. Tìm kiếm nhị phân (Binary Search)**

**Thụât toán tìm kiếm nhị phân (Binary Search)** hay còn được gọi là tìm kiếm một nửa là thụât toán tiếp kiếm được sử dụng rất nhiều trong thực tế cho phép tìm kiếm vị trí của một phần tử trong một mảng đã được sắp xếp.



**Thụât toán tìm kiếm nhị phân**thực hiện tìm kiếm một mảng đã sắp xếp bằng cách liên tục chia các khoảng tìm kiếm thành 1 nửa. Bắt đầu với một khoảng từ phần tử đầu mảng, tới cuối mảng. Nếu giá trị của phần tử cần tìm nhỏ hơn giá trị của phần từ nằm ở giữa khoảng thì thu hẹp phạm vi tìm kiếm từ đầu mảng tới giửa mảng và nguợc lại. Cứ thế tiếp tục chia phạm vi thành các nửa cho dến khi tìm thấy hoặc đã duyệt hết.

Thuật toán tìm kiếm nhị phân tỏ ra tối ưu hơn so với tìm kiếm tuyết tính ở các mảng có độ dài lớn và đã được sắp xếp. Ngược lại, tìm kiếm tuyến tính sẽ tỏ ra hiệu quả hơn khi triển khai trên các mảng nhỏ và chưa được sắp xếp.

**2. Ý tưởng triển khai thụât toán**

**Thuật toán tìm kiếm nhi phân** là một thuật toán khá thông dụng và chỉ dùng được với một mảng đã sắp xếp. Để triển khai thuật toán này hãy chắc chắn rằng mảng đã được sắp xếp. Sau đây là ý tưởng triển khai thuật toán.

* Xét một đoạn trong mảng **arr[left...right]**. Lúc này giá trị của left và right lần luợt là 0 và số phần tử của mảng - 1.
* So sánh x với phần tử nằm ở vị trí chính giữa của mảng **(mid = (left + right) /2)**. Nếu x bằng arr[mid] thì trả về vị trí và thoát vòng lặp.
* Nếu **x < arr[mid]**thì chắc chắn x sẽ nằm ở phía bên trái tức là từ **arr[left....mid-1]**.
* Nếu **x > arr[mid]** thì chắc chắn x sẽ nằm ở phía bên phải mid tức là ở khoảng **arr[mid+1...right]**.
* Tiếp tục thực hiện chia đôi các khoảng tìm kiếm tới khi nào tìm thấy được vị trí của x trong mảng hoặc khi đã duyệt hết mảng.

Người ta đã chứng minh được độ phực tạp của thụât toán tìm kiếm nhi phân trong trường hợp tốt nhất là O(1), trong trường hợp xấu nhất là O(log2n) và trung bình cũng là O(log2n). Tuy nhiên, ta không nên quên rằng trước khi sử dụng tìm kiếm nhị phân, dãy khoá phải được sắp xếp rồi, tức là thời gian chi phí cho việc sắp xếp cũng phải tính đến.

**3. Triển khai thụât toán**

Sau đây là các bước triển khai thuật toán, các bước thực hiện đều được comment ở từng đoạn.

|  |  |
| --- | --- |
| 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  38  39  40  41  42 | #include <bits/stdc++.h>  using namespace std;    //Hàm tìm kiếm nhi phân  **int** binarySearch(**int** arr[], **int** left, **int** right, **int** x) {  **int** middle;        while(left <= right) {          // Lấy vị trí ở giữa left và right          middle = (left + right) / 2;            // Nếu phần từ ở giữa bằng x thì trả về          // vị trí          if (arr[middle] == x)              return middle;            // Nếu x lớn hơn phần tử ở giữa thì          // chắc chắn nó nằm bên phải.          // Chỉ định left phần từ ở giữa + 1          if (x > arr[middle])              left = middle + 1;          else              //Ngược lại              right = middle - 1;      }        //Trả về -1 nếu không tìm thấy gía trị trong mảng.      return -1;  }  **int** main() {  **int** arr[] = {15, 20, 25, 30, 31, 44, 66};        //Lấy ra độ dài của mảng  **int** n = sizeof(arr) / sizeof(arr[0]);      //Phần từ cần tìm  **int** x = 25;        // n -1 là vị trí cuối cùng trong mảng.  **int** result = binarySearch(arr, 0, n-1, x);        cout << result;  } |

Khởi chạy chươn trình chúng ta sẽ nhận được kết quả là vị trí của 25 trong mảng.

|  |  |
| --- | --- |
| 1 | Output : 2 |

Trên đây là phần giới thiệu cũng như triển khai của thụât toán tìm kiếm nhị phân. Đây cũng là một thuật toán hay được sử dụng cũng như rât hữu ích trong quá trình giải các bài toán tìm kiếm. Rất mong bài viết sẽ hữu ích cho bạn !

# Thuật toán tìm kiếm nội suy (Interpolation Search)

**Trong bài này mình sẽ giới thiệu thuật toán Interpolation Search (tìm kiếm nội suy). Đây là một trong những thuật toán tìm kiếm được sử dụng rất nhiều vì tốc độ tìm kiếm rất nhanh và chính xác.**



Chúng ta sẽ cùng nhau tìm hiểu về Interpolation Search. Và so sánh nó với Binary Search, để xem sự khác biệt giữa hai cách tìm kiếm này. Sau đó mình sẽ đưa ra thuật toán cho phương pháp tìm kiếm này.

**Mục lục**

* [1. Tìm kiếm nội suy là gì?](https://freetuts.net/tim-kiem-noi-suy-interpolation-search-2922.html#goto-h2-0)
* [2. Thuật toán tìm tiếm nội suy (Interpolation Search)](https://freetuts.net/tim-kiem-noi-suy-interpolation-search-2922.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/tim-kiem-noi-suy-interpolation-search-2922.html#goto-h3-0)
  + [Thuật toán tìm kiếm Interpolation Search](https://freetuts.net/tim-kiem-noi-suy-interpolation-search-2922.html#goto-h3-1)
* [3. Ví dụ tìm kiếm nội suy](https://freetuts.net/tim-kiem-noi-suy-interpolation-search-2922.html#goto-h2-2)

## 1. Tìm kiếm nội suy là gì?

Thuật toán tìm kiếm nội suy là một sự cải tiến của tìm kiếm nhị phân Binary Search. Nó có xu hướng tiến gần đến vị trí, giá trị tìm kiếm. Do đó tốc độ tìm kiếm được tối ưu rất cao và nhanh hơn nhiều so với Binary Search.

Cách thức hoạt động của nó dựa trên [Binary Search](https://freetuts.net/thuat-toan-tim-kiem-nhi-phan-2634.html), nhưng có sự cải tiến hơn. Đó chính là nó tìm ra phần tử gần với giá trị tìm kiếm nhất và bắt đầu từ đó để tìm.

*Bài viết này được đăng tại [free tuts .net]*

**Ví dụ:**

Chúng ta có danh sách các sinh viên trong một lớp. Nếu chúng ta muốn tìm một bạn tên Tiến, thì chúng ta sẽ ưu tiên việc tìm kiếm từ cuối danh sách. Chứ không nên tìm kiếm từ đầu danh sách vì điều đó rất mất thời gian.

Với Interpolation Search nó sẽ linh hoạt hơn rất nhiều trong lúc tìm kiếm.

## 2. Thuật toán tìm tiếm nội suy (Interpolation Search)

**Giả xử chúng ta có:**

* Left, right là hai vị trí đầu và cuối.
* T là tập.
* X là giá trị cần tìm.

### Giải thích thuật toán

**Bước 1**:Chúng ta sẽ sử dụng công thức tìm phần tử chính giữa của tập theo cách tìm kiếm Binary Search:

|  |  |
| --- | --- |
| 1 | Search = left + (right - left) \* 1/2 |

Trong công thức trên chúng ta sẽ thay giá trị 1/2 bằng biểu thức sau:

|  |  |
| --- | --- |
| 1 | (X - T[left]) / (T[right] - T[left]) |

Sau khi thay biểu thức vào công thức sẽ được công thức mới như sau:

|  |  |
| --- | --- |
| 1 | Search = left + (X- T[left]) \* (right – left) / (T[right] – T[left]) |

**Bước 2**: Bây giờ chúng ta sẽ kiểm tra T[Search] nếu bằng X thì Search chính là vị trí cần tìm.

* Nếu Search < X thì tăng left lên một đơn vị rồi quay lại bước 1.
* Nếu Search > X thì giảm right xuống một đơn vị rồi quay lại bước 1.

### Thuật toán tìm kiếm Interpolation Search

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | **int** InterPolationSearch(**int** arr[], **int** n, **int** x)  {  **int** left = 0;  **int** right = n-1;    while (left <= right && x >= arr[left] && x <= arr[right])    {  **double** val1 = (**double**) (x - arr[left]) / (arr[right]-arr[left]);  **int** val2 = (right-left);  **int** Search = left + val1\*val2;        if (arr[Search] == x)        return Search;        if (arr[Search] < x)        left = Search + 1;      else        right = Search - 1;    }    return -1;  } |

## 3. Ví dụ tìm kiếm nội suy

Chúng ta sẽ áp dụng thuật toán trên để tìm một phần tử x trong mảng, được nhập từ bàn phím.

Sau khi đã viết được hàm InterPolationSearch(), thì việc đơn giản của chúng ta sẽ là viết hàm Main và gọi nó ra.

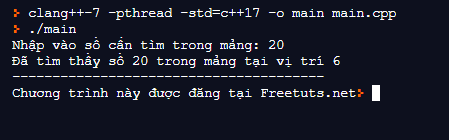
**Main:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **int** main()  {  **int** arr[] =  {10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 33, 35, 42, 47};  **int** n = sizeof(arr)/sizeof(arr[0]);  **int** x;      cout<<"Nhập vào số cần tìm trong mảng: ";    cin>>x;  **int** index = InterPolationSearch(arr, n, x);      if (index != -1)      cout << "Đã tìm thấy số "<<x<< " trong mảng tại vị trí " << index;    else      cout << "Không tìm thấy số "<<x<<" trong mảng.";    } |

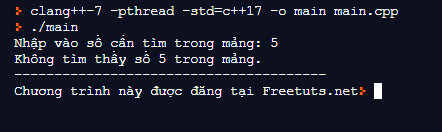
**Full Code:**

|  |  |
| --- | --- |
| 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  38  39  40  41 | #include <iostream>   using namespace std;  **int** InterPolationSearch(**int** arr[], **int** n, **int** x)  {  **int** left = 0;  **int** right = n-1;    while (left <= right && x >= arr[left] && x <= arr[right])    {  **double** val1 = (**double**) (x - arr[left]) / (arr[right]-arr[left]);  **int** val2 = (right-left);  **int** Search = left + val1\*val2;        if (arr[Search] == x)        return Search;        if (arr[Search] < x)        left = Search + 1;      else        right = Search - 1;    }    return -1;  }    **int** main()  {  **int** arr[] =  {10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 33, 35, 42, 47};  **int** n = sizeof(arr)/sizeof(arr[0]);  **int** x;      cout<<"Nhập vào số cần tìm trong mảng: ";    cin>>x;  **int** index = InterPolationSearch(arr, n, x);      if (index != -1)      cout << "Đã tìm thấy số "<<x<< " trong mảng tại vị trí " << index;    else      cout << "Không tìm thấy số "<<x<<" trong mảng.";      cout<<"\n---------------------------------------\n";    cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả 1:**



**Kêt quả 2:**



Như vậy là chúng ta đã thực hiện xong chương trình kiểm tra một số có trong mảng. Cũng như kết thúc hướng dẫn thuật toán tìm kiếm Interpolatipon Search. Chúc các bạn thực hiện thành công!!!

# Thuật toán sắp xếp nổi bọt (Bubble Sort)

**Trong bài này mình sẽ giới thiệu đến các bạn thuật toán sắp xếp nổi bọt (Bubble Sort). Đây là một thuật toán sắp xếp khá đơn giản và dễ hiểu.**



Chúng ta sẽ cùng nhau tìm hiểu về khái niệm sắp xếp nổi bọt (Bubble Sort), thuật toán sắp xếp và cách triển khai. Sau đó mình sẽ có một ví dụ đơn giản áp dụng thuật toán (mình sẽ sử dụng ngôn ngữ C++ để viết).

**Mục lục**

* [1. Sắp xếp nổi bọt (Bubble Sort) là gì?](https://freetuts.net/thuat-toan-sap-xep-noi-bot-bubble-sort-2928.html#goto-h2-0)
* [2. Thuật toán sắp xếp nổi bọt (Bubble Sort) trong C++](https://freetuts.net/thuat-toan-sap-xep-noi-bot-bubble-sort-2928.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/thuat-toan-sap-xep-noi-bot-bubble-sort-2928.html#goto-h3-0)
  + [Thuật toán sắp xếp nổi bọt C++](https://freetuts.net/thuat-toan-sap-xep-noi-bot-bubble-sort-2928.html#goto-h3-1)
* [3. Ví dụ sắp xếp nổi bọt áp dụng với mảng](https://freetuts.net/thuat-toan-sap-xep-noi-bot-bubble-sort-2928.html#goto-h2-2)

## 1. Sắp xếp nổi bọt (Bubble Sort) là gì?

**Sắp xếp nổi bọt được hiểu đơn giản như sau:**

Thuật toán sắp xếp nổi bọt (Bubble Sort) sẽ tiến hành việc so sánh các cặp phần tử liền kề nhau và tráo đổi vị trí của nó cho đúng thứ tự mà người dùng mong muốn.

*Bài viết này được đăng tại [free tuts .net]*

**Ví dụ:**

Chúng ta có một tập các số như sau: A = {20; 5; 25; 30; 22; 40}. Giả sử chúng ta muốn sắp xếp theo chiều tăng dần từ trái sang phải, thì thuật toán sẽ hoạt động như sau:

* Thuật toán sẽ bắt đầu với cặp phần tử đầu tiên là {20; 5}. Trong cặp phần tử này 20 > 5 vì vậy thuật toán sẽ hoán đổi vị trí số 20 cho vị trí số 5. Sau khi hoán đổi được cặp phần tử mới là {5; 20}.
* Tiếp đến thuật toán sẽ so sánh cặp phần tử {20;25}. Trong trường hợp này vị trí của hai phần tử đã đúng, vì vậy thuật toán sẽ giữ nguyên vị trí của nó.
* Cứ như vậy thuật toán sẽ so sánh đến hết các cặp phần tử của tập A. Sau khi so sánh đến hết sẽ được kết quả như sau: A = {5; 20; 22; 25; 30; 40}.

Thuật toán này được áp dụng cho các tập dữ liệu nhỏ, vì nó lặp hết tất cả các phần tử có trong tập. Điều này tốn rất nhiều thời gian, đầy là một thuật toán được xem là chậm nhất trong số các thuật toán sắp xếp.

## 2. Thuật toán sắp xếp nổi bọt (Bubble Sort) trong C++

**Giả sử:**

* Chúng ta có arr là một mảng chứa n phần tử.
* Một hàm Swap() để tráo đổi các phần tử (hàm này mình sẽ viết ở bên dưới).

### Giải thích thuật toán

* Trong thuật toán chúng ta cần tạo một biến ***haveSwap***để kiểm tra xem tại lần lặp hiện tại có xảy ra việc trao đổi hai số hay không.
* Sử dụng vòng lặp for để lặp tất cả các phần tử có trong mảng. Nếu phần tử thứ J > J + 1 thì tiến hành gọi hàm Swap() để tráo đổi vị trí. Sau đó gán ***haveSwap = true*** và ngược lại.
* Nếu không có Swap nào được thực hiện thì mảng đã được sắp xếp. Khi đó chúng ta sẽ thoát khỏi vòng lặp và không cần lặp thêm nữa.

### Thuật toán sắp xếp nổi bọt C++

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | void bubbleSort(**int** arr[], **int** n)  {  **int** i, j;  **bool** haveSwap = false;      for (i = 0; i < n-1; i++){      // i phần tử cuối cùng đã được sắp xếp          haveSwap = false;          for (j = 0; j < n-i-1; j++){              if (arr[j] > arr[j+1]){                  swap(arr[j], arr[j+1]);                  haveSwap = true; // Kiểm tra lần lặp này có swap không              }          }          // Nếu không có swap nào được thực hiện => mảng đã sắp xếp. Không cần lặp thêm          if(haveSwap == false){              break;          }      }  } |

## 3. Ví dụ sắp xếp nổi bọt áp dụng với mảng

Chúng ta sẽ áp dụng thuật toán trên để làm một bài toán đơn giản. Cụ thể chúng ta sẽ sắp xếp các phần tử có trong một mảng được nhập tử bàn phím.

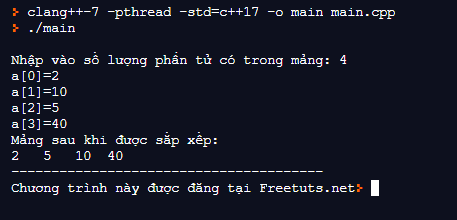
**Gợi ý:**

* Việc đầu tiên chúng ta cần tạo hàm Swap() để thực hiện tráo đổi vị trí các phần tử.
* Tiếp theo cần tạo hàm bubbleSort() để thực hiện sắp xếp các phần tử.
* Cuối cùng chỉ cần tạo hàm main và yêu cầu người dùng nhập vào các phần tử cho mảng. Sau đó gọi hàm bubbleSort() để sắp xếp là xong.

**Code Mẫu:**

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64 | #include <iostream>  #include <stdio.h>  using namespace std;  void swap(**int** &x, **int** &y)  {  **int** temp = x;      x = y;      y = temp;  }    // Hàm sắp xếp bubble sort  void bubbleSort(**int** arr[], **int** n)  {  **int** i, j;  **bool** haveSwap = false;      for (i = 0; i < n-1; i++){      // i phần tử cuối cùng đã được sắp xếp          haveSwap = false;          for (j = 0; j < n-i-1; j++){              if (arr[j] > arr[j+1]){                  swap(arr[j], arr[j+1]);                  haveSwap = true; // Kiểm tra lần lặp này có swap không              }          }          // Nếu không có swap nào được thực hiện => mảng đã sắp xếp. Không cần lặp thêm          if(haveSwap == false){              break;          }      }  }    /\* Hàm xuất mảng \*/  void printArray(**int** arr[], **int** size)  {  **int** i;      for (i=0; i < size; i++){          cout << arr[i];          cout<<"\t";        }  }    **int** main()  {  **int** n;      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);    **int** a[n];        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };        bubbleSort(a, n);        cout<<"Mảng sau khi được sắp xếp: \n";      printArray(a, n);        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình sắp xếp các phần tử có trong mảng. Cũng như kết thúc hướng dẫn thuật toán sắp xếp nổi bọt (Bubble Sort) trong C++. Chúc các bạn thực hiện thành công!!!.

# Thuật toán sắp xếp chèn (Insertion Sort)

**Trong bài này mình sẽ giới thiệu đến các bạn thuật toán sắp xếp chèn (Insertion Sort). Đây là một trong những thuật toán sắp xếp căn bản trong C++.**



Chúng ta sẽ cùng nhau tìm hiểu thuật toán sắp xếp chèn là gì và cách triển khai nó như thế nào trong C++. Ở cuối bài viết mình sẽ thực hành một bài tập đơn giản áp dụng thuật toán để các bạn dễ hiểu hơn.

**Mục lục**

* [1. Sắp xếp chèn là gì?](https://freetuts.net/thuat-toan-sap-xep-chen-insertion-sort-2930.html#goto-h2-0)
* [2. Thuật toán sắp xếp chèn (Insertion Sort)](https://freetuts.net/thuat-toan-sap-xep-chen-insertion-sort-2930.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/thuat-toan-sap-xep-chen-insertion-sort-2930.html#goto-h3-0)
  + [Thuật toán sắp xếp chèn](https://freetuts.net/thuat-toan-sap-xep-chen-insertion-sort-2930.html#goto-h3-1)
* [3. Ví dụ sắp xếp chèn viết trong C++](https://freetuts.net/thuat-toan-sap-xep-chen-insertion-sort-2930.html#goto-h2-2)

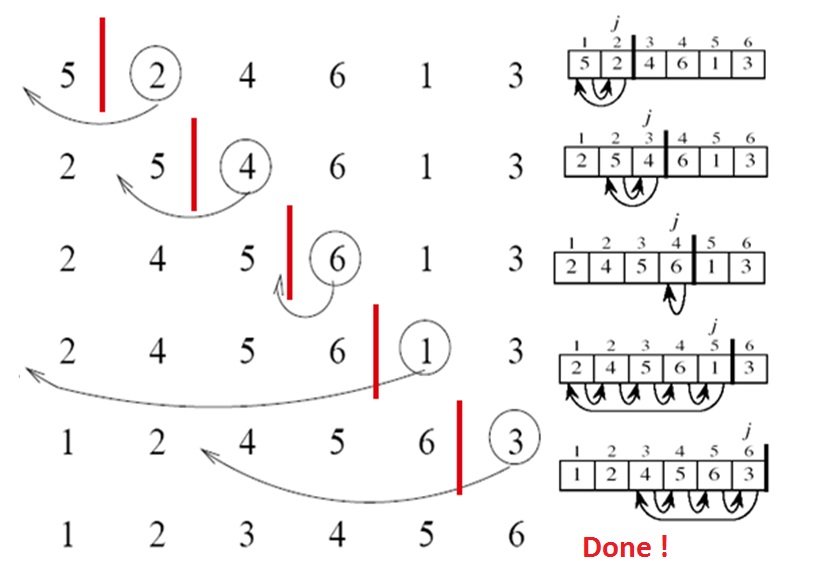
## 1. Sắp xếp chèn là gì?

Sắp xếp chèn về cơ bản cách thức hoạt động của nó khá giống với sắp xếp [Bubble Sort].

Thuật toán sắp xếp chèn (Insertion Sort) thực hiện sắp xếp các phần tử theo cách duyệt từng phần tử. Và chèn từng phần tử đó vào đúng vị trí trong mảng con. Phần tử được chuyền vào vị trí thích hợp sao cho mảng con vẫn đảm bảo sắp xếp theo đúng thứ tự.

*Bài viết này được đăng tại [free tuts .net]*

**Ví dụ minh họa:**



Như các bạn đã thấy ở hình trên, phía bên trái dấu gạch đỏ (mảng con) đã được sắp xếp theo đúng thứ tự. Còn phía bên phải vẫn chưa được sắp xếp.

Ở dòng đầu tiên, dãy số bao gồm: 5 2 4 6 1 3. Ta sẽ bắt đầu so sánh cặp phần tử đầu tiên đó chính là 5 và 2. Theo thứ tự tăng dần thì số 5 sẽ xếp sau số 2, vì vậy thuật toán sẽ chèn số 2 trước số 5. Như vậy mảng con lúc này có 2 phần tử đó là 2; 5.

Ở dòng thứ hai, tiếp tục so sánh cặp phần tử tiếp theo là 5 và 4. Trong trường hợp này 5 > 4, vì vậy số 4 sẽ được chèn vào mảng con ở vị trí sau số 2 và trước số 5. Khi đó mảng con bao gồm 2; 4; 5.

Cứ như vậy, thuật toán sẽ so sánh và chèn các phần tử đúng vào vị trí trong mảng con cho đến phần tử cuối cùng.

## 2. Thuật toán sắp xếp chèn (Insertion Sort)

Trong phần này mình sẽ giải thích thuật toán sắp xếp chèn cho các bạn hiểu, sau đó sẽ viết thuật toán ở bên dưới cho các bạn dễ hình dung.

### Giải thích thuật toán

1. Kiểm tra phần tử đầu tiên xem đã được sắp xếp hay chưa, nếu rồi thì trả về 1.
2. Lấy phần tử tiếp theo để kiểm tra.
3. So sánh với tất cả các phần tử con đã được sắp xếp.
4. Dịch chuyển tất cả các phần tử có trong mảng con nếu nó lớn hơn giá trị của phần tử đó.
5. Sau khi dịch chuyển, tiến hành chèn giá trị đó.
6. Lặp lại như vậy cho đến khi các phần tử đã được sắp xếp theo thứ tự đúng.

### Thuật toán sắp xếp chèn

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | void insertionSort(**int** arr[], **int** n)  {  **int** i, key, j;     for (i = 1; i < n; i++)     {         key = arr[i];         j = i-1;           /\* Di chuyển các phần tử có giá trị lớn hơn giá trị         key về sau một vị trí so với vị trí ban đầu         của nó \*/         while (j >= 0 && arr[j] > key)         {             arr[j+1] = arr[j];             j = j-1;         }         arr[j+1] = key;     }  } |

## 3. Ví dụ sắp xếp chèn viết trong C++

Chúng ta sẽ áp dụng thuật toán trên để làm một ví dụ thực tế. Cụ thể bài toán như sau:

* Tạo một mảng bao gồm các số nguyên được nhập từ bàn phím, sau đó thực hiện sắp xếp các phần tử trong mảng sử dụng thuật toán sắp xếp chèn.
* Hiển thị kết quả đã được sắp xếp ra màn hình.

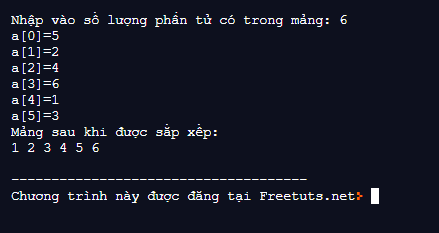
**Gợi ý:**

* Chúng ta sẽ bắt đầu với việc viết hàm insertionSort() để sắp xếp các phần tử.
* Sau đó viêt hàm printArray() để in mảng sau khi đã được sắp xếp ra màn hình.
* Cuối cùng chỉ cần tạo một mảng các số nguyên được nhập từ bàn phím trong hàm main(). Và gọi các hàm đã viết ở trên là xong.

**Code mẫu:**

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | #include <stdio.h>  #include <math.h>  #include <iostream>  using namespace std;  /\* Hàm sắp xếp sử dụng thuật toán sắp xếp chèn \*/  void insertionSort(**int** arr[], **int** n)  {  **int** i, key, j;     for (i = 1; i < n; i++)     {         key = arr[i];         j = i-1;           /\* Di chuyển các phần tử có giá trị lớn hơn giá trị         key về sau một vị trí so với vị trí ban đầu         của nó \*/         while (j >= 0 && arr[j] > key)         {             arr[j+1] = arr[j];             j = j-1;         }         arr[j+1] = key;     }  }    /\* Hàm xuất mảng \*/  void printArray(**int** arr[], **int** size)  {  **int** i;      for (i=0; i < size; i++){          cout << arr[i];          cout<<"\t";        }  }    **int** main()  {  **int** n;  **int** a[n];      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };        insertionSort(a, n);      cout<<"Mảng sau khi được sắp xếp: \n";      printArray(a, n);        cout<<"\n-------------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình sắp xếp các phần tử trong mảng, sử dụng cách sắp xếp chèn (Insertion Sort). Cũng như kết thúc hướng dẫn thuật toán sắp xếp chèn, chúc các bạn thực hiện thành công!!!

# Thuật toán sắp xếp chọn (Selection Sort)

**Trong bài này mình sẽ giới thiệu đến các bạn thuật toán sắp xếp chọn (Selection Sort). Đây là một trong những thuật toán sắp xếp căn bản trong C++.**



Chúng ta sẽ cùng nhau tìm hiểu về sắp xếp chọn là gì. Cách triển khai thuật toán trong C++ và ví dụ cụ thể áp dụng thuật toán để các bạn hiểu rõ hơn.

**Mục lục**

* [1. Sắp xếp chọn là gì?](https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html#goto-h2-0)
* [Thuật toán sắp xếp chọn (Selection Sort) trong C++.](https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html#goto-h3-0)
  + [Thuật toán sắp xếp chọn](https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html#goto-h3-1)
* [Ví dụ sắp xếp chọn trong mảng](https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html#goto-h2-2)

## 1. Sắp xếp chọn là gì?

Sắp xếp chọn là một trong những thuật toán đơn giản. Nó thực hiện công việc so sánh các phần tử trong danh sách.

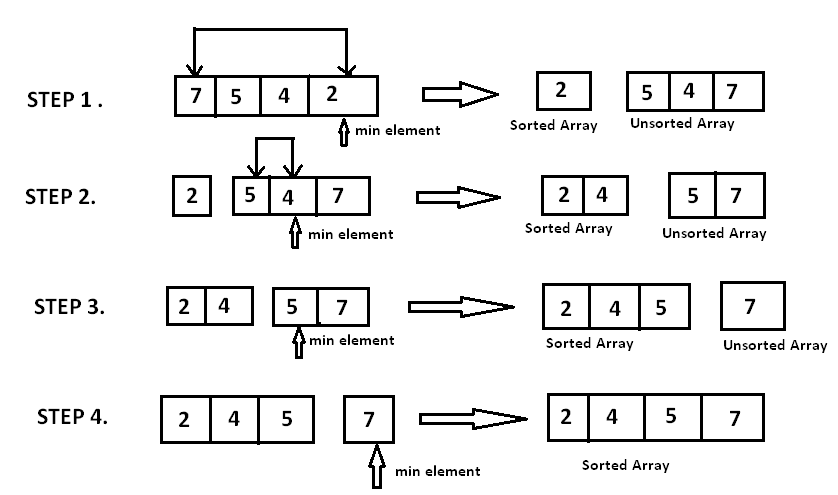
Danh sách chứa các phần tử sẽ được chia làm hai phần. Phần ở bên trái là phần được sắp xếp (**sort list**) và phần ở bên phải là phần chưa được sắp xếp (**unsorted list**).

*Bài viết này được đăng tại [free tuts .net]*

Ban đầu chưa được sắp xếp thì phần **sorted list** sẽ trống và phần **unsorted list** sẽ chứa tất cả các phần tử ban đầu.

Phần tử nhỏ nhất trong list sẽ được chọn và tráo đổi với vị trí đầu tiên trong list, tiếp đến sẽ là vị trí nhỏ thứ hai tiếp tục được tráo đổi ngay sau vị trí nhỏ nhất.

**Ví dụ minh họa:**



Như các bạn thấy ở hình trên. Ở Step1, list ban đầu là 7 5 4 2. Trong list, số 2 là nhỏ nhất vì vậy chọn số 2 vào sorted list và thay thế vị trí số 7. Như vậy phần sorted list đã có số 2 và phần unsorted list còn lại 3 số là 5 4 7.

Ở Step2, chúng ta bắt đầu tìm số nhỏ thứ hai trong list đó là số 4. Tiếp tục chọn số 4 vào sorted list ở vị trí ngay sau số 2 là số nhỏ nhất. Trong sorted list bây giờ có hai số là 2 5 và unsorted list còn lại hai số là 5 7.

Cứ như vậy Step3 và Step4 sẽ tìm số nhỏ tiếp theo trong list và chọn vào sorted list, cho đến khi các phần tử trong list đã sắp xếp xong.

## Thuật toán sắp xếp chọn (Selection Sort) trong C++.

Trong phần này mình sẽ đưa ra các bước để viết thuật toán, sau đó sẽ để thuật toán mình đã viết cho các bạn dễ hình dung.

### Giải thích thuật toán

1. Thiết lập giá trị nhỏ nhất (min) về vị trí số 0.
2. Tạo vòng lặp for để di chuyển ranh giới của **sorted list** và **unsorted list**.
3. Tìm phần tử nhỏ nhất trong list chưa được sắp xếp.
4. Sau khi tìm được phần tử nhỏ nhất thì đổi chỗ phần tử đó với phần tử đầu tiên. Ở bước này chúng ta cần viết một hàm Swap(), hàm này mình sẽ viết ở bên dưới.
5. Lặp lại cho tới khi list được sắp xếp xong.

### Thuật toán sắp xếp chọn

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | void selectionSort(**int** arr[], **int** n)  {  **int** i, j, min\_idx;      // Di chuyển ranh giới của mảng đã sắp xếp và chưa sắp xếp      for (i = 0; i < n-1; i++)      {      // Tìm phần tử nhỏ nhất trong mảng chưa sắp xếp      min\_idx = i;      for (j = i+1; j < n; j++)          if (arr[j] < arr[min\_idx])          min\_idx = j;        // Đổi chỗ phần tử nhỏ nhất với phần tử đầu tiên          swap(arr[min\_idx], arr[i]);      }  } |

Hàm swap():

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void swap(**int** &xp, **int** &yp)  {  **int** temp = xp;      xp = yp;      yp = temp;  } |

## Ví dụ sắp xếp chọn trong mảng

Trong phần ví dụ sắp xếp chọn này, chúng ta sẽ thực hành một bài toán đơn giản đó là:

* Tạo mảng gồm các phần tử được nhập từ bàn phím.
* Tiến hành sắp xếp các phần tử sử dụng phương pháp sắp xếp chọn.
* Hiển thị mảng đã sắp xếp ra màn hình.

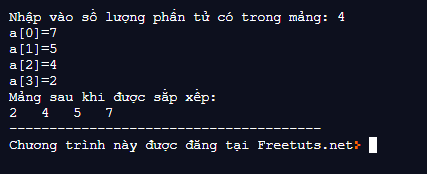
**Gợi ý:**

* Việc đầu tiên sẽ viết hàm swap() để thực hiện công việc tráo đổi vị trí khi tìm được phần tử nhỏ nhất trong mảng.
* Tiếp đến viết hàm selectionSort() để thực hiện công việc sắp xếp các phần tử trong mảng.
* Viết hàm printSort() để in mảng đã được sắp xếp.
* Tạo mảng gồm các phần tử được nhập từ bàn phím trong hàm main(). Gọi hàm selectionSort() để sắp xếp.
* Cuối cùng gọi hàm printSort() để in mảng ra màn hình.

**Code mẫu:**

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | #include <stdio.h>  #include <iostream>  using namespace std;  // Hàm đổi chỗ 2 số nguyên  void swap(**int** &xp, **int** &yp)  {  **int** temp = xp;      xp = yp;      yp = temp;  }    // Hàm selection sort  void selectionSort(**int** arr[], **int** n)  {  **int** i, j, min\_idx;      // Di chuyển ranh giới của mảng đã sắp xếp và chưa sắp xếp      for (i = 0; i < n-1; i++)      {      // Tìm phần tử nhỏ nhất trong mảng chưa sắp xếp      min\_idx = i;      for (j = i+1; j < n; j++)          if (arr[j] < arr[min\_idx])          min\_idx = j;        // Đổi chỗ phần tử nhỏ nhất với phần tử đầu tiên          swap(arr[min\_idx], arr[i]);      }  }    /\* Hàm xuất mảng \*/  void printArray(**int** arr[], **int** size)  {  **int** i;      for (i=0; i < size; i++){          cout << arr[i];          cout<<"\t";        }  }    **int** main()  {  **int** n;  **int** a[n];      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };        selectionSort(a, n);        cout<<"Mảng sau khi được sắp xếp: \n";      printArray(a, n);        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình sắp xếp mảng bằng phương pháp sắp xếp chọn. Cũng như kết thúc hướng dẫn thuật toán sắp xếp chon (Selection Sort) trong C++. Chúc các bạn thực hiện thành công!!!

# Thuật toán sắp xếp trộn (Merge Sort)

**Trong bài này mình sẽ giới thiệu đến các bạn thuật toán sắp xếp trộn (Merge Sort). Đây là một trong những thuật toán sắp xếp trong C++.**



Chúng ta sẽ cùng nhau tìm hiều về sắp xếp trộn là gì và cách triển khai thuật toán sắp xếp trộn trong C++. Ở cuối bài viết, mình sẽ giải một bài toán cụ thể để các bạn hiểu hơn về cách áp dụng thuật toán trong thực tế.

**Mục lục**

* [1. Sắp xếp trộn là gì?](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html#goto-h2-0)
* [2. Thuật toán sắp xếp trộn (Merge Sort) trong C++](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html#goto-h3-0)
  + [Code thuật toán sắp xếp trộn](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html#goto-h3-1)
* [3. Ví dụ sắp xếp trộn trong mảng](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html#goto-h2-2)

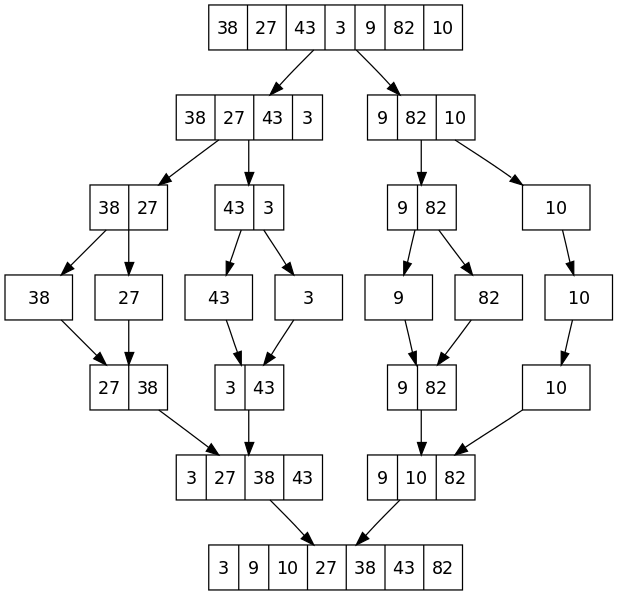
## 1. Sắp xếp trộn là gì?

Sắp xếp trộn là một thuật toán sắp xếp dựa trên giải thuật **Divide and Conquer** (Chia để trị).

Thuật toán này sẽ chia mảng thành hai nữa rồi sắp xếp trên từng nữa một. Sau đó kết hợp chúng lại với nhau thành một mảng đã được sắp xếp.

*Bài viết này được đăng tại [free tuts .net]*

**Ví dụ minh họa:**



Như các bạn đã thấy ở hình trên, dãy số ban đầu bao gôm các số: 38 27 43 3 9 82 10.

Chúng ta sẽ chia thành hai phần là một bên 4 số và một bên 3 số. Rồi tiếp tục chia 4 số đó thành hai phần là mỗi bên 2 số. Cứ chia như vậy cho đến khi được kết quả như dòng thứ 4.

Sau khi chia xong, bây giờ chúng ta bắt đầu vào việc so sanh từng phần nhỏ. Rồi gom chúng lại thành một dãy số hoàn chỉnh đã sắp xếp ở các dòng 5, 6, 7 như trong hình.

Sau khi gom lại và so sánh xong ta được dãy số mới đã sắp xếp: 3 9 10 27 38 43 82.

## 2. Thuật toán sắp xếp trộn (Merge Sort) trong C++

Trong phần này mình sẽ đưa ra các bước để viết thuật toán, sau đó mình sẽ để thuật toán đã viết sẵn cho các bạn dễ hình dung.

### Giải thích thuật toán

Trong thuật toán sẽ có hai bước đó là chia phần tử và gộp phần tử (kèm theo sắp xếp). Cụ thể trong thuật toán mình viết hàm mergeSort() để chia phần tử và hàm merge() để gộp, sắp xếp phần tử.

1. Sử dụng đệ qui để chia list thành hai nửa cho đến khi không chia thêm được nữa (hàm mergeSorrt()).
2. Tạo hai mảng tạm thời để chứa các phần tử sau khi chia, cùng với đó là hai mảng con L và R.
3. Trường hợp chỉ có một phần tử thì xem như đã được sắp xếp.
4. Sau khi chia xong, sẽ gộp các phần tử ở mảng con R và L vào mảng chính arr.
5. Kết hợp các list nhỏ đã sắp xếp với nhau thành một list mới. Sau khi kết hợp tiến hành sắp xếp chúng để tiếp tục cho lần kết hợp kế tiếp.

### Code thuật toán sắp xếp trộn

Trong thuật toán mình đã chú thích cụ thể, các bạn có thể đọc để dễ hiểu hơn.

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | void merge(**int** arr[], **int** l, **int** m, **int** r)  {  **int** i, j, k;  **int** n1 = m - l + 1;  **int** n2 =  r - m;    **int** L[n1], R[n2];//tạo 2 mảng tạm thời để chứa các phần tử sau khi chia        // Copy dữ liệu sang các mảng tạm      for (i = 0; i < n1; i++)          L[i] = arr[l + i];      for (j = 0; j < n2; j++)          R[j] = arr[m + 1+ j];        // khởi tạo các giá trị ban đầu      i = 0;      j = 0;      k = l;        //gộp hai mảng tạm thời vào mảng arr      while (i < n1 && j < n2)      {          if (L[i] <= R[j])          {              arr[k] = L[i];              i++;          }          else          {              arr[k] = R[j];              j++;          }          k++;      }        // Copy các phần tử còn lại của mảng L vào arr nếu có      while (i < n1)      {          arr[k] = L[i];          i++;          k++;      }        // Copy các phần tử còn lại của mảng R vào arr nếu có      while (j < n2)      {          arr[k] = R[j];          j++;          k++;      }  }    // l là chỉ số trái và r là chỉ số phải của mảng cần được sắp xếp  void mergeSort(**int** arr[], **int** l, **int** r)  {      if (l < r)      {  **int** m = l+(r-l)/2;          // Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng          mergeSort(arr, l, m);          mergeSort(arr, m+1, r);            merge(arr, l, m, r);      }  } |

## 3. Ví dụ sắp xếp trộn trong mảng

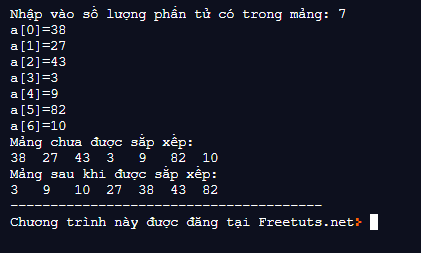
Chúng ta sẽ áp dụng thuật toán đã viết ở trên, để viết một chương trình sắp xếp các phần tử trong mảng được nhập từ bàn phím.

Tương tự như các bài trước, chỉ cần thay thế các thuật toán khác bằng thuật toán mergeSort() là được.

**Code mẫu:**

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107 | #include<stdlib.h>  #include<stdio.h>  #include <iostream>  using namespace std;  // Chúng ta cần có hai mảng con vì vậy cần tạo hai mảng con arr[l...m] và arr[m+1..r]. Sau đó gộp chúng lại  void merge(**int** arr[], **int** l, **int** m, **int** r)  {  **int** i, j, k;  **int** n1 = m - l + 1;  **int** n2 =  r - m;    **int** L[n1], R[n2];//tạo 2 mảng tạm thời để chứa các phần tử sau khi chia        // Copy dữ liệu sang các mảng tạm      for (i = 0; i < n1; i++)          L[i] = arr[l + i];      for (j = 0; j < n2; j++)          R[j] = arr[m + 1+ j];        // khởi tạo các giá trị ban đầu      i = 0;      j = 0;      k = l;        //gộp hai mảng tạm thời vào mảng arr      while (i < n1 && j < n2)      {          if (L[i] <= R[j])          {              arr[k] = L[i];              i++;          }          else          {              arr[k] = R[j];              j++;          }          k++;      }        // Copy các phần tử còn lại của mảng L vào arr nếu có      while (i < n1)      {          arr[k] = L[i];          i++;          k++;      }        // Copy các phần tử còn lại của mảng R vào arr nếu có      while (j < n2)      {          arr[k] = R[j];          j++;          k++;      }  }    // l là chỉ số trái và r là chỉ số phải của mảng cần được sắp xếp  void mergeSort(**int** arr[], **int** l, **int** r)  {      if (l < r)      {  **int** m = l+(r-l)/2;          // Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng          mergeSort(arr, l, m);          mergeSort(arr, m+1, r);            merge(arr, l, m, r);      }  }    /\* Hàm xuất mảng \*/  void printArray(**int** arr[], **int** size)  {  **int** i;      for (i=0; i < size; i++){          cout << arr[i];          cout<<"\t";        }  }    **int** main()  {  **int** n;      do{          cout << "\nNhập vào số lượng phần tử có trong mảng: ";          cin >> n;      }while(n <= 0);    **int** a[n];        for(**int** i = 0;i < n;i++){          cout<<"a["<<i<<"]=";         cin >> a[i];      };        cout<<"Mảng chưa được sắp xếp: \n";      printArray(a, n);        mergeSort(a, 0, n - 1);        cout<<"\nMảng sau khi được sắp xếp: \n";      printArray(a, n);        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình sắp xếp các phần tử trong mảng, sử dụng phương thức sắp xếp trộn. Cũng như kết thúc hướng dẫn thuật toán sắp xếp trộn trong C++. Chúc các bạn thực hiện thành công!!!

# Thuật toán sắp xếp nhanh (Quick Sort)

**Trong bài này mình sẽ giới thiệu thuật toán Quick Sort (sắp xếp nhanh), đây là thuật toán sắp xếp được xem là nhành nhất trong các thuật toán mình đã giới thiệu trước đây.**



Chúng ta sẽ cùng nhau tìm hiểu về sắp xếp nhanh là gì? Cũng như cách thức nó hoạt động và triển khai trong C++ như thế nào.

Ở cuối bài viết mình sẽ có một ví dụ sắp xếp trong mảng. Để các bạn áp dụng được thuật toán trong các bài tập thực tế.

**Mục lục**

* [1. Sắp xếp nhanh (Quick Sort) là gì?](https://freetuts.net/thuat-toan-sap-xep-nhanh-quick-sort-2940.html#goto-h2-0)
* [2. Thuật toán Quick Sort trong C++](https://freetuts.net/thuat-toan-sap-xep-nhanh-quick-sort-2940.html#goto-h2-1)
  + [Giải thích thuật toán](https://freetuts.net/thuat-toan-sap-xep-nhanh-quick-sort-2940.html#goto-h3-0)
  + [Thuật toán Quick Sort trong C++](https://freetuts.net/thuat-toan-sap-xep-nhanh-quick-sort-2940.html#goto-h3-1)
* [3. Ví dụ sắp xếp Quick Sort trong mảng](https://freetuts.net/thuat-toan-sap-xep-nhanh-quick-sort-2940.html#goto-h2-2)

## 1. Sắp xếp nhanh (Quick Sort) là gì?

Về cơ bản thuật toán sắp xếp Quick Sort khá giống như [Merge Sort](https://freetuts.net/thuat-toan-sap-xep-tron-merge-sort-2932.html). Đây là một thuật toán áp dụng cách thức chia để trị (**Divide and Conquer**).

*Bài viết này được đăng tại [free tuts .net]*

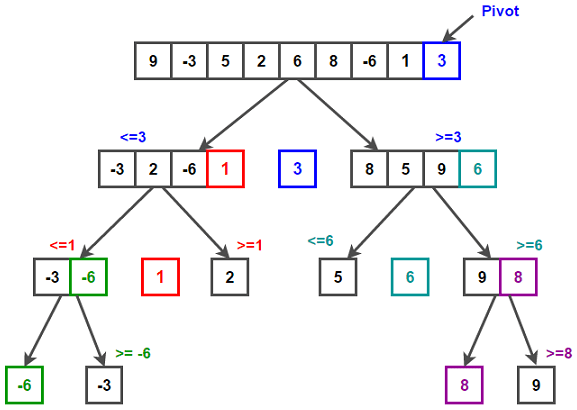
Thuật toán sẽ chọn mốt phần tử trong mảng làm điểm đánh dấu (**pivot**). Khi chọn được điểm đánh dấu, nó sẽ chia mảng thành hai mảng con bằng cách so sánh với pivot đã chọn. Một mảng bao gồm các phần tử nhỏ hơn hoặc bằng pivot và mảng còn lại sẽ lớn hơn hoặc bằng pivot.

Tốc độ sắp xếp của thuật toán tùy thuộc vào việc lựa chọn pivot, có một số cách chọn như sau:

1. Chọn phần tử đầu tiên của mảng.
2. Chọn phần tử cuối cùng của mảng.
3. Chọn phần tử có giá trị nằm giữa mảng.
4. Chọn Random một phần tử của mảng.

Trên đây là một số cách chọn thông dụng, được sử dụng nhiều nhất. Tùy thuộc vào bài toán mà ta chọn pivot cho phù hợp.

Các bạn có thể xem hình minh họa dưới đây để có thể hiểu hơn.



Khi chúng ta chọn số 3 làm pivot, thì mảng được chia làm hai phần. Phần bên trái nhỏ hơn hoặc bằng 3, phần bên phải lớn hơn hoặc bằng 3, cả hai phần đều được sắp xếp tằng dần.

Tiếp tục chọn pivot cho mỗi phần. Phần bên trái có pivot là số 1 và phần bên phải có pivot là số 6.

Cứ như vậy chúng ta sẽ chia phần ra cho đến khi không chia được nữa và sắp xếp theo đúng thứ tự.

## 2. Thuật toán Quick Sort trong C++

### Giải thích thuật toán

Trong phần này chúng ta có hai giai đoạn. Giai đoạn một là giai đoạn phân đoạn mảng (partition()) và giai đoạn hai là giai đoạn sắp xếp (quickSort()).

* Chọn pivot cho mảng, ở đây mình sẽ chọn pivot là số cuối cùng của mảng.
* Tạo hai biến là left và right để trỏ tới bên trái và bên phải của danh sách.
* Thực hiện so sánh các phần tử với pivot. Nếu phần tử nhỏ hơn pivot thì dịch chuyển qua bên trái và ngược lại.
* Sau khi dịch chuyển thực hiện công việc sắp xếp các phần tử trong mảng con mới, trước khi tiếp tục phân đoạn tiếp theo.

### Thuật toán Quick Sort trong C++

Ở phần trên mình đã nêu ra các bước viết thuật toán. Để chi tiết hơn mình đã có chú thích rõ ràng, cụ thể trong từng dòng code trong thuật toán dưới đây. Các bạn hãy đọc thật kỹ nhé:

**Hàm partition()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | **int** partition (**int** arr[], **int** low, **int** high)  {  **int** pivot = arr[high];    // khai báo phần tử đánh dâu pivot  **int** left = low;   //khai báo biến bên trái  **int** right = high - 1; //khai báo biến bên phải      while(true){          while(left <= right && arr[left] < pivot) left++; // tìm phần tử >= phần tử pivot trong mảng          while(right >= left && arr[right] > pivot) right--; // tìm phần tử <= phần tử pivot trong mảng          if (left >= right) break; // sau khi duyệt xong thì thoát khỏi vòng lặp          swap(arr[left], arr[right]); // nếu chưa xong thì sử dụng hàm swap() để tráo đổi.          left++; // Vì left hiện tại đã xét, nên cần tăng          right--; // Vì right hiện tại đã xét, nên cần giảm      }      swap(arr[left], arr[high]);      return left; // Trả về chỉ số sẽ dùng để chia đôi mảng  } |

**Hàm quicksort()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | void quickSort(**int** arr[], **int** low, **int** high)  {      if (low < high)      {          /\* index là chỉ số nơi phần tử này đã đứng đúng vị trí           và đây là phần tử chia mảng làm 2 mảng con trái & phải \*/  **int** index = partition(arr, low, high);            // Gọi đệ quy sắp xếp 2 mảng con trái và phải          quickSort(arr, low, index - 1);          quickSort(arr, index + 1, high);      }  } |

**Hàm swap()**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | void swap(**int** &a, **int** &b)  {  **int** t = a;      a = b;      b = t;  } |

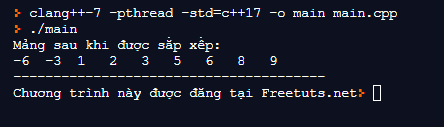
## 3. Ví dụ sắp xếp Quick Sort trong mảng

Để minh họa cho hình ảnh ở trên, mình sẽ làm ví dụ áp dụng thuật toán sắp xếp nhanh (Quick Sort). Sắp xếp các phần tử trong mảng arr[] = {9, -3, 5, 2, 6, 8, -6, 1, 3} theo thứ tự tăng dần.

**Code mẫu:**

|  |  |
| --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | #include<stdio.h>  #include<iostream>  using namespace std;    //tạo hàm swap để tráo đổi các vị trí  void swap(**int** &a, **int** &b)  {  **int** t = a;      a = b;      b = t;  }    //  phân đoạn trong mảng  **int** partition (**int** arr[], **int** low, **int** high)  {  **int** pivot = arr[high];    // khai báo phần tử đánh dâu pivot  **int** left = low;   //khai báo biến bên trái  **int** right = high - 1; //khai báo biến bên phải      while(true){          while(left <= right && arr[left] < pivot) left++; // tìm phần tử >= phần tử pivot trong mảng          while(right >= left && arr[right] > pivot) right--; // tìm phần tử <= phần tử pivot trong mảng          if (left >= right) break; // sau khi duyệt xong thì thoát khỏi vòng lặp          swap(arr[left], arr[right]); // nếu chưa xong thì sử dụng hàm swap() để tráo đổi.          left++; // Vì left hiện tại đã xét, nên cần tăng          right--; // Vì right hiện tại đã xét, nên cần giảm      }      swap(arr[left], arr[high]);      return left; // Trả về chỉ số sẽ dùng để chia đôi mảng  }    /\* Hàm thực hiện giải thuật quick sort \*/  void quickSort(**int** arr[], **int** low, **int** high)  {      if (low < high)      {          // index là chỉ số nơi phần tử này đã đứng đúng vị trí và đây là phần tử chia mảng làm 2 mảng con trái & phải  **int** index = partition(arr, low, high);            // Gọi đệ quy sắp xếp 2 mảng con trái và phải          quickSort(arr, low, index - 1);          quickSort(arr, index + 1, high);      }  }    /\* Hàm xuất mảng \*/  void printArray(**int** arr[], **int** size)  {  **int** i;      for (i=0; i < size; i++){          cout << arr[i];          cout<<"\t";        }  }    **int** main()  {  **int** arr[] = {9, -3, 5, 2, 6, 8, -6, 1, 3};  **int** n = sizeof(arr)/sizeof(arr[0]);      quickSort(arr, 0, n-1);        cout<<"Mảng sau khi được sắp xếp: \n";      printArray(arr, n);        cout<<"\n---------------------------------------\n";      cout<<"Chương trình này được đăng tại Freetuts.net";  } |

**Kết quả:**



Như vậy là chúng ta đã thực hiện xong chương trình sắp xếp mảng bằng phương pháp Quick Sort. Cũng như kết thúc hướng dẫn thuật toán sắp xếp nhanh (Quick Sort) trong C++. Chúc các bạn thực hiện thành công!!!