BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**BÁO CÁO THỰC TẬP CƠ SỞ**

**ĐỀ TÀI: MÔ PHỎNG CÁC THUẬT TOÁN SẮP XẾP**

**Giáo viên hướng dẫn: Bùi Chí Thành**

**Sinh viên thực hiện: Lê Minh Thành**

**MSSV: 63131263**

**Khánh Hoà – 2023**

**MỤC LỤC**

[DANH MỤC HÌNH VẼ 5](#_Toc155620758)

[DANH MỤC BẢNG 6](#_Toc155620759)

[Chương 1. Giới thiệu đề tài 7](#_Toc155620760)

[1.1 Lý do chọn đề tài 7](#_Toc155620761)

[1.2 Mục tiêu và nhiệm vụ 7](#_Toc155620762)

[Chương 2. Cơ sở lý thuyết 8](#_Toc155620763)

[2.1 Ngôn ngữ C# 8](#_Toc155620764)

[2.1.1 Khái niệm 8](#_Toc155620765)

[2.1.2 Ứng dụng của ngôn ngữ lập trình C# 8](#_Toc155620766)

[2.2 Thuật toán 8](#_Toc155620767)

[2.2.1 Khái niệm thuật toán 8](#_Toc155620768)

[2.2.2 Đặc điểm thuật toán 9](#_Toc155620769)

[2.2.3 Tổng hợp các thuật toán sắp xếp hiện nay 9](#_Toc155620770)

[Chương 3. Chương trình ứng dụng thuật toán sắp xếp 10](#_Toc155620771)

[3.1 Sơ lược mô phỏng minh họa trên C# 10](#_Toc155620772)

[3.1.1 Các chức năng 10](#_Toc155620773)

[3.1.2 Các kỹ thuật xử lý 11](#_Toc155620774)

[3.1.2.1 Phân tích hiệu ứng 11](#_Toc155620775)

[3.1.2.2 Phân tích chức năng 11](#_Toc155620776)

[3.1.3 Các kỹ thuật sử dụng 12](#_Toc155620777)

[3.1.3.1 Xử lý bất đồng bộ với Thread và Task 12](#_Toc155620778)

[3.1.3.2 Xử lý việc thay đổi tốc độ sắp xếp 13](#_Toc155620779)

[3.1.3.3 Xử lý việc di chuyển các phần tử và các biến 13](#_Toc155620780)

[3.1.3.4 Xử lý việc tính thời gian thực hiện quá trình sắp xếp 13](#_Toc155620781)

[3.1.3.5 Xử lý debug 14](#_Toc155620782)

[3.2 Các thuật toán sắp xếp đơn giản 14](#_Toc155620783)

[3.2.1 Thuật toán Interchange Sort (Đổi chỗ trực tiếp) 14](#_Toc155620784)

[3.2.1.1 Ý tưởng thuật toán 14](#_Toc155620785)

[3.2.1.2 Cài đặt mã giả 14](#_Toc155620786)

[3.2.1.3 Đánh giá độ phức tạp 14](#_Toc155620787)

[3.2.2 Thuật toán Selection Sort (Sắp xếp chọn trực tiếp) 15](#_Toc155620788)

[3.2.2.1 Ý tưởng thuật toán 15](#_Toc155620789)

[3.2.2.2 Cài đặt mã giả 15](#_Toc155620790)

[3.2.2.3 Đánh giá độ phức tạp 15](#_Toc155620791)

[3.2.3 Thuật toán Bubble Sort (Sắp xếp nổi bọt) 15](#_Toc155620792)

[3.2.3.1 Ý tưởng thuật toán 15](#_Toc155620793)

[3.2.3.2 Cài đặt mã giả 16](#_Toc155620794)

[3.2.3.3 Đánh giá độ phức tạp 16](#_Toc155620795)

[3.2.4 Thuật toán Insertion Sort (Sắp xếp chèn trực tiếp) 16](#_Toc155620796)

[3.2.4.1 Ý tưởng thuật toán 16](#_Toc155620797)

[3.2.4.2 Cài đặt mã giả 16](#_Toc155620798)

[3.2.4.3 Đánh giá độ phức tạp 17](#_Toc155620799)

[3.3 Giao diện phần mềm 17](#_Toc155620800)

[3.3.1 Khởi tạo dãy số 17](#_Toc155620801)

[3.3.2 Bảng điều khiển 18](#_Toc155620802)

[3.3.3 Bảng lựa chọn thuật toán 19](#_Toc155620803)

[3.3.4 Hiện ý tưởng thuật toán, dãy chưa sắp xếp và hiện code C++ 19](#_Toc155620804)

[3.3.5 Chế độ sắp xếp tăng giảm, debug, chế độ hủy 20](#_Toc155620805)

[3.3.6 Thuật toán Interchange Sort 21](#_Toc155620806)

[3.3.7 Thuật toán Selection Sort 23](#_Toc155620807)

[3.3.8 Thuật toán Insertion Sort 24](#_Toc155620808)

[3.3.9 Thuật toán Bubble Sort 26](#_Toc155620809)

[Chương 4. Kết luận 28](#_Toc155620810)

[4.1 Kết quả đạt được 28](#_Toc155620811)

[4.2 Hạn chế 28](#_Toc155620812)

[4.3 Hướng phát triển 28](#_Toc155620813)

[Tài liệu tham khảo 29](#_Toc155620814)

[PHỤ LỤC 30](#_Toc155620815)

# **DANH MỤC HÌNH VẼ**

[Hình 1 Bảng khởi tạo dãy số. 18](#_Toc155614333)

[Hình 2 Nhập một dãy. 18](#_Toc155614334)

[Hình 3 Nhập bằng tay. 19](#_Toc155614335)

[Hình 4 Bảng điều khiển. 19](#_Toc155614336)

[Hình 5 Bảng lựa chọn thuật toán. 19](#_Toc155614337)

[Hình 6 Ý tưởng thuật toán. 20](#_Toc155614338)

[Hình 7 Dãy chưa sắp xếp. 20](#_Toc155614339)

[Hình 8 Code C++. 20](#_Toc155614340)

[Hình 9 Hướng sắp xếp tăng giảm. 21](#_Toc155614341)

[Hình 10 Chế độ Debug. 21](#_Toc155614342)

[Hình 11 Chế độ hủy. 21](#_Toc155614343)

[Hình 12 Thanh điều chỉnh tốc độ. 22](#_Toc155614344)

[Hình 13 Interchange Sort trước khi thực hiện sắp xếp. 22](#_Toc155614345)

[Hình 14 Interchange Sort sau khi thực hiện sắp xếp. 23](#_Toc155614346)

[Hình 15 Selection Sort trước khi thực hiện sắp xếp. 24](#_Toc155614347)

[Hình 16 Selection Sort sau khi thực hiện sắp xếp. 24](#_Toc155614348)

[Hình 17 Insertion Sort trước khi thực hiện sắp xếp. 25](#_Toc155614349)

[Hình 18 Insertion Sort sau khi thực hiện sắp xếp. 26](#_Toc155614350)

[Hình 19 Bubble Sort trước khi thực hiện sắp xếp. 26](#_Toc155614351)

[Hình 20 Bubble Sort sau khi thực hiện sắp xếp. 27](#_Toc155614352)

# **DANH MỤC BẢNG**

[Bảng 3. 1 Bảng đánh giá độ phức tạp của Interchange Sort. 14](#_Toc155618130)

[Bảng 3. 2 Bảng đánh giá độ phức tạp của Selection Sort. 15](#_Toc155618131)

[Bảng 3. 3 Bảng đánh giá độ phức tạp Bubble Sort. 16](#_Toc155618132)

[Bảng 3. 4 Bảng đánh giá độ phức tạp Insertion Sort. 17](#_Toc155618133)

1. Giới thiệu đề tài
   1. Lý do chọn đề tài

Cấu trúc dữ liệu và thuật toán là môn học rất quan trọng trong lĩnh vực công nghệ thông tin. Chúng được coi là thành phần cơ bản của lập trình, với công thức "Chương trình = Cấu trúc dữ liệu + Thuật toán". Trên thực tế, cấu trúc dữ liệu là cách để lưu trữ và quản lý dữ liệu một cách hiệu quả trên máy tính. Thuật toán, hay còn gọi là giải thuật, là tập hợp các hướng dẫn được thiết kế để máy tính thực hiện theo một trình tự nhất định để đạt được kết quả mong muốn.

Để giúp sinh viên hiểu rõ hơn về cấu trúc dữ liệu và thuật toán, giảng viên thường sử dụng các công cụ mô phỏng để minh họa các thay đổi trong quá trình thực thi thuật toán và cấu trúc dữ liệu. Các hệ thống mô phỏng thuật toán đang được phát triển ngày càng nhiều, và hầu hết chúng phức tạp hơn so với các hệ thống thực tế. Có rất nhiều thuật toán sắp xếp khác nhau và chúng có phạm vi rất rộng. Do đó, việc xây dựng một báo cáo thực tập về "Mô phỏng thuật toán sắp xếp" là một nhiệm vụ phù hợp.

* 1. Mục tiêu và nhiệm vụ

• Nghiên cứu tổng quát về mô phỏng thuật toán.

• Ôn tập và tìm hiểu một số thuật toán sắp xếp

• Xây dựng chương trình mô phỏng để hình dung rõ hơn quá trình thực hiện và phân loại chúng.

• Áp dụng các nghiên cứu để tạo một bản trình diễn mô phỏng thuật toán sắp xếp trên màn hình bảng điều khiển và trong C#.

1. Cơ sở lý thuyết
   1. Ngôn ngữ C#
      1. Khái niệm

Ngôn ngữ C# (phát âm là "C Sharp") là ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft, làm điểm khởi đầu cho bản thiết kế .NET của họ.

* + 1. Ứng dụng của ngôn ngữ lập trình C#

• *Ứng dụng Windows*: Với sự trợ giúp của .Net framework, “C#” được sử dụng để phát triển các ứng dụng dựa trên Windows cho máy tính để bàn. Nhiều ứng dụng Windows phổ biến như công cụ Microsoft Office, Skype, Photoshop và Visual Studio được phát triển bằng ngôn ngữ này.

• *Thành phần và Điều khiển*: Thành phần và điều khiển là các thư viện có thể được sử dụng để tạo ra thứ gì đó dễ dàng phân phối và chia sẻ. Thư viện GPS là một ví dụ tuyệt vời về thư viện có thể được tạo bởi một lập trình viên và dễ dàng phân phối cho các lập trình viên khác để sử dụng trong ứng dụng của họ. Nó cũng được sử dụng để xây dựng các thành phần máy chủ và hơn thế nữa.

• *Ứng dụng web*: Với sự trợ giúp của .NET framework, C# có khả năng tạo nhiều ứng dụng web bằng ASP.NET. Đây là một ngôn ngữ phổ biến khác mà bất kỳ ai cũng có thể học ngay lập tức khi muốn chạy các ứng dụng web đúng cách trên máy chủ web. Các ứng dụng Windows chạy cả trên máy chủ và trong trình duyệt của máy khách, tùy thuộc vào cách viết mã. Nếu C# được sử dụng làm mã hóa trong phần phụ trợ, mã C# sẽ chạy trên máy chủ và HTML Frontend chạy trong trình duyệt của máy khách.

* 1. Thuật toán
     1. Khái niệm thuật toán

Thuật toán là một chuỗi hữu hạn các thao tác được sắp xếp theo một thứ tự nhất định sao cho sau khi thực hiện chuỗi thao tác này, từ đầu vào của bài toán ta nhận được đầu ra cần tìm.

* + 1. Đặc điểm thuật toán

Các thuật toán có một số thuộc tính chung, đó là:

• *Đầu vào*: Một thuật toán có các giá trị đầu vào từ một bộ xác định.

• *Đầu ra*: Từ mỗi bộ giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra. Các giá trị đầu ra là giải pháp cho vấn đề.

• *Tính tất định*: Các bước của thuật toán phải được xác định chính xác.

• *Tính chính xác*: Một thuật toán phải đưa ra các giá trị đầu ra chính xác cho từng bộ giá trị đầu vào.

• *Tính hữu hạn*: Một thuật toán phải tạo ra các giá trị đầu ra sau một số bước hữu hạn (có thể rất lớn) cho mỗi bộ đầu vào.

• *Tính tổng quát*: Thuật toán cần áp dụng được cho tất cả các bộ dữ liệu đầu vào của bài toán chứ không chỉ một bộ giá trị đầu vào cụ thể.

* + 1. Tổng hợp các thuật toán sắp xếp hiện nay

Trong lĩnh vực khoa học máy tính và toán học, thuật toán sắp xếp được sử dụng để sắp xếp các phần tử trong danh sách (hoặc mảng) theo thứ tự tăng dần hoặc giảm dần. Thông thường, chúng ta xem xét các trường hợp sắp xếp các số.

Quá trình sắp xếp là quá trình xử lý danh sách các phần tử (hoặc thông tin) để sắp xếp chúng theo một tiêu chuẩn nhất định, dựa trên nội dung thông tin được lưu trữ trong mỗi phần tử. Hiệu suất của mỗi thuật toán phụ thuộc vào thuộc tính của cấu trúc dữ liệu cụ thể mà nó áp dụng.

Có nhiều thuật toán sắp xếp khác nhau như Interchange Sort, Selection Sort, Bubble Sort, Insertion Sort và nhiều hơn nữa. Tuy nhiên, trong báo cáo này, tôi muốn trình bày các giải pháp sắp xếp cổ điển để mang đến một cái nhìn khoa học hơn về một vài thuật toán sắp xếp.

1. Chương trình ứng dụng thuật toán sắp xếp
   1. Sơ lược mô phỏng minh họa trên C#
      1. Các chức năng

Trong mô phỏng minh họa bằng C#, phần mềm của tôi có các chức năng sau:

+ Bao gồm 4 thuật toán sắp xếp điển hình trong cấu trúc dữ liệu và thuật toán.

+ Tạo ngẫu nhiên một dãy phần tử mới theo từng thuật toán sắp xếp phù hợp.

+ Xóa bảng: Khi xóa bảng, tất cả các mục trên màn hình chính sẽ không còn xuất hiện.

+ Hủy quá trình: Khi hủy quá trình sắp xếp, các mục trên màn hình chính sẽ trở về trạng thái chưa sắp xếp ban đầu.

+ Giới hạn số phần tử có thể thêm vào dãy số (tối đa 20 phần tử).

+ Hỗ trợ chức năng tạm dừng hoặc tiếp tục thực thi ứng dụng.

+ Điều chỉnh tốc độ thực hiện các thuật toán sắp xếp trong quá trình thực hiện.

+ Có cửa sổ hiển thị mã C++ của các thuật toán sắp xếp tương ứng để so sánh với ảnh mô phỏng.

+ Hiển thị ý tưởng thuật toán và hiển thị dãy số thay đổi khi sắp xếp.

+ Khởi tạo dãy số cần sắp xếp bằng cách click chuột vào phần khởi tạo rồi chọn chế độ.

+ Tạo ngẫu nhiên với số phần tử ngẫu nhiên.

+ Nhập thủ công: Nhập từng phần tử với mã số mục tương ứng khi gõ vào ô số mục.

+ Lựa chọn chế độ debug cho phép thực thi từng dòng lệnh theo mã C++ của form chính.

+ Bảng chưa sắp xếp: Hiển thị bảng trước khi sắp xếp.

+ Hiển thị quá trình sắp xếp và kết quả sau khi sắp xếp dãy số.

* + 1. Các kỹ thuật xử lý
       1. Phân tích hiệu ứng

Để mô phỏng một phần tử trong mảng và hiển thị trực tiếp trên giao diện người dùng (GUI), chúng ta có thể tạo một thành phần điều khiển người dùng gọi là "Node" để lưu trữ giá trị của phần tử mảng và hiển thị nó. Người dùng có thể dễ dàng thay đổi tốc độ của quá trình mô phỏng.

Khi các phần tử di chuyển, chúng ta có thể chọn một phần tử đại diện cho thuật toán theo từng bước tương ứng với cách sắp xếp các phần tử. Việc hiển thị quá trình sắp xếp giúp người dùng dễ dàng theo dõi và hiểu hơn về thuật toán.

Ngoài ra, các biến thuật toán cũng có thể được hiển thị trực quan trên giao diện để phù hợp với vị trí của phần tử và có các giá trị tương ứng với giá trị của các biến thuật toán.

* + - 1. Phân tích chức năng
* Khởi tạo mảng (3 cách):
* Tạo ngẫu nhiên: Tạo một mảng có giá trị và số lượng phần tử bất kỳ.
* Nhập tay: cho người dùng nhập giá trị của từng phần tử một. Người dùng có thể thay đổi giá trị của một phần tử bất kì theo ý muốn.
* Nhập một dãy: cho người dùng nhập một dãy số liên tiếp và tạo mảng các phần tử dựa vào dãy số vừa nhập. Người dùng có thể lưu lại dãy vừa nhập cho các lần nhập sau nếu muốn.
* Bảng điều khiển:
* Bắt đầu sắp xếp: Bắt đầu sắp xếp các phần tử dựa trên thuật toán đã chọn, mặc định bắt đầu với Interchange Sort.
* Tạm dừng: Tạm dừng quá trình sắp xếp.
* Thời gian thực hiện: Hiển thị tổng thời gian thực hiện để hoàn thành sắp xếp.
* Chế độ hủy :
* Xóa mảng : xóa mảng đã tạo
* Hủy quá trình : hủy quá trình sắp xếp và quay lại mảng ban đầu
* Các chức năng phụ:
* Thay đổi tốc độ của quá trình sắp xếp.
* Cập nhật thông tin về quá trình sắp xếp.
* Cho phép thuật toán chạy theo hai hướng: tăng và giảm.
  + 1. Các kỹ thuật sử dụng
       1. Xử lý bất đồng bộ với Thread và Task

Để có thể mô phỏng việc di chuyển của các phần tử khi sắp xếp ta cần sử dụng kỹ thuật lập trình bất đồng bộ. Để lập trình bất đồng bộ ta có thể sử dụng Thread hoặc Task hoặc sử dụng cả hai.

Để có thể nhìn thấy các phần tử di chuyển mượt mà khi sắp xếp ta cần dùng đến hàm Thread.Sleep(<thời gian>) để cho tiến trình dừng lại trong một khoản thời gian nhất định và giúp cho người dùng kịp nhận ra sự thay đổi vị trí của các phần tử.

Để chọn từng dòng code trong khung Code C/C++ thì ta cần tạo một tiến trình chạy song song với tiến trình chính để khi tiến trình thực hiện sẽ không ngăn cản (block) tiến trình chính tiếp nhận các sự kiện (event) xảy ra.

Để có thể có hai chuyển động (hai hàm di chuyển phần tử) cùng chạy một lúc thì ta cần tạo ra hai tiến trình chạy song song với tiến trình chính.

Khi sử dụng Thread ta có thể hủy Thread bất kì khi nào mình muốn nhưng Thread lại không hỗ trợ các thư việc giúp thông báo khi tiến trình đã hoàn tất (callback). Vì thế ta sử dụng Task, Task hỗ trợ đủ các hàm thông báo việc tiến trình kết thúc nhưng lại thiếu hàm hủy Task đang thực hiện, việc hủy Task phải thông qua một CancellationTokenSource và cần phải thực hiện nhiều việc mới có thể có thể hủy một Task đang thực hiện.

Vì các nhược điểm và ưu điểm trên nên ta kết hợp việc sử dụng Thread và Task lại để thực hiện quá trình sắp xếp. Ta cho tạo một Thread để chạy việc chọn từng dòng code, bên trong Thread đó ta tạo ra hai Task con đê thực hiện việc di chuyển các phần tử. Vì vậy ta vừa có thể báo việc hoàn tất di chuyển của các phần tử và vừa có thể hủy quá trình bất cứ khi nào.

Khi xử lý đa tiến trình ta cũng gặp phải một vấn đề khá quan trọng đó là việc các tiến trình không cho phép tiến trình khác thay đổi hay chỉnh sửa các control được tạo ra bên trong tiến trình của mình. Ví dụ: khi ta tạo ra tiến trình con để thay đổi đoạn Text trên một Button của tiến trình chính thì Compiler sẽ báo lỗi. Khi đó ta cần thông báo cho Compiler biết là sẽ bỏ qua và không kiểm ta lỗi này

Việc tạm dừng các tiến trình ta sẽ sử dụng đên ManualResetEvent đây là một lớp có thể dùng để tạm dừng (block) các tiến trình đang chạy và tiếp tục (resume) các tiến trình khi cần.

* + - 1. Xử lý việc thay đổi tốc độ sắp xếp

Do ở mỗi vòng lặp khi thay đổi tọa độ của phần tử ta đều lấy giá trị tốc độ hiện tại được lưu trong tham số nên khi thay đổi giá trị tốc độ hiện tại thì các vòng lặp trong tương lai sẽ sử dụng giá trị tốc độ mới. Điều này sẽ làm thay đổi tốc độ di chuyển các phần tử, dẫn đến thay đổi tốc độ sắp xếp.

* + - 1. Xử lý việc di chuyển các phần tử và các biến

Để xác định được tọa độ đặt và tọa độ di chuyển các phần tử và các biến ta cần dựa vào chiều dài, rộng của panel xác định được tọa độ gốc. Từ tọa độ gốc ta dựa vào vị trí phần tử trong mảng để tính ra tọa độ mà phần tử sẽ được đặt trên panel và cũng dùng cách tương tự để xác định tọa độ đích khi cần di chuyển phần tử. Đối với các biến có trong các thuật toán sắp xếp ta cũng làm tương tự như cách trên.

Ngoài ra các biến và các phần tử cũng được đổi màu để người sử dụng dễ dàng phân biệt.

* + - 1. Xử lý việc tính thời gian thực hiện quá trình sắp xếp

Thời gian của quá trình sắp xếp được tính bằng việc sử dụng Timer. Thời gian được tính từ lúc quá trình sắp xếp bắt đầu đến lúc quá trình kết thúc.

Nếu chạy trong chế độ Debug thì khoảng thời gian này sẽ không được tính.

* + - 1. Xử lý debug

Để chạy chế độ Debug ta đặt một ManualResetEvent vào hàm chuyển qua từng dòng code. Mỗi khi thực hiện xong một lệnh thì quá trình sẽ tự động được dừng lại. Khi đó chương trình đợi người dùng nhấn vào nút Debug thì sẽ ra lệnh cho trương trình chạy lệnh tiếp theo. Cứ tiếp tục lặp lại các bước trên ta sẽ được chế độ chạy Debug.

* 1. Các thuật toán sắp xếp đơn giản
     1. Thuật toán Interchange Sort (Đổi chỗ trực tiếp)
        1. Ý tưởng thuật toán

Xuất phát từ đầu dãy,tìm tất cả các cặp nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi phần tử này với phần tử tương ứng trong cặp nghịch thế .Lặp lại xử lý trên với các phần tử tiếp theo.

* + - 1. Cài đặt mã giả
* **Bước 1**: i = 0;//Bắt đầu từ đầu dãy
* **Bước 2**: j=i+1;//Tìm a[j] < a[i] với j > i.
* **Bước 3**:

Khi j < n thì kiểm tra: nếu a[j] < a[i] thì hoán vị a[j] và a[i].

Gán j=j+1; rồi thực hiện lại bước 3.

* **Bước 4**: i=i+1; nếu i < n-1 thì lặp lại bước 2, ngược lại -> Dừng.
  + - 1. Đánh giá độ phức tạp

|  |  |
| --- | --- |
| Trường hợp | Độ phức tạp |
| Tốt nhất | O(n) |
| Xấu nhất | O(n2) |

* + - * 1. Bảng đánh giá độ phức tạp của Interchange Sort.
    1. Thuật toán Selection Sort (Sắp xếp chọn trực tiếp)
       1. Ý tưởng thuật toán

Chọn phần tử nhỏ nhất hoặc lớn nhất trong N phần tử trong dãy hiện hành. Đưa phần tử này về vị trí đầu dãy hiện hành. Xem dãy hiện hành chỉ còn N-1 phần tử của dãy hiện hành ban đầu. Bắt đầu từ vị trí thứ 2. Lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử.

* + - 1. Cài đặt mã giả
* **Bước 1:** i = 0;
* **Bước 2:** Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[n-1].
* **Bước 3:** Đổi chỗ a[min] và a[i].
* **Bước 4:** Nếu i < n-1 thì gán i = i+1; rồi lặp lại bước 2, ngược lại -> Dừng.
  + - 1. Đánh giá độ phức tạp

|  |  |
| --- | --- |
| Trường hợp | Độ phức tạp |
| Tốt nhất | O(n2) |
| Xấu nhất | O(n2) |

* + - * 1. Bảng đánh giá độ phức tạp của Selection Sort.
    1. Thuật toán Bubble Sort (Sắp xếp nổi bọt)
       1. Ý tưởng thuật toán

Xuất phát từ cuối dãy,đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn hoặc lớn hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo,do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

* + - 1. Cài đặt mã giả
* **Bước 1**: i = 0;//lần xử lý đầu tiên
* **Bước 2**: j = n-1;//duyệt từ cuối dãy ngược về vị trí i
* Trong khi (j > i) xét trường hợp: Nếu a[j]<a[j-1] thì đổi chổ a[j] và a[j-1].
* j = j-1;
* **Bước 3**: i = i+1; //lần xử lý kế tiếp
* Nếu i = n-1 -> hết dãy -> Dừng.
* Ngược lại -> Lặp lại Bước 2.
  + - 1. Đánh giá độ phức tạp

|  |  |
| --- | --- |
| Trường hợp | Độ phức tạp |
| Tốt nhất | O(n) |
| Xấu nhất | O(n2) |

* + - * 1. Bảng đánh giá độ phức tạp Bubble Sort.
    1. Thuật toán Insertion Sort (Sắp xếp chèn trực tiếp)
       1. Ý tưởng thuật toán

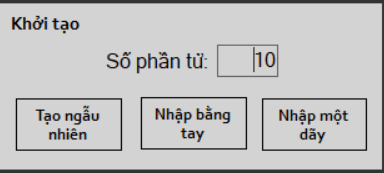
Giả sử có một dãy a(0),a(1),...,a(n-1) trong đó i phần tử đầu tiên a(0),a(1),...,a(i-1) đã có thứ tự. Tìm cách chèn phần tử a(i) vào vị trí thích hợp của đoạn đã được sắp để có dãy mới a(0),a(1),...,a(i) trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử a(k-1) và a(k) thỏa a(k-1)<a(i)<a(k) (1<=k<=i).

* + - 1. Cài đặt mã giả
* **Bước 1**: i = 1;//giả sử có đoạn a[0] đã được sắp xếp
* **Bước 2**: x = a[i];
* **Bước 3**:
* Tìm vị trí pos thích hợp trong đoạn a[0] đến a[i-1] để chèn a[i] vào danh sách.
* Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i].
* **Bước 4**: a[pos] = x;//chèn x, có đoạn a[0],…,a[i] đã được sắp.
* **Bước 5**: i = i+1; nếu i < n -> lặp lại bước 2, ngược lại -> Dừng.
  + - 1. Đánh giá độ phức tạp

|  |  |
| --- | --- |
| Trường hợp | Độ phức tạp |
| Tốt nhất | O(n) |
| Xấu nhất | O(n2) |

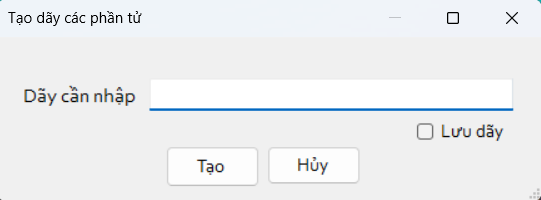
* + - * 1. Bảng đánh giá độ phức tạp Insertion Sort.
  1. Giao diện phần mềm
     1. Khởi tạo dãy số

Người dùng có thể khởi tạo input đầu vào bằng ba cách: Tạo ngẫu nhiên, nhập bằng tay, nhập một dãy. Chức năng tạo ngẫu nhiên một dãy số với số phần tử tương ứng được nhập ở ô số phần tử.



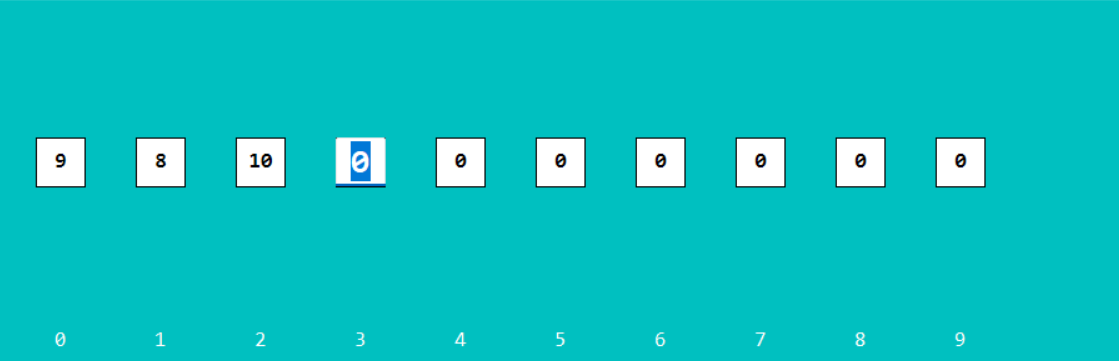
1. Bảng khởi tạo dãy số.

Chức năng nhập một dãy số bằng tay trên 1 Form mới. Có thể lưu dãy số phần tử để lần sau có thể sử dụng lại dãy số phần tử đã lưu trước đó.



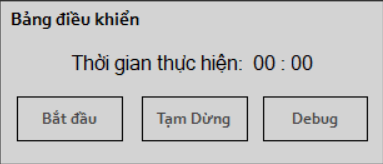
1. Nhập một dãy.

Khi nhấn vào nhập bằng tay với số phần tử tương ứng được nhập ở ô số phần tử. Các phần tử sẽ xuất hiện trên Panel quá trình sắp xếp. Ta dễ dàng nhập lần lượt từng phần tử thông qua phím Enter hoặc Tap đối với mỗi phần tử. Ta cũng có thể thay đổi giá trị bằng cách nhấn trực tiếp vào phần tử đó.



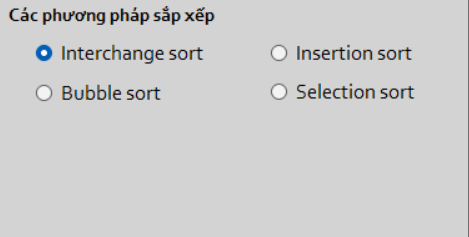
1. Nhập bằng tay.
   * 1. Bảng điều khiển

Người dùng có thể bắt đầu, tạm dừng, nút tiếp tục sẽ hiện lên khi bạn ấn vào nút tạm dừng, chế độ debug và cuối cùng hiện thời gian thực hiện sắp xếp.



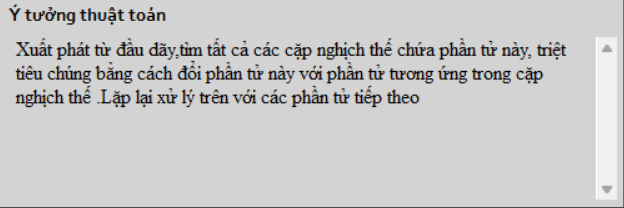
1. Bảng điều khiển.
   * 1. Bảng lựa chọn thuật toán

Người dùng có thể lựa chọn thuật toán để sắp xếp.



1. Bảng lựa chọn thuật toán.
   * 1. Hiện ý tưởng thuật toán, dãy chưa sắp xếp và hiện code C++

Ý tưởng thuật toán sẽ hiện lên khi bạn ấn vào 1 thuật toán bất kì.



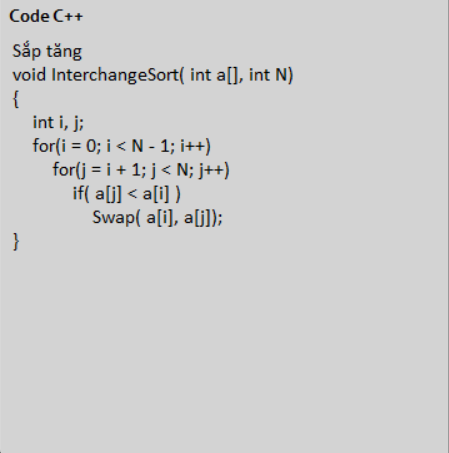
1. Ý tưởng thuật toán.

Dãy chưa sắp xếp hiện lên dãy để người dùng có thể so sánh với dãy đã sắp xếp với thuật toán bất kì ở phía trên.



1. Dãy chưa sắp xếp.

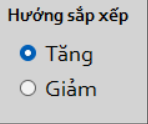
Code C++ hiển thị code của thuật toán sắp xếp mà người dùng chọn tương ứng.



1. Code C++.

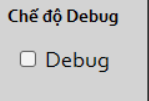
3.3.5 Chế độ sắp xếp tăng giảm, debug, chế độ hủy

Lựa chọn hướng sắp xếp tăng hoặc giảm để sắp xếp theo đúng yêu cầu của người dùng.



1. Hướng sắp xếp tăng giảm.

Tùy chỉnh lựa chọn chế độ debug hoặc không debug.

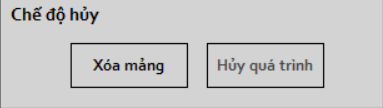


1. Chế độ Debug.

* Cho phép thực hiện chế độ hủy trong quá trình sắp xếp:

+ Xóa mảng: xóa tất cả các phần tử trên giao diện kể cả khi đang sắp xếp.

+ Hủy quá trình: Hủy tất cả các phần tử đang sắp xếp và trở lại dãy số ban đầu chưa sắp xếp.



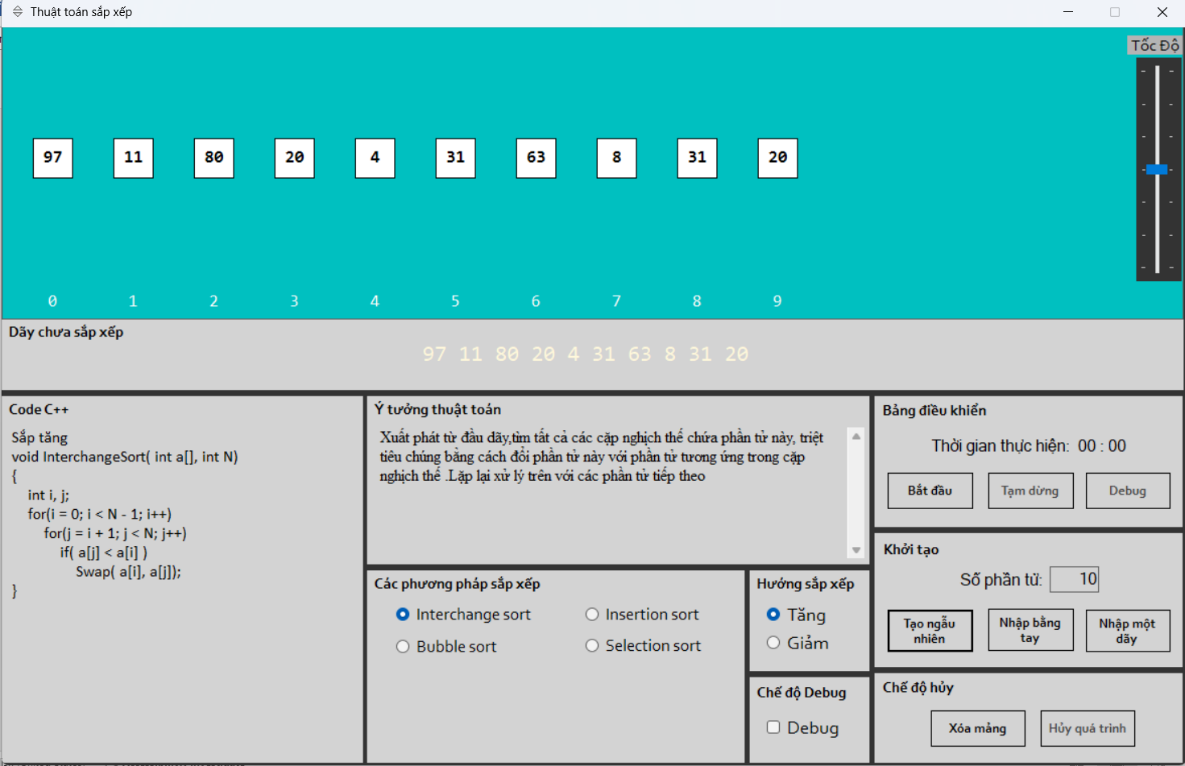
1. Chế độ hủy.

Chức năng điều chình tốc độ của quá trình sắp xếp: khi thanh tốc độ càng lớn thời gian sắp xếp sẽ nhanh hơn.

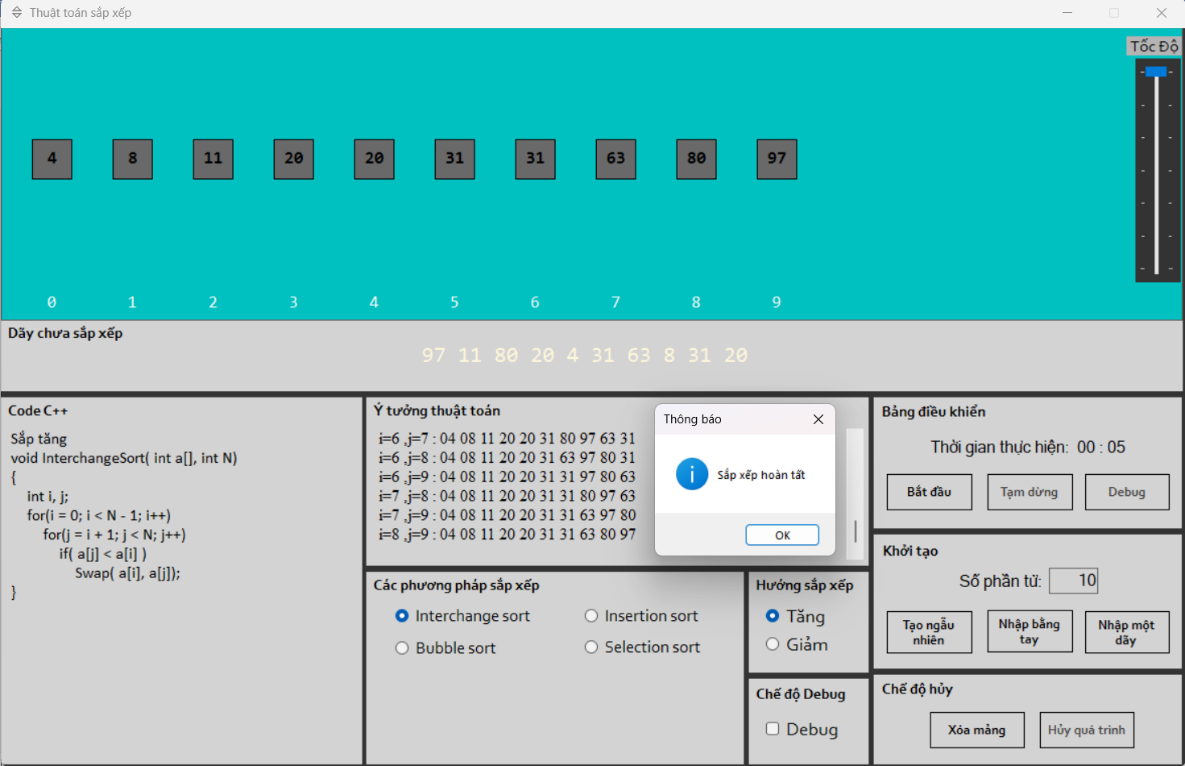


1. Thanh điều chỉnh tốc độ.
   * 1. Thuật toán Interchange Sort

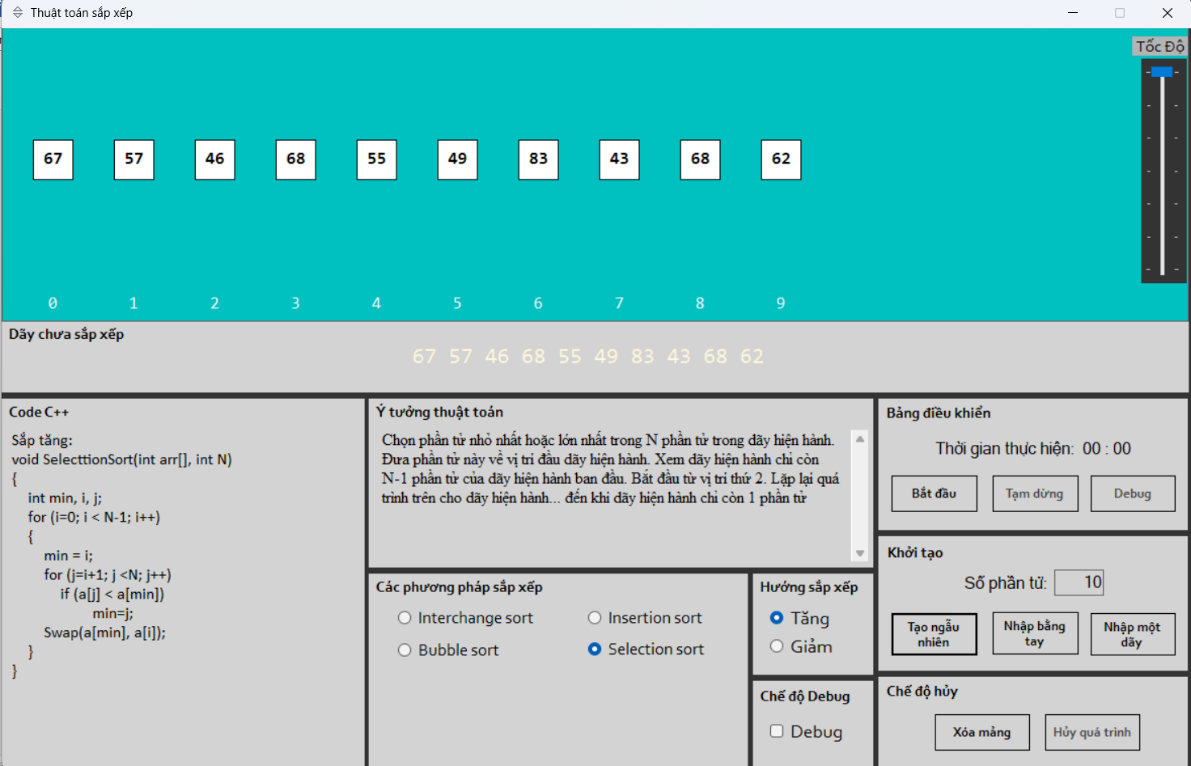
Người dùng chọn thuật toán Interchange Sort và có thể chọn kiểu sắp xếp tăng dần hoặc giảm dần, trong quá trình thực hiện người dùng có thể điều chỉnh tốc độ sắp xếp qua thanh điều khiển và có thể chọn debug chương trình sẽ chạy từng bước của thuật toán để người dùng có thể dễ dàng nắm bắt quá trình.



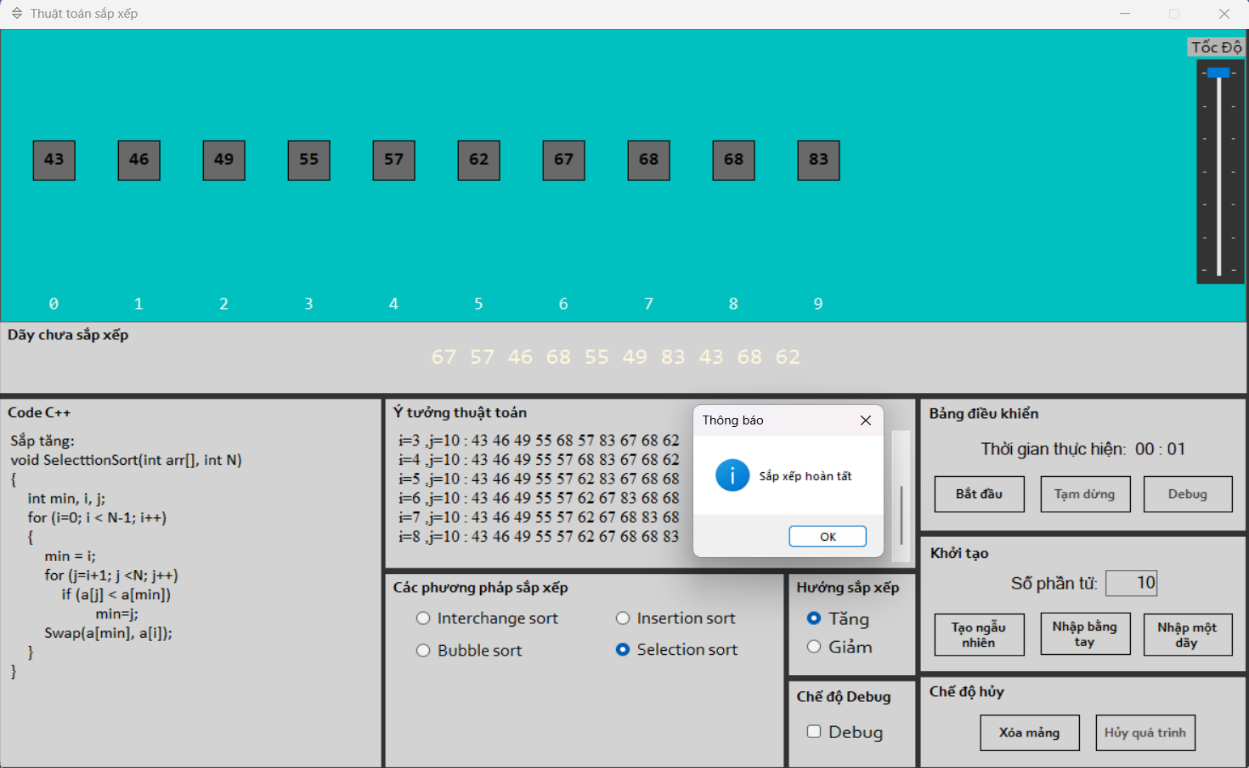
1. Interchange Sort trước khi thực hiện sắp xếp.

Đây là mảng sau khi đã sắp xếp xong chương trình, nó sẽ xuất hiện một thông báo mảng đã sắp xếp xong, và hiển thị kết quả sắp xếp thuật toán Interchange Sort. 

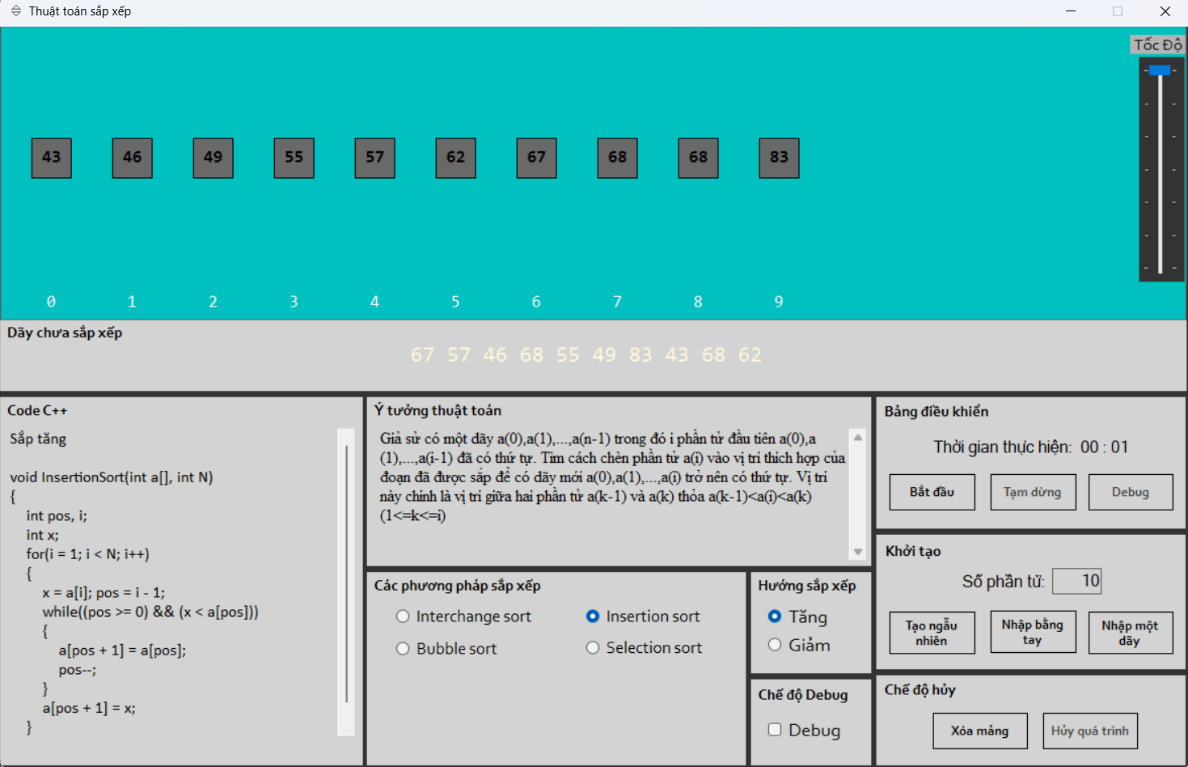
1. Interchange Sort sau khi thực hiện sắp xếp.
   * 1. Thuật toán Selection Sort

Người dùng chọn thuật toán Selection Sort và có thể chọn kiểu sắp xếp tăng dần hoặc giảm dần, trong quá trình thực hiện người dùng có thể điều chỉnh tốc độ sắp xếp qua thanh điều khiển và có thể chọn debug chương trình sẽ chạy từng bước của thuật toán để người dùng có thể dễ dàng nắm bắt quá trình. 

1. Selection Sort trước khi thực hiện sắp xếp.

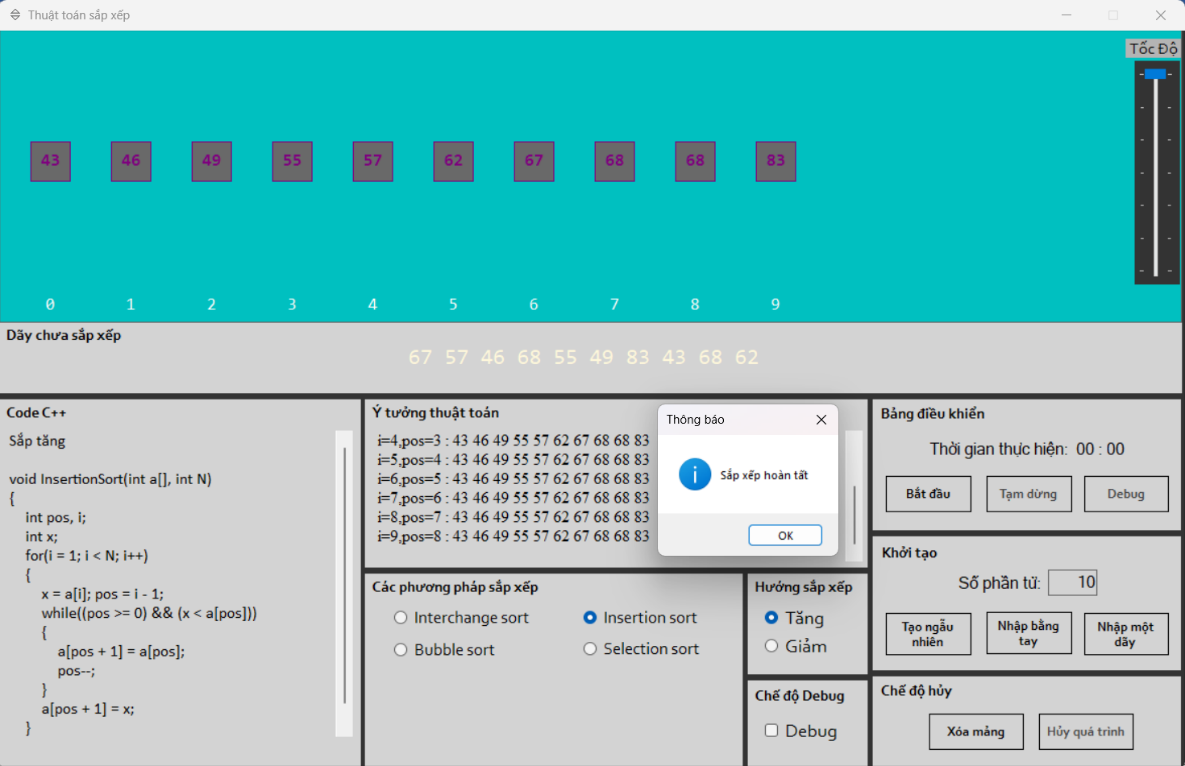
Đây là mảng sau khi đã sắp xếp xong chương trình, nó sẽ xuất hiện một thông báo mảng đã sắp xếp xong, và hiển thị kết quả sắp xếp thuật toán Selection Sort. 

1. Selection Sort sau khi thực hiện sắp xếp.
   * 1. Thuật toán Insertion Sort

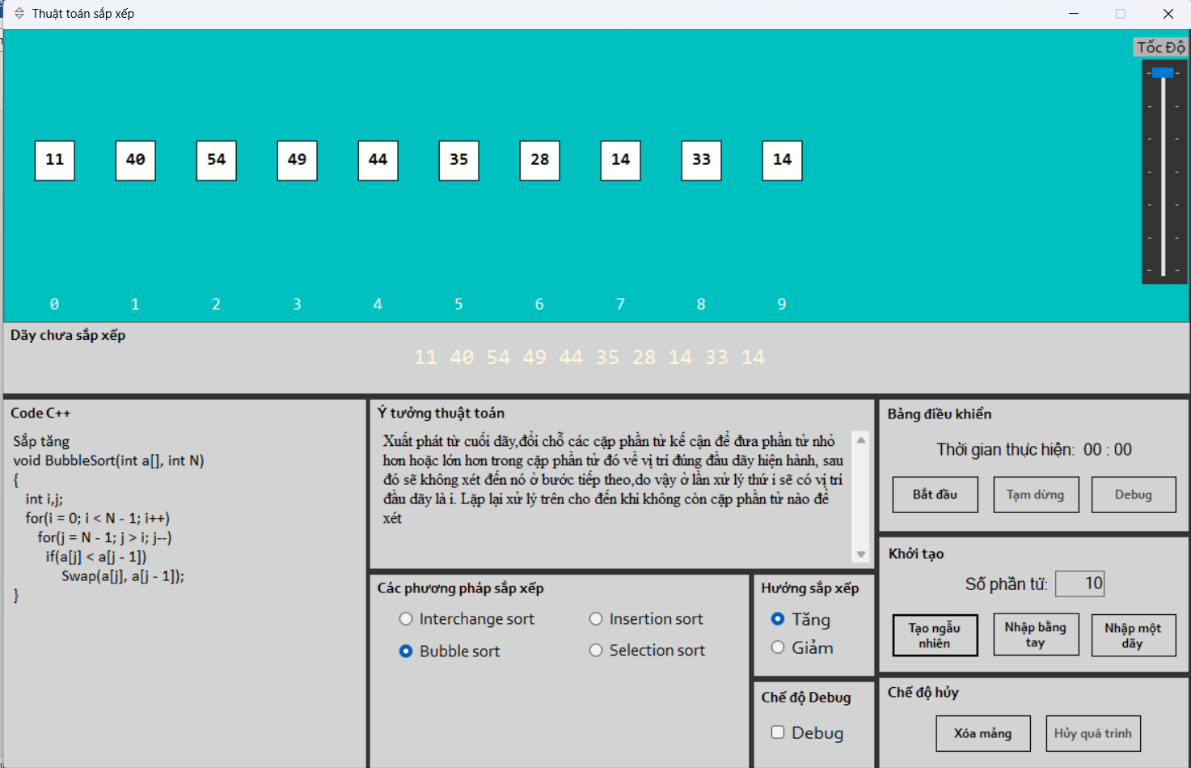
Người dùng chọn thuật toán Insertion Sort và có thể chọn kiểu sắp xếp tăng dần hoặc giảm dần, trong quá trình thực hiện người dùng có thể điều chỉnh tốc độ sắp xếp qua thanh điều khiển và có thể chọn debug chương trình sẽ chạy từng bước của thuật toán để người dùng có thể dễ dàng nắm bắt quá trình.

1. Insertion Sort trước khi thực hiện sắp xếp.

Đây là mảng sau khi đã sắp xếp xong chương trình, nó sẽ xuất hiện một thông báo mảng đã sắp xếp xong, và hiển thị kết quả sắp xếp thuật toán Insertion Sort.

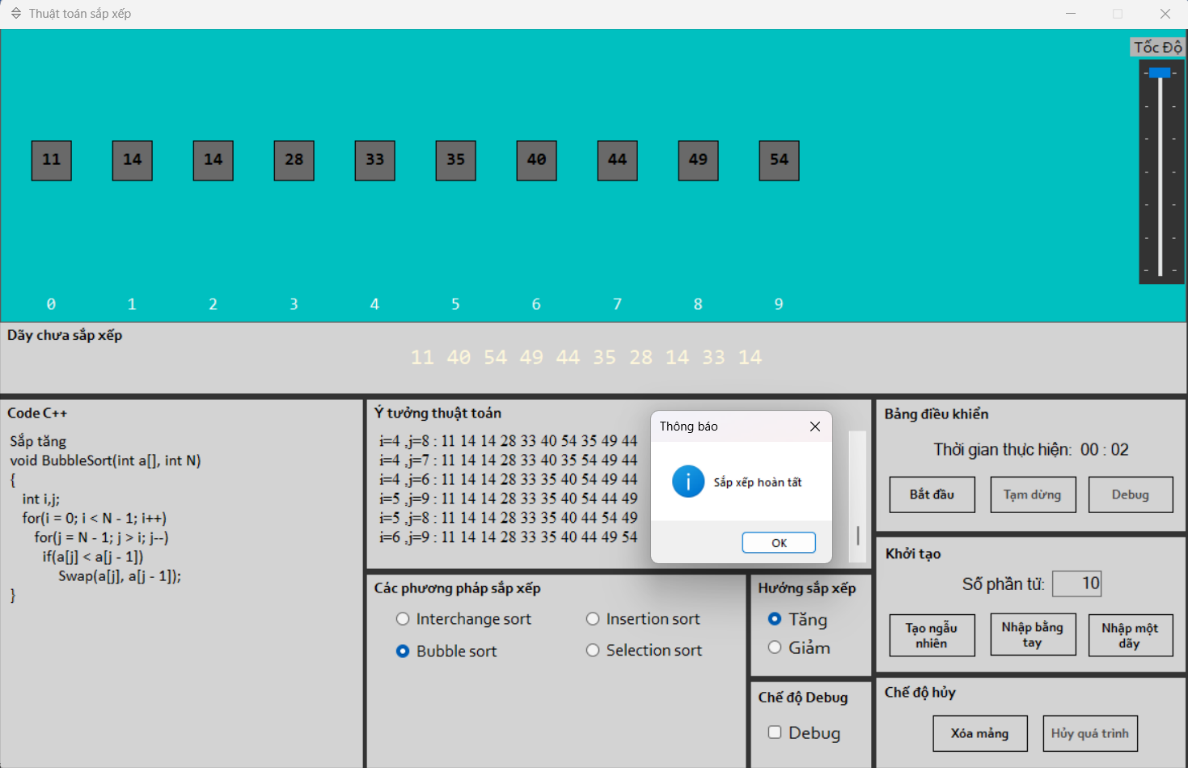


1. Insertion Sort sau khi thực hiện sắp xếp.
   * 1. Thuật toán Bubble Sort

Người dùng chọn thuật toán Bubble Sort và có thể chọn kiểu sắp xếp tăng dần hoặc giảm dần, trong quá trình thực hiện người dùng có thể điều chỉnh tốc độ sắp xếp qua thanh điều khiển và có thể chọn debug chương trình sẽ chạy từng bước của thuật toán để người dùng có thể dễ dàng nắm bắt quá trình.

1. Bubble Sort trước khi thực hiện sắp xếp.

Đây là mảng sau khi đã sắp xếp xong chương trình, nó sẽ xuất hiện một thông báo mảng đã sắp xếp xong, và hiển thị kết quả sắp xếp thuật toán Bubble Sort.



1. Bubble Sort sau khi thực hiện sắp xếp.
2. Kết luận
   1. Kết quả đạt được

* Hoàn thành chương trình mô phỏng thuật toán sắp xếp như mục tiêu ban đầu có thể hiện node giữa màn hình sửa node trực tiếp trên màn hình chờ, mô phỏng thành công các thuật toán sắp xếp cùng khung ý tưởng và các tham số hiện thị trên màng hình chờ.
* Bước đầu nắm bắt được những kỹ thuật lập trình của C#, winform như timer, multithread, backgroundworker, đa ngôn ngữ sử dụng resources, tự tạo 1 user controls.
  1. Hạn chế
* Giao diện chương trình chưa được đẹp mắt và thân thiện với người dùng.
* Xử lý code còn rườm rà.
* Số lượng phần tử hiển thị hạn chế.
* Gặp một số lỗi khó khắc phục trong quá trình xử lý đa luồng multithread.
* Cung cấp ít thuật toán.
  1. Hướng phát triển
* Cải thiện giao diện đẹp, tối ưu hơn cho người dùng sử dụng.
* Thêm nhiều chức năng như so sánh độ phức tạp của thuật toán (có animation) từ hai thuật toán trở lên…
* Đồng bộ tốc độ xử lý của bộ đếm thời gian và thread.
* Thêm nhiều tùy chọn cho người dùng trong phần cài đặt.
* Thêm nhiều chức năng mới cho phần mềm.

# **Tài liệu tham khảo**

Nguyễn Đình Hoàng Sơn, *Slỉdes Bài giảng Cấu trúc dữ liệu và giải thuật,* đại học Nha Trang.

Phạm Thị Kim Ngoan (2023), *Slides Bài giảng Lập trình hướng đối tượng,* đại học Nha Trang.

Thread :

<https://msdn.microsoft.com/en-us/library/aa645740(v=vs.71).aspx>

<https://yinyangit.wordpress.com/2011/04/11/c-thread-basic/>

<https://www.dotnetperls.com/thread>

Timer : <https://msdn.microsoft.com/en-us/library/system.timers.timer(v=vs.110).aspx>

Task :

<https://msdn.microsoft.com/en-us/library/system.threading.tasks.task(v=vs.110).aspx>

Manual Reset Event:

<https://yinyangit.wordpress.com/2012/12/04/c-thread-signaling-auto-va-manual-reset-event/>

Có xem tham khảo thêm ở stackoverflow.com và các forum khác.

# **PHỤ LỤC**

**Phần tử và các hàm có trong phần tử**

- Để tạo một phần tử ta dùng một Button và thêm vào đó các thuộc tính mà phần tử cần: vị trí hiện tại, giá trị,… Ta đặt nên cho một phần tử là một Node . Ngoài ra mỗi Node còn chứa một TextBox bên trong nó có nhiệm vụ để cho người dùng có thể nhập giá trị cho Node mỗi khi nhấn vào (nhập tay).

|  |
| --- |
| class Node : Button  {  public int giaTri;  public int vitriHienTai;  public TextBox nhapTayTexbox;  public Node(int vitrihientai, int giatri)  {  // Node : property + event  this.FlatStyle = FlatStyle.Flat;  this.BackColor = ThamSo.MauNenNode;  this.TextAlign = ContentAlignment.MiddleCenter;  this.Size = new Size(ThamSo.KichCoNode, ThamSo.KichCoNode);  this.Padding = new Padding(0);  this.Font = new Font("Consolas", ThamSo.KichCoNode / 3.2f, FontStyle.Bold);  this.UseCompatibleTextRendering = true;    this.Text = giatri.ToString();  giaTri = giatri;  vitriHienTai = vitrihientai;  this.GotFocus += new EventHandler(Node\_GotFocus);  // NhapTay TextBox : property + event  nhapTayTexbox = new TextBox();  nhapTayTexbox.MaxLength = 2;  nhapTayTexbox.TextAlign = HorizontalAlignment.Center;  nhapTayTexbox.BorderStyle = BorderStyle.Fixed3D;  nhapTayTexbox.Visible = false;  nhapTayTexbox.Size = new Size(ThamSo.KichCoNode, ThamSo.KichCoNode);  nhapTayTexbox.Font = new Font("Consolas", ThamSo.KichCoNode/2, FontStyle.Bold);  this.Controls.Add(nhapTayTexbox);  nhapTayTexbox.KeyPress += new KeyPressEventHandler(nhapTayTexbox\_KeyPress);  nhapTayTexbox.KeyDown += new KeyEventHandler(nhapTayTexbox\_KeyDown);  nhapTayTexbox.TextChanged += new EventHandler(nhapTayTexbox\_TextChanged);  nhapTayTexbox.LostFocus += new EventHandler(nhapTayTexbox\_LostFocus);  }  } |

- Để xử lý việc nhập giá trị bằng tay cho Node ta cần một số hàm chính sau:

Hàm Node\_GotFocus : mỗi khi Node được focus vào thì textbox nhập tay hiện lên cho người dùng nhập.

|  |
| --- |
| private void Node\_GotFocus(object sender, EventArgs e)  {  if (nhapTayTexbox.Enabled == true) // Nếu textbox bị tắt (khi node đang sắp xếp) thì texbox không đc bật lên để sửa  {  nhapTayTexbox.BackColor = this.BackColor;  nhapTayTexbox.Visible = true;  nhapTayTexbox.Text = this.Text;  nhapTayTexbox.SelectAll();  nhapTayTexbox.Focus();  }  } |

Hàm nhapTayTexbox\_LostFocus : khi textbox nhập tay bị lost focus thì tắt textbox đi và lưu giá trị của textbox vào trong Node. Ngoài ra còn chạy một delegate để thông báo giá trị Node vừa được thay đổi.

|  |
| --- |
| public static Action NodeValueChangedHandler;  private void nhapTayTexbox\_LostFocus(object sender, EventArgs e)  {  nhapTayTexbox.Visible = false;  this.Text = nhapTayTexbox.Text;  this.giaTri = int.Parse(nhapTayTexbox.Text);  NodeValueChangedHandler();  } |

Hàm nhapTayTexbox\_TextChanged : không cho người dùng nhập giá trị rỗng.

|  |
| --- |
| private void nhapTayTexbox\_TextChanged(object sender, EventArgs e)  {  if (nhapTayTexbox.Text.Count() == 0)  {  nhapTayTexbox.Text = "0";  nhapTayTexbox.SelectAll();  nhapTayTexbox.Focus();  }  } |

Hàm nhapTayTexbox\_KeyDown : nếu nhấn Enter thì tắt textbox nhập tay và lưu giá trị vào Node.

|  |
| --- |
| private void nhapTayTexbox\_KeyDown(object sender, KeyEventArgs e)  {  if (e.KeyCode == Keys.Enter)  {  nhapTayTexbox.Visible = false;  this.Text = nhapTayTexbox.Text;  }  } |

Hàm nhapTayTexbox\_KeyPress : cho cho phép nhập các phím là số khi nhập giá trị vào textbox.

|  |
| --- |
| private void nhapTayTexbox\_KeyPress(object sender, KeyPressEventArgs e)  {  if (e.KeyChar != '\b') // nếu là nút BackSpace thì bỏ qua bước check này --> cho phép nút Backspace hoạt động  {  e.Handled = !char.IsNumber(e.KeyChar);  // Nếu : Handled == true thì event bị hủy  // Nhưng : [isNumber(True) + not] = false --> Handled = false --> cho phép nhập --> nếu là số thì cho phép nhập  if (nhapTayTexbox.Text.Count() == 0)  {  nhapTayTexbox.Text = "0";  }  }  } |

- Để xử lý di chuyển các Node lên, xuống, qua trái, qua phải ta cần các hàm sau:

Hàm ChuyenXuong : chuyển Node đi xuống. Dựa vào khoản cách đã trên dưới và tọa độ của Node hiện tại ta sẽ tính ra tọa độ Node sẽ đến. Mỗi lần Node di chuyển ta tăng lên n đơn vị khoản cách. Mỗi lần Node di chuyển thread cũng sẽ được dừng trong m mili giây để tạo ra hiệu ứng chuyển động. Ngoài ra hàm còn giữ

|  |
| --- |
| public void ChuyenXuong()  {  int y\_ViTriMoi = this.Location.Y + ThamSo.KhoanCachTrenDuoiNode;  while (this.Location.Y < y\_ViTriMoi)  {  pauseStatus.WaitOne(Timeout.Infinite);  Thread.Sleep(ThamSo.TocDo);  this.Location = new Point(this.Location.X, this.Location.Y + 2);  }  } |

Hàm ChuyenLen : chuyển Node đi lên

|  |
| --- |
| public void ChuyenLen()  {  int y\_ViTriMoi = this.Location.Y - ThamSo.KhoanCachTrenDuoiNode;  while (this.Location.Y > y\_ViTriMoi)  {  pauseStatus.WaitOne(Timeout.Infinite);  Thread.Sleep(ThamSo.TocDo);  this.Location = new Point(this.Location.X, this.Location.Y - 2);  }  } |

Hàm ChuyenQua : chuyển Node đi qua trái hoặc qua phải tùy theo vị trí hiện tại của Node và vị trí mới mà Node chuyển đến

|  |
| --- |
| public void ChuyenQua(int vitriMoi)  {  int x\_vitririMoi;  if (vitriMoi > this.vitriHienTai)  {  x\_vitririMoi = this.Location.X + ((vitriMoi - vitriHienTai) \* ThamSo.KhoanCachGiuaCacNode);  while (this.Location.X < x\_vitririMoi)  {  pauseStatus.WaitOne(Timeout.Infinite);  Thread.Sleep(ThamSo.TocDo);  this.Location = new Point(this.Location.X + 2, this.Location.Y);  }  }  else  {  x\_vitririMoi = this.Location.X - ((vitriHienTai - vitriMoi) \* ThamSo.KhoanCachGiuaCacNode);  while (this.Location.X > x\_vitririMoi)  {  pauseStatus.WaitOne(Timeout.Infinite);  Thread.Sleep(ThamSo.TocDo);  //Task.Delay(ThamSo.TocDoMax).Wait();  this.Location = new Point(this.Location.X - 2, this.Location.Y);  }  }  } |

**Các hàm tạo mảng, xóa mảng và các hàm bổ trợ**

- Để tạo một mảng các phần tử ta cần tạo ra một mảng các Node và lưu các Node lại bên trong một danh sách liên kết đơn để thuận tiện cho việc điều khiển

Một số các thuộc tính cần để tạo một mảng:

root : vị trí tọa độ gốc để đặt Node đầu tiên

isRunning : cho biết quá trình sắp xếp có đang chạy hay không

nodeArr : chứa mảng các Node

labelSTTArr: chứa dãy số thứ tự bên dưới mảng

bienArr: chứa label các biến có trong thuật toán

daySoChuaSapXepLabel : label dãy số ban đầu

|  |
| --- |
| private Point root;  public bool isRunning = false;  private List<Node> nodeArr = new List<Node>();  private List<Label> labelSTTArr = new List<Label>(); // dãy số thứ tự  private Dictionary<string, Label> bienArr;  private Label daySoChuaSapXepLabel; |

- Các hàm hỗ trợ cho hàm tạo mảng:

Hàm ChinhKichCoNode : điều chỉnh kích cỡ Node sao cho phù hợp với số lượng phần tử

|  |
| --- |
| private void ChinhKichCoNode()  {  int soPhanTu = int.Parse(soPhanTuTextBox.Text);  //Điều chỉnh khích cỡ node phù hợp với số phần tử  if (soPhanTu > 14 && soPhanTu <= ThamSo.SoLuongPhanTuMax)  {  ThamSo.KichCoNode = 30 + (18 - soPhanTu) \* 2;  ThamSo.KhoanCachGiuaCacNode = ThamSo.KichCoNode \* 2;  }  else if ((Properties.Settings.Default.KichCoNode > 40 || Properties.Settings.Default.KhoanCachGiuaCacNode > 80) && soPhanTu > 11)  {  ThamSo.KichCoNode = 45;  ThamSo.KhoanCachGiuaCacNode = 85;  }  else if (soPhanTu <= 14)  {  ThamSo.KichCoNode = Properties.Settings.Default.KichCoNode;  ThamSo.KhoanCachGiuaCacNode = Properties.Settings.Default.KhoanCachGiuaCacNode;  }  } |

Hàm TaoLabelSoThuTuChoMang: điều chỉnh kích cỡ Node sao cho phù hợp với số lượng phần tử

|  |
| --- |
| private void TaoLabelSoThuTuChoMang(int i)  {  Label temp = new Label();  temp.Location = new Point(root.X + i \* ThamSo.KhoanCachGiuaCacNode, sortingPanel.Size.Height - 30);  temp.Font = new Font("Consolas", ThamSo.KichCoNode / 3.3f, FontStyle.Regular);  temp.BackColor = Color.Transparent;  temp.ForeColor = Color.White;  temp.Size = new Size(ThamSo.KichCoNode, ThamSo.KichCoNode / 2);  temp.TextAlign = ContentAlignment.MiddleCenter;  temp.Text = i.ToString();  this.sortingPanel.Controls.Add(temp);  labelSTTArr.Add(temp);  } |

- Các hàm tạo mảng:

Hàm TaoMangNgauNhien : tạo một mảng với giá trị của các phần tử là ngẫu nhiên từ 0 đến 99

|  |
| --- |
| private void TaoMangNgauNhien(int soPhanTu)  {  XoaMang();  ChinhKichCoNode();  Random rd = new Random();  for (int i = 0; i < soPhanTu; i++)  {  int giaTri = rd.Next(0, 99);  Node temp = new Node(i, giaTri);  this.sortingPanel.Controls.Add(temp);  nodeArr.Add(temp);  temp.Location = new Point(root.X + i \* ThamSo.KhoanCachGiuaCacNode, root.Y - ThamSo.KichCoNode / 2);  TaoLabelSoThuTuChoMang(i);  }  CapNhatDaySoChuaSapXep();  } |

Hàm NhapTay : cho nhập tay giá trị của từng phần tử trong mảng. Hàm này có sử dụng đến thuộc tính nhập tay của Node

|  |
| --- |
| private void NhapTay(int soPhanTu)  {  XoaMang();  ChinhKichCoNode();  for (int i = 0; i < soPhanTu; i++)  {  int giaTri = 0;  Node temp = new Node(i, giaTri);  this.sortingPanel.Controls.Add(temp);  nodeArr.Add(temp);  temp.Location = new Point(root.X + i \* ThamSo.KhoanCachGiuaCacNode, root.Y - ThamSo.KichCoNode / 2);  TaoLabelSoThuTuChoMang(i);  }  nodeArr[0].Focus();  } |

Hàm NhapMotDay : nhập một dãy số liên tiếp các giá trị của các phần tử trong mảng. Hàm này cần sự hỗ trợ của một form giúp nhận dãy số

|  |
| --- |
| private void NhapMotDay(List<int> dayCanNhap)  {  XoaMang();  ChinhKichCoNode();  for (int i = 0; i < dayCanNhap.Count; i++)  {  int giaTri = dayCanNhap[i];  Node temp = new Node(i, giaTri);  this.sortingPanel.Controls.Add(temp);  nodeArr.Add(temp);  temp.Location = new Point(root.X + i \* ThamSo.KhoanCachGiuaCacNode, root.Y - ThamSo.KichCoNode / 2);  TaoLabelSoThuTuChoMang(i);  }  CapNhatDaySoChuaSapXep();    } |

- Hàm xóa mảng:

Hàm XoaMang : xóa các Node hiện có, xóa các label số thứ tự, tắt chế độ tạm dừng nếu đang tạm dừng, hủy các quá trình sắp xếp nếu đang chạy, xóa label dãy số ban đầu, đặt lại khung ý tưởng và khung code, đặt lại bộ đếm thời gian chạy

|  |
| --- |
| private void XoaMang()  {  // Xóa các node  foreach (Control node in nodeArr)  {  node.Dispose();  }  nodeArr.Clear();  // Xóa các label  foreach (Control label in labelSTTArr)  {  label.Dispose();  }  labelSTTArr.Clear();  // Tắt chế độ tạm dừng nếu đang Tạm dừng  if (CodeListBoxIsPause == true || Node.IsPause == true)  {  TamDung();  }  // Cancel các thread đang hoạt động  if (sortingThread != null)  {  sortingThread.Abort();  }  foreach (Label item in bienArr.Values)  {  item.Visible = false;  }  SortRunOrStop(false);  // Xóa label daySoChuaSapXep  daySoChuaSapXepLabel.Text = "";  // đặt lại yTuongTextBox và codeTextbox  CapNhatYTuongVaCode();  // reset lại thời gian chạy  thoiGianChayTimer.Stop();  thoiGianChay\_GiayLabel.Text = thoiGianChay\_PhutLabel.Text = "00";  } |

**Các hàm bắt đầu và hủy quá trình sắp xếp và các hàm hỗ trợ**

- Các hàm hỗ trợ cho hàm bắt đầu sắp xếp:

Hàm SortRunOrStop : hàm thực hiện điều chỉnh cần thiết khi quá trình sắp xếp bắt đầu và dừng lại . Ví dụ như tắt/bật các button, tắt/bật chế độ nhập tay cho các Node, dừng bộ đếm thời gian

|  |
| --- |
| public void SortRunOrStop(bool IsRun)  {  // Nếu dừng mà mảng rỗng thì tắt batDauButton  if (nodeArr != null)  batDauButton.Enabled = !IsRun;  cancelSortingButton.Enabled = IsRun;  isRunning = IsRun;  codeListBox.SelectedIndex = -1;  taoNgauNghienButton.Enabled = !IsRun;  nhapMotDayButton.Enabled = !IsRun;  nhapTayButton.Enabled = !IsRun;  debugButton.Enabled = IsRun;  // Nếu debug đang bật thì tamdungButton tắt  if (!debugCheckBox.Checked)  tamDungButton.Enabled = IsRun;  foreach (Node node in nodeArr)  {  // tắt chế độ nhập tay khi chạy sắp xếp  node.nhapTayTexbox.Enabled = !IsRun;  // Chuyển về màu mặc định khi bắt đầu sắp xếp  if (IsRun)  {  node.BackColor = ThamSo.MauNenNode;  }  }  // tắt các biến label đang hiện thị  foreach (Label label in bienArr.Values)  {  label.Visible = false;  }  if (!IsRun)  {  thoiGianChayTimer.Stop();  }  } |

- Hàm để bắt đầu quá trình sắp xếp

Hàm BatDauSapXep : hàm này sẽ bắt đầu quá trình sắp xếp bằng việc tạo ra một tiến trình sắp xếp, gán hàm cần chạy vào và bắt đầu tiến trình sắp xếp đó

|  |
| --- |
| Thread sortingThread;  Action hamSapXep;  public void BatDauSapXep()  {  SortRunOrStop(true);  if (sortingThread != null) // Hủy thread nếu vẫn đang chạy  {  sortingThread.Abort();  }  // reset timer về mặc định  thoiGianChay\_GiayLabel.Text = thoiGianChay\_PhutLabel.Text = "00";  thoiGianChayTimer.Start();  sortingThread = new Thread(new ThreadStart(hamSapXep));  sortingThread.Start();  // Dùng thread vì : có thể abort khi nào cần, task k làm đc  // cả task và thread đều k thể tạo control và add vào trong thread khác --> dùng Invoke để giải quyết  } |

- Hàm hủy quá trình sắp xếp:

Hàm HuyQuaTrinh : hàm này hủy quá trình đang sắp xếp bằng cách xóa mảng hiện có và tạo lại mảng mới với giá trị là dãy số ban đầu

|  |
| --- |
| private void HuyQuaTrinh(int soPhanTu)  {  string[] mangCu = daySoChuaSapXepLabel.Text.Split();  XoaMang();  for (int i = 0; i < soPhanTu; i++)  {  int giaTri = int.Parse(mangCu[i]);  Node temp = new Node(i, giaTri);  this.sortingPanel.Controls.Add(temp);  nodeArr.Add(temp);  temp.Location = new Point(root.X + i \* ThamSo.KhoanCachGiuaCacNode, root.Y - ThamSo.KichCoNode / 2);  TaoLabelSoThuTuChoMang(i);  }  CapNhatDaySoChuaSapXep();  } |

**Các hàm di chuyển vị trí của Node và hoán vị hai Node**

Hàm DiChuyenNodeDen : hàm này di chuyển một Node đến một vị trí chỉ định

|  |
| --- |
| public void DiChuyenNodeDen(object oNode, object oVitriMoi) // public delegate void ParameterizedThreadStart(object obj);  {  int vitriMoi = (int)oVitriMoi;  Node node = (Node)oNode;  if (vitriMoi > node.vitriHienTai)  {  node.ChuyenLen();  node.ChuyenQua(vitriMoi);  node.ChuyenXuong();  }  else if (vitriMoi < node.vitriHienTai)  {  node.ChuyenXuong();  node.ChuyenQua(vitriMoi);  node.ChuyenLen();  }  // Gán lại giá trị vị trí hiện tại  node.vitriHienTai = vitriMoi;  // Khi dùng thread thì dùng delegate Callback lại  //this.BeginInvoke(moveIsStopped); // define: public Action moveIsStopped = null;  } |

Hàm HoanVi2Node : hàm này giúp đổi chổ hai Node trong mảng. Hàm tạo ra hai Task với mỗi task di chuyển một Node đến vị trí của Node còn lại. Khi thực hiện xong nó sẽ đổi màu hai Node và đổi vị trí của hai Node bên trong nodeArr

|  |
| --- |
| Task hoanVi1Task;  Task hoanVi2Task;  private void HoanVi2Node(int vitriNodeA, int vitriNodeB)  {  // Cách dùng task  hoanVi1Task = Task.Factory.StartNew(() => { DiChuyenNodeDen(nodeArr[vitriNodeA], vitriNodeB); });  hoanVi2Task = Task.Factory.StartNew(() => { DiChuyenNodeDen(nodeArr[vitriNodeB], vitriNodeA); });  // note: Task.Factory.StartNew = THÊM task vào cuối hàng đợi và CHẠY ngay khi có thể  Task.WaitAll(hoanVi1Task, hoanVi1Task);  // Thay đổi vị trí của node trong mảng nodeArray  if (nodeArr.Count != 0) //check xem nếu mảng còn tồn tại --> trong trường hợp mảng đã bị hủy  {  // Đổi màu 2 node sau khi sắp xếp  Color tempColor = nodeArr[vitriNodeA].BackColor;  nodeArr[vitriNodeA].BackColor = nodeArr[vitriNodeB].BackColor;//nodeArr[vitriNodeB].BackColor;  nodeArr[vitriNodeB].BackColor = tempColor;    Node t = nodeArr[vitriNodeA];  nodeArr[vitriNodeA] = nodeArr[vitriNodeB];  nodeArr[vitriNodeB] = t;  }  } |

**Hàm tạm dừng quá trình sắp xếp**

Hàm TamDung : hàm này kiểm tra xem Node có đang di chuyển hay không nếu không thì tạm dừng Node lại còn có thì cho Node tiếp tục di chuyển. Hàm này điều khiển ManualResetEvent của cả Node và ManualResetEvent có trong hàm chọn từng dòng code.

|  |
| --- |
| public static ManualResetEvent codeListBoxPauseStatus = new ManualResetEvent(true);  public static bool CodeListBoxIsPause = false;  void TamDung()  {  if (Node.IsPause)  {  Node.pauseStatus.Set(); // hàm để resume  Node.IsPause = false;  tamDungButton.Text = "Tạm dừng";  thoiGianChayTimer.Start();  }  else  {  Node.pauseStatus.Reset(); // hàm để pause  Node.IsPause = true;  tamDungButton.Text = "Tiếp tục";  thoiGianChayTimer.Stop();  }  if (CodeListBoxIsPause)  {  codeListBoxPauseStatus.Set();  CodeListBoxIsPause = false;  tamDungButton.Text = "Tạm dừng";  thoiGianChayTimer.Start();  }  else  {  codeListBoxPauseStatus.Reset();  CodeListBoxIsPause = true;  tamDungButton.Text = "Tiếp tục";  thoiGianChayTimer.Stop();  }  } |

**Hàm chọn từng dòng code trong khung Code C/C++ và chế độ Debug**

Hàm ChonDongChoCodeListBox : hàm nhận giá trị vị trí hàng được chọn trong khung Code C/C++ và chọn hàng đó. Hàm này giữ một ManualResetEvent giúp cho việc tạm dừng code và còn giúp cho việc Debug. Mỗi khi chạy chọn một lệnh nếu đang ở trong chế độ Debug thì cho dừng code lại đến khi nhấn nút Debug thì mới tiếp tục chạy.

|  |
| --- |
| public void ChonDongChoCodeListBox(int viTri)  {  Thread.Sleep(ThamSo.TocDo \* 30);  codeListBoxPauseStatus.WaitOne(Timeout.Infinite); // có thể pause mỗi khi chuyển line  codeListBox.SelectedIndex = viTri;    // nếu đang trong chế độ Debug thì dừng sau mỗi câu lệnh chạy xong sẽ dừng lại  if (debugCheckBox.Checked)  {  codeListBoxPauseStatus.Reset();  CodeListBoxIsPause = true;  }  } |

**Các hàm xóa mảng**

- Hàm XoaMang : để xóa tất cả các phần tử sau khi đã nhấn vào button bắt đầu hoặc ngay cả khi đang trong quá trình sắp xếp mà chủ yếu theo yêu cầu của người dùng. Khi thực hiện xóa mảng:

Xóa tất cả các node đang hiển thị trên dãy sắp xếp.

Xóa tất cả các Label tương ứng với dãy số sắp xếp trên giao diện.

Tắt chế độ tạm dừng nếu đang tạm dừng.

Hủy bỏ tất cả các thread đang hoạt động trên chương trình.

Xóa tất cả các phần tử Label trên dãy số chưa sắp xếp.

Reset lại thời gian để đo cho một phương thức sắp xếp.

|  |
| --- |
| private void XoaMang()  {  // Xóa các node  foreach (Control node in nodeArr)  {  node.Dispose();  }  nodeArr.Clear();  // Xóa các label  foreach (Control label in labelSTTArr)  {  label.Dispose();  }  labelSTTArr.Clear();  // Tắt chế độ tạm dừng nếu đang Tạm dừng  if (CodeListBoxIsPause == true || Node.IsPause == true)  {  TamDung();  }  // Cancel các thread đang hoạt động  if (sortingThread != null)  {  sortingThread.Abort();  }  foreach (Label item in bienArr.Values)  {  item.Visible = false;  }  SortRunOrStop(false);  // Xóa label daySoChuaSapXep  daySoChuaSapXepLabel.Text = "";  // đặt lại yTuongTextBox và codeTextbox  CapNhatYTuongVaCode();  // reset lại thời gian chạy  thoiGianChayTimer.Stop();  thoiGianChay\_GiayLabel.Text = thoiGianChay\_PhutLabel.Text = "00";  } |

**Cập nhật trạng thái ý tưởng thuật toán, dãy chưa sắp xếp**

- Khi bắt đầu quá trình sắp xếp, tùy thuộc vào phương pháp sắp xếp mà sẽ có những biến và số lượng biến khác nhau phù hợp cho phương pháp sắp xếp đó. Trong quá trình sắp xếp, những phần tử của dãy số sẽ đổi chỗ cho nhau tùy thuộc vào thuật toán mà người sử dụng chọn. Do đó ý tưởng thuật toán sẽ cập nhật liên tục trong quá trình sắp xếp.

Hàm CapNhatYTuongTextBox :Cập nhật cho trạng thái mảng vào Ý tưởng TextBox

|  |
| --- |
| public void CapNhatYTuongTextBox(int i, int j, string chuDau, string chuCuoi)  {  string temp = "\r\n" + chuDau + "=" + i + " ," + chuCuoi + "=" + j + " : ";  for (int h = 0; h < nodeArr.Count; h++)  {  temp += nodeArr[h].giaTri.ToString("D2") + " ";  }  yTuongTextBox.AppendText(temp);  } |

Hàm CapNhatDaySoChuaSapXep :Cập nhật dãy số chưa sắp xếp tương ứng với dãy số ban đầu khi mới khởi tạo.

|  |
| --- |
| private void CapNhatDaySoChuaSapXep()  {  daySoChuaSapXepLabel.Text = "";  foreach (Node node in nodeArr)  {  node.BackColor = ThamSo.MauNenNode;  daySoChuaSapXepLabel.Text += node.Text + " ";  }  } |

**Phần code sẽ được cập nhật vào khung Code C/C++:**

|  |
| --- |
| public static void InterchangeSort(bool tang = true)  {  string yTuong = @"Xuất phát từ đầu dãy,tìm tất cả các cặp nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi phần tử này với phần tử tương ứng trong cặp nghịch thế .Lặp lại xử lý trên với các phần tử tiếp theo";  string[] code = ChuyenText(  @"Sắp tăng  void InterchangeSort( int a[], int N)  {  int i, j;  for(i = 0; i < N - 1; i++)  for(j = i + 1; j < N; j++)  if( a[j] < a[i] )  Swap( a[i], a[j]);  }");  yTuongTextBox.Clear();  yTuongTextBox.Text = yTuong;  codeListBox.Items.Clear();  foreach (string item in code)  {  codeListBox.Items.Add(item);  }  if (!tang)  {  codeListBox.Items[0] = "Sắp giảm";  codeListBox.Items[6] = " if( a[j] > a[i] )";  }  } |

**Các hàm sắp xếp mảng**

- Hàm InterchangeSort : sắp xếp mảng theo thuật toán InterchangeSort theo hai hướng tăng và giảm. Có thể hiện các biến trong thuật toán. Có điều chỉnh màu của các biến và các Node

|  |
| --- |
| public void InterchangeSort()  {  yTuongTextBox.Clear();  ChonDongChoCodeListBox(3);  int i = 0, j = 0;  ChonDongChoCodeListBox(4);  nodeArr[i].BackColor = ThamSo.MauNodeDangXet;  bienArr["i"].Text = "i = " + i;  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* i, sortingPanel.Height - 50);  bienArr["i"].Visible = true;  for (i = 0; i < nodeArr.Count - 1; i++)  {  ChonDongChoCodeListBox(5);  j = i + 1;  bienArr["j"].Text = "j = " + j;  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* j, sortingPanel.Height - 50);  bienArr["j"].Visible = true;  if(j != nodeArr.Count)  nodeArr[j].BackColor = ThamSo.MauNodeDangXet;  for (j = i + 1; j < nodeArr.Count; j++)  {  ChonDongChoCodeListBox(6);  bool thucHien = false; // dùng để xét tăng/giảm , nếu bằng true thì code sẽ chạy  if (tangRadioButton.Checked == true)  {  if (nodeArr[i].giaTri > nodeArr[j].giaTri)  thucHien = true;  }  else  {  if (nodeArr[i].giaTri < nodeArr[j].giaTri)  thucHien = true;  }  if (thucHien)  {  ChonDongChoCodeListBox(7);  HoanVi2Node(i, j);  // Cập nhật trạng thái của mảng lên yTuongTextBox -- AppendText : thêm dòng mới rồi scroll textbox tới dòng vừa thêm  CapNhatYTuongTextBox(i, j,"i","j");  }  ChonDongChoCodeListBox(5);  bienArr["j"].Text = "j = " + (j + 1);  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (j + 1), sortingPanel.Height - 50);  if (j+1 != nodeArr.Count)  nodeArr[j + 1].BackColor = ThamSo.MauNodeDangXet;  nodeArr[j].BackColor = ThamSo.MauNenNode;  }  ChonDongChoCodeListBox(4);  bienArr["i"].Text = "i = " + (i + 1);  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (i + 1), sortingPanel.Height - 50);  if (i + 1 != nodeArr.Count)  nodeArr[i + 1].BackColor = ThamSo.MauNodeDangXet;  // Đổi màu các node đã xét qua  nodeArr[i].BackColor = ThamSo.MauNodeDaXetQua;  }  ChonDongChoCodeListBox(4);  // Đổi màu node i cuối  nodeArr[nodeArr.Count - 1].BackColor = ThamSo.MauNodeDaXetQua;  // Kết thúc  SortRunOrStop(false);  if (nodeArr.Count != 0) // Nếu mảng bị huy trong lúc chạy thì k cần in ra kết quả  MessageBox.Show("Sắp xếp hoàn tất","Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Information);  } |

Hàm SelectionSort:

|  |
| --- |
| public void SelectionSort()  {  yTuongTextBox.Clear();  ChonDongChoCodeListBox(3);  int i = 0, j = 0, min;  ChonDongChoCodeListBox(4);  nodeArr[i].BackColor = ThamSo.MauNodeDangXet;  bienArr["i"].Text = "i = " + i;  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* i, sortingPanel.Height - 50);  bienArr["i"].Visible = true;  for (i = 0; i < nodeArr.Count - 1; i++)  {  ChonDongChoCodeListBox(6);  min = i;  bienArr["min"].Text = "min = " + min;  bienArr["min"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* min -10, 40);  bienArr["min"].SendToBack();  bienArr["min"].Visible = true; // Đặt visible ở dây tránh trường hợp khi hiện lên vẫn còn nằm ở vị trí cũ  nodeArr[min].BackColor = Color.LightGreen;  ChonDongChoCodeListBox(7);  j = i + 1;  bienArr["j"].Text = "j = " + j;  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* j, sortingPanel.Height - 50);  bienArr["j"].Visible = true;  if (j != nodeArr.Count)  nodeArr[j].BackColor = ThamSo.MauNodeDangXet;  for (j = i+1; j < nodeArr.Count; j++)  {  ChonDongChoCodeListBox(8);  bool thucHien = false; // dùng để xét tăng/giảm , nếu bằng true thì code sẽ chạy  if (tangRadioButton.Checked == true)  {  if (nodeArr[j].giaTri < nodeArr[min].giaTri)  thucHien = true;  }  else  {  if (nodeArr[j].giaTri > nodeArr[min].giaTri)  thucHien = true;  }  if (thucHien)  {  // đổi lại màu nền khi k còn là node min  if (min == i)  nodeArr[min].BackColor = ThamSo.MauNodeDangXet;  else  nodeArr[min].BackColor = ThamSo.MauNenNode;  min = j;  bienArr["min"].Text = "min = " + min;  nodeArr[min].BackColor = Color.LightGreen;  bienArr["min"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* min - 10, 40);  }  ChonDongChoCodeListBox(7);  bienArr["j"].Text = "j = " + (j + 1);  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (j + 1), sortingPanel.Height - 50);  if (j + 1 != nodeArr.Count)  nodeArr[j + 1].BackColor = ThamSo.MauNodeDangXet;  // Nếu là min thì k đổi màu  if (j != min)  nodeArr[j].BackColor = ThamSo.MauNenNode;  }  ChonDongChoCodeListBox(10);  HoanVi2Node(i, min);  Task.WaitAll(hoanVi1Task, hoanVi2Task);  CapNhatYTuongTextBox(i, j,"i","j");  // đổi lại màu bth cho node min sau khi đã swap xong  nodeArr[min].BackColor = ThamSo.MauNenNode;  ChonDongChoCodeListBox(4);  bienArr["i"].Text = "i = " + (i + 1);  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (i + 1), sortingPanel.Height - 50);  if (i + 1 != nodeArr.Count)  nodeArr[i + 1].BackColor = ThamSo.MauNodeDangXet;  // Đổi màu các node đã xét qua  nodeArr[i].BackColor = ThamSo.MauNodeDaXetQua;  }  ChonDongChoCodeListBox(4);  // Đổi màu node i cuối  nodeArr[nodeArr.Count - 1].BackColor = ThamSo.MauNodeDaXetQua;  SortRunOrStop(false);  if (nodeArr.Count != 0)  MessageBox.Show("Sắp xếp hoàn tất","Thông báo",MessageBoxButtons.OK,MessageBoxIcon.Information);  } |

Hàm InsertionSort

|  |
| --- |
| public void InsertionSort()  {  yTuongTextBox.Clear();  int i, pos, x;  Node nodeTam, nodeTam2;  nodeArr[0].BackColor = ThamSo.MauNodeDangXet;  for (i = 1; i < nodeArr.Count; i++)  {  ChonDongChoCodeListBox(6);  nodeArr[i].BackColor = ThamSo.MauNodeDangXet;  nodeArr[i].ForeColor = Color.Red;  x = nodeArr[i].giaTri;  ChonDongChoCodeListBox(8);  nodeTam = nodeArr[i];  pos = i - 1;  bienArr["i"].Text = "i = " + i;  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* i, sortingPanel.Height - 50);  bienArr["i"].Visible = true;  nodeArr[i].ChuyenLen();  nodeArr[0].BackColor = ThamSo.MauNodeDaXetQua;  nodeArr[0].ForeColor = Color.Purple;  ChonDongChoCodeListBox(9);  bienArr["pos"].Text = "pos = " + pos;  bienArr["pos"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* pos - 10, sortingPanel.Height - 50);  bienArr["pos"].Visible = true;  if (tangRadioButton.Checked == true)  {  while ((pos >= 0) && (nodeArr[pos].giaTri > x))  {  ChonDongChoCodeListBox(11);  nodeArr[pos].ChuyenQua(pos + 1);  nodeArr[pos].vitriHienTai = pos + 1;  nodeTam2 = nodeArr[pos + 1];  nodeArr[pos + 1] = nodeArr[pos];  nodeArr[pos] = nodeTam2;  ChonDongChoCodeListBox(12);  pos--;  }  }  else  {  while ((pos >= 0) && (nodeArr[pos].giaTri < x))  {  ChonDongChoCodeListBox(11);  nodeArr[pos].ChuyenQua(pos + 1);  nodeArr[pos].vitriHienTai = pos + 1;  nodeTam2 = nodeArr[pos + 1];  nodeArr[pos + 1] = nodeArr[pos];  nodeArr[pos] = nodeTam2;  ChonDongChoCodeListBox(12);  pos--;  }  }  ChonDongChoCodeListBox(14);  nodeTam.ChuyenQua(pos + 1);  nodeTam.ChuyenXuong();  nodeArr[pos + 1] = nodeTam;  nodeTam.vitriHienTai = pos + 1;  nodeArr[pos + 1].BackColor = ThamSo.MauNodeDaXetQua;  nodeArr[pos + 1].ForeColor = Color.Purple;  //Cập nhật ý tưởng Text Box  string temp = "\r\ni=" + i + ",pos=" + (i - 1) + " : ";  for (int h = 0; h < nodeArr.Count; h++)  {  temp += nodeArr[h].giaTri + " ";  }  yTuongTextBox.AppendText(temp);  }  SortRunOrStop(false);  if (nodeArr.Count != 0)  {  MessageBox.Show("Sắp xếp hoàn tất", "Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Information);  }  } |

Hàm BubbleSort :

|  |
| --- |
| public void BubbleSort()  {  // Clear yTuongTextBox  yTuongTextBox.Clear();  ChonDongChoCodeListBox(4);  int i = 0, j = 0;  nodeArr[i].BackColor = ThamSo.MauNodeDaXetQua;  bienArr["i"].Text = "i = " + i;  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* i, sortingPanel.Height - 50);  bienArr["i"].Visible = true;  for (i = 0; i < nodeArr.Count - 1; i++)  {  ChonDongChoCodeListBox(5);  j = nodeArr.Count - 1;  bienArr["j"].Text = "j = " + j;  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* j, sortingPanel.Height - 50);  bienArr["j"].Visible = true;  for (j = nodeArr.Count - 1; j > i; j--)  {  //Đổi màu các Node đang xét  if (j != i)  nodeArr[j].BackColor = ThamSo.MauNodeDangXet;  nodeArr[j - 1].BackColor = ThamSo.MauNodeDangXet;  ChonDongChoCodeListBox(6);  bool thucHien = false; // dùng để xét tăng/giảm , nếu bằng true thì code sẽ chạy  if (tangRadioButton.Checked == true)  {  if (nodeArr[j].giaTri < nodeArr[j - 1].giaTri)  thucHien = true;  }  else  {  if (nodeArr[j].giaTri > nodeArr[j - 1].giaTri)  thucHien = true;  }  if (thucHien)  {  ChonDongChoCodeListBox(7);  HoanVi2Node(j, j - 1);  // Cập nhật trạng thái của mảng lên yTuongTextBox -- AppendText : thêm dòng mới rồi scroll textbox tới dòng vừa thêm  CapNhatYTuongTextBox(i, j, "i", "j");  }  if (j - 1 != i)  {  ChonDongChoCodeListBox(5);  bienArr["j"].Text = "j = " + (j - 1);  bienArr["j"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (j - 1), sortingPanel.Height - 50);  }  //Đổi màu Node đã xét qua  if (j != i)  nodeArr[j].BackColor = ThamSo.MauNenNode;  }  // Đổi màu các node đã xét qua  nodeArr[i].BackColor = ThamSo.MauNodeDaXetQua;  ChonDongChoCodeListBox(4);  bienArr["i"].Text = "i = " + (i + 1);  bienArr["i"].Location = new Point(root.X + ThamSo.KhoanCachGiuaCacNode \* (i + 1), sortingPanel.Height - 50);  //Đổi màu Node cuối cùng  if (i != nodeArr.Count)  {  ChonDongChoCodeListBox(4);  nodeArr[i + 1].BackColor = ThamSo.MauNodeDaXetQua;  }  }  ChonDongChoCodeListBox(4);  // Đổi màu node i cuối  nodeArr[nodeArr.Count - 1].BackColor = ThamSo.MauNodeDaXetQua;  // Kết thúc  SortRunOrStop(false);  if (nodeArr.Count != 0) // Nếu mảng bị hủy trong lúc chạy thì k cần in ra kết quả  MessageBox.Show("Sắp xếp hoàn tất","Thông báo",MessageBoxButtons.OK,MessageBoxIcon.Information);  } |