BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**BÁO CÁO THỰC TẬP CƠ SỞ (CÔNG NGHỆ THÔNG TIN)**

**MÔ PHỎNG QUÁ TRÌNH SẮP XẾP DÃY SỐ BẰNG THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT) BẰNG CÔNG CỤ ĐỒ HOẠ GRAPHICS.H**

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

**Sinh viên thực hiện: Nguyễn Khắc Duy Hưng**

**Mã số sinh viên: 63132095**

**Lớp: 63.CNTT-4**

**Khánh Hoà – tháng 12/2023**

**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:** Mô phỏng quá trình sắp xếp dãy số bằng thuật toán sắp xếp trộn (Merge Sort) bằng công cụ đồ hoạ Graphics.h

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

**Sinh viên được hướng dẫn:** Nguyễn Khắc Duy Hưng

**MSSV:** 63132095

**Khoá**: 63 **Ngành:** Công nghệ thông tin

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 27/11/2023 | Nhận đề tài hướng dẫn và định hướng giải quyết vấn đề. Sinh viên trình bày kế hoạch thực hiện. |  |
| 2 | 04/12/2023 | Sinh viên thực hiện thiết kế giao diện đồ hoạ bằng việc tạo các nút, đổi màu cho nút khi click chuột vào nút đó, khung chứa giao diện dãy số, xử lý sự kiện chuột, tạo hình tròn cho các phần tử và in giá trị của mỗi phần tử lên trung tâm các hình tròn đó. |  |
| 3 | 11/12/2023 | Sinh viên hoàn thiện các thuật toán tạo mảng bằng các cách đưa dữ liệu đầu vào đã đề ra như đọc file, nhập bàn phím, tạo ngẫu nhiên,… và cách Reset mảng và Exit chương trình. |  |
| 4 | 18/12/2023 | Sinh viên tiến hành nghiên cứu thuật toán, cách sắp xếp và di chuyển các phần tử để từ đó mô phỏng đồ hoạ cho từng bước sắp xếp của thuật toán. Trong quá trình sắp xếp sẽ tiến hành đổi màu để nhận dạng cho các số đang xét và đã xét xong. |  |
| 5 | 25/12/2023 | Sinh viên thực hiện demo giao diện và sửa các trường hợp lỗi phát sinh khi sắp xếp như chạy sai màu các số khi đang xét, thực hiện việc tránh các dãy số mới bị chồng lên các dãy số cũ khi lấy đầu vào khác liên tiếp như nhấn nhiều lần nút Random mà không cần Reset. |  |
| 6 | 01/01/2024 | Sinh viên nộp bản thảo lần cuối sau khi đã chỉnh sửa các yêu cầu như đã đề ra. |  |

**MỤC LỤC**

[MỤC LỤC ii](#_Toc155451978)

[MỤC LỤC HÌNH ẢNH iv](#_Toc155451979)

[LỜI CẢM ƠN vi](#_Toc155451980)

[TÓM TẮT vii](#_Toc155451981)

[Chương 1. TỔNG QUAN VỀ ĐỀ TÀI 1](#_Toc155451982)

[1.1. DEV C++ 1](#_Toc155451983)

[1.2. THƯ VIỆN ĐỒ HOẠ GRAPHICS.H 1](#_Toc155451984)

[1.3. THUẬT TOÁN SẮP XẾP 2](#_Toc155451985)

[1.3.1. Một số thuật toán sắp xếp phổ biến 2](#_Toc155451986)

[1.3.2. So sánh các thuật toán sắp xếp 3](#_Toc155451987)

[1.4. THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT) 3](#_Toc155451988)

[1.4.1. Giới thiệu về thuật toán 3](#_Toc155451989)

[1.4.2. Ý tưởng về thuật toán 3](#_Toc155451990)

[1.4.3. Lưu đồ thuật toán 4](#_Toc155451991)

[1.4.4. Ví dụ minh hoạ 4](#_Toc155451992)

[1.4.5. Đánh giá thuật toán 5](#_Toc155451993)

[1.4.5.1. Độ phức tạp 5](#_Toc155451994)

[1.4.5.2. Ưu và nhược điểm 5](#_Toc155451995)

[Chương 2. PHƯƠNG PHÁP NGHIÊN CỨU 6](#_Toc155451996)

[2.1. PHẦN CHUNG CỦA SẢN PHẨM 6](#_Toc155451997)

[2.1.1. Khai báo thư viện và biến toàn cục 6](#_Toc155451998)

[2.1.2. Hàm chương trình chính 6](#_Toc155451999)

[2.2. TỔNG QUAN VỀ THIẾT KẾ GIAO DIỆN 7](#_Toc155452000)

[2.2.1. Sơ lược về giao diện sản phẩm 7](#_Toc155452001)

[2.2.2. Xây dựng giao diện 8](#_Toc155452002)

[2.2.2.1. Khởi tạo cửa sổ đồ hoạ BGI 8](#_Toc155452003)

[2.2.2.2. Xây dựng các thực thể trên cửa sổ BGI 8](#_Toc155452004)

[2.2.2.2.1. Xây dựng khung để chứa dãy số 9](#_Toc155452005)

[2.2.2.2.2. Xây dựng các nút (Button) 9](#_Toc155452006)

[2.2.2.2.3. Thiết kế Text các thông tin của tác giả 13](#_Toc155452007)

[2.2.2.2.4. Thiết kế các dãy số ban đầu (Dòng 1) 14](#_Toc155452008)

[2.2.2.2.5. Thiết kế mũi tên khi sắp xếp 15](#_Toc155452009)

[2.2.2.2.6. Thiết kế 2 dãy số nửa trái và phải được tách từ dãy số lớn (Dòng 2) 17](#_Toc155452010)

[2.2.2.2.7. Thiết kế các dãy số được tách từ dãy số ở dòng 2 (Dòng 3) 19](#_Toc155452011)

[2.2.2.2.8. Thiết kế các dãy số cho lần tách cuối cùng (Dòng 4 + 5) 22](#_Toc155452012)

[2.2.2.2.9. Thiết kế dãy số để sắp xếp (Dòng cuối cùng) 27](#_Toc155452013)

[2.3. XỬ LÝ SỰ KIỆN CHUỘT 28](#_Toc155452014)

[2.3.1. Kiểm tra khi nhấn nút Load File 31](#_Toc155452015)

[2.3.2. Kiểm tra khi nhấn nút Keyboard 32](#_Toc155452016)

[2.3.3. Kiểm tra khi nhấn nút Input Array 32](#_Toc155452017)

[2.3.4. Kiểm tra khi nhấn nút tạo ngẫu nhiên 10 số 33](#_Toc155452018)

[2.3.5. Kiểm tra khi nhấn nút tạo ngẫu nhiên 2 – 10 số 33](#_Toc155452019)

[2.3.6. Kiểm tra khi nhấn nút Reset 34](#_Toc155452020)

[2.3.7. Kiểm tra khi nhấn nút Exit 34](#_Toc155452021)

[2.3.8. Kiểm tra khi nhấn nút Sắp xếp tăng dần 35](#_Toc155452022)

[2.3.9. Kiểm tra khi nhấn nút Sắp xếp giảm dần 35](#_Toc155452023)

[2.4. PHÂN TÍCH THUẬT TOÁN 36](#_Toc155452024)

[2.4.1. Thiết lập tạo mảng 36](#_Toc155452025)

[2.4.1.1. Tạo mảng bằng cách đọc dữ liệu từ file 36](#_Toc155452026)

[2.4.1.2. Tạo mảng bằng cách nhập dữ liệu từ bàn phím (Trường hợp cố định 10 phần tử) 37](#_Toc155452027)

[2.4.1.3. Tạo mảng bằng cách nhập dữ liệu từ bàn phím (Trường hợp từ 2 đến 10 phần tử) 37](#_Toc155452028)

[2.4.1.4. Tạo mảng bằng cách tạo ngẫu nhiên (Trường hợp cố định 10 phần tử) 38](#_Toc155452029)

[2.4.1.5. Tạo mảng bằng cách tạo ngẫu nhiên (Trường hợp từ 2-10 phần tử) 38](#_Toc155452030)

[2.4.2. Thiết lập đồ hoạ cho thuật toán MERGE SORT 39](#_Toc155452031)

[2.4.2.1. Sắp xếp tăng dần 39](#_Toc155452032)

[2.4.2.2. Sắp xếp giảm dần 43](#_Toc155452033)

[Chương 3. KẾT QUẢ THỰC HIỆN 44](#_Toc155452034)

[Chương 4. KẾT LUẬN VÀ ĐÁNH GIÁ BẢN THÂN 51](#_Toc155452035)

[4.1. Kết quả đạt được 51](#_Toc155452036)

[4.2. Nhược điểm 51](#_Toc155452037)

[TÀI LIỆU THAM KHẢO 52](#_Toc155452038)

# **MỤC LỤC HÌNH ẢNH**

[Hình 1.1. Giới thiệu về Dev C++ 1](#_Toc155452039)

[Hình 1.2. Các đối tượng hình học của thư viện Graphics.h 2](#_Toc155452040)

[Hình 1.3. Tổng quan về so sánh của một số thuật toán sắp xếp 3](#_Toc155452041)

[Hình 2.1. Tất cả về thư viện và biến toàn cục mà bài sử dụng 6](#_Toc155460195)

[Hình 2.2. Hàm chương trình chính *main()* 6](#_Toc155460196)

[Hình 2.3. Màn hình giao diện bắt đầu chương trình 7](#_Toc155460197)

[Hình 2.4. Cấu trúc của các dãy số khi sắp xếp 7](#_Toc155460198)

[Hình 2.5. Kích thước cửa sổ đồ hoạ BGI 8](#_Toc155460199)

[Hình 2.6. Toàn bộ hàm *giaodien()* 8](#_Toc155460200)

[Hình 2.7. Kích thước của khung 9](#_Toc155460201)

[Hình 2.8. Các Button của chương trình 9](#_Toc155460202)

[Hình 2.9. Hiển thị text các thông tin của tác giả 13](#_Toc155460203)

[Hình 2.10. Các dãy số ban đầu của dòng 1 14](#_Toc155460204)

[Hình 2.11. Toàn bộ nội dung của hàm *veds()* dùng để vẽ các dãy số ở Dòng 1 15](#_Toc155460205)

[Hình 2.12. Thiết kế mũi tên trong quá trình sắp xếp 16](#_Toc155460206)

[Hình 2.13. Toàn bộ nội dung của hàm *arrow()* để vẽ mũi tên 16](#_Toc155460207)

[Hình 2.14. Hai dãy số nửa trái và nửa phải của Dòng 2 17](#_Toc155460208)

[Hình 2.15. Toàn bộ nội dung của hàm *veds\_chia()* để vẽ các dãy số ở Dòng 2 18](#_Toc155460209)

[Hình 2.16. Các dãy số được tách tiếp theo ở dòng 3 19](#_Toc155460210)

[Hình 2.17. Toàn bộ nội dung của hàm *veds\_sub()* để vẽ các dãy số ở Dòng 3 21](#_Toc155460211)

[Hình 2.18. Nếu số lượng bé hơn 9 phần tử thì việc tách kết thúc ở dòng 4 22](#_Toc155460212)

[Hình 2.19. Nếu số lượng từ 9 phần tử trở lên thì việc tách kết thúc ở dòng 5 23](#_Toc155460213)

[Hình 2.20. Toàn bộ nội dung của hàm *veds\_sub\_sub()* 26](#_Toc155460214)

[Hình 2.21. Dãy số thực hiện sắp xếp trên dòng cuối cùng 27](#_Toc155460215)

[Hình 2.22. Toàn bộ nội dung của hàm *veds\_sort()* 27](#_Toc155460216)

[Hình 2.23. Tóm gọn nội dung chính của hàm *mouse()* 29](#_Toc155460217)

[Hình 2.24. Khi click chuột bất kỳ vào màn hình BGI thì trên Console Window sẽ hiển thị toạ độ chuột ở đó 30](#_Toc155460218)

[Hình 2.25. Các câu lệnh khi click chuột phải sẽ thoát vòng lặp while 30](#_Toc155460219)

[Hình 2.26. Toàn bộ nội dung của khu vực nút Load File 31](#_Toc155460220)

[Hình 2.27. Toàn bộ nội dung của khu vực nút Keyboard 32](#_Toc155460221)

[Hình 2.28. Toàn bộ nội dung của khu vực nút Input Array 32](#_Toc155460222)

[Hình 2.29. Toàn bộ nội dung của khu vực nút Random 33](#_Toc155460223)

[Hình 2.30. Toàn bộ nội dung của khu vực Random Array 33](#_Toc155460224)

[Hình 2.31. Toàn bộ nội dung của khu vực nút Reset 34](#_Toc155460225)

[Hình 2.32. Toàn bộ nội dung của khu vực nút Exit 34](#_Toc155460226)

[Hình 2.33. Toàn bộ nội dung của khu vực nút Sắp xếp tăng dần 35](#_Toc155460227)

[Hình 2.34. Toàn bộ nội dung của khu vực nút Sắp xếp giảm dần 35](#_Toc155460228)

[Hình 2.35. Toàn bộ nội dung của hàm *readfile()* 36](#_Toc155460229)

[Hình 2.36. Hàm *inmang()* để hiển thị giá trị trên Console Windows 36](#_Toc155460230)

[Hình 2.37. Toàn bộ nội dung của hàm *keyboard()* 37](#_Toc155460231)

[Hình 2.38. Toàn bộ nội dung của hàm *input\_array()* 37](#_Toc155460232)

[Hình 2.39. Toàn bộ nội dung của hàm tạo ngẫu nhiên 10 số 38](#_Toc155460233)

[Hình 2.40. Toàn bộ nội dung của hàm tạo ngẫu nhiên từ 2 đến 10 số 38](#_Toc155460234)

[Hình 2.41. Toàn bộ nội dung của hàm *merge\_tangdan()* 39](#_Toc155460235)

[Hình 2.42. Toàn bộ nội dung của hàm *mergesort\_tangdan()* 42](#_Toc155460236)

[Hình 2.43. Dãy số ở dòng cuối cùng sau khi thực hiện sắp xếp tăng dần 42](#_Toc155460237)

[Hình 2.44. Toàn bộ nội dung của hàm *mergesort\_giamdan()* 43](#_Toc155460238)

[Hình 2.45. Dãy số ở dòng cuối cùng sau khi thực hiện sắp xếp giảm dần 43](#_Toc155460239)

[Hình 3.1. Vẽ dãy số từ dữ liệu file trên dòng 1 44](#_Toc155456246)

[Hình 3.2. Vẽ mô phỏng các dãy số khi tách trên dòng 2 44](#_Toc155456247)

[Hình 3.3. Vẽ mô phỏng các dãy số sau lần tách tiếp theo trên dòng 3 45](#_Toc155456248)

[Hình 3.4. Vẽ mô phỏng các dãy số cho lần tách tiếp theo trên dòng 4 46](#_Toc155456249)

[Hình 3.5. Vẽ dãy số trên dòng cuối cùng để thực hiện việc sắp xếp 46](#_Toc155456250)

[Hình 3.6. MergeSort sẽ sắp xếp cặp số đầu tiên 47](#_Toc155456251)

[Hình 3.7. Thuật toán sẽ sắp xếp 2 cặp số tiếp theo 47](#_Toc155456252)

[Hình 3.8. Thuật toán gộp lại rồi sắp xếp, kết thúc sắp xếp mảng con bên trái 48](#_Toc155456253)

[Hình 3.9. Tiến đến mảng nửa bên phải, thuật toán cũng sắp xếp 2 cặp số đầu tiên 48](#_Toc155456254)

[Hình 3.10. Thuật toán xét 2 cặp số tiếp theo rồi sắp xếp 49](#_Toc155456255)

[Hình 3.11. Thuật toán sẽ gộp lại và sắp xếp. Kết thúc mảng nửa phải 49](#_Toc155456256)

[Hình 3.12. Thuật toán tiến hành gộp và sắp xếp mảng nửa trái và nửa phải 50](#_Toc155456257)

# **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 thực hiện đề tài này.

Qua đây, em xin chân thành cảm ơn thầy Đoàn Vũ Thịnh, người đã trực tiếp quan tâm và hướng dẫn 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**

Đồ hoạ máy tính là một lĩnh vực quan trọng trong công nghệ thông tin, nơi các hình ảnh và đối tượng được tạo ra và hiển thị trực quan trên màn hình máy tính. Đồ hoạ máy tính không chỉ giúp làm đẹp giao diện người dùng mà còn có ứng dụng rộng rãi trong mô phỏng, giáo dục, và trực quan hóa dữ liệu.

Thuật toán sắp xếp là một khái niệm quan trọng trong lĩnh vực khoa học máy tính và lập trình. Nó đề cập đến quá trình sắp xếp các phần tử trong một tập hợp theo một thứ tự nhất định, nhằm tối ưu hóa thời gian và tăng hiệu suất khi xử lý dữ liệu. Có nhiều thuật toán sắp xếp khác nhau, mỗi thuật toán có đặc điểm và ưu điểm riêng, dựa trên độ phức tạp thời gian, không gian và các yếu tố khác.

Khi kết hợp thuật toán sắp xếp với công cụ đồ hoạ máy tính như Graphics.h trong ngôn ngữ lập trình C++, chúng ta có cơ hội tạo ra một trải nghiệm học tập và hiểu biết động hấp dẫn. Việc mô phỏng quá trình sắp xếp dãy số thông qua đồ hoạ máy tính không chỉ giúp sinh viên thấy rõ từng bước của thuật toán mà còn giúp họ hình dung và theo dõi quá trình này một cách trực quan.

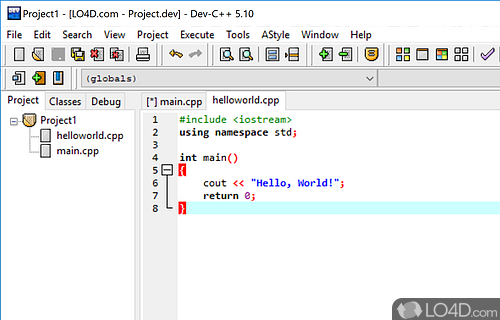
Sản phẩm đã minh họa được quá trình mô phỏng được thực hiện thông qua việc trực quan hóa các bước của thuật toán MergeSort. Các bước bao gồm chia dãy số thành các nửa, sắp xếp từng nửa, và sau đó trộn hai nửa đã sắp xếp thành một dãy số hoàn chỉnh. Sản phẩm thực hiện tốt với nhiều cách đưa dữ liệu đầu vào như đọc file, nhập từ bàn phím hoặc tạo nhanh ngẫu nhiên,… và có thể sắp xếp được dựa trên các dữ liệu đầu vào đó.

Chương 1. TỔNG QUAN VỀ ĐỀ TÀI

1.1. DEV C++

Dev C++ là một môi trường phát triển tích hợp (IDE) phổ biến được sử dụng cho việc lập trình bằng ngôn ngữ lập trình C và C++. Được phát triển bởi Bloodshed Software, Dev C++ cung cấp một giao diện đơn giản và thuận tiện, làm cho quá trình lập trình trở nên dễ dàng đối với cả người mới bắt đầu và những người có kinh nghiệm.

Dev-C++ được đóng gói và hỗ trợ nhiều trình biên dịch C++ phổ biến như MinGW hoặc TDM-GCC 64bit của GCC (GNU Compoler Collection). Dev-C++ cũng có thể được sử dụng kết hợp với Cygwin hoặc bất kỳ trình biên dịch dựa trên GCC nào khác.



**Hình 1.1. Giới thiệu về Dev C++**

Giao diện của Dev C++ đơn giản và thân thiện với người dùng, điều này làm cho quá trình lập trình trở nên trực quan và dễ tiếp cận. Ngoài ra, IDE này hỗ trợ nhiều tính năng như quản lý dự án, trình soạn thảo mã nguồn tùy chỉnh, và khả năng tạo và chạy chương trình một cách thuận lợi.

Dự án phát triển Dev-C++ được lưu trữ trên SourceForge. Dev-C++ nguyên được phát triển bởi một lập trình viên có tên là Colin Laplace và chỉ chạy trên hệ điều hành Microsoft Windows.

1.2. THƯ VIỆN ĐỒ HOẠ GRAPHICS.H

Thư viện đồ hoạ graphics.h là một thư viện phổ biến được sử dụng trong ngôn ngữ lập trình C và C++ để tạo và quản lý đồ hoạ trên màn hình máy tính. Thư viện này thường được sử dụng để vẽ đồ họa cơ bản, tạo ra giao diện người dùng đơn giản, và mô phỏng các hình ảnh đồ họa.

A group of shapes on a black background

Description automatically generated

**Hình 1.2. Các đối tượng hình học của thư viện Graphics.h**

Thư viện graphics.h cung cấp một loạt các hàm và công cụ để điều khiển đồ hoạ, bao gồm việc vẽ các đối tượng hình học như đường thẳng, hình chữ nhật, hình tròn, v.v. Thư viện này cũng hỗ trợ việc thao tác màu sắc, font chữ và các hiệu ứng đơn giản khác.

1.3. THUẬT TOÁN SẮP XẾP

Sắp xếp là một quá trình quan trọng trong lập trình và khoa học máy tính, nhằm sắp xếp các phần tử trong một dãy số theo một thứ tự nhất định. Mục tiêu chính của việc sắp xếp là tạo ra một trình tự có thứ tự từ dữ liệu không có trình tự hoặc có thứ tự không đúng.

### **1.3.1. Một số thuật toán sắp xếp phổ biến**

* Sắp xếp nổi bọt (Bubble Sort): Thuật toán
* so sánh các phần tử để đặt các phần tử lớn nhất vào vị trí cuối cùng.
* Sắp xếp chèn (Insertion Sort): Thuật toán so sánh các phần tử để quyết định vị trí của một phần tử trong mảng đã được sắp xếp một phần.
* Sắp xếp lựa chọn (Selection Sort): Thuật toán so sánh các phần tử để đặt các phần tử nhỏ nhất vào vị trí phía trước.
* Sắp xếp nhanh (Quick Sort): Thuật toán so sánh các phần tử của phân vùng mảng chưa được sắp xếp thành hai nửa khác nhau xung quanh giá trị pivot.
* Sắp xếp trộn (Merge Sort): Thuật toán so sánh các phần tử của hai phần tử đã sắp xếp để hợp nhất chúng thành mảng được sắp xếp cuối cùng.

1.3.2. So sánh các thuật toán sắp xếp

A table of text with black and white text

Description automatically generated with medium confidence

**Hình 1.3. Tổng quan về so sánh của một số thuật toán sắp xếp**

1.4. THUẬT TOÁN SẮP XẾP TRỘN (MERGE SORT)

1.4.1. Giới thiệu về thuật toán

Trong khoa học máy tính, sắp xếp trộn (Merge Sort) là một thuật toán sắp xếp để sắp xếp các danh sách (hoặc bất kỳ cấu trúc dữ liệu nào có thể truy cập tuần tự, v.d. luồng tập tin) theo một trật tự nào đó. Nó được phát triển bởi John von Neumann vào năm 1945 và được tối ưu hóa bởi John von Neumann và Hermann A. H. Goldstine vào năm 1948.

Merge Sort dựa trên nguyên tắc “chia để trị” (divide and conquer). Thuật toán này hoạt động bằng cách chia dãy số cần sắp xếp thành các đoạn nhỏ, sắp xếp từng đoạn rồi kết hợp chúng lại với nhau theo trật tự đúng để tạo ra dãy số đã sắp xếp.

Merge Sort đảm bảo tính ổn định trong quá trình sắp xếp, điều này có nghĩa là nếu có hai phần tử có giá trị bằng nhau, thì thứ tự của chúng trong dãy số gốc sẽ được duy trì trong dãy số đã sắp xếp.

1.4.2. Ý tưởng về thuật toán

Sử dụng ý tưởng đệ quy “chia để trị” với quan điểm việc sắp xếp tăng dần một dãy số sẽ tương ứng với các thao tác sau:

* Chia (Divide): Chia dãy gồm n phần tử cần sắp xếp thành 2 dãy, mỗi dãy có n/2 phần tử (nửa trái và nửa phải).
* Trị (Conquer): Sắp xếp mỗi dãy con một cách đệ quy sử dụng sắp xếp trộn. Khi dãy chỉ còn một phần tử thì trả lại phần tử này (tức gọi hàm đệ quy mergeSort(…) trên lần lượt nửa trái và nửa phải).
* Tổ hợp (Combine): Mỗi nửa sau khi được sắp xếp sẽ được kết hợp, trộn lẫn với nhau để tạo ra một mảng hoàn chỉnh đã được sắp xếp như mong muốn.

1.4.3. Lưu đồ thuật toán

A diagram of a algorithm

Description automatically generated

**Bảng 1.3. Lưu đồ của thuật toán**

1.4.4. Ví dụ minh hoạ



**Bảng 1.4. Ví dụ minh hoạ cho thuật toán MergeSort**

1.4.5. Đánh giá thuật toán

1.4.5.1. Độ phức tạp

* Độ phức tạp về thời gian

Ta gọi T(n) là độ phức tạp của thuật toán MergeSort khi mảng đầu vào có n phần tử.

Độ phức tạp của hai thủ tục gọi đệ quy lần lượt là T([n/2]) và T([n/2]). Phần còn lại của phép phân tích là tính độ phức tạp của thủ tục merge.

Vì thủ tục merge có các lệnh điều kiện if-else nên không thể đếm chính xác số thao tác cơ bản được. Nên ta sử dụng O(.), để có thể khẳng định số thao tác cơ bản là O(n) vì ta chỉ lặp n lần và mỗi lần lặp ta thực O(1) phép tính cơ bản. Do đó:

Ta có: T(n) = T([n/2]) + T([n/2]) + O(n) = 2T([n/2]) + O(n)

Phương trình trên, ta chỉ cần áp dụng định lý Thợ rút gọn (*master theorem*), thu được:

T(n) = O(nlogn)

Với cả 3 trường hợp: Best, Avarage và Worst thì Merge Sort đều có độ phức tạp là O(nlogn) cho thấy thuật toán Merge Sort có tốc độ giải quyết khá nhanh.

* Độ phức tạp về không gian: có độ phức tạp O(n) cho cả 3 trường hợp Best, Avarage và Worst, Merge Sort khá tốn bộ nhớ vì trong nó cần phải sao chép các phần tử vào một mảng tạm. Cho nên cần phải có n không gian cho việc sắp xếp.

1.4.5.2. Ưu và nhược điểm

* Ưu điểm
* Đơn giản và dễ hiểu, thời gian sắp xếp với độ phức tạp đã được giảm xuống một cách đáng kể (O(nlog(n)) trên cả 3 trường hợp đều chạy như nhau so với các giải thuật Insertion Sort, Selection Sort, Bubble Sort.
* Giải thuật Merge Sort giữ được tính ổn định tương đối của các phần tử.
* Merge Sort xử lý khá tốt với dữ liệu lớn đặc biệt là dạng list, file.
* Nhược điểm
* Giải thuật Merge Sort do cần phải sử dụng thêm vùng nhớ bên ngoài (vùng nhớ dùng thêm này tỉ lệ với số lượng phần tử n) nên giải thuật không có tính chất tại chỗ.
* Không gian bộ nhớ bổ sung cần nhiều hơn các thuật toán sắp xếp so sánh O(n\*log(n)) khác như Heapsort hay Quicksort.

Chương 2. PHƯƠNG PHÁP NGHIÊN CỨU

2.1. PHẦN CHUNG CỦA SẢN PHẨM

2.1.1. Khai báo thư viện và biến toàn cục

|  |  |
| --- | --- |
| A screenshot of a computer screen  Description automatically generated | |
|  | A screen shot of a computer program  Description automatically generated |

**Hình 2.1. Tất cả về thư viện và biến toàn cục mà bài sử dụng**

### **2.1.2. Hàm chương trình chính**

A screen shot of a computer code

Description automatically generated

**Hình 2.2. Hàm chương trình chính main()**

2.2. TỔNG QUAN VỀ THIẾT KẾ GIAO DIỆN

2.2.1. Sơ lược về giao diện sản phẩm

Giao diện chính được mô tả các thông tin như: Họ tên sinh viên thực tập, họ tên giáo viên hướng dẫn, mã số sinh viên, thư điện tử của sinh viên; Khung để hiển thị mảng trước khi sắp xếp, mảng trong quá trình sắp xếp và mảng sau khi đã được sắp xếp; Các nút (button) thể hiện từng chức năng riêng ở bên cạnh khung lớn.

A black screen with white text

Description automatically generated

**Hình 2.3. Màn hình giao diện bắt đầu chương trình**

Mô tả cấu trúc giao diện khi sắp xếp sẽ chia thành 5 dòng như sau:

Lưu ý: Dòng 1 để hiển thị dãy số ban đầu khi mới tạo. Các dòng 2 đến 5 chỉ để mô phỏng làm cách nào mà thuật toán MergeSort có thể tách mảng lớn thành các mảng con nhỏ hơn. Dòng 5 sẽ được MergeSort áp dụng lên để sắp xếp.

A screenshot of a computer game

Description automatically generated

**Hình 2.4. Cấu trúc của các dãy số khi sắp xếp**

2.2.2. Xây dựng giao diện

2.2.2.1. Khởi tạo cửa sổ đồ hoạ BGI

Tại hàm chính *main(),* ta sử dụng hàm *initwindow(width, height)* với width = 1700 và height = 1000 được sử dụng để khởi tạo một cửa sổ đồ họa có độ rộng là 1700 và chiều cao là 1000 trên màn hình do thư viện đồ hoạ graphics.h cung cấp.

A black screen with red text

Description automatically generated

**Hình 2.5. Kích thước cửa sổ đồ hoạ BGI**

2.2.2.2. Xây dựng các thực thể trên cửa sổ BGI

Để hiển thị các thực thể (khung, nút và các dãy số) lên cửa sổ đồ hoạ. Ta sẽ viết các đoạn lệnh tại hàm *giaodien()*.

Toàn bộ nội dung của hàm *giaodien()* như sau:

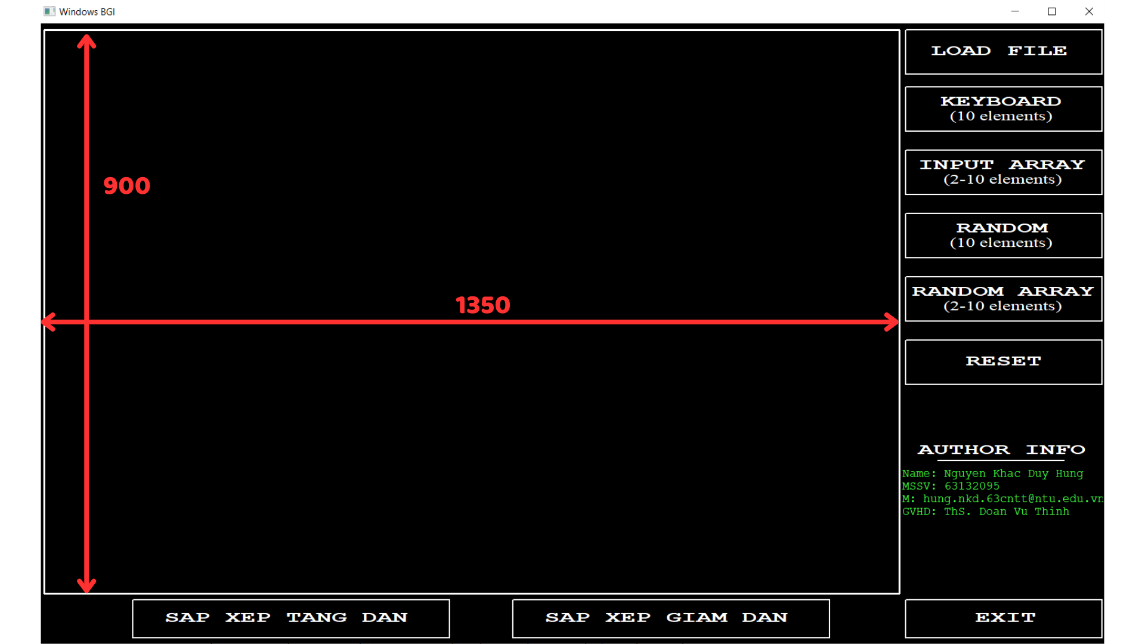
|  |  |
| --- | --- |
| A screen shot of a computer program  Description automatically generated | A screen shot of a computer program  Description automatically generated |

**Hình 2.6. Toàn bộ hàm giaodien()**

2.2.2.2.1. Xây dựng khung để chứa dãy số

Để vẽ khung, ta sẽ cần sử dụng các hàm do thư viện *graphics.h* cung cấp như sau:

* *setlinestyle(3, 1, 3)*: Thiết lập đường vẽ có tham số thứ 1 *linestyle* = 3 là kiểu đường chấm gạch ngang. Tham số thứ 2 *upattern* = 1 là mẫu nhị phân (tạo ra đường chấm) và tham số thứ 3 *thickness* = 3 để thể hiện độ dày của nét vẽ.
* *rectangle(10, 10, 1360, 900)*: Vẽ một hình chữ nhật có góc trái trên cùng có toạ độ (10, 10) và góc phải dưới cùng có toạ độ (1360, 900). Kích thước của hình chữ nhật là 1350 px (1360 - 10) theo chiều ngang và 890 px (900 - 10) theo chiều dọc.



**Hình 2.7. Kích thước của khung**

2.2.2.2.2. Xây dựng các nút (Button)

A screenshot of a computer

Description automatically generated

**Hình 2.8. Các Button của chương trình**

* Nút Load File: dùng để đọc dữ liệu từ một file từ đó có thể tạo ra mảng dựa trên dữ liệu đó.
* *setcolor(loadfilestatus ? YELLOW : WHITE)*: Tạo màu cho nút Load File. Nếu biến *loadfilestatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 10, 1680, 80)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 10) và góc phải dưới cùng có toạ độ (1680, 80). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (80 - 10) theo chiều dọc.
* *outtextxy(1410, 30, "LOAD FILE")*: Hiển thị chuỗi văn bản “LOAD FILE” bắt đầu từ vị trí có toạ độ là (1410, 30) trên cửa sổ BGI.
* Nút Keyboard: dùng để nhập 10 giá trị phần tử tại Console Windows.
* *setcolor(keyboardstatus ? YELLOW : WHITE)*: Tạo màu cho nút Keyboard. Nếu biến *keyboardstatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 100, 1680, 170)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 100) và góc phải dưới cùng có toạ độ (1680, 170). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (170 - 100) theo chiều dọc.
* *outtextxy(1425, 110, "KEYBOARD")*: Hiển thị chuỗi văn bản “KEYBOARD” bắt đầu từ vị trí có toạ độ là (1425, 110) trên cửa sổ BGI.
* *outtextxy(1440,135,"(10 elements)")*: Hiển thị chuỗi văn bản “(10 elements)” bắt đầu từ vị trí có toạ độ là (1440, 135) trên cửa sổ BGI.
* Nút Input Array: dùng để nhập số lượng phần tử từ 2 đến 10 và nhập giá trị của từng phần tử tương ứng với số lượng phần tử đó tại Console Windows.
* *setcolor(inputstatus ? YELLOW : WHITE)*: Tạo màu cho nút Input Array. Nếu biến *inputstatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 200, 1680, 270)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 200) và góc phải dưới cùng có toạ độ (1680, 270). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (270 - 200) theo chiều dọc.
* *outtextxy(1390,210,"INPUT ARRAY")*: Hiển thị chuỗi văn bản “INPUT ARRAY” bắt đầu từ vị trí có toạ độ là (1390, 210) trên cửa sổ BGI.
* *outtextxy(1430,235,"(2-10 elements)")*: Hiển thị chuỗi văn bản “(2-10 elements)” bắt đầu từ vị trí có toạ độ là (1430, 235) trên cửa sổ BGI.
* Nút Random: Tạo ngẫu nhiên 10 phần tử.
* *setcolor(randstatus ? YELLOW : WHITE)*: Tạo màu cho nút Random. Nếu biến *randstatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 300, 1680, 370)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 300) và góc phải dưới cùng có toạ độ (1680, 370). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (370 - 300) theo chiều dọc.
* *outtextxy(1450,310,"RANDOM")*: Hiển thị chuỗi văn bản “RANDOM” bắt đầu từ vị trí có toạ độ là (1450, 310) trên cửa sổ BGI.
* *outtextxy(1440,335,"(10 elements)")*: Hiển thị chuỗi văn bản “(10 elements)” bắt đầu từ vị trí có toạ độ là (1440, 335) trên cửa sổ BGI.
* Nút Random Array: Tạo ngẫu nhiên 2 đến 10 phần tử.
* *setcolor(randarraystatus ? YELLOW : WHITE)*: Tạo màu cho nút Random Array. Nếu biến *randarraystatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 400, 1680, 470)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 400) và góc phải dưới cùng có toạ độ (1680, 470). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (470 - 400) theo chiều dọc.
* *outtextxy(1380,410,"RANDOM ARRAY")*:Hiển thị chuỗi văn bản “RANDOM ARRAY” bắt đầu từ vị trí có toạ độ là (1380, 410) trên cửa sổ BGI.
* *outtextxy(1430,435,"(2-10 elements)")*: Hiển thị chuỗi văn bản “(2-10 elements)” bắt đầu từ vị trí có toạ độ là (1430, 435) trên cửa sổ BGI.
* Nút RESET: dùng để làm mới màn hình BGI.
* *setcolor(resetstatus ? YELLOW : WHITE)*: Tạo màu cho nút RESET. Nếu biến *resetstatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 500, 1680, 570)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 500) và góc phải dưới cùng có toạ độ (1680, 570). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (570 - 500) theo chiều dọc.
* *outtextxy(1465,520,"RESET")*: Hiển thị chuỗi văn bản “RESET” bắt đầu từ vị trí có toạ độ là (1465, 520) trên cửa sổ BGI.
* Nút EXIT: Thoát khỏi chương trình đồ hoạ BGI.
* *setcolor(exitstatus ? YELLOW : WHITE)*: Tạo màu cho nút EXIT. Nếu biến *exitstatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(1370, 900, 1680, 970)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (1370, 900) và góc phải dưới cùng có toạ độ (1680, 970). Kích thước của hình chữ nhật này là 310 px (1680 - 1370) theo chiều ngang và 70 px (970 - 900) theo chiều dọc.
* *outtextxy(1480,920,"EXIT")*: Hiển thị chuỗi văn bản “EXIT” bắt đầu từ vị trí có toạ độ là (1480, 920) trên cửa sổ BGI.
* Nút SẮP XẾP TĂNG DẦN: Thực thi sắp xếp tăng dần.
* *setcolor(increasestatus ? YELLOW : WHITE)*: Tạo màu cho nút SẮP XẾP TĂNG DẦN. Nếu biến *increasestatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(150, 910, 650, 970)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (150, 910) và góc phải dưới cùng có toạ độ (650, 970). Kích thước của hình chữ nhật này là 500 px (650 - 150) theo chiều ngang và 60 px (970 - 910) theo chiều dọc.
* *outtextxy(200, 925, "SAP XEP TANG DAN")*: Hiển thị chuỗi văn bản “SAP XEP TANG DAN” bắt đầu từ vị trí có toạ độ là (200, 925) trên cửa sổ BGI.
* Nút SẮP XẾP GIẢM DẦN: Thực thi sắp xếp giải dần.
* *setcolor(decreasestatus ? YELLOW : WHITE)*: Tạo màu cho nút SẮP XẾP GIẢM DẦN. Nếu biến *decreasestatus* có giá trị khác 0(đúng) thì sẽ chuyển màu vàng (YELLOW) và ngược lại nếu biến có giá trị là 0 (sai) thì sẽ chuyển màu trắng (WHITE). Phần thay đổi giá trị của biến sẽ nằm ở hàm *mouse().*
* *rectangle(750, 910, 1250, 970)*: Vẽ hình chữ nhật có góc trái trên cùng có toạ độ (750, 910) và góc phải dưới cùng có toạ độ (1250, 970). Kích thước của hình chữ nhật này là 500 px (1250 - 750) theo chiều ngang và 60 px (970 - 910) theo chiều dọc.
* *outtextxy(900, 925, "SAP XEP GIAM DAN")*: Hiển thị chuỗi văn bản “SAP XEP GIAM DAN” bắt đầu từ vị trí có toạ độ là (900, 925) trên cửa sổ BGI.

2.2.2.2.3. Thiết kế Text các thông tin của tác giả

A screen shot of a computer

Description automatically generated

**Hình 2.9. Hiển thị text các thông tin của tác giả**

*outtextxy(1390, 660, "AUTHOR INFO")*: Hiển thị chuỗi văn bản “AUTHOR INFO” bắt đầu từ vị trí có toạ độ là (1390, 660) trên cửa sổ BGI.

*setcolor(LIGHTGREEN)*: Thay đổi màu sang màu xanh lá nhạt.

*outtextxy(1365,700,"Name: Nguyen Khac Duy Hung")*: Hiển thị chuỗi văn bản “Name: Nguyen Khac Duy Hung” bắt đầu từ vị trí có toạ độ là (1365, 700) trên cửa sổ BGI.

*outtextxy(1365,720,"MSSV: 63132095")*: Hiển thị chuỗi văn bản “MSSV: 63132095” bắt đầu từ vị trí có toạ độ là (1365, 720) trên cửa sổ BGI.

*outtextxy(1365,740,"M: hung.nkd.63cntt@ntu.edu.vn")*: Hiển thị chuỗi văn bản “hung.nkd.63cntt@ntu.edu.vn” bắt đầu từ vị trí có toạ độ là (1365, 740) trên cửa sổ BGI.

*outtextxy(1365,760,"GVHD: ThS. Doan Vu Thinh")*: Hiển thị chuỗi văn bản “GVHD: ThS. Doan Vu Thinh” bắt đầu từ vị trí có toạ độ là (1365, 760) trên cửa sổ BGI.

2.2.2.2.4. Thiết kế các dãy số ban đầu (Dòng 1)

A screenshot of a computer

Description automatically generated

**Hình 2.10. Các dãy số ban đầu của dòng 1**

Dãy số của dòng 1 được tạo bởi hàm *veds().* Toàn bộ nội dung của hàm như sau:

|  |  |
| --- | --- |
|  |  |
| A computer screen shot of a program code  Description automatically generated | |

**Hình 2.11. Toàn bộ nội dung của hàm veds() dùng để vẽ các dãy số ở Dòng 1**

* Các điều kiện if (n == ...): Các điều kiện này dựa trên số lượng phần tử n của mảng để xác định vị trí bắt đầu để vẽ hình tròn cho phần tử đầu tiên. Mỗi trường hợp tương ứng với số lượng phần tử cụ thể.
* Vòng lặp for chạy từ i = 0 đến i = n - 1, lặp qua từng phần tử của mảng để vẽ đồ hoạ hình tròn cho các điểm.
* *int x = toado\_x + i \* (2 \* RADIUS + CIRCLE\_GAP):* Tính toán vị trí x của hình tròn thứ i dựa trên vị trí ban đầu toado\_x, bán kính RADIUS, và khoảng cách giữa các hình tròn CIRCLE\_GAP. Vị trí x này được tính để các hình tròn không chồng lên nhau.
* *int y = toado\_y:* Vị trí y của hình tròn là toado\_y, tức là một giá trị cố định
* *circle(x, y, RADIUS):* Vẽ hình tròn tại vị trí (x, y) với bán kính RADIUS.
* *sprintf(buffer, "%d", a[i]):* Chuyển đổi giá trị của phần tử thứ i trong mảng a thành chuỗi và lưu vào buffer. Hàm sprintf là hàm có chức năng giống như printf, nhưng nó in vào một chuỗi thay vì in ra màn hình.
* *outtextxy(x - textwidth(buffer) / 2, y - textheight(buffer) / 2, buffer):* In chuỗi buffer lên màn hình tại vị trí *(x - textwidth(buffer) / 2, y - textheight(buffer) / 2)*. Các hàm textwidth và textheight là để lấy chiều rộng và chiều cao của chuỗi, giúp đặt vị trí in chuỗi chính xác giữa hình tròn.

2.2.2.2.5. Thiết kế mũi tên khi sắp xếp

Khi ta thực hiện việc thao tác tạo ra các dãy số từ các chức năng nằm ở bên phải khung. Sau đó ta tiến hành nhấn nút “sắp xếp tăng dần” hoặc “sắp xếp giảm dần” thì một loạt các giao diện được vẽ ra, trong đó có việc vẽ mũi tên thông qua hàm arrow() để làm trực quan hơn trong quá trình sắp xếp.

A screenshot of a phone

Description automatically generated

**Hình 2.12. Thiết kế mũi tên trong quá trình sắp xếp**

Toàn bộ nội dung của hàm arrow() như sau:

A computer screen shot of a program code

Description automatically generated

**Hình 2.13. Toàn bộ nội dung của hàm arrow() để vẽ mũi tên**

Hàm *arrow()* được sử dụng để vẽ mũi tên trên đoạn đường từ điểm (x1, y1) đến điểm (x2, y2).

* *setcolor(LIGHTCYAN):* Đặt màu vẽ là màu xanh nhạt
* *setlinestyle(SOLID\_LINE, 0, THICK\_WIDTH):* Đặt kiểu đường vẽ là đường thẳng liền (SOLID\_LINE), không đặt kiểu nét (0), và đặt độ dày cho đường vẽ (THICK\_WIDTH).
* *line(x1, y1, x2, y2):* Vẽ đoạn đường thẳng từ điểm (x1, y1) đến (x2, y2).
* *int arrowSize = 10*: Đặt kích thước của mũi tên.
* *double angle = atan2(y2 - y1, x2 - x1):* Tính góc của đoạn đường so với trục x bằng hàm atan2. Góc này sẽ được sử dụng để tính toán toạ độ của các điểm để vẽ mũi tên.
* *x3, y3, x4, y4*: Tính toán toạ độ của các điểm (x3, y3) và (x4, y4) để vẽ mũi tên.
* *line(x2, y2, x3, y3):* Vẽ đoạn đường từ điểm (x2, y2) đến (x3, y3), tạo nên một phần của mũi tên.
* *line(x2, y2, x4, y4):* Vẽ đoạn đường từ điểm (x2, y2) đến (x4, y4), tạo nên phần còn lại của mũi tên

2.2.2.2.6. Thiết kế 2 dãy số nửa trái và phải được tách từ dãy số lớn (Dòng 2)

Khi ta thực hiện việc thao tác chuột để tạo ra dãy số từ các chức năng nằm ở bên phải khung. Sau đó ta thực hiện việc thao tác vào nút sắp xếp tăng dần hoặc giảm dần. Thì theo Thuật toán Merge Sort sẽ chia thành 2 mảng con (nửa bên trái và nửa bên phải) từ mảng ban đầu nằm ở Dòng 1. Đây là phần thiết kế giao diện cho việc chia mảng lớn đó thông qua hàm *veds\_chia()*

A screenshot of a black background

Description automatically generated

**Hình 2.14. Hai dãy số nửa trái và nửa phải của Dòng 2**

Toàn bộ nội dung hàm *veds\_chia()* như sau:

|  |  |  |
| --- | --- | --- |
| A screenshot of a computer program  Description automatically generated | A screenshot of a computer program  Description automatically generated | |
| A computer screen shot of a program code  Description automatically generated | | |
| A screenshot of a computer program  Description automatically generated | | A screenshot of a computer program  Description automatically generated |

**Hình 2.15. Toàn bộ nội dung của hàm veds\_chia() để vẽ các dãy số ở Dòng 2**

Các điều kiện if (n == ...): Các điều kiện này dựa trên số lượng phần tử n của mảng để xác định vị trí bắt đầu của mảng con và khoảng cách giữa chúng. Mỗi trường hợp tương ứng với một số lượng phần tử cụ thể.

* Vòng lặp for đầu tiên (vẽ mảng con bên trái):
* *setcolor(WHITE):* Đặt màu vẽ là trắng.
* Tính toán vị trí (x, y) của hình tròn tại vị trí bắt đầu của mảng con bên trái.
* *circle(x, y, RADIUS):* Vẽ hình tròn với bán kính RADIUS tại vị trí (x, y).
* *sprintf(buffer, "%d", a[i]):* Chuyển giá trị của phần tử thứ i thành chuỗi và lưu vào buffer.
* *outtextxy(x - textwidth(buffer) / 2, y - textheight(buffer) / 2, buffer):* In giá trị của phần tử lên trung tâm của hình tròn.
* *(mid + 2) \* (2 \* RADIUS + CIRCLE\_GAP) + ARRAY\_GAP + ...:* Tính toán vị trí bắt đầu của mảng con bên phải dựa trên vị trí kết thúc của mảng con bên trái và thêm vào một khoảng cách để ngăn cách giữa hai mảng con.
* Vòng lặp for thứ hai (vẽ mảng con bên phải): Tương tự như vòng lặp for đầu tiên.
* Gọi phần hàm vẽ mũi tên: Dùng hàm *arrow()* để vẽ mũi tên. Các điểm đầu và cuối của mũi tên được tính toán để trỏ đến các phần tử tương ứng của hai mảng con.

2.2.2.2.7. Thiết kế các dãy số được tách từ dãy số ở dòng 2 (Dòng 3)

Trong quá trình sắp xếp, đối với số lượng từ 3 phần tử trở lên thì việc sau khi in 2 mảng con nửa trái và nửa phải từ hàm *veds\_chia()* (Dòng 2), thì 2 mảng con đó sẽ tiếp tục chia tiếp nhỏ hơn thông qua hàm *veds\_sub().*

A screen shot of a video game

Description automatically generated

**Hình 2.16. Các dãy số được tách tiếp theo ở dòng 3**

Toàn bộ nội dung của hàm *veds\_sub():*

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**Hình 2.17. Toàn bộ nội dung của hàm veds\_sub() để vẽ các dãy số ở Dòng 3**

* Các điều kiện if (n == ...) ban đầu: Các điều kiện này dựa trên số lượng phần tử n của mảng để xác định vị trí bắt đầu vẽ hình tròn của phần tử đầu tiên. Mỗi trường hợp tương ứng với một số lượng phần tử cụ thể.
* Vẽ các mảng con nhỏ hơn (nửa trái và nửa phải) ở mảng bên trái:
* Vòng lặp đầu tiên vẽ mảng con bên trái từ *left* đến *mid\_sub* = *mid/2*. Sử dụng toado\_x và toado\_y để xác định vị trí và vẽ các phần tử trong mảng con bên trái.
* Dùng các điều kiện if-else để điều chỉnh khoảng cách giữa hai mảng con.
* Tiếp tục vòng lặp từ *mid\_sub + 1* đến *mid*, vẽ mảng con bên phải. Cũng tương tự như trên, sử dụng toado\_x và toado\_y để xác định vị trí và vẽ các phần tử trong mảng con bên phải.
* Đưa các điều kiện if-else để điều chỉnh khoảng cách giữa 2 mảng lớn.
* Vẽ các mảng con nhỏ hơn (nửa trái và nửa phải) ở mảng bên phải:
* Khai báo biến *mid\_sub\_right\_1 = mid + 1 + (right - mid - 1) / 2* và *mid\_sub\_right\_2 = mid\_sub\_right\_1 + 1* để lấy các giá trị ở nửa mảng bên phải còn lại.
* Vòng lặp đầu tiên vẽ mảng con bên trái từ *mid + 1* đến *mid\_sub\_right\_1*. Sử dụng toado\_x và toado\_y để xác định vị trí và vẽ các phần tử trong mảng con bên trái.
* Dùng các điều kiện if-else để điều chỉnh khoảng cách giữa hai mảng con.
* Tiếp tục vòng lặp từ *mid\_sub\_right\_2* đến *right*, vẽ mảng con bên phải. Cũng tương tự như trên, sử dụng toado\_x và toado\_y để xác định vị trí và vẽ các phần tử trong mảng con bên phải.
* Gọi hàm vẽ mũi tên: Sử dụng các điều kiện if-else dựa trên số lượng phần tử trong mảng để vẽ các mũi tên chỉ hướng sự chia mảng con.

2.2.2.2.8. Thiết kế các dãy số cho lần tách cuối cùng (Dòng 4 + 5)

Khi quá trình sắp xếp diễn ra ở lần tách của hàm *veds\_sub()*, thì ta sẽ tiếp tục chia nhỏ tiếp cho đến khi các phần tử trở thành là các phần tử đơn. Đối với số lượng phần tử bé hơn 9 thì đây là lần tách cuối cùng kết thúc ở dòng 4 (Line 4), nhưng đối với số lượng phần tử từ 9 và 10 thì sẽ diễn ra 2 lần tách, trên 2 dòng 4 và 5 (Line 4 và 5), các trường hợp này được chia đầy đủ trong hàm *veds\_sub\_sub()*.

A black background with white circles and arrows

Description automatically generated

**Hình 2.18. Nếu số lượng bé hơn 9 phần tử thì việc tách kết thúc ở dòng 4**

A black background with white circles and arrows

Description automatically generated

**Hình 2.19. Nếu số lượng từ 9 phần tử trở lên thì việc tách kết thúc ở dòng 5**

Toàn bộ nội dung của hàm *veds\_sub\_sub():*

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Hình 2.20. Toàn bộ nội dung của hàm veds\_sub\_sub()**

Mục đích của hàm *veds\_sub\_sub()* là vẽ ra các điểm đơn cho lần tách cuối cùng. Mỗi trường hợp số lượng phần tử khác nhau sẽ có cách lấy điểm khác nhau. Nhìn chung tổng quan như sau:

* toado\_x và toado\_y là tọa độ của điểm bắt đầu vẽ mảng con. Mỗi trường hợp sẽ đặt lại toado\_x và toado\_y cho phù hợp với khung hình.
* Sử dụng các vòng lặp để lấy từng phần tử và vẽ chúng thành các hình tròn.
* Gọi hàm *arrow()* để vẽ các mũi tên.

2.2.2.2.9. Thiết kế dãy số để sắp xếp (Dòng cuối cùng)

A screenshot of a computer

Description automatically generated

**Hình 2.21. Dãy số thực hiện sắp xếp trên dòng cuối cùng**

|  |  |
| --- | --- |
| A screenshot of a computer program  Description automatically generated | A screen shot of a computer program  Description automatically generated |

**Hình 2.22. Toàn bộ nội dung của hàm veds\_sort()**

2.3. XỬ LÝ SỰ KIỆN CHUỘT

Thư viện BGI (Borland Graphics Interface) cung cấp sẵn 1 số phương thức để có thể thao tác với chuột và bàn phím.

Thư viện BGI nhận các loại sự kiện chuột như:

* WM\_MOUSEMOVE: Thường được sử dụng để xử lý sự kiện khi chuột di chuyển, ví dụ như để vẽ các đối tượng theo vị trí mới của chuột.
* WM\_LBUTTONDBLCLK: Thường được sử dụng để xác định hành động khi người dùng nhấn đúp chuột trái.
* WM\_LBUTTONDOWN: Thường được sử dụng để xử lý sự kiện khi người dùng nhấn chuột trái.
* WM\_LBUTTONUP: Thường được sử dụng để xác định hành động khi người dùng nhả chuột trái.
* WM\_MBUTTONDBLCLK: Có thể được sử dụng để xác định hành động khi người dùng nhấn đúp nút giữa chuột.
* WM\_MBUTTONDOWN: Có thể được sử dụng để xử lý sự kiện khi người dùng nhấn nút giữa chuột.
* WM\_MBUTTONUP: Có thể được sử dụng để xác định hành động khi người dùng nhả nút giữa chuột.
* WM\_RBUTTONDBLCLK: Thường được sử dụng để xác định hành động khi người dùng nhấn đúp chuột phải.
* WM\_RBUTTONDOWN: Thường được sử dụng để xử lý sự kiện khi người dùng nhấn chuột phải.
* WM\_RBUTTONUP: Thường được sử dụng để xác định hành động khi người dùng nhả chuột phải.

Trong đó RBUTTON là chuột phải, LBUTTON là chuột trái và MBUTTON là chuột giữa.

Các loại sự kiện chuột ở trên sẽ giúp ta dễ dàng thao tác với các hàm sau:

* *bool ismouseclick(int kind)*: Trả về true khi có sự kiện click xảy ra mà chưa được xử lý, ngược lại là false.
* *void getmouseclick(int kind, int& x, int& y)*: Gán x và y là giá trị tọa độ chuột tại thời điểm được click và đánh dấu đã xử lý 1 sự kiện click. Nếu tại thời điểm gọi hàm mà không còn sự kiện click nào xảy ra thì giá trị của x và y là -1. Hàm này thường được dùng chung với hàm ismouseclick để xử lý sự kiện 1 cách linh hoạt hơn.
* *void clearmouseclick(int kind)*: Hủy các sự kiện chuột tùy thuộc vào loại sự kiện được truyền vào tham số, sau khi thực hiện xong hàm này thì giá trị của ismouseclick sẽ được reset về false.

Tương tự như với chuột, thư viện BGI cung cấp các hàm sau để xử lý sự kiện phím:

* *int getch()*: Trả về mã ASCII của ký tự vừa được nhấn, nếu có 1 ký tự đặc biệt được nhất ví dụ như: HOME, UP, DOWN, … thì lần đầu tiên hàm getch() sẽ trả về giá trị 0. Lần thứ 2 sẽ trả về 1 trong các mã sau: KEY\_HOME, KEY\_UP, KEY\_F5,…
* *int kbhit()*: Kiểm tra xem tại thời điểm gọi hàm có sự kiện phím nào chưa được xử lý không, trả về 0 không có sự kiện nào xảy ra.

Trong bài làm, hàm *mouse()* có chức năng xử lý sự kiện chuột trong một vòng lặp vô hạn. Mỗi lần chuột được click, hàm sẽ kiểm tra vị trí của chuột và thực hiện các hành động tương ứng với từng khu vực trên màn hình.

Tóm gọn nội dung hàm *mouse()* như sau:

A screen shot of a computer

Description automatically generated

**Hình 2.23. Tóm gọn nội dung chính của hàm mouse()**

Đầu tiên, ta khai báo biến *x\_mouse* và *y\_mouse* đểchứa giá trị tọa độ của điểm mà người dùng đã thực hiện sự kiện click chuột trái hoặc chuột phải.

* Khi có sự kiện click chuột trái (WM\_LBUTTONDOWN), hàm sẽ gọi getmouseclick(WM\_LBUTTONDOWN, x\_mouse, y\_mouse) để lấy giá trị tọa độ của điểm mà người dùng đã click chuột trái.
* Tương tự, khi có sự kiện click chuột phải (WM\_RBUTTONDOWN), hàm sẽ gọi getmouseclick(WM\_RBUTTONDOWN, x\_mouse, y\_mouse) để lấy giá trị tọa độ của điểm mà người dùng đã click chuột phải.
* Để làm rõ hơn vì việc 2 biến *x\_mouse* và ­*y\_mouse* thì ở trong hàm mouse() có viết một đoạn lệnh hàm *printf("\n(%d, %d)", x\_mouse, y\_mouse)* để in ra toạ độ của *x\_mouse* và *y\_mouse* ở màn hình cửa sổ console.

A black screen with white text

Description automatically generated

**Hình 2.24. Khi click chuột bất kỳ vào màn hình BGI thì trên Console Window sẽ hiển thị toạ độ chuột ở đó**

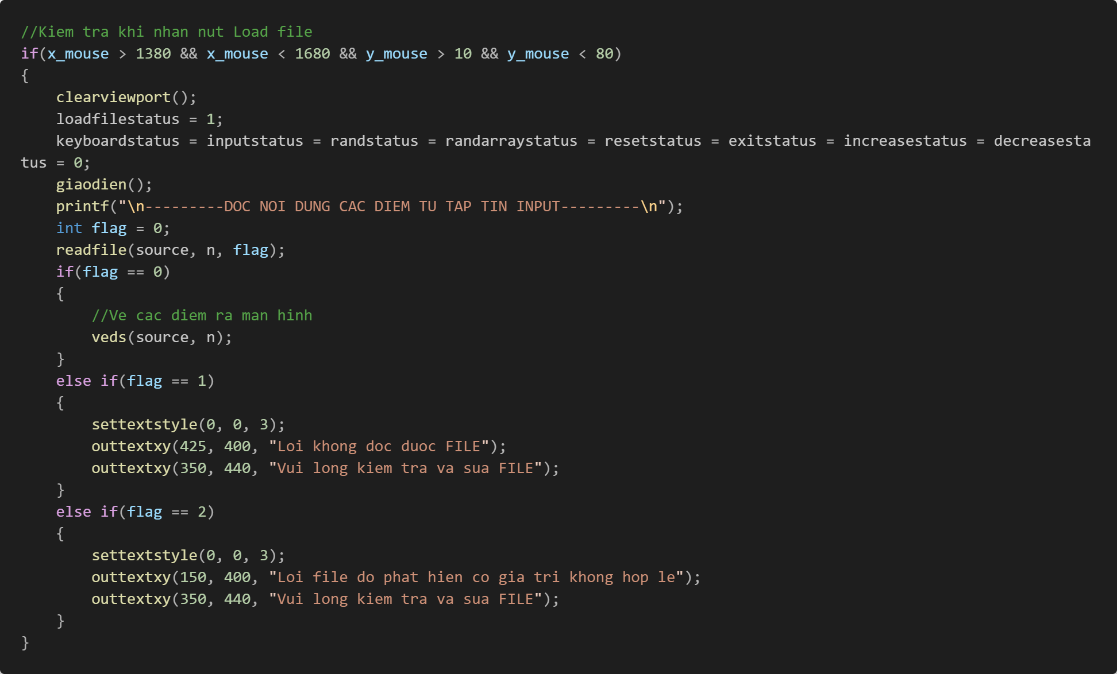
Biến *enable\_mouse* có vai trò quan trọng để kiểm soát vòng lặp của hàm mouse(). Nếu enable\_mouse bằng 1, chương trình sẽ thực hiện các kiểm tra và xử lý sự kiện chuột trong vòng lặp này. Khi có sự kiện chuột phải (WM\_RBUTTONDOWN), biến enable\_mouse sẽ được đặt thành 0, làm kết thúc vòng lặp và thoát khỏi hàm mouse(). Điều này dẫn đến kết thúc chương trình hoặc thực hiện một hành động cụ thể khi người dùng chọn thoát.

A black screen with text

Description automatically generated

**Hình 2.25. Các câu lệnh khi click chuột phải sẽ thoát vòng lặp while**

**2.3.1. Kiểm tra khi nhấn nút Load File**

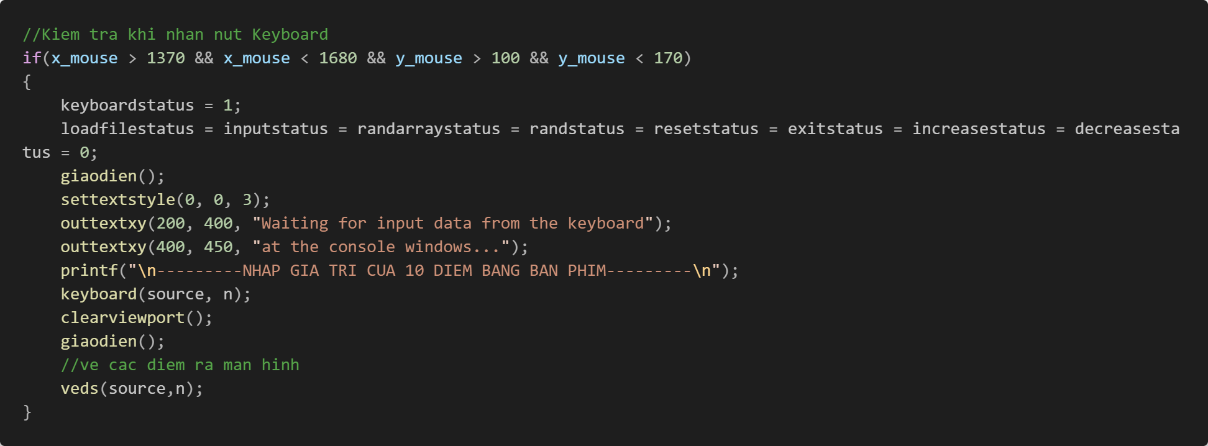


**Hình 2.26. Toàn bộ nội dung của khu vực nút Load File**

Mô tả như sau:

* Điều kiện nếu *x\_mouse > 1380 && x\_mouse < 1680 && y\_mouse > 10 && y\_mouse < 80*: Đây là khu vực của nút “Load File” nhằm kiểm tra xem sự kiện click chuột trái có xảy ra trong một khu vực này của màn hình không. Nếu điều kiện kiểm tra là đúng (chuột được click trong khu vực xác định), thì một loạt các lệnh bên trong điều kiện sẽ hoạt động.
* *clearviewport():* hàm này viết đầu tiên vì khi người dùng nhấn nút này sang nút khác thì sẽ xóa nội dung của view cũ (màn hình đồ họa) để chuẩn bị vẽ nội dung mới.
* Gán biến *loadfilestatus = 1*: Có nghĩa khi click chuột trái vào khu vực này, thì toàn bộ giao diện của nút “Load File” sẽ thay đổi màu. Ở hàm *giaodien()* ta có câu lệnh như sau: *setcolor(loadfilestatus? YELLOW : WHITE)*, thì biến *loadfilestatus* là true thì sẽ chuyển sang màu vàng.
* Các biến còn lại sẽ gán = 0 vì nút “Load File” được khởi động nên các nút khác sẽ không cần đổi màu.
* Ta gọi lại hàm *giaodien()* để tiến hành thay đổi giao diện màu của nút được nhấn.
* Biến *flag:* Ghi gán *flag* = 0 nghĩa là đọc file thành công, hiển thị dãy số thông qua hàm *veds()*, *flag* = 1 nghĩa là đọc file thất bại do không tìm thấy file và *flag* = 2 nghĩa là lỗi do có thể sai về dữ liệu.

### **2.3.2. Kiểm tra khi nhấn nút Keyboard**

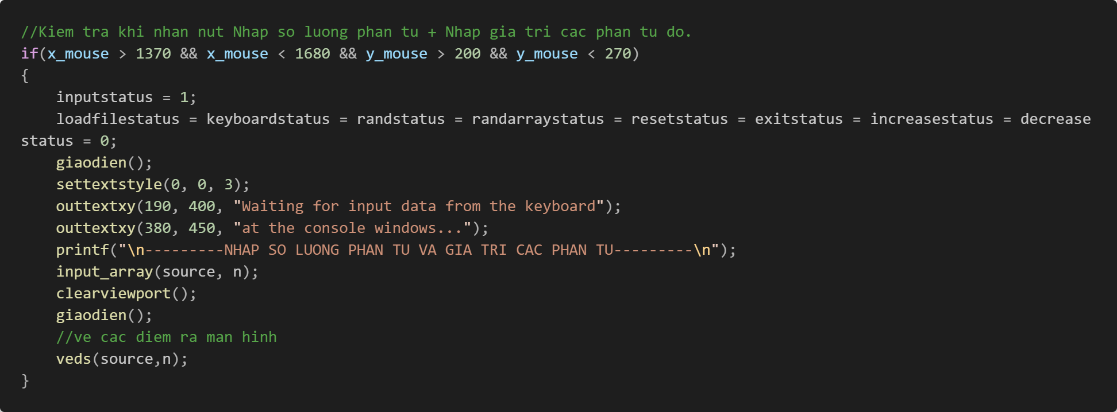


**Hình 2.27. Toàn bộ nội dung của khu vực nút Keyboard**

Mô tả như sau:

* Đầu tiên khi click chuột vào vùng có điều kiện như trong hình của nút Keyboard thì một loạt các câu lệnh trong điều kiện sẽ xảy ra.
* Tương tự khi click vào, nút Keyboard sẽ chuyển sang màu vàng (vì biến *keyboardstatus = 1*) và các nút khác màu trắng do gán = 0, gọi lại *giaodien()* để vẽ phần đổi màu này.
* Phần in ra giao diện đồ hoạ dòng chữ *“Waiting for input data from the keyboard at the console windows…”* ở giữa khung hình để chờ người dùng nhập dữ liệu từ bàn phím ở cửa sổ console.
* Nếu người dùng nhập dữ liệu xong từ hàm *keyboard()* thì hàm *clearviewport()* hoạt động để xoá phần dòng chữ ở trên để tạo ra các dãy số từ dữ liệu vừa nhập ở hàm *veds().*

### **2.3.3. Kiểm tra khi nhấn nút Input Array**



**Hình 2.28. Toàn bộ nội dung của khu vực nút Input Array**

Mô tả như sau: Khi click chuột vào vùng điều kiện của nút Input Array thì một loại các câu lệnh bên trong sẽ xảy ra, tương tự như nút Keyboard.

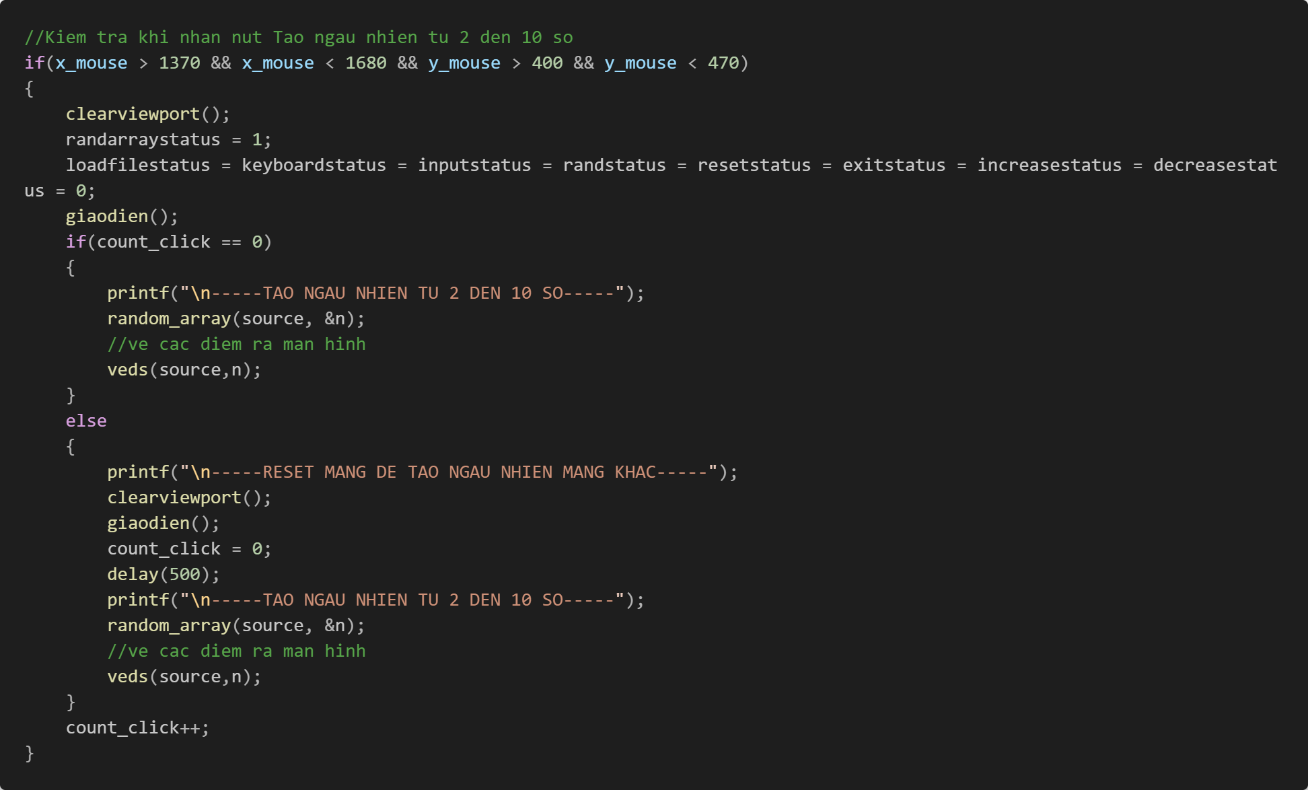
### **2.3.4. Kiểm tra khi nhấn nút tạo ngẫu nhiên 10 số**

A computer screen shot of a program

Description automatically generated

**Hình 2.29. Toàn bộ nội dung của khu vực nút Random**

### **2.3.5. Kiểm tra khi nhấn nút tạo ngẫu nhiên 2 – 10 số**



**Hình 2.30. Toàn bộ nội dung của khu vực Random Array**

Trong phần điều kiện này, biến *count\_click* để theo dõi việc nhấn vào vùng của nút “Random Array” hay chưa và ngăn việc khi nhấn nhiều lần nút “Random Array” sẽ bị chèn đồ hoạ của các dãy số lên nhau. Gán *count\_click* = 0 nghĩa là chưa nhấn lần nào. Ngược lại nếu *count\_click* = 1 nghĩa là đã nhấn nút này trước đó nên ở phần điều kiện của *count\_click* = 1 sẽ có hàm *clearviewport()* để reset màn hình các dãy số đã tạo trước đó để hiển thị các dãy số mới. Sau đó sẽ gán lại *count\_click* = 0.

### **2.3.6. Kiểm tra khi nhấn nút Reset**

Chức năng Reset để xoá dãy số trước đó để thực hiện việc hiển thị dãy số khác.

A computer screen shot of a program

Description automatically generated

**Hình 2.31. Toàn bộ nội dung của khu vực nút Reset**

### **2.3.7. Kiểm tra khi nhấn nút Exit**

A computer screen with text

Description automatically generated

**Hình 2.32. Toàn bộ nội dung của khu vực nút Exit**

Chức năng Exit để thoát cửa sổ đồ hoạ.

* closegraph(): Đóng cửa sổ đồ họa, kết thúc chế độ đồ họa.
* FreeConsole(): Giải phóng bộ nhớ của console.
* ExitProcess(0): Kết thúc quá trình thực thi của chương trình với mã trạng thái 0, tức là thoát chương trình một cách thành công.

### **2.3.8. Kiểm tra khi nhấn nút Sắp xếp tăng dần**

|  |  |
| --- | --- |
|  |  |

**Hình 2.33. Toàn bộ nội dung của khu vực nút Sắp xếp tăng dần**

### **2.3.9. Kiểm tra khi nhấn nút Sắp xếp giảm dần**

|  |  |
| --- | --- |
|  |  |

**Hình 2.34. Toàn bộ nội dung của khu vực nút Sắp xếp giảm dần**

## **2.4. PHÂN TÍCH THUẬT TOÁN**

### **2.4.1. Thiết lập tạo mảng**

#### **2.4.1.1. Tạo mảng bằng cách đọc dữ liệu từ file**

**A computer screen shot of a black screen with colorful text

Description automatically generated**

**Hình 2.35. Toàn bộ nội dung của hàm readfile()**

Hàm *readfile()* được thiết kế để đọc dữ liệu với một tệp tin (file) đầu vào có tên “points.inp” và lưu trữ các giá trị vào mảng a. Đồng thời cũng in về số lượng điểm và giá trị của chúng ở màn hình Console thông qua hàm *inmang().*

Đặt cờ *flag*: Nếu flag = 0 nghĩa là đọc file thành công, flag = 1 nghĩa là do không tìm thấy file, flag = 2 nghĩa là lỗi file do có 1 giá trị phần tử sai.

A computer screen with text and symbols

Description automatically generated

**Hình 2.36. Hàm inmang() để hiển thị giá trị trên Console Windows**

#### **2.4.1.2. Tạo mảng bằng cách nhập dữ liệu từ bàn phím (Trường hợp cố định 10 phần tử)**

Hàm *keyboard()* để cho phép người dùng có thể nhập dữ liệu từ bàn phím trên cửa sổ Console. Sau khi nhập đủ 10 giá trị phần tử thì sẽ tự động hiển thị các dãy số đồ hoạ trên màn hình BGI.

**A computer screen shot of text

Description automatically generated**

**Hình 2.37. Toàn bộ nội dung của hàm keyboard()**

#### **2.4.1.3. Tạo mảng bằng cách nhập dữ liệu từ bàn phím (Trường hợp từ 2 đến 10 phần tử)**

Hàm *input\_array()* được thiết kế để nhận dữ liệu đầu vào từ người dùng khi nhập dữ liệu từ bàn phím tương tự hàm *keyboard()* nhưng cho phép người dùng có thể nhập số lượng phần tử từ 2 đến 10 sau đó có thể nhập giá trị của từng phần tử đó (các giá trị phần tử cũng sẽ giới hạn từ 0 đến 9) và lưu trữ các giá trị vào mảng a.

|  |  |
| --- | --- |
|  |  |

**Hình 2.38. Toàn bộ nội dung của hàm input\_array()**

#### **2.4.1.4. Tạo mảng bằng cách tạo ngẫu nhiên (Trường hợp cố định 10 phần tử)**

|  |  |
| --- | --- |
|  |  |

**Hình 2.39. Toàn bộ nội dung của hàm tạo ngẫu nhiên 10 số**

Hàm *random\_num()* sử dụng hàm rand() để tạo ra một số ngẫu nhiên từ 0 đến 9 và kiểm tra xem số đó đã tồn tại trong mảng hay chưa (kiểm tra các số có bị trùng không). Nếu chưa tồn tại, số đó được thêm vào mảng a.

Để kiểm tra các số có bị trùng thì sẽ được bổ trợ bởi hàm *check\_dup()*.

#### **2.4.1.5. Tạo mảng bằng cách tạo ngẫu nhiên (Trường hợp từ 2-10 phần tử)**

**A screen shot of a computer program

Description automatically generated**

**Hình 2.40. Toàn bộ nội dung của hàm tạo ngẫu nhiên từ 2 đến 10 số**

### **2.4.2. Thiết lập đồ hoạ cho thuật toán MERGE SORT**

#### **2.4.2.1. Sắp xếp tăng dần**

* Hàm *merge\_tangdan()*

|  |  |
| --- | --- |
|  |  |

**Hình 2.41. Toàn bộ nội dung của hàm merge\_tangdan()**

Hàm *merge()* sẽ ghép hai dãy con lại thành một dãy có thứ tự, dãy con nhỏ nhất ta cần ghép sẽ có 1 phần tử. Thực chất đó là ghép hai mảng liền nhau đã sắp xếp thành một mảng có thứ tự.

Chi tiết từng đoạn lệnh như sau:

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int L[n1], R[n2];

    int temp[n1 + n2];

* n1 và n2 là kích thước của hai mảng con L và R (n1 là số lượng phần tử nửa trái và n2 là số lượng phần tử nửa phải).
* L và R là hai mảng con để lưu trữ dữ liệu.
* temp là mảng tạm để lưu trữ các phần tử khi trộn.

    for (int i = 0; i < n1; i++)

        L[i] = a[left + i];

    for (int j = 0; j < n2; j++)

        R[j] = a[mid + 1 + j];

    int i = 0, j = 0, k = 0;

    int index = left;

* Sao chép dữ liệu từ mảng con gốc a vào các mảng con L và R.
* Khai báo hai biến i và j chạy để duyệt mảng con và lưu k làm vị trí bắt đầu của mảng chính.
* Biến *index* để chạy từng phần tử dành riêng cho hàm *veds()*.

    while (i < n1 && j < n2)

    {

        if (L[i] <= R[j])

        {

            temp[k] = L[i];

            i++;

        }

        else

        {

            temp[k] = R[j];

            j++;

        }

        k++;

        // Hien thi mang sau moi buoc sap xep

        veds\_sort(a, index);

        delay(500);

        index++;

    }

* Trong khi cả hai mảng con L và R vẫn còn phần tử chưa trộn, so sánh từng phần tử và chọn phần tử nhỏ hơn để đưa vào mảng temp.
* Trong quá trình này, hàm *veds\_sort* được gọi để hiển thị mảng sau mỗi bước sắp xếp.

while (i < n1)

    {

        temp[k] = L[i];

        i++;

        k++;

        // Hien thi mang sau moi buoc sap xep

        veds\_sort(a, k);

        delay(500);

        k++;

    }

    while (j < n2)

    {

        temp[k] = R[j];

        j++;

        k++;

        // Hien thi mang sau moi buoc sap xep

        veds\_sort(a, k);

        delay(500);

        k++;

    }

* Sau vòng lặp trên, 1 trong 2 mảng đã duyệt hết phần tử, hoặc cả hai cùng hết thì sẽ xử lý các phần tử còn lại trong cả hai mảng con L và R.
* Trong quá trình này, hàm *veds\_sort* được gọi để hiển thị mảng sau mỗi bước sắp xếp.

    // Sao chep mang temp vao mang goc

    for (int i = 0; i < k; i++)

    {

        a[left + i] = temp[i];

        // Hien thi mang sau moi buoc sap xep

        veds\_sort(a, left + i);

        delay(500);

    }

* Sao chép dữ liệu từ mảng tạm temp vào mảng gốc a và gọi *veds\_sort* để hiển thị mảng cho bước sắp xếp này.

    // Hien thi mang sau khi tron

    for (int i = left; i <= right; i++)

     {

        setcolor(GREEN);

        int x = toado\_x + i \* (2 \* RADIUS + CIRCLE\_GAP);

        int y = toado\_y;

        circle(x, y, RADIUS);

        char buffer[5];

        sprintf(buffer, "%d", a[i]);

        outtextxy(x - textwidth(buffer) / 2, y - textheight(buffer) / 2, buffer);

        setcolor(WHITE);

    }

* Thiết kế hiển thị vòng tròn các phần tử của mảng sau khi trộn lên màn hình.
* Hàm *mergesort\_tangdan()*

Hàm mergeSort sẽ thực hiện:

* Chia nhỏ mảng ban đầu thành 2 mảng con.
* Thực hiện đệ quy mergeSort hai mảng con đó.
* Cuối cùng thì trộn hai mảng con lại bằng hàm merge đã viết ở trên

A computer screen shot of a code

Description automatically generated

**Hình 2.42. Toàn bộ nội dung của hàm mergesort\_tangdan()**

* Chia mảng thành hai phần bằng cách khai báo biến mid để tính giá trị trung bình của chỉ số left và right.
* Gọi đệ quy hàm mergesort\_tangdan để sắp xếp từng nửa của mảng.
* Gọi hàm merge\_tangdan để trộn hai mảng con đã được sắp xếp thành một mảng duy nhất.

A screenshot of a video game

Description automatically generated

**Hình 2.43. Dãy số ở dòng cuối cùng sau khi thực hiện sắp xếp tăng dần**

#### **2.4.2.2. Sắp xếp giảm dần**

* Hàm *merge\_giamdan()*

Tương tự như hàm *merge\_tangdan()* nhưng điều kiện để lấy phần tử sẽ khác. Phần điều kiện sẽ so sánh từng phần tử và chọn phần tử lớn hơn để đưa vào mảng temp.

    while (i < n1 && j < n2)

    {

        if (L[i] >= R[j])

        {

            temp[k] = L[i];

            i++;

        }

    }

* Hàm *mergesort\_giamdan()*

A computer screen shot of a code

Description automatically generated

**Hình 2.44. Toàn bộ nội dung của hàm mergesort\_giamdan()**

A screenshot of a video game

Description automatically generated

**Hình 2.45. Dãy số ở dòng cuối cùng sau khi thực hiện sắp xếp giảm dần**

# **Chương 3. KẾT QUẢ THỰC HIỆN**

Bước 1: Ta thực hiện nhấn một trong các nút khởi tạo mảng bất kỳ (VD: chọn đọc dữ liệu từ file). Dữ liệu có 8 phần tử cho mảng a gồm 2, 6, 3, 1, 0, 8, 7, 4 và thực hiện sắp xếp tăng dần theo giải thuật *MergeSort().*

**A screen shot of a computer

Description automatically generated**

**Hình 3.1. Vẽ dãy số từ dữ liệu file trên dòng 1**

Bước 2: Hàm MergeSort sẽ thực hiện việc tính mid, gọi 2 hàm và sắp xếp lặp đi lặp lại khi hoàn tất. Ta có mid = (left + right)/2 = 8/2 = 4 🡪 mid ở phần tử thứ 4 có giá trị 1.

* *mergesort(a, left, mid)* để chia mảng nửa bên trái (2, 6, 3, 1).
* *mergesort(a, mid + 1, right)* để chia mảng nửa bên phải (0, 8, 7, 4).

**A screen shot of a game

Description automatically generated**

**Hình 3.2. Vẽ mô phỏng các dãy số khi tách trên dòng 2**

* Ở mảng nửa bên trái {2,6,3,1} có mid = 4/2 = 2 🡪 mid ở phần tử 2 có giá trị là 6. Gọi 2 hàm đệ quy, ta được mảng nửa bên trái là {2, 6} và nửa bên phải là {3, 1}.
* Ở mảng nửa bên phải {0,8,7,4} có mid = (5 + 8)/2 = 6.5 ≈ 6 🡪 mid ở phần tử thứ 6 có giá trị là 8. Gọi 2 hàm đệ quy, ta được mảng nửa bên trái là {0, 8} và nửa bên phải là {7, 4}.

**A screen shot of a computer game

Description automatically generated**

**Hình 3.3. Vẽ mô phỏng các dãy số sau lần tách tiếp theo trên dòng 3**

* Mảng {2,6} ta tính mid = 2/2 = 1 (mid ở vị trí 1 có giá trị 2). Gọi 2 hàm đệ quy tách ra được mảng nửa trái 2 và nửa phải 6.
* Mảng {3,1} ta tính mid = (3+4)/2 = 3.5 = 3 (mid ở vị trí 3 có giá trị 3). Gọi 2 hàm đệ quy tách ra được mảng nửa trái 3 và nửa phải 1.
* Mảng {0,8} ta tính mid = (5+6)/2 = 5.5 = 5 (mid ở vị trí 5 có giá trị 0). Gọi 2 hàm đệ quy tách ra được mảng nửa trái 0 và nửa phải 8.
* Mảng {7,4} ta tính mid = (7+8)/2 = 7.5 = 7 (mid ở vị trí 7 có giá trị 7). Gọi 2 hàm đệ quy tách ra được mảng nửa trái 7 và nửa phải 4.

**A screenshot of a video game

Description automatically generated**

**Hình 3.4. Vẽ mô phỏng các dãy số cho lần tách tiếp theo trên dòng 4**

Bước 3: Quá trình sắp xếp và trộn bằng hàm *merge()* sẽ được diễn ra tại dãy số trên dòng cuối cùng.

**A screenshot of a video game

Description automatically generated**

**Hình 3.5. Vẽ dãy số trên dòng cuối cùng để thực hiện việc sắp xếp**

* Đầu tiên, thuật toán sẽ chạy 2 cặp số đầu {2,6} (khi giải thuật chạy thì các vòng tròn đổi màu đỏ nhằm xét và trộn nhau). Trộn xong sau đó sắp xếp lại thành {2,6}. Sắp xếp xong sẽ chuyển sang màu xanh để đánh dấu hoàn tất trong giai đoạn này.

**A screenshot of a video game

Description automatically generated**

**Hình 3.6. MergeSort sẽ sắp xếp cặp số đầu tiên**

* Tiếp theo đến cặp số {3,1}. Tương tự 2 số này sẽ trong trạng thái xét và trộn nhau. Trộn xong sẽ sắp xếp thành {1,3} và chuyển sang màu xanh nhằm sắp xếp xong trong giai đoạn này.

**A screenshot of a video game

Description automatically generated**

**Hình 3.7. Thuật toán sẽ sắp xếp 2 cặp số tiếp theo**

* Sau đó, thuật toán xét toàn bộ mảng bên trái {2,6,1,3} và trộn lại rồi sắp xếp lại thành {1,2,3,6} và chuyển sang màu xanh để đánh dấu sắp xếp xong trong giai đoạn này.

**A screenshot of a video game

Description automatically generated**

**Hình 3.8. Thuật toán gộp lại rồi sắp xếp, kết thúc sắp xếp mảng con bên trái**

* Đến nửa mảng bên phải, ta xét cặp số đầu tiên {0,8} và trộn lại rồi sắp xếp thành {0,8} và chuyển sang màu xanh để đánh dấu sắp xếp xong trong giai đoạn này.

A screenshot of a video game

Description automatically generated

**Hình 3.9. Tiến đến mảng nửa bên phải, thuật toán cũng sắp xếp 2 cặp số đầu tiên**

* Tiếp theo là xét cặp số tiếp theo {7,4} và trộn lại rồi sắp xếp thành {4,7}, chuyển sang màu xanh để đánh dấu sắp xếp xong trong giai đoạn này.

**A screenshot of a video game

Description automatically generated**

**Hình 3.10. Thuật toán xét 2 cặp số tiếp theo rồi sắp xếp**

* Sau đó, thuật toán xét toàn bộ mảng bên phải {0,8,4,7} và trộn lại rồi sắp xếp lại thành {0,4,7,8} và chuyển sang màu xanh để đánh dấu sắp xếp xong trong giai đoạn này.

A screenshot of a computer game

Description automatically generated

**Hình 3.11. Thuật toán sẽ gộp lại và sắp xếp. Kết thúc mảng nửa phải**

* Cuối cùng, thuật toán xét toàn bộ mảng nửa bên trái và nửa bên phải rồi trộn với nhau. Sau đó sắp xếp lại thành {0,1,2,3,4,5,6,7,8} rồi chuyển sang màu xanh 🡪 Hoàn tất.

A screenshot of a video game

Description automatically generated

**Hình 3.12. Thuật toán tiến hành gộp và sắp xếp mảng nửa trái và nửa phải**

# **Chương 4. KẾT LUẬN VÀ ĐÁNH GIÁ BẢN THÂN**

## **4.1. Kết quả đạt được**

* Hoàn thành quá trình mô phỏng sắp xếp trộn MergeSort.
* Hoàn thành các mục tiêu, kế hoạch đề ra như tạo các nút chức năng, khung,…
* Rèn luyện tốt hơn về các kỹ năng vẽ đồ hoạ của môn Kỹ thuật đồ hoạ.

## **4.2. Nhược điểm**

* Số lượng phần tử và giá trị của phần tử còn hạn chế (đều dưới 10).
* Vì mô phỏng cách thuật toán tách các dãy số nên cần vẽ nhiều dãy số để minh hoạ một cách trực quan khiến cho việc xử lý code trở nên rườm rà và dài dòng.

# **TÀI LIỆU THAM KHẢO**

[1] Đoàn Vũ Thịnh, *Bài giảng Kỹ thuật đồ họa*, 2019, Đại học Nha Trang.

[2] Nguyễn Đức Thuần, *Bài giảng cấu trúc dữ liệu và giải thuật*, 2023.

[3] Donald Hearn, *Computer Graphics, C Version (2nd Edition)*, 1996.

[4] Đoàn Vũ Thịnh, *Hướng dẫn giải chi tiết và lập trình Kỹ thuật đồ hoạ*, 2021.

[5] Lê Minh Hoàng, *Giải thuật và lập trình*, 2006.