BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**SỬ DỤNG THƯ VIỆN GRAPHICS.H MÔ PHỎNG GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT)**

**Giảng viên hướng dẫn : ThS. Đoàn Vũ Thịnh**

**Sinh viên thực hiện : Lê Thanh Phước**

**Mã số sinh viên : 64131856**

**Lớp : 64.CNTT-3**

**Lớp : 64.CNTT-3**

Khánh Hòa - 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**SỬ DỤNG THƯ VIỆN GRAPHICS.H MÔ PHỎNG GIẢI THUẬT SẮP XẾP NHANH (QUICK SORT)**

GVHD : ThS. Đoàn Vũ Thịnh

SVTH : Lê Thanh Phước

MSSV : 64131856

Lớp : 64.CNTT-3

Khánh Hòa - 2024

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**Khoa: Công nghệ Thông tin**

**PHIẾU THEO DÕI TIẾN ĐỘ VÀ ĐÁNH GIÁ BÁO CÁO THỰC TẬP CƠ SỞ**

**Tên đề tài:** Sử dụng công cụ đồ họa graphics.h để mô phỏng quá trình sắp xếp dãy số bằng giải thuật Quick Sort.

**Giảng viên hướng dẫn:** ThS. Đoàn Vũ Thịnh

**Sinh viên được hướng dẫn:** Lê Thanh Phước

**MSSV:** 64131856

**Khóa:** 64 **Ngành:** Công nghệ Thông tin

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 28/11/2024 | Nhận đề tài hướng dẫn và định hướng giải quyết vấn đề. Sinh viên trình bày kế hoạch thực hiện. |  |
| 2 | 6/12/2024 | Sinh viên thực hiện mô phỏng giao diện thông qua việc tạo và đổi màu các button khi click, xử lý sự kiện chuột, các thanh bar với độ cao tương ứng với giá trị và khung giao diện chứa dãy số cần sắp xếp. |  |
| 3 | 12/12/2024 | Sinh viên tiến hành thực hiện thuật toán đã đề ra với việc tạo ra các cách nhập dữ liệu đầu vào đa dạng như bàn phím, đọc file, random. |  |
| 4 | 21/12/2024 | Sinh viên tiếp tục thực hiện phân tích thuật toán, nắm rõ cách sắp xếp và di chuyển của các phần tử để tạo các chức năng sắp xếp tăng dần, giảm dần, tạo thêm hai nút chức năng là làm mới và thoát chương trình. |  |
| 5 | 28/12/2024 | Sinh viên thực hiện demo và sửa những lỗi phát sinh trong quá trình sắp xếp như lỗi dư thừa điểm ảnh và tô màu trong lúc di chuyển các phần tử. |  |
| 6 | 5/1/2025 | Sinh viên nộp bản thảo lần cuối sau khi đã chỉnh sửa các yêu cầu như đã đề ra. |  |

# **MỤC LỤC**

**[MỤC LỤC I](#_Toc28594)**

**[DANH MỤC HÌNH III](#_Toc32704)**

**[LỜI CẢM ƠN V](#_Toc17494)**

**[TÓM TẮT VI](#_Toc32323)**

**[CHƯƠNG 1. TỔNG QUAN 1](#_Toc18338)**

[1.1. Dev-C++ 1](#_Toc7040)

[1.2. Thư viện Graphic.h 1](#_Toc4003)

[1.3. Thuật toán Quick Sort (Sắp xếp nhanh) 2](#_Toc9463)

[1.3.1. Sơ lược về thuật toán. 2](#_Toc3315)

[1.3.2. Ý tưởng chính của thuật toán. 3](#_Toc9607)

[1.3.3. Độ phức tạp của thuật toán Quick Sort. 4](#_Toc20703)

[1.3.4. So sánh độ phức tạp về thời gian của các thuật toán. 5](#_Toc31631)

[1.3.5. Lưu đồ thuật toán Quick Sort. 7](#_Toc4483)

**[CHƯƠNG 2. PHƯƠNG PHÁP NGHIÊN CỨU 10](#_Toc30987)**

[2.1. Tổng quan về giao diện 10](#_Toc17400)

[2.1.1. Cài đặt Dev-C++ và thư viện graphic.h 10](#_Toc9620)

[2.1.2. Xử lý sự kiện chuột với thư viện BGI 10](#_Toc9419)

[2.1.3. Xây dựng giao diện 12](#_Toc14703)

[2.2. Phân tích thuật toán 20](#_Toc2392)

[2.2.1. Khởi tạo mảng 21](#_Toc25105)

[2.2.2. Thuật toán Quick Sort 29](#_Toc26807)

[2.2.3. Button chức năng 39](#_Toc9154)

**[CHƯƠNG 3. KẾT QUẢ THỰC HIỆN 41](#_Toc16735)**

**[CHƯƠNG 4. KẾT LUẬN 43](#_Toc2365)**

[4.1. Ưu điểm 43](#_Toc26804)

[4.2. Nhược điểm 43](#_Toc6537)

**[TÀI LIỆU THAM KHẢO 44](#_Toc31760)**

# 

# **DANH MỤC HÌNH**

[Hình 1.1. Hình ảnh minh họa sắp xếp nhanh 3](#_Toc24859)

[Hình 1.2. Hình ảnh lưu đồ thuật toán Quick Sort. 7](#_Toc23031)

[Hình 2.1. Hình ảnh minh họa graphic.h 10](#_Toc11912)

[Hình 2.2. Hình ảnh lấy tọa độ khi sử dụng sự kiện chuột. 12](#_Toc2202)

[Hình 2.3. Hình ảnh giao diện lúc mới mở. 13](#_Toc29572)

[Hình 2.4. Hình ảnh code xây dựng Button. 14](#_Toc1559)

[Hình 2.5. Hình ảnh code xây dựng màu button 17](#_Toc22621)

[Hình 2.6. Hình ảnh giao diện màu button. 18](#_Toc30652)

[Hình 2.7. Hình ảnh code xây dựng text. 18](#_Toc31352)

[Hình 2.8. Hình ảnh code xây dựng thanh bar. 19](#_Toc9575)

[Hình 2.9. Hình ảnh thanh bar 20](#_Toc18978)

[Hình 2.10. Hình ảnh code nhập số lượng phần tử từ bàn phím. 22](#_Toc28300)

[Hình 2.11. Hình ảnh nhập số lượng phần tử từ bàn phím. 23](#_Toc32748)

[Hình 2.12. Hình ảnh giao diện nhập số lượng phần tử từ bàn phím. 23](#_Toc27806)

[Hình 2.13. Hình ảnh code nhập phần tử từ bàn phím với số lượng mặc định. 24](#_Toc11781)

[Hình 2.14. Hình ảnh nhập phần tử từ bàn phím với số lượng mặc định. 24](#_Toc21903)

[Hình 2.15. Hình ảnh giao diện nhập phần tử từ bàn phím với số lượng mặc định. 25](#_Toc21795)

[Hình 2.16. Hình ảnh code tạo số lượng phần tử và giá trị ngẫu nhiên. 26](#_Toc1923)

[Hình 2.17. Hình ảnh tạo số lượng phần tử và giá trị ngẫu nhiên. 27](#_Toc25086)

[Hình 2.18. Hình ảnh giao diện tạo số lượng phần tử và giá trị ngẫu nhiên. 27](#_Toc12012)

[Hình 2.19. Hình ảnh code tạo các phần tử bằng cách đọc file. 28](#_Toc24433)

[Hình 2.20. Hình ảnh tạo các phần tử bằng cách đọc file. 28](#_Toc17060)

[Hình 2.21. Hình ảnh giao diện tạo các phần tử bằng cách đọc file. 29](#_Toc19122)

[Hình 2.22. Hình ảnh code thuật toán Qick Sort sắp xếp tăng dần. 31](#_Toc11717)

[Hình 2.23. Hình ảnh sau khi sắp xếp tăng dần. 33](#_Toc8773)

[Hình 2.24. Hình ảnh code thuật toán Qick Sort sắp xếp giảm dần. 36](#_Toc32075)

[Hình 2.25. Hình ảnh sau khi sắp xếp giảm dần. 38](#_Toc19864)

[Hình 3.1. Hình ảnh kết quả khởi tạo mảng. 41](#_Toc22471)

[Hình 3.2. Hình ảnh kết quả đổi màu xanh cho các phần tử đang so sánh và đã sắp xếp. 41](#_Toc10855)

[Hình 3.3. Hình ảnh kết quả Quick Sort sắp xếp tăng dần. 42](#_Toc30638)

[Hình 3.4. Hình ảnh kết quả Quick Sort sắp xếp giảm dần. 42](#_Toc13914)

# 

# **LỜI CẢM ƠN**

Để có thể hoàn thành đợt thực tập lần này, em xin chân thành cảm ơn đến quý thầy cô khoa Công nghệ Thông tin đã tạo điều kiện hỗ trợ và giúp đỡ em trong quá trính học tập và nghiên cứu đề tài này.

Qua đây, em xin chân thành cảm ơn thầy Đoàn Vũ Thịnh, người đã trực tiếp quan tâm và hướng dẫn chúng em hoàn thành tốt đợt thực tập trong thời gian qua.

Do kiến thức còn hạn chế và thời gian thực hiện còn ngắn nên bài báo cáo của em còn nhiều thiếu sót, kính mong sự góp ý của quý thầy cô.

Em xin chân thành cảm ơn!

# **TÓM TẮT**

Trong khoa học máy tính, thuật toán sắp xếp là một thuật toán đặt các phần tử của danh sách thành một thứ tự. Thứ tự thường được áp dụng nhiều nhất là thứ tự số và thứ tự từ điển, có thể là thứ tự tăng dần hoặc giảm dần. Sắp xếp hiệu quả rất quan trọng để tối ưu hóa hiệu quả của các thuật toán khác (như thuật toán tìm kiếm và hợp nhất) yêu cầu dữ liệu đầu vào phải nằm trong danh sách được sắp xếp. Quá trình sắp xếp còn hữu ích trong việc biểu diễn dữ liệu một cách có trật tự và tạo ra đầu ra dễ đọc cho con người. Tất cả các bước thực hiện, từ triển khai thuật toán đến hiển thị kết quả trên màn hình, đều được thực hiện trong môi trường C/C++ thông qua sử dụng phần mềm Dev-C++ kết hợp với thư viện graphics.h.

Sản phẩm đã minh họa được từng bước sắp xếp mảng sử dụng giải thuật sắp xếp Qick Sort. Đồng thời cũng chỉ ra các ưu điểm và hạn chế của thuật toán và cách khắc phục các nhược điểm đó. Ứng dụng chạy mượt mà với dữ liệu được lấy từ file, nhập từ bàn phím và tương tác thuận lợi với sự hỗ trợ của chuột.

Toàn bộ mã nguồn của báo cáo được tải lên theo địa ch

# **CHƯƠNG 1. TỔNG QUAN**

## **Dev-C++**

Dev-C++ là một môi trường phát triển tích hợp (IDE) cho ngôn ngữ lập trình C và C++. Nó được thiết kế để cung cấp một giao diện dễ sử dụng cho việc phát triển ứng dụng trong hai ngôn ngữ lập trình này. Dưới đây là một đặc điểm về Dev-C++:

* Môi trường Phát triển Tích hợp (IDE): Dev-C++ cung cấp một môi trường phát triển tích hợp mạnh mẽ và thuận tiện. Nó bao gồm một trình soạn thảo mã nguồn, trình biên dịch, và một loạt các công cụ hỗ trợ quản lý dự án.
* Hỗ trợ nhiều trình biên dịch: Dev-C++ hỗ trợ nhiều trình biên dịch, bao gồm MinGW, một phiên bản của GCC (GNU Compiler Collection) dành cho Windows. Điều này cho phép lập trình viên lựa chọn trình biên dịch phù hợp với nhu cầu của họ.
* Giao diện đồ họa sử dụng thư viện graphics.h: Dev-C++ tích hợp với thư viện graphics.h để hỗ trợ việc vẽ đồ họa đơn giản trong ứng dụng C/C++. Điều này làm cho nó trở thành một công cụ hữu ích cho việc phát triển ứng dụng đồ họa cơ bản.
* Quản lý dự án: IDE này cung cấp các tính năng quản lý dự án, giúp lập trình viên tổ chức mã nguồn của họ một cách hiệu quả. Nó hỗ trợ việc tạo, xóa, và quản lý tệp và thư mục trong dự án.
* Debugging và Profiling: Dev-C++ hỗ trợ các tính năng quan trọng như debugging và profiling, giúp lập trình viên xác định và sửa lỗi trong mã nguồn của họ và tối ưu hóa hiệu suất.
* Hỗ trợ Chuột: IDE này hỗ trợ tương tác với chuột, giúp làm cho quá trình phát triển và thử nghiệm ứng dụng trở nên thuận lợi hơn.
* Tích hợp Git: Dev-C++ có tích hợp hỗ trợ cho hệ thống quản lý phiên bản Git, giúp lập trình viên quản lý mã nguồn của họ một cách dễ dàng.

## **Thư viện Graphic.h**

Do sử dụng Dev C++ làm trình biên dịch cho triển khai thuật toán, việc này không thể thực hiện trên môi trường Windows một cách trực tiếp. Vì vậy, Michael đã sáng tạo một môi trường giả lập đồ họa dựa trên Borland C và phát triển thư viện được gọi là Graphics.h để hỗ trợ quá trình này. Để đạt được mục tiêu này, Michael đã tiến hành thay đổi thư viện BGI, chuyển đổi nó thành thư viện mới có tên là WinBGIm, được tối ưu hóa để hoạt động mạnh mẽ trên hệ điều hành Windows. Nhờ vào sự cải tiến này, bạn có thể sử dụng các hàm đặc biệt của Borland một cách hiệu quả khi lập trình với Dev C++.

Thư viện graphic.h bao gồm các hàm để:

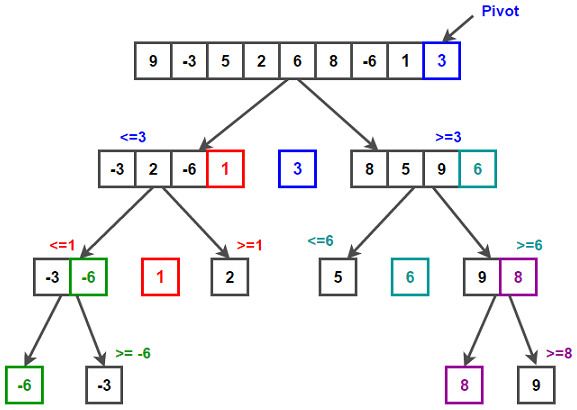
* Khởi tạo và kết thúc cửa sổ đồ họa.
* Vẽ các đường, hình chữ nhật, hình tròn, v.v.
* Thêm văn bản vào cửa sổ đồ họa.
* Kiểm soát màu sắc và độ sáng của đồ họa.

## **Thuật toán Quick Sort (Sắp xếp nhanh)**

### **1.3.1. Sơ lược về thuật toán.**

**Thuật toán sắp xếp nhanh**hay còn gọi là **QuickSort Algorithm** là một trong [6 thuật toán sắp xếp thông dụng](https://duongdinh24.com/tag/thuat-toan-sap-xep/" \t "https://duongdinh24.com/thuat-toan-quick-sort/_blank) nhất của khoa học máy tính. Thuật toán sử dụng tư tưởng chia để trị nên còn được ví là part sort và thuộc loại sắp xếp phức tạp. Từ dãy ban đầu, người ta sẽ chia dãy thành hai phần nhỏ theo một quy tắc xác định, từ đó giúp cải thiện tốc độ của việc săp xếp.

Ví dụ minh họa: Sắp xếp tăng dần dãy số 9,-3,5,2,6,8,-6,1,3 bằng thuật toán sắp xếp nhanh.



Hình 1.1. Hình ảnh minh họa sắp xếp nhanh

### **1.3.2. Ý tưởng chính của thuật toán.**

**Quicksort** là thuật toán ứng dụng ý tưởng chia nhỏ để trị. Đầu tiên nó chia mảng đầu vào thành hai mảng con một nửa nhỏ hơn, một nửa lớn hơn dựa vào một phần tử trung gian. Sau đó, nó sắp xếp đệ quy các mảng con để giải quyết bài toán.

Mấu chốt của thuật toán ở việc chia mảng con hay còn gọi là thuật toán phân đoạn. Cách giải quyết có thể tóm tắt như sau:

**Bước 1: Chọn chốt (pivot)**  
Chọn một phần tử trong mảng làm phần tử trung gian để chia đôi mảng gọi là **pivot**. Thông thường ta thường chọn phần tử đầu tiên, phần tử ở giữa mảng hoặc phần tử cuối cùng của mảng để làm pivot. Trong trường hợp trên là chọn phần tử cuối (3) làm pivot.

**Bước 2: Phân chia danh sách**  
Phân chia danh sách thành hai mảng con :

* Các phần tử nhỏ hơn pivot(< 3) nằm ở bên trái(-3,2,-6,1).
* Các phần tử lớn hơn hoặc bằng pivot(>3 hoặc =3) nằm ở bên phải(8,5,9,6).

Sau khi phân chia danh sách di chuyển pivot về đúng vị trí giữa của 2 mảng con.

**Bước 3: Sắp xếp đệ quy**  
Áp dụng thuật toán Quick Sort đệ quy lên hai phần danh sách bên trái và bên phải (trừ pivot).

**Bước 4: Kết hợp danh sách**  
Sau khi hai phần danh sách con đã được sắp xếp, ghép chúng lại với pivot ở giữa để hoàn thành danh sách sắp xếp.

**Kết quả:**  
Danh sách được sắp xếp theo thứ tự tăng dần sau khi thuật toán hoàn thành

(-6,-3,1,2,5,6,8,9).

### **1.3.3. Độ phức tạp của thuật toán Quick Sort.**

Độ phức tạp của thuật toán Quick Sort phụ thuộc vào cách chọn **pivot** và tình trạng của dãy ban đầu. Chúng ta sẽ phân tích từng trường hợp cụ thể như sau:

Trường hợp tốt nhất:

Pivot chia danh sách thành hai phần bằng nhau (hoặc gần bằng nhau) sau mỗi lần phân chia.

Khi đó, tại mỗi bước, danh sách được chia làm đôi, và số tầng đệ quy sẽ là log2(n).

Tại mỗi tầng, tổng số phép so sánh cần thực hiện để phân chia là n.

**Tổng độ phức tạp:** T(n)=n.log2(n).

**Độ phức tạp tốt nhất:** O(nlogn).

Trường hợp xấu nhất:

Pivot được chọn không tối ưu, khiến một phần luôn chứa toàn bộ các phần tử (tức là danh sách không được chia đều). Ví dụ:

* + Danh sách đã có thứ tự tăng dần/giảm dần, nhưng pivot luôn là phần tử đầu tiên hoặc cuối cùng.
  + Khi đó, tại mỗi tầng, chỉ một phần tử bị loại, và độ sâu của đệ quy là nnn.
  + Tổng số phép so sánh cần thực hiện là: T(n)=n+(n−1)+(n−2)+…+1=n(n+1)2T(n) = n + (n-1) + (n-2) + … + 1 = n(n+1)/2.

Tổng độ phức tạp:T(n) = O(n^2).

Độ phức tạp xấu nhất: O(n^2).

Trường hợp trung bình:

* Với các dãy ngẫu nhiên và pivot được chọn ngẫu nhiên hoặc gần giữa, danh sách sẽ chia thành hai phần tương đối cân bằng ở mỗi bước.
* Số tầng đệ quy trung bình là log2(n), và tổng số phép so sánh tại mỗi tầng là n.

**Tổng độ phức tạp:** T(n)=n.log2(n).

**Độ phức tạp trung bình:** O(nlogn).

### **1.3.4. So sánh độ phức tạp về thời gian của các thuật toán.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuật toán sắp xếp** | **Thời gian thực hiện** | | |
| **Nhanh nhất** | **Trung bình** | **Chậm nhất** |
| Bubble Sort | O(n) | O(n2) | O(n2) |
| Selection Sort | O(n2) | O(n2) | O(n2) |
| Insertion Sort | O(n) | O(n2) | O(n2) |

**Ưu điểm và nhược điểm**

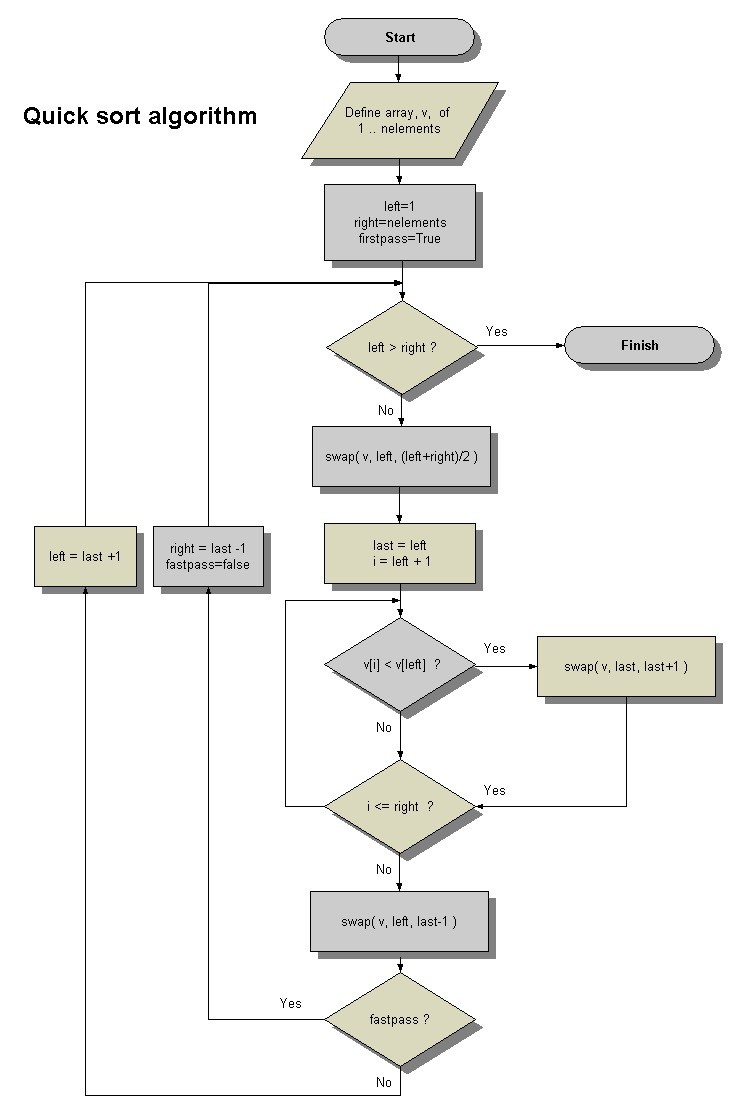
#### **Ưu điểm:**

* **Hiệu suất cao:** Quick Sort có hiệu suất cao, đặc biệt là với dữ liệu lớn. Với độ phức tạp trung bình là O(nlogn), thuật toán thường nhanh hơn so với nhiều thuật toán sắp xếp khác như Insertion Sort hay Bubble Sort.
* **Thích hợp cho dữ liệu ngẫu nhiên:** Quick Sort hoạt động rất tốt trên các tập dữ liệu ngẫu nhiên, nhờ khả năng phân hoạch và chia nhỏ hiệu quả.
* **Tiết kiệm bộ nhớ:** Phiên bản sử dụng chia đổi tại chỗ (in-place) không yêu cầu bộ nhớ phụ lớn như Merge Sort, làm giảm chi phí bộ nhớ.
* **Ứng dụng rộng rãi:** Được sử dụng rộng rãi trong thực tế nhờ tính hiệu quả và khả năng áp dụng linh hoạt.

#### ****Nhược điểm:****

* **Trường hợp xấu nhất:** Nếu pivot được chọn không tốt (như luôn chọn phần tử đầu tiên hoặc cuối cùng), độ phức tạp có thể lên đến O(n^2), làm giảm hiệu suất.
* **Không ổn định:** Quick Sort không giữ được sự ổn định của các phần tử có cùng giá trị. Vị trí của chúng có thể thay đổi sau quá trình sắp xếp.
* **Đòi hỏi kỹ thuật chọn pivot:** Cần các chiến lược chọn pivot hợp lý (như chọn pivot ngẫu nhiên hoặc phần tử giữa) để tránh trường hợp xấu.
* **Khó triển khai:** So với các thuật toán đơn giản như InsertionSort, việc triển khai Quick Sort phức tạp hơn, đặc biệt là trong việc quản lý đệ quy và phân hoạch.

### **1.3.5. Lưu đồ thuật toán Quick Sort.**



Hình 1.2. Hình ảnh lưu đồ thuật toán Quick Sort.

Ngưồn: http://www.hypergraphics.co.uk/SCDiagrams.html

**Mô tả lưu đồ:**

**1. Khởi tạo**

* **Start**: Bắt đầu thuật toán.
* Khai báo mảng v gồm các phần tử từ 1..nelements.
* Gán các biến:
  + left = 1 (điểm bắt đầu bên trái),
  + right = nelements (điểm bắt đầu bên phải),
  + firstpass = True (đánh dấu lần lặp đầu tiên).

**2. Kiểm tra điều kiện dừng**

* **left > right?**: Nếu vị trí trái đã vượt qua vị trí phải, thuật toán dừng (**Finish**).
* Nếu không, tiếp tục.

**3. Hoán đổi Pivot**

* **swap(v, left, (left+right)/2)**: Hoán đổi phần tử tại vị trí left với phần tử tại vị trí giữa của left và right.  
  Mục tiêu: Lấy phần tử pivot để phân chia mảng.

**4. Phân chia vùng**

* Gán:
  + last = left (vị trí cuối của vùng bên trái),
  + i = left + 1 (chỉ số để duyệt mảng từ trái sang phải).
* Kiểm tra từng phần tử để phân chia vào vùng left (nhỏ hơn pivot) hoặc right (lớn hơn hoặc bằng pivot).

**5. Kiểm tra và hoán đổi**

* **v[i] < v[left]?**
  + **Yes**: Hoán đổi v[last+1] với v[i], sau đó tăng last.
  + **No**: Tiếp tục duyệt.
* **i <= right?**
  + **Yes**: Tăng i và quay lại kiểm tra.
  + **No**: Đã duyệt xong mảng.

**6. Hoán đổi Pivot về vị trí cuối**

* **swap(v, left, last)**: Đưa phần tử pivot về đúng vị trí phân chia, nằm giữa vùng left và right.

**7. Kiểm tra lần lặp đầu tiên**

* **fastpass?**
  + **Yes**: Nếu đây là lần đầu phân chia, quay lại bước lặp tiếp theo.
  + **No**: Tiếp tục chia đệ quy hai nửa mảng.

**8. Kết thúc**

* Thuật toán kết thúc khi không còn phần tử nào để phân chia, mọi phần tử đã được sắp xếp.

# **CHƯƠNG 2. PHƯƠNG PHÁP NGHIÊN CỨU**

## **2.1. Tổng quan về giao diện**

### **2.1.1. Cài đặt Dev-C++ và thư viện graphic.h**

Tải file cài đặt phần mềm Dev-C++ theo đường dẫn (<https://sourceforge.net/projects/orwelldevcpp/>). Sau đó mở file vừa tải, và tiến hành cài đặt. Thư viện graphics.h được tiến hành cài đặt theo các bước:

**Bước 1:** Copy 6-ConsoleAppGraphics và ConsoleApp\_cpp\_graph

Paste C:\...\Dev-Cpp\Templates

**Bước 2:** Copy graphics và winbgim

Paste C:\... Dev-Cpp \MinGW64\x86\_64-w64-mingw32\include

**Bước 3:** Copy libbgi.a

Paste C:\...\Dev-Cpp\MinGW64\x86\_64-w64-mingw32\lib

**Bước 4:** Ở DevC++ => New Project => Console Graphics Application

**Bước 5:** Thay đổi Tools - Compiler Option: TDM – GCC 4.9.2 32 bit Release

**Bước 6:** Sử dụng đoạn code mẫu bên dưới để test thư viện winbgim

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Hình 2.1. Hình ảnh minh họa graphic.h

### **2.1.2. Xử lý sự kiện chuột với thư viện BGI**

Thư viện BGI nhận các loại sự kiện chuột như:

* WM\_MOUSEMOVE: bắt sự kiện khi di chuyển chuột trong màn hình đồ họa.
* WM\_LBUTTONDBLCLK: bắt sự kiện khi nhấn đúp chuột trái.
* WM\_LBUTTONDOWN: bắt sự kiện khi nhấn chuột trái.
* WM\_LBUTTONUP: bắt sự kiện khi thả sau khi nhấn chuột trái.

Tương tự với chuột giữa và phải.

* WM\_MBUTTONDBLCLK
* WM\_MBUTTONDOWN
* WM\_MBUTTONUP
* WM\_RBUTTONDBLCLK
* WM\_RBUTTONDOWN
* WM\_RBUTTONUP

Các hàm bắt sự kiện click chuột:

void getmouseclick( int kind, int& x, int& y );

-> lấy tọa độ x,y ( theo pixel) mà ở đó có click

bool ismouseclick( int kind );

-> trả về true nếu xảy ra click

Thư viện BGI cung cấp sẵn 1 số phương thức để có thể thao tác với chuột và bàn phím. Đơn giản nhất như bộ đôi: int x\_mouse; int y\_mouse;

Cả hai phương thức này trả về tọa độ của chuột trên khung cửa sổ ứng dụng tại thời điểm hàm này được gọi. Khi vị trí của chuột nằm ngoài khung ứng dụng thì việc gọi hàm này sẽ xảy ra 2 trường hợp:

* Nếu chuột chưa từng di chuyển vào phạm vi cửa sổ ứng dụng giá trị trả về là 0.
* Nếu chuột đã từng ở trong phạm vi cửa sổ ứng dụng và sau đó di chuyển ra ngoài, giá trị trả về là tọa độ lần cuối cùng của chuột trong cửa sổ ứng dụng.

Đoạn chương trình sau mô tả cách lấy tọa độ khi nhấn chuột trái, “Exiting…” khi nhấn chuột phải và in ra màn hình:

void mouse(){

int x\_mouse; int y\_mouse;

while(enable\_click){

//get even mouse click

if(ismouseclick(WM\_LBUTTONDOWN)){

//get position of mouse

getmouseclick(WM\_LBUTTONDOWN,x\_mouse, y\_mouse);

//printf x,y mouse to console

printf("(%d,%d)",x\_mouse,y\_mouse);

}

//check right mouse click to exit loop

if(ismouseclick(WM\_RBUTTONDOWN)){

getmouseclick(WM\_RBUTTONDOWN,x\_mouse, y\_mouse);

printf("\nExiting...");

break;

}

delay(100);//wait for next cycle mouse click

}

}

A screenshot of a computer

Description automatically generated

Hình 2.2. Hình ảnh lấy tọa độ khi sử dụng sự kiện chuột.

### **2.1.3. Xây dựng giao diện**

Giao diện được dùng để mô tả các thông tin của đề tài như: tên giáo viên hướng dẫn, tên sinh viên thực hiện, mã số sinh viên, lớp… mảng trước khi được sắp xếp, mảng sau khi sắp xếp. Các Button dùng để tạo mảng, nhập số liệu mảng, đọc dữ liệu từ file, tạo mảng ngẫu nhiên, chọn cách sắp xếp tăng hay giảm dần, nút làm mới lại mảng và nút thoát chương trình.



Hình 2.3. Hình ảnh giao diện lúc mới mở.

**Xây dựng khung**

setcolor(9);

setlinestyle(0,1,3);

rectangle(20,20,975,600);

Để xây dựng được khung như trên giao diện, đầu tiên ta sử dụng setcolor(9) để thiết lập màu vẽ trong thư viện đồ họa cụ thể là màu xanh dương nhạt. Tiếp theo ta thiết lập kiểu đường vẽ setlinestyle(0, 1, 3). Cụ thể, đối số đầu tiên (0) là kiểu đường vẽ nét liền, đối số thứ hai là một biểu diễn nhị phân của mẫu đường vẽ và sử dụng số 1 để vẽ đường đơn liền, và đối số cuối cùng dùng để vẽ độ dày của đường vẽ là 3. Cuối cùng để có được một hình chữ nhật hoàn chỉnh ta sử dụng hàm rectangle(20, 20, 975, 600). Hình chữ nhật từ điểm có tọa độ góc trái trên là (20, 20) và tọa độ góc phải dưới là (975, 600). Hình chữ nhật này có chiều rộng là 955 pixel và chiều cao là 580 pixel.

**Xây dựng Button**



Hình 2.4. Hình ảnh code xây dựng Button.

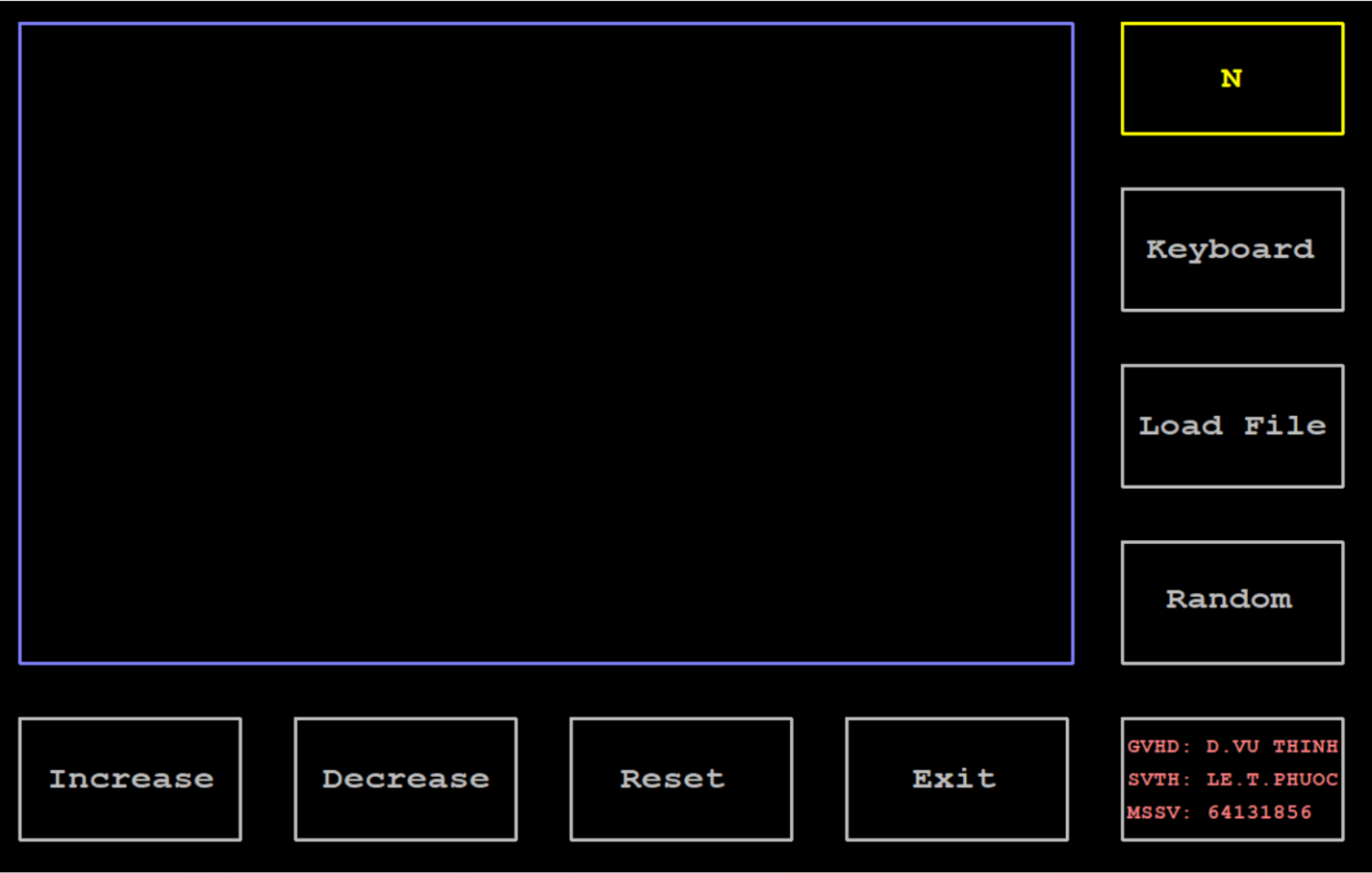
* Button “N”:
* Sử dụng rectangle(1020,20,1220,120) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (1020, 20) và tọa độ góc phải dưới là (1220, 120). Nút “N” có chiều rộng là 200 pixel và chiều cao là 100 pixel.
* Sử dụng settextstyle(10,0,4); outtextxy(1110,55,"N") để tạo chữ "N" tại tọa độ (1110, 55) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Keyboard”:
* Sử dụng rectangle(1020,170,1220,280) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (1020, 170) và tọa độ góc phải dưới là (1220, 280). Nút “Keyboard” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(1042,210,"Keyboard") để tạo chữ "Keyboard" tại tọa độ (1042, 210) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Load File”:
* Sử dụng rectangle(1020,330,1220,440) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (1020, 330) và tọa độ góc phải dưới là (1220, 440). Nút “Load File” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(1035,370,"Load File") để tạo chữ "Load File" tại tọa độ (1035, 350) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Random”:
* Sử dụng rectangle(1020,490,1220,600) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (1020, 490) và tọa độ góc phải dưới là (1220, 600). Nút “Random” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(1060,527,"Random") để tạo chữ "Random" tại tọa độ (1060, 527) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Increase”:
* Sử dụng rectangle(20,650,220,760) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (20, 650) và tọa độ góc phải dưới là (220, 760). Nút “Increase” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(45,690,"Increase") để tạo chữ "Increase" tại tọa độ (45,690) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Decrease”:
* Sử dụng rectangle(270,650,470,760) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (270, 650) và tọa độ góc phải dưới là (470, 760). Nút “Decrease” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(295,690,"Decrease") để tạo chữ "Decrease" tại tọa độ (295,690) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Reset”:
* Sử dụng rectangle(520,650,720,760) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (520, 650) và tọa độ góc phải dưới là (720, 760). Nút “Reset” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(565,690,"Reset") để tạo chữ "Reset" tại tọa độ (565,690) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.
* Button “Exit”:
* Sử dụng rectangle(770,650,970,760) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (770, 650) và tọa độ góc phải dưới là (970, 760). Nút “Exit” có chiều rộng là 200 pixel và chiều cao là 110 pixel.
* Sử dụng settextstyle(10,0,4), outtextxy(830,690,"Exit") để tạo chữ "Exit" tại tọa độ (830,690) với kiểu chữ số 10, không in đậm (0), và kích thước là 4.

**Xây dựng màu Button**



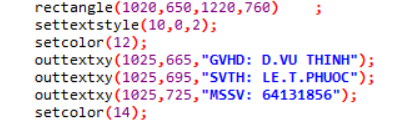
Hình 2.5. Hình ảnh code xây dựng màu button

Sử dụng setcolor(14) để chuyển thành màu vàng cho các button được click và setcolor(7) để chuyển các button thành màu xám, ngoài trừ khung main border vẫn giữ nguyên màu xanh dương nhạt(setcolor(9)) và màu chữ trong khung name vẫn giữ nguyên màu đỏ nhạt(setcolor(12)). Ví dụ khi click vào Button “N” để nhập số lượng phần tử thì button “N” sẽ chuyển sang màu vàng và các button còn lại sẽ có màu xám nhạt ,ngoại trừ khung main border vẫn giữ nguyên màu xanh dương nhạt và màu chữ trong khung name vẫn giữ nguyên màu đỏ nhạt. Tương tự với khi click với các button khác.



Hình 2.6. Hình ảnh giao diện màu button.

**Xây dựng text**



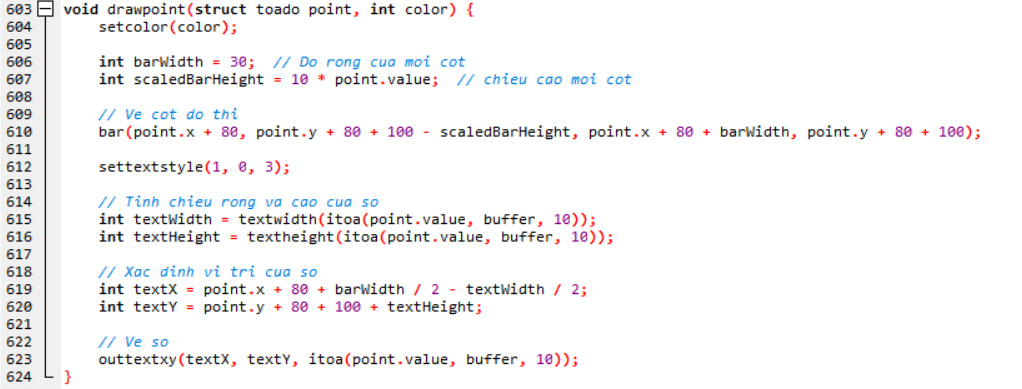
Hình 2.7. Hình ảnh code xây dựng text.

Sử dụng rectangle(1020,650,1220,760) để vẽ hình chữ nhật từ điểm có tọa độ góc trái trên là (1020,650) và tọa độ góc phải dưới là (1220,760). Hình chữ nhật này có chiều rộng là 200 pixel và chiều cao là 110 pixel.

Sử dụng settextstyle(10,0,2) để tạo văn bản với kiểu chữ số 10, không in đậm (0), và kích thước là 2.

* outtextxy(1025,665,"GVHD: D.VU THINH") để tạo “GVHD: D.VU THINH” tại tọa độ (1025, 665).
* outtextxy(1025,695,"SVTH: LE.T.PHUOC") để tạo “SVTH: LE.T.PHUOC” tại tọa độ (1025, 695).
* outtextxy(1025,725," MSSV: 64131856") để tạo “MSSV: 64131856” tại tọa độ (1020, 725).

**Xây dựng thanh bar**



Hình 2.8. Hình ảnh code xây dựng thanh bar.

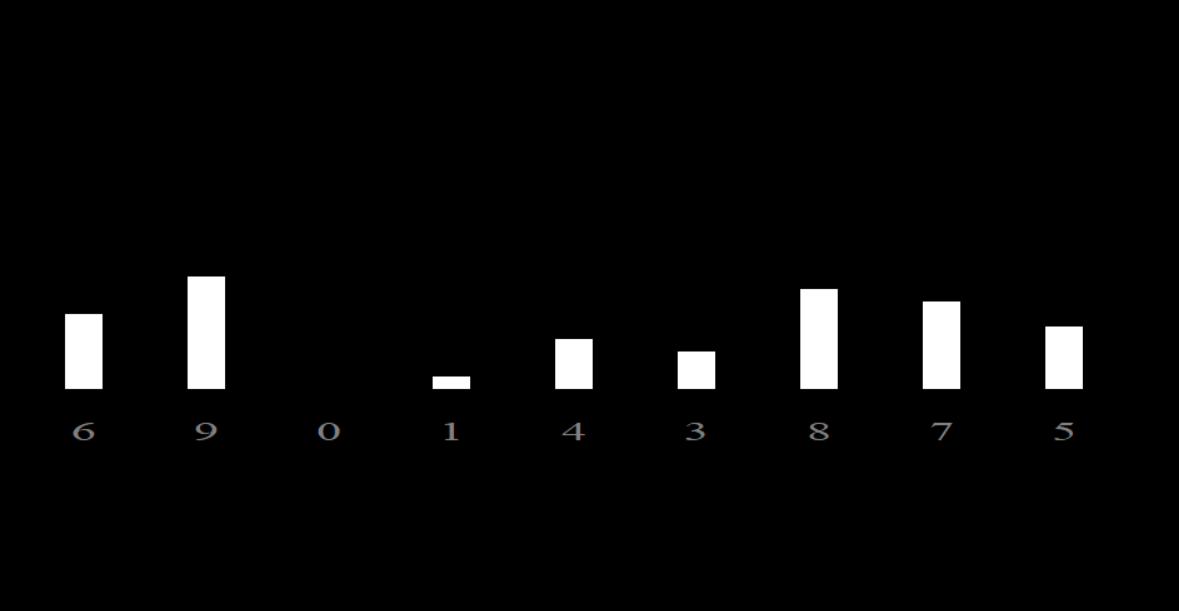
Sử dụng int barWidth = 30 để đặt độ rộng của mỗi cột đồ thị là 30 pixel, int scaledBarHeight = 10 \* point.value để đặt chiều cao của cột đồ thị dựa trên giá trị của điểm được đưa vào, ở đây nhân với một hệ số 10 vì muốn phóng đại chiều cao của cột lên 10 lần. Chiều cao của cột sẽ tỷ lệ thuận với giá trị của point.value.

Sử dụng hàm bar (point.x + 80, point.y + 80 + 100 - scaledBarHeight, point.x + 80 + barWidth, point.y + 80 + 100) để vẽ một hình chữ nhật, tạo thành một cột đồ thị. Các tham số của bar là tọa độ của hai đỉnh đối diện của hình chữ nhật: (x1, y1, x2, y2). Ở đây, x1 và y1 là tọa độ của góc trên bên trái của hình chữ nhật, x2 và y2 là tọa độ của góc dưới bên phải của hình chữ nhật. Các giá trị này được tính toán dựa trên tọa độ của điểm (point.x và point.y), chiều rộng của cột, và chiều cao của cột.

Sử dụng settextstyle(1,0,3) để tạo số với kiểu chữ số 1, không in đậm (0), và kích thước là 3.

Tính toán chiều rộng và chiều cao của số dựa trên giá trị của điểm, sử dụng hàm itoa để chuyển giá trị của điểm thành chuỗi. Để số được đặt ở giữa cột đồ thị, tọa độ x được tính bằng công thức point.x + 80 + barWidth / 2 - textWidth / 2 và tọa độ y là point.y + 80 + 100 + textHeight.

Để vẽ số tại vị trí và với giá trị đã tính toán. Đối số cuối cùng là giá trị số, được chuyển đổi từ giá trị của điểm sang chuỗi bằng itoa bằng cách sửu dụng outtextxy(textX, textY, itoa(point.value, buffer, 10.



Hình 2.9. Hình ảnh thanh bar

## **2.2. Phân tích thuật toán**

***Khai báo thư viện***

#include<stdio.h>

#include<winbgim.h>

#include<stdlib.h>

#define MAX 100

#define INPUT "input.txt"

#include <cstdlib>

#include <ctime>

***Khai báo biến toàn cục và hàm***

*//bien toan cuc*

int sodinh;*//so phan tu can sap xep*

FILE \*fp;

char buffer[10];

int enable\_click =1;

int sorted = 0;*//co kiem tra xem co sap xep hay chua*

struct toado{

int x;

int y;

int value;

*// Toán toan so sánh dua trên giá tri value*

bool operator<=(const toado& other) const {

return value <= other.value;

}

};

*//khai bao ham*

void readfile();

void printpoints(struct toado m[MAX], int size);

void GUI\_init();

void GUI\_inputN();

void GUI\_Random();

void GUI\_Loadfile();

void GUI\_Keyboard();

void mouse();

void InputN();

void Keyboard();

void Random();

void Reset();

void Exit();

void drawpoint(struct toado point,int color);

void drawpoints(struct toado points[MAX],int size);

void GUI\_increase();

void GUI\_decrease();

int partitionIncrease(struct toado arr[MAX], int left, int right);

void quickSortIncrease(struct toado arr[MAX], int left, int right);

void IncreaseSort(struct toado arr[MAX], int n);

int partitionDecrease(struct toado arr[MAX], int left, int right);

void quickSortDecrease(struct toado arr[MAX], int left, int right);

void DecreaseSort(struct toado arr[MAX], int n);

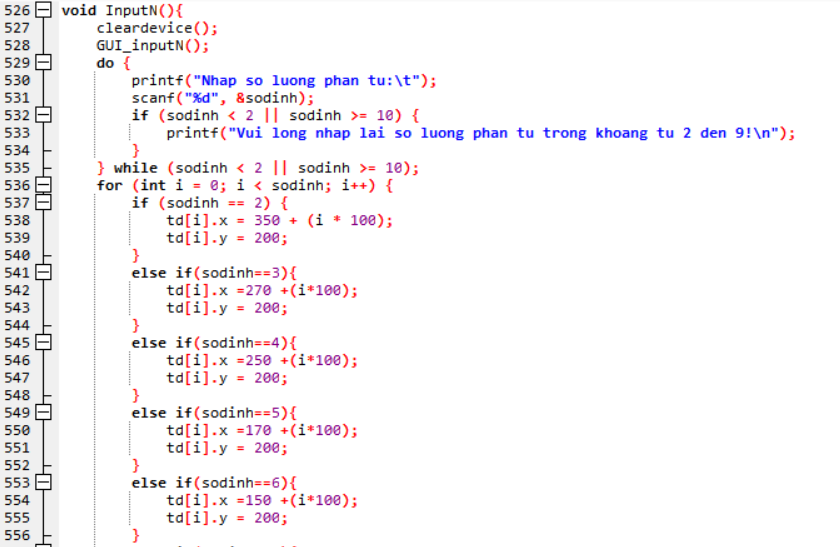
void Increase();

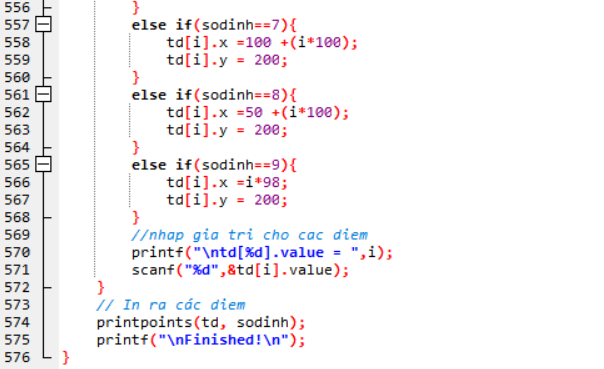
void Decrease();

### **2.2.1. Khởi tạo mảng**

**Nhập số lượng phần tử từ bàn phím**

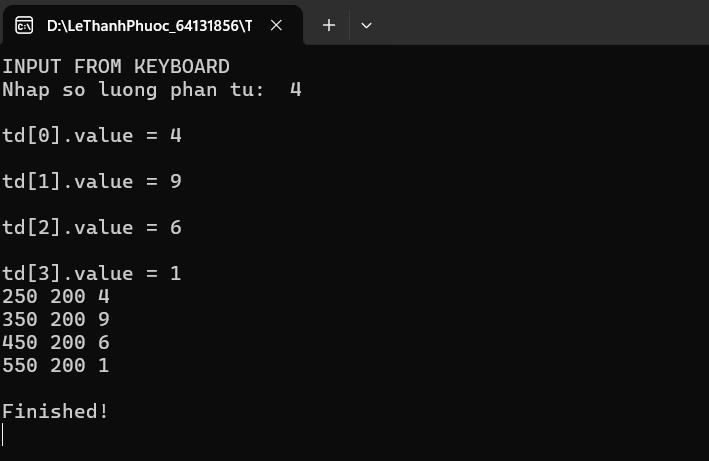
Người dùng sẽ nhập số lượng phần tử trong khoảng từ 2 đến 9 và giá trị phần tử mà họ muốn sắp xếp. Tọa độ của các phần tử sẽ được cố định sẵn trong code. Sau khi nhập xong số lượng phần tử, tọa độ và giá trị của các phần tử sẽ được in ra.





Hình 2.10. Hình ảnh code nhập số lượng phần tử từ bàn phím.

Ví dụ: Nhập số lượng phần tử là 4 và giá trị lần lượt là 4,9,6,1.



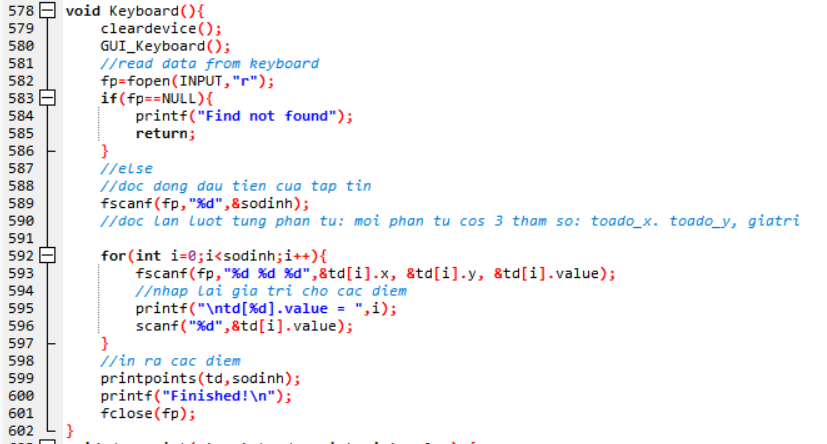
Hình 2.11. Hình ảnh nhập số lượng phần tử từ bàn phím.



Hình 2.12. Hình ảnh giao diện nhập số lượng phần tử từ bàn phím.

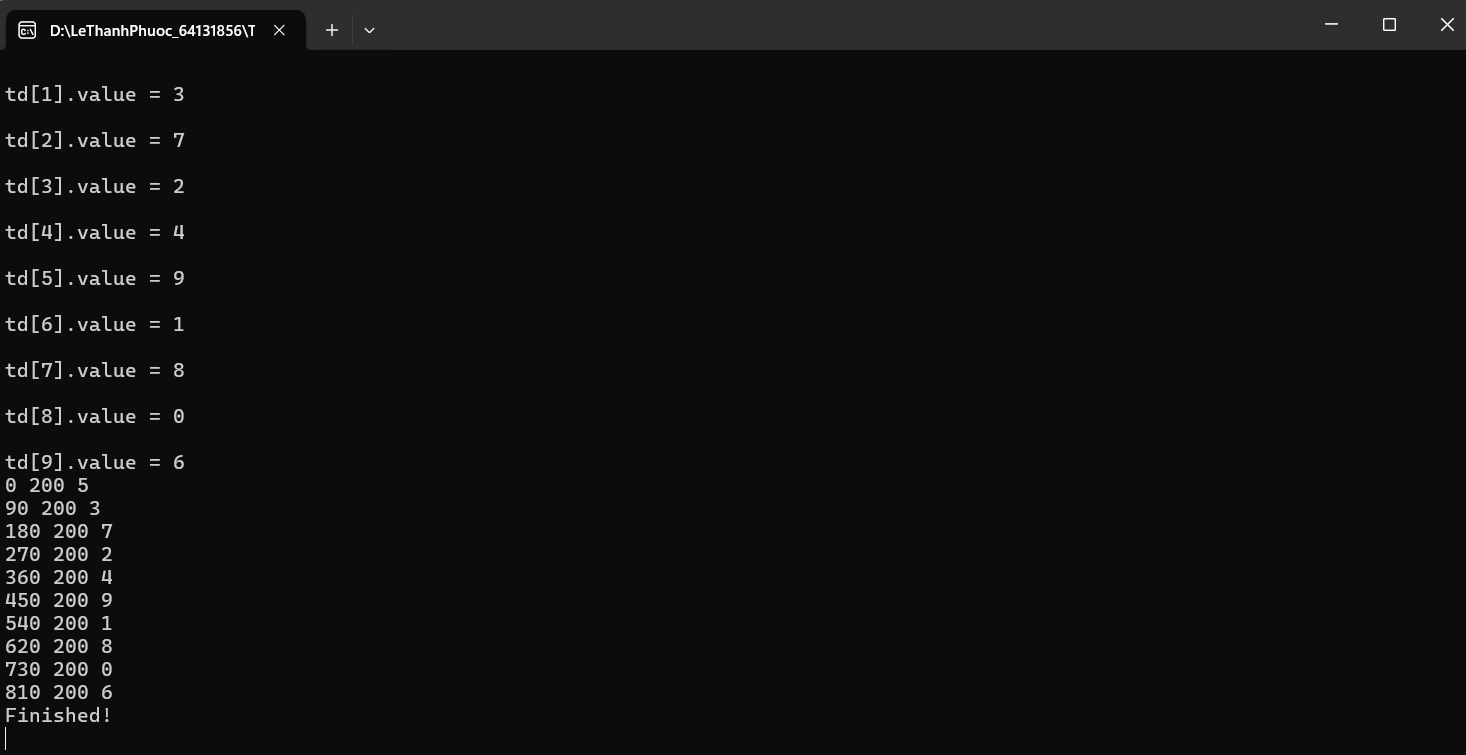
**Nhập phần tử từ bàn phím với số lượng mặc định**

Chúng ta sẽ cố định thuật toán Quick Sort sắp xếp 10 phần tử. Tọa độ của các phần tử được lấy từ file và giá trị của 10 phần tử sẽ được nhập từ bàn phím. Sau khi nhập xong tọa độ và giá trị của các phần tử sẽ được in ra.



Hình 2.13. Hình ảnh code nhập phần tử từ bàn phím với số lượng mặc định.

Ví dụ: Nhập giá trị cho 10 phần tử lần lượt là 5, 3, 7, 2, 4, 9, 1, 8, 0, 6.



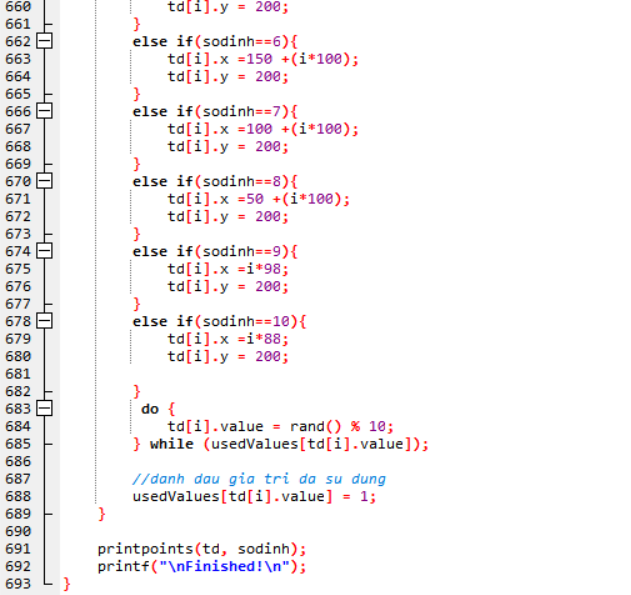
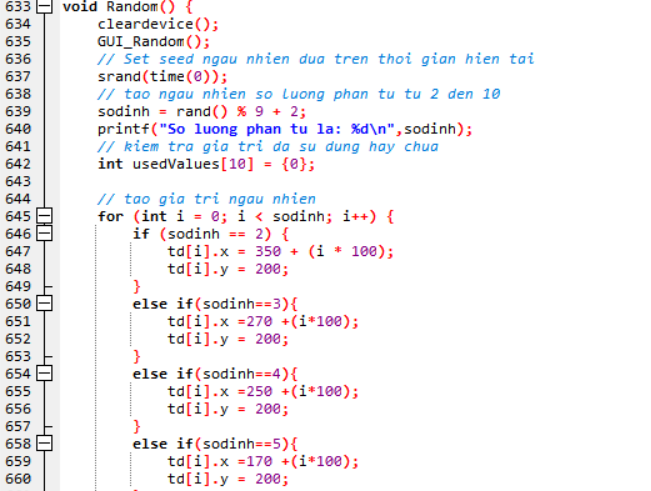
Hình 2.14. Hình ảnh nhập phần tử từ bàn phím với số lượng mặc định.



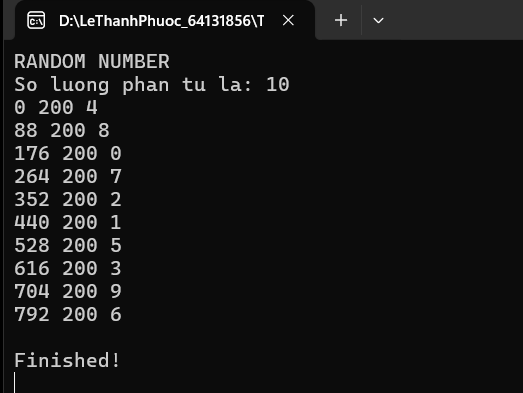
Hình 2.15. Hình ảnh giao diện nhập phần tử từ bàn phím với số lượng mặc định.

**Tạo số lượng phần tử và giá trị ngẫu nhiên**

Ta sử dụng hàm random() hai lần, một lần để tạo số lượng ngẫu nhiên và một hàm để tạo ra các giá trị ngẫu nhiên không trùng nhau. Sau mỗi lần click button “Random”, giao diện sẽ được xóa và vẽ lại để tránh trường hợp các số bị chồng lên nhau. Tọa độ của các số được cài đặt trong code. Sau khi tạo ngẫu nhiên, số lượng phần tử, tọa độ và giá trị sẽ được in ra.



Hình 2.16. Hình ảnh code tạo số lượng phần tử và giá trị ngẫu nhiên.

Ví dụ: khi click vào nút Random các phần tử ngẫu nhiên sẽ được tạo ra.  


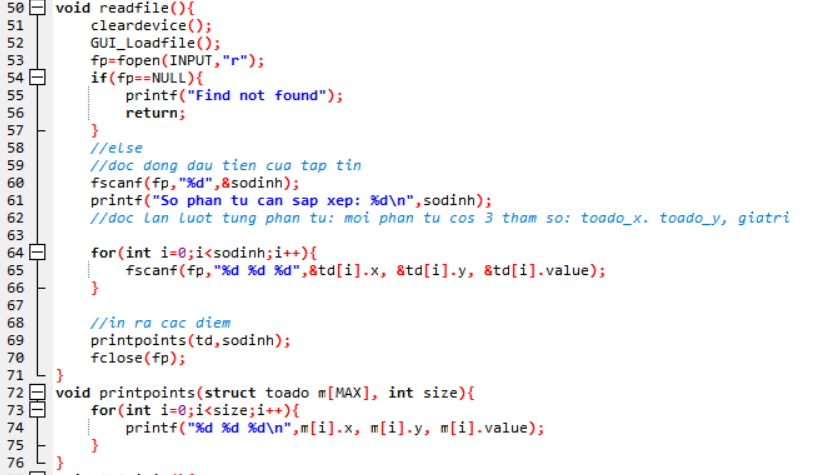
Hình 2.17. Hình ảnh tạo số lượng phần tử và giá trị ngẫu nhiên.



Hình 2.18. Hình ảnh giao diện tạo số lượng phần tử và giá trị ngẫu nhiên.

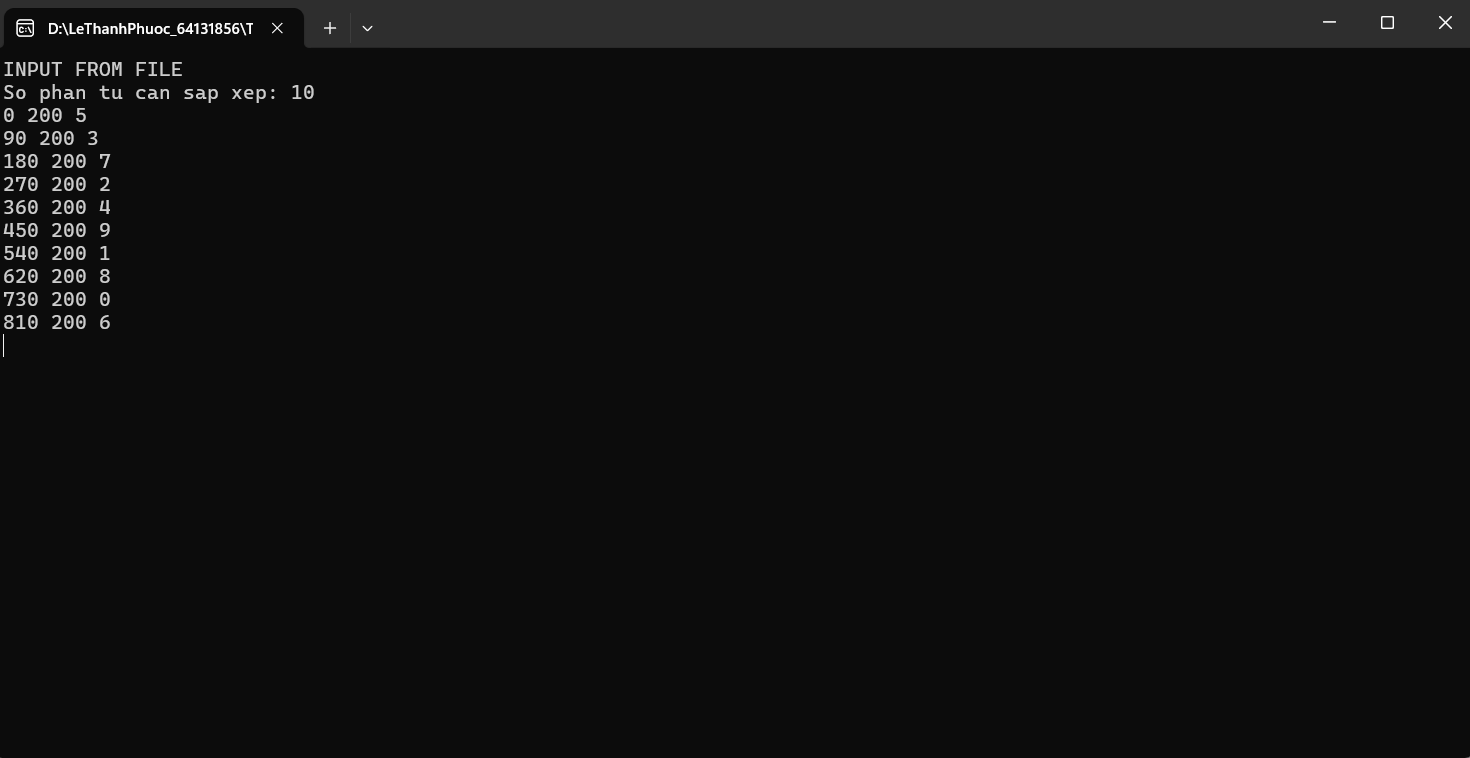
**Tạo các phần tử bằng cách đọc file**

Chúng ta sẽ cố định thuật toán Quick Sort sắp xếp 10 phần tử trong file. Tọa độ của các phần tử và giá trị của chúng sẽ được lấy từ file. Sau khi lấy từ file sso lượng phần tử, tọa độ và giá trị của các phần tử sẽ được in ra.

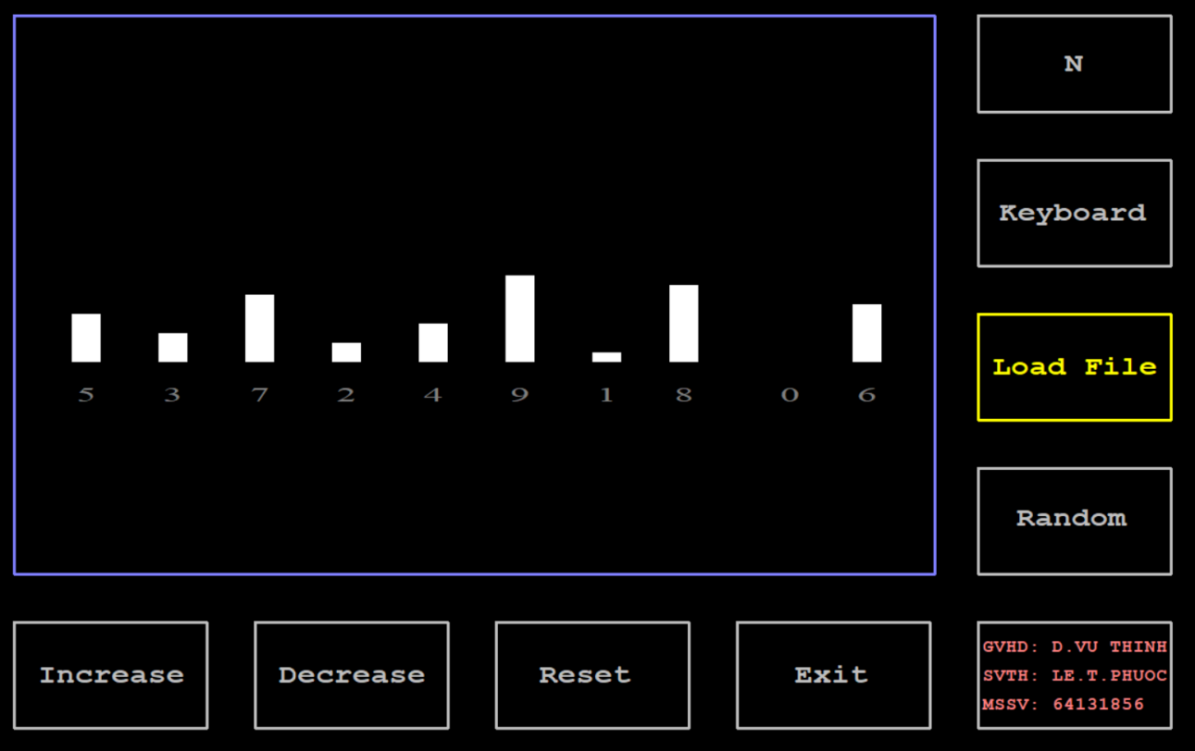


Hình 2.19. Hình ảnh code tạo các phần tử bằng cách đọc file.

Ví dụ: tạo file chứa 10 phần tử có giá trị là 5,3,7,2,4,9,1,8,0,6.



Hình 2.20. Hình ảnh tạo các phần tử bằng cách đọc file.

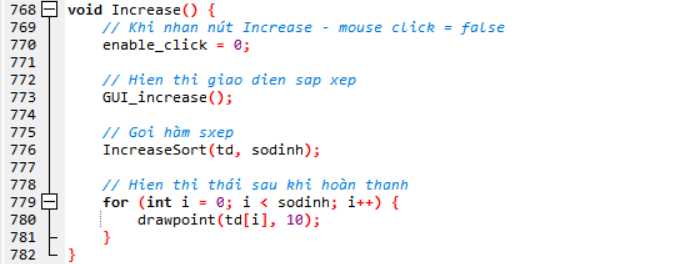
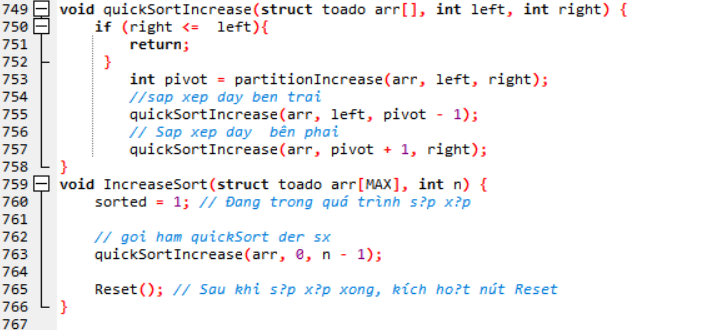
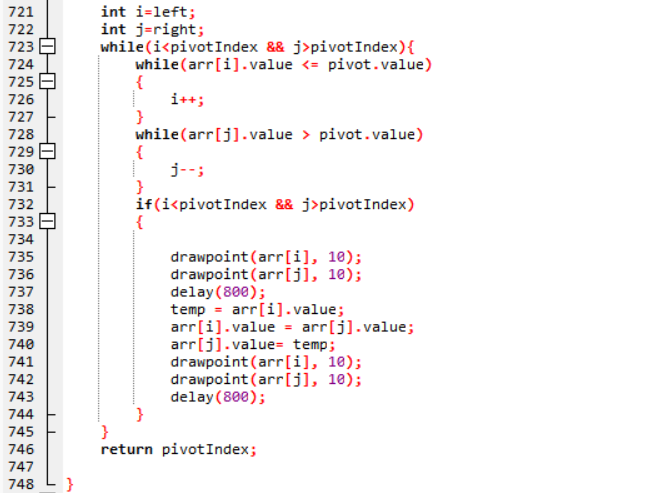
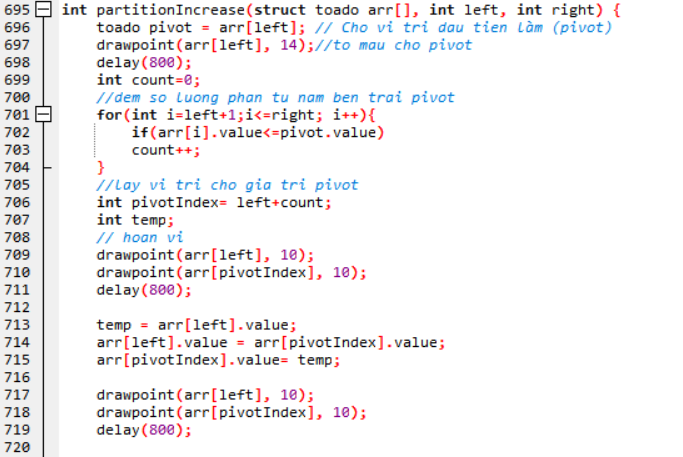


Hình 2.21. Hình ảnh giao diện tạo các phần tử bằng cách đọc file.

### **2.2.2. Thuật toán Quick Sort**

**Sắp xếp tăng dần**

Thuật toán QuickSort tăng dần sử dụng phương pháp chia để trị, trong đó mảng được sắp xếp bằng cách chọn một phần tử làm điểm trục (**pivot**) để chia mảng thành hai phần: các phần tử nhỏ hơn hoặc bằng **pivot** nằm bên trái và các phần tử lớn hơn nằm bên phải. Đầu tiên, thuật toán chọn phần tử đầu tiên của mảng làm **pivot**, sau đó đếm số lượng phần tử nhỏ hơn hoặc bằng **pivot** để xác định vị trí chính xác của nó sau khi sắp xếp. **Pivot** được hoán đổi với phần tử ở vị trí đã tính toán để đảm bảo nó nằm đúng chỗ. Tiếp theo, hai con trỏ bắt đầu từ hai đầu mảng, lần lượt tìm và hoán đổi các cặp phần tử không đúng vị trí sao cho phần tử nhỏ hơn **pivot** nằm bên trái và phần tử lớn hơn nằm bên phải. Khi quá trình này hoàn tất, thuật toán trả về vị trí cuối cùng của **pivot** và áp dụng đệ quy trên hai mảng con bên trái và bên phải. Quá trình này tiếp diễn cho đến khi toàn bộ mảng được sắp xếp. Trong quá trình sắp xếp, các hàm trực quan hóa như drawpoint và delay được sử dụng để đánh dấu các phần tử đang xử lý và hiển thị sự thay đổi của mảng, giúp dễ dàng theo dõi từng bước của thuật toán.



Hình 2.22. Hình ảnh code thuật toán Qick Sort sắp xếp tăng dần.

#### Hàm partitionIncrease

Hàm này thực hiện chia mảng thành hai phần dựa trên giá trị **pivot**, đảm bảo các phần tử nhỏ hơn hoặc bằng **pivot** nằm bên trái, và các phần tử lớn hơn nằm bên phải.

* toado pivot = arr[left];: Chọn phần tử đầu tiên trong đoạn cần xử lý làm **pivot**.
* drawpoint(arr[left], 14); delay(800);: Đánh dấu **pivot** bằng màu xanh lá cây và thêm độ trễ để hiển thị trực quan.
* int count = 0;: Khởi tạo biến đếm số phần tử nhỏ hơn hoặc bằng **pivot**.
* for (int i = left + 1; i <= right; i++): Duyệt qua các phần tử trong mảng.
  + if (arr[i].value <= pivot.value) count++;: Tăng biến đếm nếu giá trị nhỏ hơn hoặc bằng **pivot**.
* int pivotIndex = left + count;: Tính vị trí cuối cùng của **pivot** sau khi sắp xếp.
* Đoạn hoán vị:
  + Đánh dấu **pivot** và phần tử ở vị trí sẽ đổi chỗ với màu vàng (drawpoint).
  + Hoán đổi giá trị của hai phần tử (temp), cập nhật **pivot** vào đúng vị trí.
  + Xóa màu sau khi hoàn thành hoán đổi.
* Hai con trỏ i và j:
  + while (i < pivotIndex && j > pivotIndex): Duyệt đến khi các con trỏ gặp nhau.
  + Các lệnh if và while bên trong đảm bảo phần tử nhỏ hơn **pivot** ở bên trái, phần tử lớn hơn **pivot** ở bên phải.
  + Hoán đổi các phần tử nếu chúng sai vị trí.
* return pivotIndex;: Trả về vị trí của **pivot** sau khi chia mảng.

#### Hàm quickSortIncrease

Hàm này thực hiện sắp xếp đệ quy mảng theo thứ tự tăng dần.

* if (right <= left) return;: Nếu mảng chỉ có một phần tử hoặc rỗng, không cần sắp xếp.
* int pivot = partitionIncrease(arr, left, right);: Gọi hàm partitionIncrease để chia mảng, đồng thời nhận vị trí **pivot**.
* quickSortIncrease(arr, left, pivot - 1);: Gọi đệ quy để sắp xếp đoạn mảng bên trái **pivot**.
* quickSortIncrease(arr, pivot + 1, right);: Gọi đệ quy để sắp xếp đoạn mảng bên phải **pivot**.

#### Hàm IncreaseSort

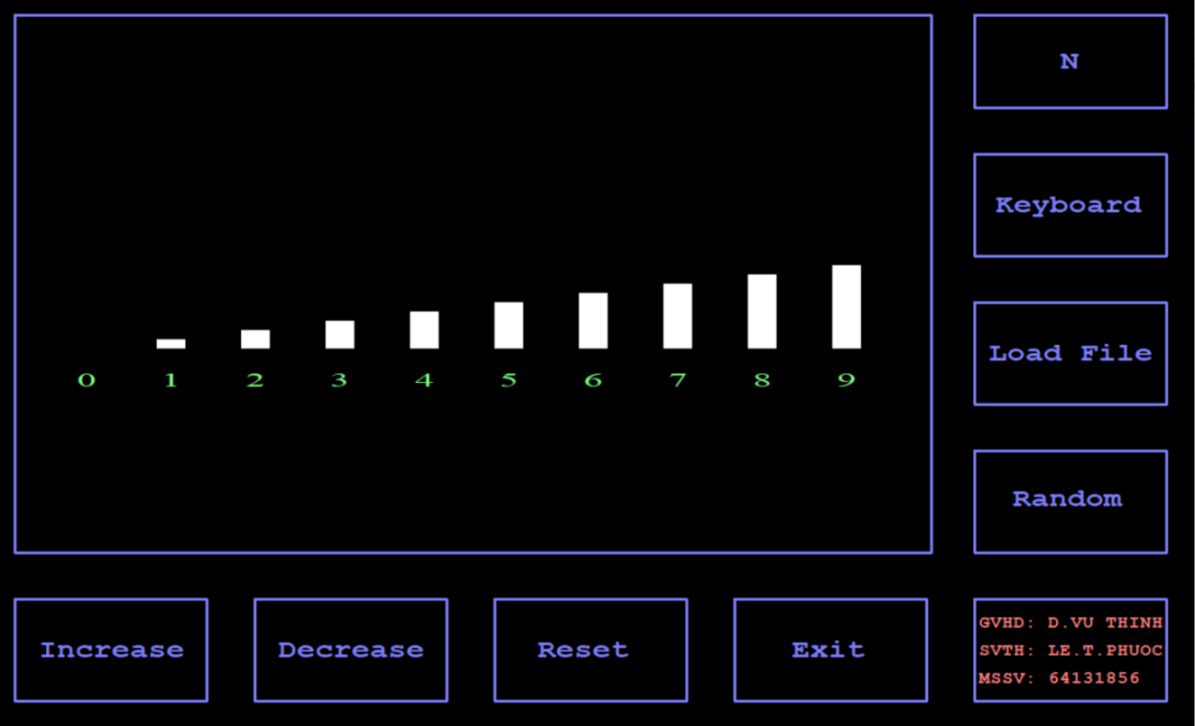
Hàm này đóng vai trò khởi chạy toàn bộ thuật toán QuickSort.

* sorted = 1;: Đặt trạng thái hệ thống thành "đang sắp xếp".
* quickSortIncrease(arr, 0, n - 1);: Gọi thuật toán QuickSort để sắp xếp toàn bộ mảng từ đầu đến cuối.
* Reset();: Sau khi sắp xếp xong, kích hoạt giao diện "đặt lại" (reset).

#### Hàm Increase

Hàm này xử lý sự kiện người dùng nhấn nút để thực hiện sắp xếp.

* enable\_click = 0;: Vô hiệu hóa tương tác chuột trong quá trình sắp xếp.
* GUI\_increase();: Hiển thị giao diện minh họa cho quá trình sắp xếp tăng dần.
* IncreaseSort(td, sodinh);: Gọi hàm IncreaseSort để sắp xếp mảng tọa độ td với sodinh là số phần tử.
* for (int i = 0; i < sodinh; i++): Sau khi hoàn tất, lặp qua toàn bộ mảng để vẽ lại từng phần tử đã được sắp xếp với màu trắng (drawpoint(td[i], 10);).



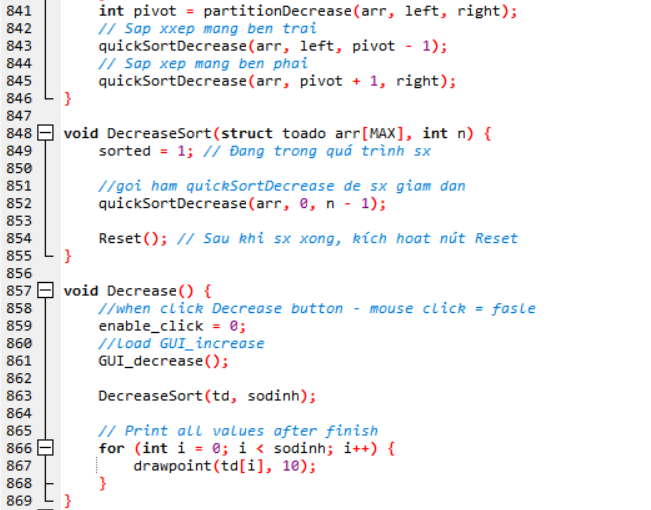
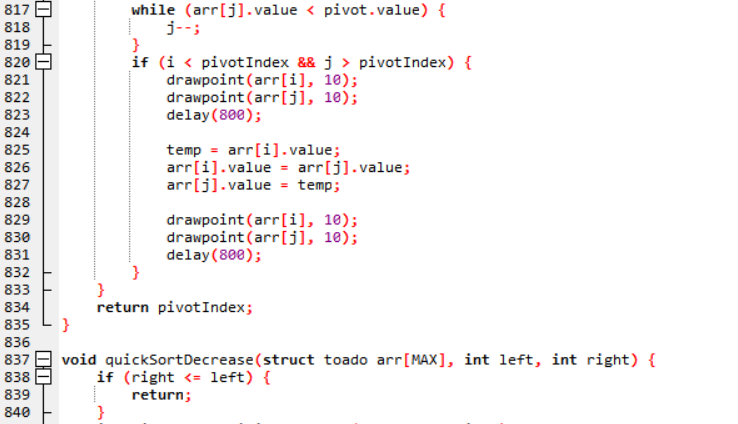
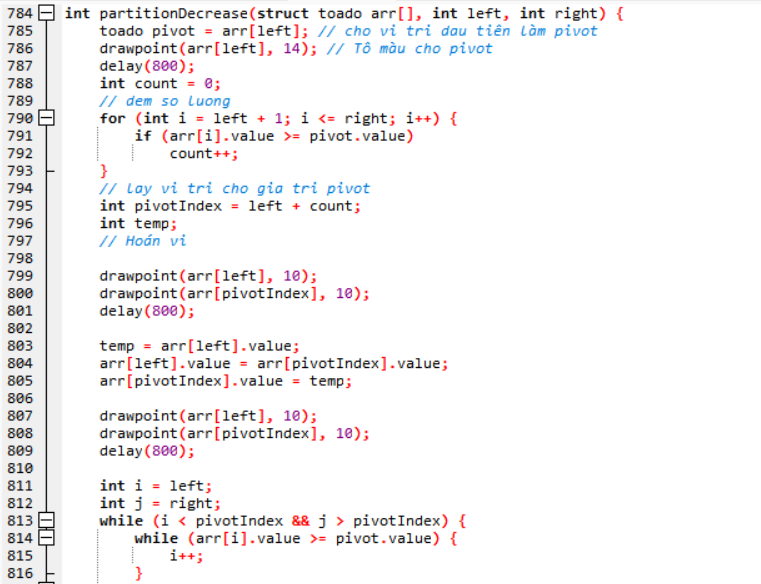
Hình 2.23. Hình ảnh sau khi sắp xếp tăng dần.

**Sắp xếp giảm dần**

Thuật toán QuickSort giảm dần cũng sử dụng phương pháp chia để trị, nhưng mảng được sắp xếp theo thứ tự giảm dần thay vì tăng dần. Trong thuật toán này, một phần tử được chọn làm điểm trục (**pivot**) để chia mảng thành hai phần: các phần tử lớn hơn hoặc bằng **pivot** nằm bên trái và các phần tử nhỏ hơn nằm bên phải. Đầu tiên, thuật toán chọn phần tử đầu tiên của mảng làm **pivot**, sau đó đếm số lượng phần tử lớn hơn hoặc bằng **pivot** để xác định vị trí chính xác của nó sau khi sắp xếp. **Pivot** được hoán đổi với phần tử ở vị trí đã tính toán để đảm bảo nó nằm đúng chỗ.

Tiếp theo, hai con trỏ bắt đầu từ hai đầu mảng, lần lượt tìm và hoán đổi các cặp phần tử không đúng vị trí sao cho phần tử lớn hơn **pivot** nằm bên trái và phần tử nhỏ hơn nằm bên phải. Khi quá trình này hoàn tất, thuật toán trả về vị trí cuối cùng của **pivot** và áp dụng đệ quy trên hai mảng con bên trái và bên phải.

Quá trình này tiếp diễn cho đến khi toàn bộ mảng được sắp xếp theo thứ tự giảm dần. Trong quá trình sắp xếp, các hàm trực quan hóa như drawpoint và delay được sử dụng để đánh dấu các phần tử đang xử lý và hiển thị sự thay đổi của mảng, giúp dễ dàng theo dõi từng bước của thuật toán. Sự thay đổi chính trong thuật toán giảm dần so với tăng dần là việc so sánh giá trị trong các điều kiện (>= thay cho <=), đảm bảo rằng các phần tử lớn hơn **pivot** luôn nằm bên trái.



Hình 2.24. Hình ảnh code thuật toán Qick Sort sắp xếp giảm dần.

Hàm partitionDecrease

Hàm này thực hiện việc phân hoạch mảng dựa trên một phần tử trung tâm (pivot), sắp xếp các phần tử lớn hơn hoặc bằng pivot ở bên trái, và các phần tử nhỏ hơn ở bên phải.

* toado pivot = arr[left];: Chọn phần tử đầu tiên của mảng làm pivot.
* drawpoint(arr[left], 14);: Hiển thị phần tử pivot với màu đặc biệt để dễ dàng nhận biết trong giao diện đồ họa.
* int count = 0;: Khởi tạo biến đếm số phần tử có giá trị lớn hơn hoặc bằng pivot.
* for (int i = left + 1; i <= right; i++): Duyệt qua các phần tử còn lại trong mảng.
* if (arr[i].value >= pivot.value): Kiểm tra xem giá trị hiện tại có lớn hơn hoặc bằng giá trị của pivot hay không. Nếu có, tăng biến đếm count.
* int pivotIndex = left + count;: Xác định vị trí cuối cùng mà pivot sẽ được đặt sau khi phân hoạch.
* temp = arr[left].value;: Lưu giá trị của pivot.
* arr[left].value = arr[pivotIndex].value;: Đưa phần tử tại vị trí pivotIndex về vị trí ban đầu của pivot.
* arr[pivotIndex].value = temp;: Đặt pivot vào đúng vị trí pivotIndex.
* int i = left; int j = right;: Khởi tạo hai con trỏ để duyệt từ hai đầu mảng.
* while (i < pivotIndex && j > pivotIndex): Lặp lại cho đến khi hai con trỏ gặp nhau ở vị trí pivot.
* while (arr[i].value >= pivot.value): Di chuyển con trỏ i qua phải đến khi gặp phần tử nhỏ hơn pivot.
* while (arr[j].value < pivot.value): Di chuyển con trỏ j qua trái đến khi gặp phần tử lớn hơn hoặc bằng pivot.
* if (i < pivotIndex && j > pivotIndex): Nếu con trỏ i và j chưa vượt qua pivot, thực hiện hoán đổi giá trị giữa hai vị trí này.
* return pivotIndex;: Trả về vị trí của pivot sau khi phân hoạch.

Hàm quickSortDecrease

Hàm này triển khai thuật toán QuickSort để sắp xếp mảng giảm dần bằng cách đệ quy.

* if (right <= left): Nếu mảng con không hợp lệ (chỉ có một phần tử hoặc rỗng), dừng đệ quy.
* int pivot = partitionDecrease(arr, left, right);: Phân hoạch mảng và nhận vị trí cuối cùng của pivot.
* quickSortDecrease(arr, left, pivot - 1);: Gọi đệ quy sắp xếp phần mảng bên trái (các phần tử lớn hơn hoặc bằng pivot).
* quickSortDecrease(arr, pivot + 1, right);: Gọi đệ quy sắp xếp phần mảng bên phải (các phần tử nhỏ hơn pivot).

Hàm DecreaseSort

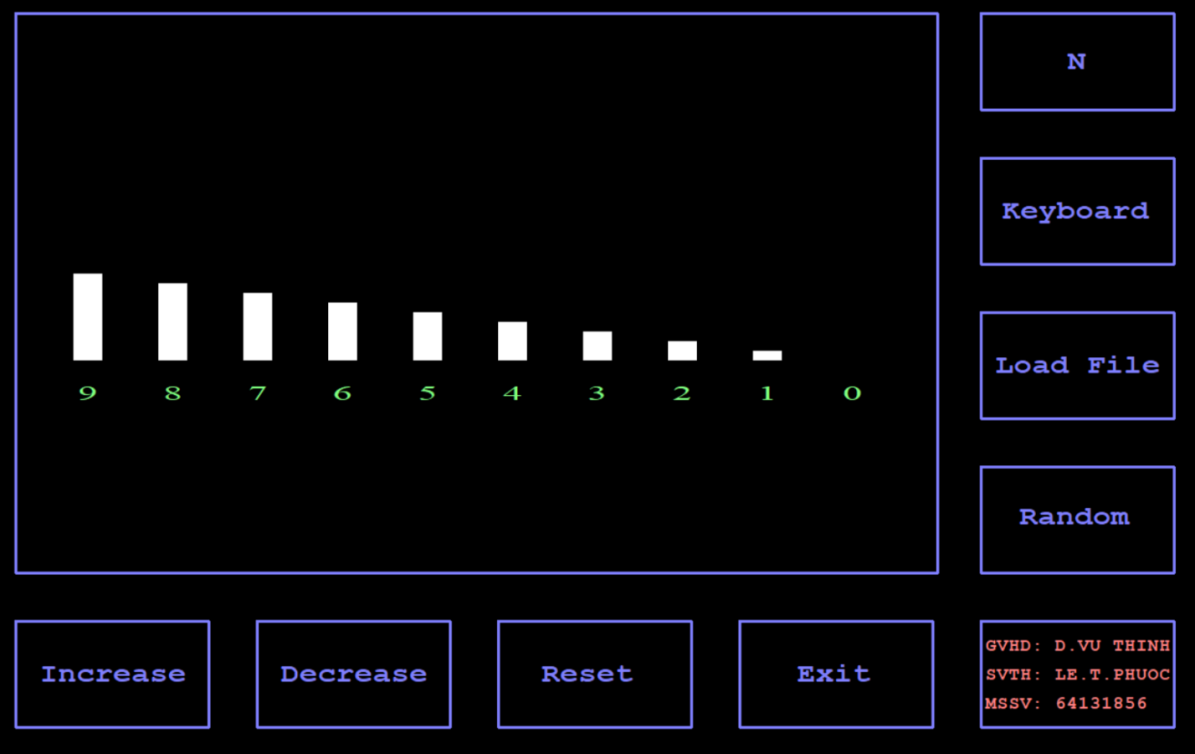
Hàm này gọi thuật toán QuickSort để sắp xếp toàn bộ mảng theo thứ tự giảm dần.

* sorted = 1;: Đánh dấu trạng thái là đang sắp xếp, có thể được sử dụng trong giao diện để khóa các chức năng khác.
* quickSortDecrease(arr, 0, n - 1);: Gọi hàm quickSortDecrease để sắp xếp toàn bộ mảng.
* Reset();: Kích hoạt nút "Reset" trên giao diện sau khi quá trình sắp xếp hoàn tất.

Hàm Decrease

Hàm này được gọi khi người dùng nhấn vào nút "Decrease" trên giao diện.

* enable\_click = 0;: Vô hiệu hóa thao tác chuột trong khi đang sắp xếp.
* GUI\_decrease();: Hiển thị giao diện liên quan đến sắp xếp giảm dần (ví dụ: hiển thị thông báo "Đang sắp xếp...").
* DecreaseSort(td, sodinh);: Gọi hàm DecreaseSort để thực hiện sắp xếp giảm dần mảng tọa độ td.
* for (int i = 0; i < sodinh; i++) { drawpoint(td[i], 10); }: Vẽ lại tất cả các điểm đã được sắp xếp lên giao diện. Màu sắc hoặc kích thước của các điểm có thể thay đổi tùy thuộc vào trạng thái.



Hình 2.25. Hình ảnh sau khi sắp xếp giảm dần.

### **2.2.3. Button chức năng**

**Button Reset**

Nút “Reset” được sử dụng để đặt lại trạng thái của giao diện người dùng về trạng thái ban đầu sau khi đã thực hiện một số thay đổi như khởi tạo mảng, sắp xếp mảng, và để chuẩn bị cho các tương tác tiếp theo.

void Reset() {

sorted = 0; //dat co ve 0

enable\_click = 1;

cleardevice();

GUI\_init();

}

Gán giá trị 0 cho biến sorted để theo dõi trạng thái của quá trình sắp xếp (có sắp xếp hay không). Việc gán giá trị 0 đại diện cho việc mảng chưa được sắp xếp. Gán giá trị 1 cho biến enable\_click được sử dụng để kiểm soát việc kích hoạt chức năng click chuột, việc gán giá trị 1 đại diện cho việc cho phép thực hiện các hành động liên quan đến click chuột sau khi đã thực hiện đặt lại. Để xóa nội dung hiện tại trên thiết bị đồ họa ta sử dụng hàm cleardevice(). Điều này có thể đưa giao diện về trạng thái mặc định. Sau đó gọi hàm GUI\_init() để khởi tạo lại giao diện người dùng về trạng thái ban đầu.

**Button Exit**

Nút “Exit” được sử dụng để đảm bảo việc giải phóng tất cả các tài nguyên đồ họa và chấm dứt chương trình một cách sạch sẽ khi người dùng quyết định thoát khỏi ứng dụng.

void Exit() {

closegraph();

exit(0);

}

Để đóng lại hoặc giải phóng tất cả các tài nguyên đồ họa đã được sử dụng trong chương trình ta sử dụng một hàm từ thư viện đồ họa (closegraph()). Hàm này có thể giải phóng bộ nhớ đồ họa, đóng cửa sổ đồ họa và các tài nguyên khác liên quan đến đồ họa. Hàm exit (0) dùng để kết thúc chương trình và thoát khỏi quá trình thực thi. Tham số truyền vào là 0 biểu thị rằng chương trình đã kết thúc mà không có lỗi.

# **CHƯƠNG 3. KẾT QUẢ THỰC HIỆN**

**Link youtube demo thuật toán Quick Sort:** <https://youtu.be/2pboTvfjQes>

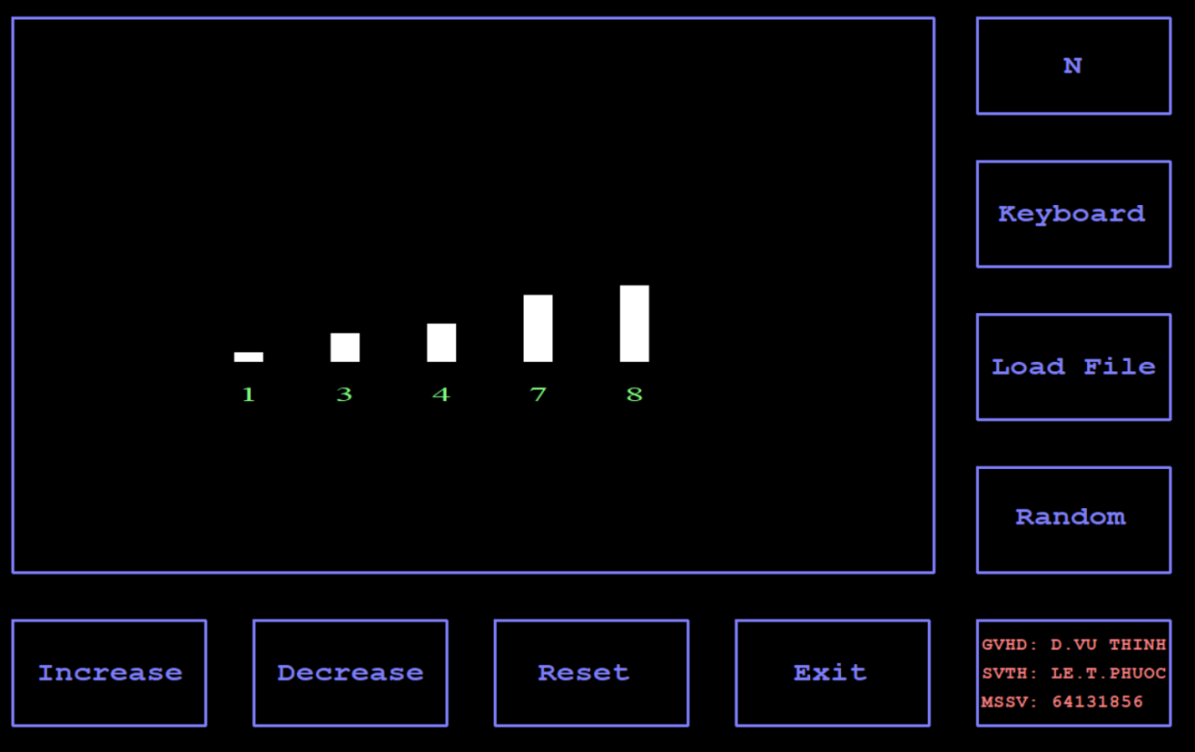
**Link github:**



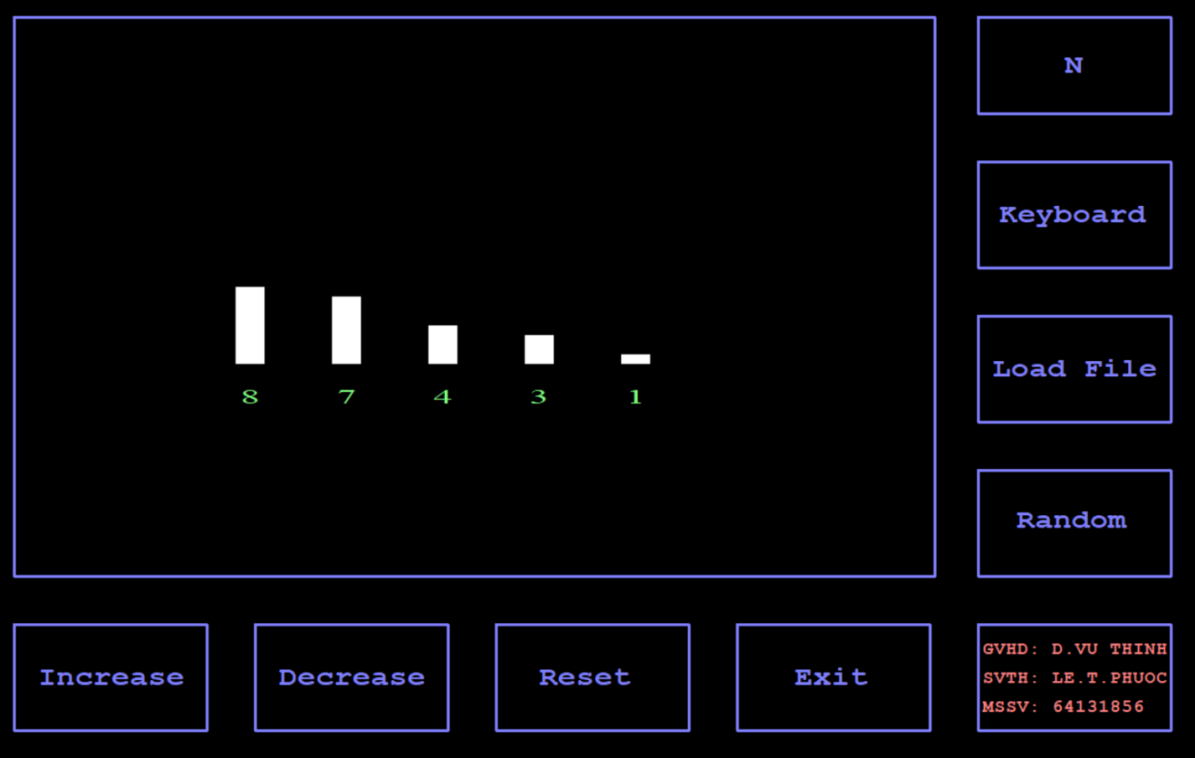
Hình 3.1. Hình ảnh kết quả khởi tạo mảng.



Hình 3.2. Hình ảnh kết quả đổi màu xanh cho các phần tử đang so sánh và đã sắp xếp.



Hình 3.3. Hình ảnh kết quả Quick Sort sắp xếp tăng dần.



Hình 3.4. Hình ảnh kết quả Quick Sort sắp xếp giảm dần.

# **CHƯƠNG 4. KẾT LUẬN**

## **4.1. Ưu điểm**

Sau một thời gian tìm hiểu, nghiên cứu, lập trình thì đề tài cũng đã được mô phỏng trên giao diện Dev-C++ quá trình sắp xếp dãy số bằng giải thuật Quick Sort sử dụng công cụ đồ hoạ graphics.h. Giao diện trình bày được đầy đủ các chức năng, tiến trình của việc sắp xếp nhanh, có nhiều sự lựa chọn khởi tạo mảng trong việc sắp xếp tăng dần và giảm dần như nhập từ bàn phím, sử dụng chuột hay tạo ngẫu nhiên. Các công đoạn được trình bày cụ thể, dễ hiểu, dễ nắm bắt, thông tin được đưa ra rõ ràng, dễ nhìn. Khi quy trình sắp xếp nhanh được kích hoạt, ta có thể sắp xếp được các mảng theo một trật tự nhất định, có thể là tăng dần hoặc giảm dần.

## **4.2. Nhược điểm**

Tuy nhiên, đề tài vẫn chưa chưa khắc phục được nhược điểm lỗi khi tiến hành sắp xếp tăng dần hoặc giảm dần khi cần phải di chuyển vị trí các thanh bar lớn hơn thì sẽ gây ra hiện tượng chồng lắp lên thanh bar nhỏ hơn. Cách khắc phục là xóa màn hình và vẽ lại sau mỗi lần di chuyển thanh bar nhưng màn hình sẽ bị nhấp nháy liên tục và gây khó chịu. Em sẽ cố gắng tìm hiểu và cải tiến nhược điểm trên trong quá trình lập trình.

**TÀI LIỆU THAM KHẢO**

1. Geeksforgeeks, quick-sort-algorithm. Từ<https://www.geeksforgeeks.org/quick-sort-algorithm/>Thạc Bình Cường (2023), *Cấu trúc dữ liệu và giải thuật,* NXB Bách Khoa Hà Nội, Hà Nội.
2. Hải Hà (2019). Các thuật toán sắp xếp cơ bản. Truy cập ngày 25/12/2023, từ <<https://viblo.asia/p/cac-thuat-toan-sap-xep-co-ban-Eb85ooNO52G>>.
3. Đỗ Xuân Lôi, *Cấu trúc dữ liệu và Giải thuật*, NXB GD 2005.
4. Lương Chi Mai (2000), *Nhập môn đồ họa máy tính*, NXB Khoa học và Kỹ thuật, Hà Nội.
5. TS. Nguyễn Hữu Tài (2017), *Đồ họa máy tính,* NXB Đại học Huế, Huế.
6. Đoàn Vũ Thịnh, *Bài giảng Kỹ thuật đồ họa*, 2019, Đại học Nha Trang
7. Sodhi, T. S., Kaur, S., & Kaur, S. (2013). “Enhanced Qick sort algorithm”. *International journal of Computer applications*, 64(21).
8. Shirley, P., Ashikhmin, M., & Marschner, S. (2009). *Fundamentals of computer graphics,* AK Peters/CRC Press.
9. Willms, A. (2008*). C++-der interaktive Programmierkurs: CD-ROM mit 8 Stunden Lernprogramm und Entwicklungsumgebung Dev-C++;* *Begleitbuch (Vol. 1).* Pearson Deutschland GmbH.
10. Youtube của TITV, Bài 18. Thuật Toán Sắp Xếp Nhanh (Quick Sort) | Cấu Trúc Dữ Liệu Và Giải Thuật từ<https://www.youtube.com/watch?v=n0dYgxhDLpY>.
11. HyperGraphics website: http://www.hypergraphics.co.uk/SCDiagrams.html.