

KHOA KỸ THUẬT VÀ CÔNG NGHỆ  
BỘ MÔN CÔNG NGHỆ THÔNG TIN



THỰC TẬP ĐỒ ÁN CƠ SỞ NGÀNH  
HỌC KỲ 1, NĂM HỌC 2023-2024  
**MÔ PHỎNG CÁC PHƯƠNG PHÁP  
SẮP XẾP CƠ BẢN**

*Giáo viên hướng dẫn:*  
Họ tên: Lê Minh Tự

*Sinh viên thực hiện:*  
Họ tên: Trần Quốc Trọng  
MSSV: 110121121  
Lớp: DA21TTB

*Trà Vinh, tháng..... năm.....*

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Trà Vinh, ngày ..... tháng ..... năm .....

**Giáo viên hướng dẫn**  
(Ký tên và ghi rõ họ tên)

[illegible]

**Thành viên hội đồng**  
(Ký tên và ghi rõ họ tên)

## LỜI CẢM ƠN

Trong thời gian làm đồ án cơ sở ngành, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô và bạn bè. Em xin gửi lời cảm ơn chân thành đến thầy Lê Minh Tự giảng viên của khoa Công Nghệ Thông Tin người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình làm khoá luận.

Em cũng xin chân thành cảm ơn các thầy cô giáo trong trường nói chung, các thầy cô trong khoa Công Nghệ Thông Tin nói riêng đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập. Với điều kiện thời gian cũng như kinh nghiệm còn hạn chế của một học viên dẫn đến không thể tránh được những thiếu sót. Em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để tôi có điều kiện bổ sung, nâng cao ý thức của mình, phục vụ tốt hơn công tác thực tế sau này.

## TÓM TẮT NIÊN LUẬN/ĐỒ ÁN CƠ SỞ NGÀNH

Hiện nay, trong đa số các hệ thống lưu trữ và quản lý dữ liệu, việc thực hiện tìm kiếm thông tin nhanh chóng đóng vai trò quan trọng, ví dụ như tra cứu từ điển, tìm sách trong thư viện, và các tác vụ khác. Để đạt được hiệu suất tìm kiếm nhanh chóng, việc sắp xếp và tổ chức dữ liệu trước là quan trọng, và một hệ thống ngăn nắp theo một trật tự cụ thể sẽ giúp chúng ta thực hiện các tác vụ này một cách hiệu quả.

Mặc dù việc học về sắp xếp và tìm kiếm là một môn cơ bản quan trọng đối với sinh viên chuyên ngành công nghệ thông tin, nhưng hiện vẫn còn thiếu các phần mềm hỗ trợ chất lượng để giúp sinh viên trong quá trình học. Nhận thức về nhu cầu này, em đã phát triển một sản phẩm mô phỏng trực quan về các thuật toán sắp xếp, giải thích cách chúng hoạt động từng bước một một cách dễ hiểu nhất.

Sản phẩm của em không chỉ giải thích cách hoạt động của thuật toán mà còn giúp hiểu rõ hơn về cấu trúc dữ liệu tương ứng. Ứng dụng sử dụng nhiều hình ảnh đồ họa sinh động để minh họa cách hoạt động của các cấu trúc dữ liệu một cách sinh động, nhằm giúp người học tiếp cận một cách dễ dàng mà không tốn quá nhiều thời gian đọc hiểu. Qua đó, ứng dụng giúp sinh viên học môn Cấu trúc dữ liệu và giải thuật trở nên thú vị, dễ tiếp cận hơn và có ứng dụng thực tế cao. Có nhiều giải thuật tìm kiếm và sắp xếp được tích hợp, và hiệu suất của từng giải thuật phụ thuộc vào đặc tính cụ thể của cấu trúc dữ liệu mà nó ảnh hưởng. Điều này giúp người dùng lựa chọn phương pháp sắp xếp phù hợp với nhu cầu và điều kiện cụ thể của dự án.

## MỞ ĐẦU

Để mô phỏng hiệu ứng sắp xếp trực quan bằng đồ họa C++, chúng ta cần thực hiện các bước sau:

**Tạo mảng chứa các phần tử:** Mảng này sẽ chứa các phần tử cần sắp xếp. Mỗi phần tử trong mảng sẽ được biểu diễn bằng một hình tròn trên đồ họa C++.

**Di chuyển các phần tử:** Các phần tử sẽ di chuyển một cách trực quan và đồng bộ trên đồ họa C++ theo các thuật toán sắp xếp.

**Biểu diễn các biến trên đồ họa:** Các biến trong thuật toán sẽ được biểu diễn trên đồ họa C++ một cách trực quan, phù hợp với vị trí của phần tử và có giá trị tương ứng với giá trị của biến trong thuật toán.

**Sắp xếp theo nhiều thuật toán khác nhau:** Chương trình sẽ mô phỏng 5 thuật toán sắp xếp tiêu biểu nhất trong cấu trúc dữ liệu và giải thuật. Mỗi thuật toán sẽ tạo ra một dãy số ngẫu nhiên tương ứng.

**Xóa mảng sau khi sắp xếp:** Khi sắp xếp hoàn tất, chương trình sẽ tự động xóa toàn bộ mảng và thoát khỏi cửa sổ đồ họa.

**Tạo ngẫu nhiên 10 phần tử:** Chương trình sẽ tạo ngẫu nhiên 10 phần tử, tương ứng với 10 hình tròn trong mô phỏng sắp xếp.

**Nhập các phần tử bằng tay:** Người dùng có thể nhập từng phần tử tương ứng với 10 hình tròn trong mô phỏng sắp xếp.

**Hiển thị dãy số chưa được sắp xếp:** Chương trình sẽ hiển thị dãy số chưa được sắp xếp trên cửa sổ đồ họa.

**Hiển thị quá trình sắp xếp và kết quả:** Chương trình sẽ hiển thị quá trình sắp xếp và kết quả sau khi sắp xếp xong của dãy số.

## MỤC LỤC

LỜI CẢM ƠN.....	4
MỞ ĐẦU .....	6
MỤC LỤC .....	7
DANH MỤC HÌNH .....	8
CHƯƠNG 1: TỔNG QUAN .....	9
1.1. Thuật toán.....	9
1.1.1. Khái niệm thuật toán sắp xếp .....	9
1.1.2. Các Đặc Trưng của Thuật Toán .....	9
1.2. Lập trình trên Code Block .....	9
1.2.1. Khái niệm .....	9
1.2.2. Kiến trúc ứng dụng CodeBlocks .....	10
CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT .....	11
2.1. Lý thuyết của các thuật toán sắp xếp:.....	11
2.1.1. Lý thuyết của Bubble Sort.....	11
2.1.2. Lý thuyết của Insertion Sort .....	11
2.1.3. Lý thuyết của Insertion Sort .....	11
2.1.4. Lý thuyết của Quick Sort.....	11
2.1.5. Lý thuyết của Merge Sort .....	12
2.2. Giới thiệu CodeBlocks và cách cài đặt:.....	12
2.2.1. Giới thiệu CodeBlocks .....	12
2.2.2. Các bước cài đặt CodeBlocks.....	12
CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU.....	14
3.1. Phân tử và các hàm có trong phân tử.....	14
3.2. Các hàm sắp xếp mảng .....	23
3.2.1. Hàm sắp xếp mảng trên đồ họa .....	23
3.2.2. Hàm Quick Sort .....	25
3.2.3. Hàm MergeSort .....	27
3.2.4. Hàm Bubble Sort .....	29
3.2.5. Hàm Insertion Sort .....	29
3.2.6. Hàm Selection Sort.....	30
CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU .....	32
4.1 Kết quả đạt được.....	32
4.1.1. Giao diện của màn hình chính .....	32
4.1.2. Giao diện chọn chọn thuật toán sắp xếp .....	32
4.1.3. Giao diện chọn cách sắp xếp .....	33
4.1.4. Giao diện đang sắp xếp.....	33
4.1.5. Giao diện sắp xếp hoàn tất.....	34
CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....	35
5.1. Kết quả đạt được.....	35
5.2. Hạn chế .....	35
5.3. Hướng phát triển.....	35
DANH MỤC TÀI LIỆU THAM KHẢO .....	36

## DANH MỤC HÌNH

Hình 2.1. Giao diện bắt đầu cài đặt .....	12
Hình 2.2. Giao diện đồng ý thỏa thuận.....	12
Hình 2.3. Giao diện chọn chỗ lưu file cài đặt.....	13
Hình 2.4. Giao diện đang cài đặt .....	13
Hình 2.5. Giao diện cài đặt hoàn tất .....	13
Hình 4. 1. Giao diện chính.....	32
Hình 4. 2. Giao diện chọn thuật toán sắp xếp.....	33
Hình 4. 3. Giao diện chọn chữ năng tang dần hoặc giảm dần .....	33
Hình 4. 4. Giao diện đang tiến hành sắp xếp .....	34
Hình 4. 5. Giao diện khi sắp xếp hoàn tất.....	34



## CHƯƠNG 1: TỔNG QUAN

### 1.1. Thuật toán

#### 1.1.1. Khái niệm thuật toán sắp xếp

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

Sắp xếp là quá trình bố trí lại các phần tử trong một tập hợp theo một trình tự nào đó nhằm mục đích giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn, cùng với độ chính xác cao.

#### 1.1.2. Các Đặc Trưng của Thuật Toán

Thuật toán có những đặc trưng cụ thể để đảm bảo tính hiệu quả và tính đúng đắn trong quá trình thực hiện. Dưới đây là các đặc trưng quan trọng của thuật toán:

- Đầu vào (Input): Một thuật toán có các giá trị đầu vào từ một tập đã được xác định từ lúc khởi tạo.
- Đầu ra (Output): Từ mỗi tập các giá trị đầu vào, thuật toán sẽ tạo ra các giá trị đầu ra, các giá trị đầu ra chính là nghiệm của bài toán.
- Tính dừng: Sau một số hữu hạn bước thuật toán phải dừng.
- Tính xác định: Ở mỗi bước, các bước thao tác phải hết sức rõ ràng, không gây nên sự nhập nhằng. Nói rõ hơn, trong cùng một điều kiện hai bộ xử lý cùng thực hiện một bước của thuật toán phải cho những kết quả như nhau.
- Tính đúng đắn (hiệu quả): Trước hết thuật toán cần đúng đắn, nghĩa là sau khi đưa dữ liệu vào thuật toán hoạt động và đưa ra kết quả như ý muốn.
- Tính phổ dụng: Thuật toán có thể giải bất kỳ một bài toán nào trong lớp các bài toán, cụ thể là thuật toán có thể có các đầu vào là các bộ dữ liệu khác nhau trong một miền xác định.

### 1.2. Lập trình trên Code Block

#### 1.2.1. Khái niệm

Code::Block môi trường phát triển tích hợp (IDE) miễn phí được xây dựng để đáp ứng các nhu cầu khắt khe nhất của người dùng trong lập trình các phần mềm được viết bằng ngôn ngữ C, C++ hoặc Fortran. Ngoài ra, còn có thể hoạt động được trên nhiều nền tảng nổi tiếng như MacOS, Linux hoặc Windows và hỗ trợ nhiều trình biên dịch như MS Visual C++ và GNU GCC.

Bên cạnh đó, Code::Blocks còn có giao diện thân thiện, tổ hợp tính năng đa dạng và có thể dễ dàng thêm các tính năng mới. Điều này có được là do thiết kế trên khung plugin cho phép FDE này mở rộng tính năng bằng cách cài đặt hoặc tạo các mã plugin.

Ví dụ như chức năng biên dịch và gỡ lỗi trên Code::Blocks được bổ sung bởi các plugin.

### **1.2.2. Kiến trúc ứng dụng CodeBlocks**

Kiến trúc của CodeBlocks được thiết kế để cung cấp một môi trường phát triển toàn diện và linh hoạt cho người phát triển phần mềm, từ việc soạn thảo mã nguồn đến quản lý và triển khai dự án.

## CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

### 2.1. Lý thuyết của các thuật toán sắp xếp:

#### 2.1.1. Lý thuyết của Bubble Sort

Bubble Sort là một thuật toán sắp xếp cơ bản được thiết kế để sắp xếp dãy số hoặc các phần tử của một mảng theo thứ tự tăng dần hoặc giảm dần. Thuật toán này hoạt động bằng cách so sánh từng cặp phần tử liên tiếp và đổi chỗ chúng nếu chúng không theo thứ tự mong muốn. Quá trình này được lặp lại cho đến khi toàn bộ mảng đã được sắp xếp.

Thuật toán Bubble Sort được sử dụng rộng rãi và giảng dạy trong các khóa học cơ bản về lập trình và cấu trúc dữ liệu. Mặc dù nó không hiệu quả cho các tập dữ liệu lớn, nhưng nó vẫn được sử dụng để giảng dạy về khái niệm sắp xếp và để giới thiệu người học với các thuật toán cơ bản.

#### 2.1.2. Lý thuyết của Insertion Sort

Insertion Sort là một thuật toán sắp xếp cơ bản và hiệu quả cho các tập dữ liệu nhỏ hoặc gần như đã sắp xếp. Thuật toán này hoạt động bằng cách chia mảng thành hai phần: một phần đã sắp xếp và một phần chưa sắp xếp. Trong mỗi bước, nó chọn một phần tử từ phần chưa sắp xếp và đặt nó vào đúng vị trí trong phần đã sắp xếp, làm tăng kích thước của phần đã sắp xếp.

Insertion Sort thường được giới thiệu như một cách tiếp cận trực quan và dễ hiểu cho người mới học về thuật toán và sắp xếp. Mặc dù không nổi tiếng như một số thuật toán khác, Insertion Sort vẫn đóng vai trò quan trọng trong giáo dục và là một phần của cơ sở kiến thức cho lập trình viên.

#### 2.1.3. Lý thuyết của Insertion Sort

- Selection Sort là một thuật toán sắp xếp cơ bản được thiết kế để sắp xếp dãy số hoặc các phần tử của một mảng. Thuật toán này hoạt động bằng cách tìm kiếm phần tử nhỏ nhất (hoặc lớn nhất, tùy thuộc vào chiều sắp xếp mong muốn) trong mảng và đổi chỗ nó với phần tử đầu tiên. Quá trình này được lặp lại cho đến khi toàn bộ mảng đã được sắp xếp.

#### 2.1.4. Lý thuyết của Quick Sort

Quick Sort là một thuật toán sắp xếp hiệu quả và phổ biến, thuộc loại thuật toán chia để trị. Nó được phát triển bởi Tony Hoare vào năm 1960 và là một trong những thuật toán sắp xếp nhanh nhất trong thực tế.

### 2.1.5. Lý thuyết của Merge Sort

Merge Sort là một thuật toán sắp xếp hiệu quả và ổn định, thuộc loại thuật toán chia để trị. Nó được phát triển để giải quyết vấn đề về hiệu suất của các thuật toán sắp xếp đơn, đặc biệt là trên các dữ liệu lớn.

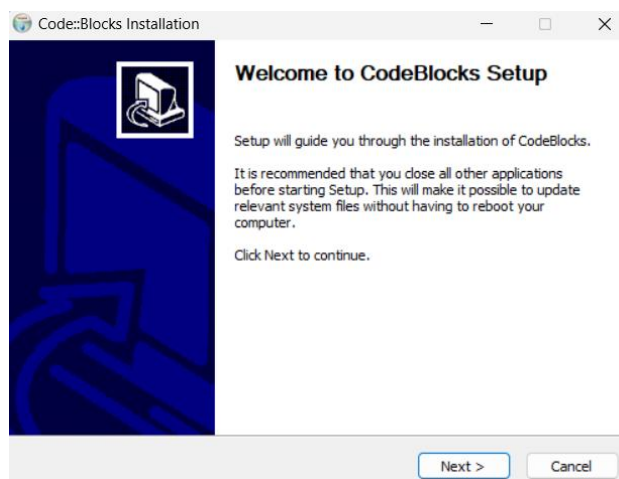
Thuật toán Merge Sort thường được coi là một trong những ý tưởng quan trọng và sáng tạo trong lĩnh vực thuật toán và nó đã đặt nền tảng cho nhiều ứng dụng và nghiên cứu trong thời gian tiếp theo.

## 2.2. Giới thiệu CodeBlocks và cách cài đặt:

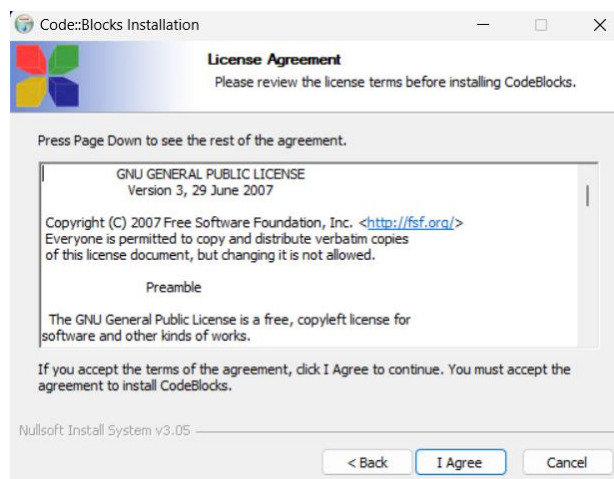
### 2.2.1. Giới thiệu CodeBlocks

CodeBlocks là một môi trường phát triển tích hợp (IDE) mạnh mẽ, đa nền tảng và mã nguồn mở, được thiết kế để hỗ trợ người phát triển trong quá trình viết và biên dịch mã nguồn. Với sự linh hoạt và đa chức năng, CodeBlocks là sự chọn lựa ưu việt cho các dự án lập trình C, C++, và Fortran.

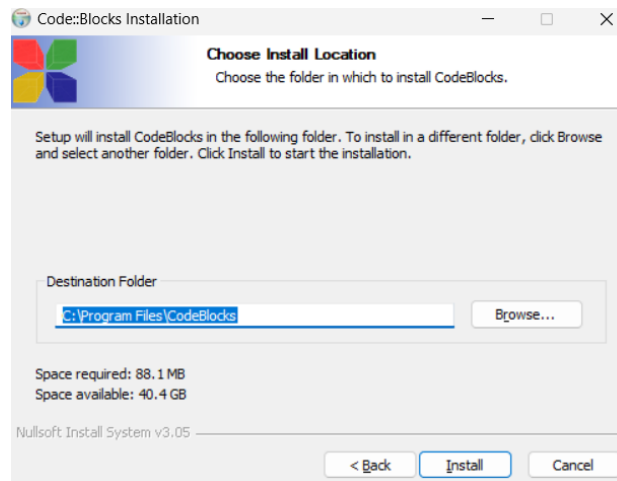
### 2.2.2. Các bước cài đặt CodeBlocks



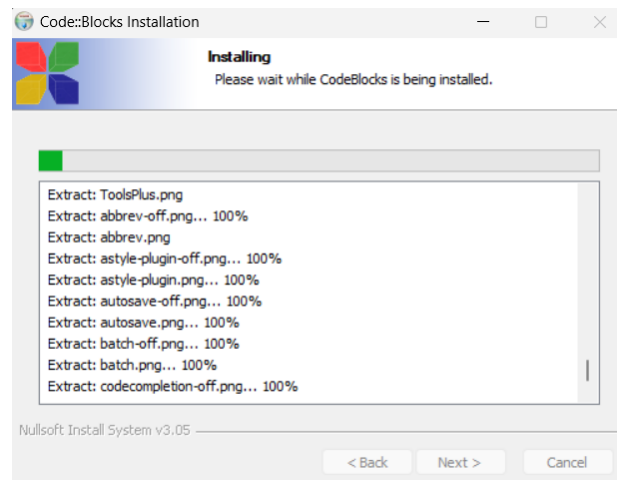
Hình 2.1. Giao diện bắt đầu cài đặt



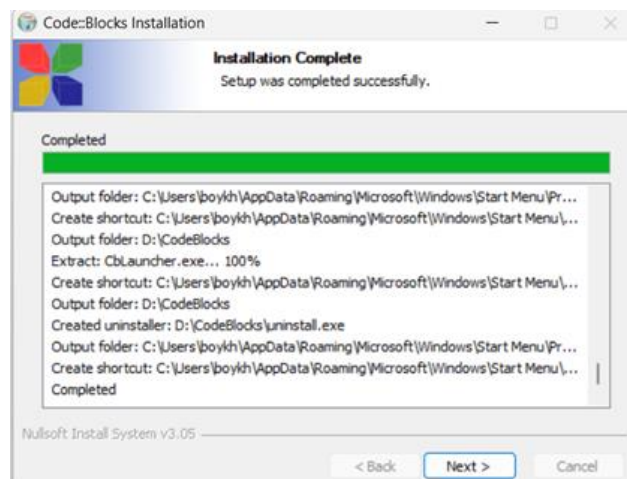
Hình 2.2. Giao diện đồng ý thỏa thuận



Hình 2.3. Giao diện chọn chỗ lưu file cài đặt



Hình 2.4. Giao diện đang cài đặt



Hình 2.5. Giao diện cài đặt hoàn tất

## CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU

### 3.1. Phân tử và các hàm có trong phân tử

Để tạo mảng sắp xếp thuật toán bằng đồ họa ta dùng thư viện đồ họa của `graphic.h` để thể hiện rõ quá trình sắp xếp bằng đồ họa.

Đầu tiên vẽ bảng xác định kích thước, vật thể và văn bản trong nền đồ họa. Và khởi tạo chiều cao và chiều rộng của màn hình đồ họa bằng `int screenWidth` và `int screenHeight`. Tạo các hình tròn và các số trong bảng đồ họa bằng các `void initCicre` và để in các hình tròn vào và các số ta dùng `void clearCircle`.

```
include <graphics.h>
#include <iostream>
#include <time.h>

void TextColor(int color);
void gotoXY(int i, int j);

using namespace std;
bool complete = false;
bool ASC = true;
int n = 10, choice;

int screenWidth = 1100;
int screenHeight = 600;
int startX = 100;
int startY = 200;
int MAX_VALUE = 100;
int circleRadius = 50;

void initCicre(int x, int y, int r, string s, int color) {
    //if(r <= circleRadius)
    r = circleRadius*0.8;
    setcolor(color);
    circle(x, y, r);
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
    setcolor(14); // Màu vàng
```

```

    int textHeight = 100; // Chiều cao của văn bản bên trong hình tròn
    int textWidth = s.length() * 15; // Độ rộng của văn bản
    int textX = x - textWidth / 2;
    int textY = y - textHeight / 8; // Điều chỉnh vị trí văn bản bên trong hình tròn
    outtextxy(textX, textY, const_cast<char*>(s.c_str())); // Vẽ văn bản lên màn hình
}

void clearCircle(int x, int y, int r, string s) {
    //if(r <= circleRadius)
    r = circleRadius*0.8;
    setcolor(BLACK);
    circle(x, y, r);
    setcolor(BLACK); // Màu vàng
    int textHeight = 100; // Chiều cao của văn bản bên trong hình tròn

    int textWidth = s.length() * 16; // Độ rộng của văn bản
    int textX = x - textWidth / 2;
    int textY = y - textHeight / 2; // Điều chỉnh vị trí văn bản bên trong hình tròn
    outtextxy(textX, textY, const_cast<char*>(s.c_str())); // Vẽ lên màn hình
}

```

*Bảng 3. 1. Bảng mô tả khởi tạo môi trường*

Để tạo khoảng cách giữa các hình tròn với nhau ta dùng hàm visualize để khoảng cách của 2 hình tròn cũng như đổi màu trong quá trình sắp xếp.

```

void visualize(int* arr, int x = -1, int y = -1, int z = -1) {
    cleardevice();
    int color;
    for (int i = 0; i < n; i++) {
        int circleX = startX + i * circleRadius * 2;
        int circleY = startY;
        if (complete || i == x || i == z) {
            color = GREEN; // Màu xanh lá cây
        } else if (i == y) {
            color = MAGENTA; // Màu tím
        } else {
            color = LIGHTGRAY; // Màu xám
        }
    }
}

```

```

        std::string text = std::to_string(arr[i]);
        initCicre(circleX, circleY, circleRadius*arr[i]/MAX_VALUE, text, color);
    }
}

```

*Bảng 3.2. Bảng mô tả hàm visualize tạo khoảng cách*

Để tạo dãy số ngẫu nhiên ta dùng hàm generateRandomArray để tạo một hàm số gồm các giá trị ngẫu nhiên.

```

int* generateRandomArray(int size, int minVal, int maxVal) {
    srand(time(0));
    int* arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = rand() % (maxVal - minVal + 1) + minVal;
    }
    return arr;
}

```

*Bảng 3. 3. Bảng mô tả của hàm generateRandomArray*

Để tạo dãy số từ dãy số được nhập từ bàn phím của người dùng ta dùng hàm inputArray để tạo một hàm số gồm các giá trị được nhập từ trước đó.

```

int* inputArray(int size) {
    int* arr = new int[size];
    std::cout << "Nhap mang " << size << " so nguyen: ";
    for (int i = 0; i < size; i++) {
        std::cin >> arr[i];
    }
    return arr;
}

```

*Bảng 3. 4. Bảng mô tả hàm inputArray*

Để xem dãy số số được nhập từ bàn phím của người dùng trước đó ta dùng hàm displayArray để xem dãy số đã được nhập từ người dùng.

```

// Hàm hiển thị mảng
void displayArray(int* arr) {
    visualize(arr);
}

```

*Bảng 3. 5. Bảng mô tả của hàm displayArray*

Để hiển thị các menu cũng như các lựa chọn của chức năng của ứng dụng ta dùng hàm ShowMenu để hiển thị các menu cho người dùng chọn.

```

void ShowMenu(int type = 0) {
    system("cls");
}

```



```

std::cout << "===== MENU =====" << std::endl;

if (type == 0) {
    std::cout << "1. Tao mang ngau nhien" << std::endl;
    std::cout << "2. Nhap mang" << std::endl;
}
else if(type == 1){
    std::cout << "1. Sap xep su dung Bubble Sort" << std::endl;
    std::cout << "2. Sap xep su dung Insertion Sort" << std::endl;
    std::cout << "3. Sap xep su dung Selection Sort" << std::endl;
    std::cout << "4. Sap xep su dung Quick Sort" << std::endl;
    std::cout << "5. Sap xep su dung Merge Sort" << std::endl;
    std::cout << "6. Hien thi mang" << std::endl;
}
else {
    std::cout << "1. Tang dan" << std::endl;
    std::cout << "2. Giam dan" << std::endl;
}

std::cout << "0. Thoat" << std::endl;
std::cout << "===== " << std::endl;
std::cout << "Nhap chuc nang: ";
}

```

*Bảng 3. 6. Bảng mô tả thuật hàm ShowMenu*

Để xét các lựa chọn của người dùng trong bảng menu cũng như kiểm tra các giá trị các giá trị được nhập vào có hợp lệ hay không. Ta dùng hàm Show để kiểm tra và hiện ra các bản menu để chọn.

```

void Show(int* arr) {
    do {
        ShowMenu(1);
        cin >> choice;
        switch (choice) {

```

```
case 0: {
    return;
}
case 1: {
    if (arr != nullptr) {
        int choice2;
        do {
            ShowMenu(2);
            cin >> choice2;
            if (choice2 == 1) {
                ASC = true;
            }
            else if (choice2 == 2) {
                ASC = false;
            }
        } while (choice2 < 0 || choice2 > 2);
        complete = false;
        bubbleSort(arr);
        complete = true;
        visualize(arr);
        std::cout << "Mang da duoc sap xep su dung Bubble Sort." << std::endl;
    }
    else {
        std::cout << "Mang chua duoc khoi tao." << std::endl;
    }
    break;
}
case 2: {
    if (arr != nullptr) {
        int choice2;
```

```

do {
    ShowMenu(2);
    cin >> choice2;
    if (choice2 == 1) {
        ASC = true;
    }
    else if (choice2 == 2) {
        ASC = false;
    }
} while (choice2 < 0 || choice2 > 2);
complete = false;
insertionSort(arr);
complete = true;
visualize(arr);
std::cout << "Mang da duoc sap xep su dung Insertion Sort." << std::endl;
}
else {
    std::cout << "Mang chua duoc khoi tao." << std::endl;
}
break;
}
case 3: {
    if (arr != nullptr) {
        int choice2;
        do {
            ShowMenu(2);
            cin >> choice2;
            if (choice2 == 1) {
                ASC = true;
            }

```

```

        else if (choice2 == 2) {
            ASC = false;
        }
    } while (choice2 < 0 || choice2 > 2);
    complete = false;
    selectionSort(arr);
    complete = true;
    visualize(arr);
    std::cout << "Mang da duoc sap xep su dung Selection Sort." << std::endl;
}
else {
    std::cout << "Mang chua duoc khoi tao." << std::endl;
}
break;
}
case 4: {
    if (arr != nullptr) {
        int choice2;
        do {
            ShowMenu(2);
            cin >> choice2;
            if (choice2 == 1) {
                ASC = true;
            }
            else if (choice2 == 2) {
                ASC = false;
            }
        } while (choice2 < 0 || choice2 > 2);
        complete = false;
        quickSort(arr, 0, n - 1);
    }
}

```

```
        complete = true;

        visualize(arr);

        std::cout << "Mang da duoc sap xep su dung Quick Sort." << std::endl;
    }
    else {
        std::cout << "Mang chua duoc khoi tao." << std::endl;
    }
    break;
}

case 5: {
    if (arr != nullptr) {
        int choice2;
        do {
            ShowMenu(2);
            cin >> choice2;
            if (choice2 == 1) {
                ASC = true;
            }
            else if (choice2 == 2) {
                ASC = false;
            }
        } while (choice2 < 0 || choice2 > 2);
        complete = false;
        mergeSort(arr, 0, n - 1);
        complete = true;
        visualize(arr);
        std::cout << "Mang da duoc sap xep su dung Merge Sort." << std::endl;
    }
    else {
        std::cout << "Mang chua duoc khoi tao." << std::endl;
```

```

    }
    break;
}
case 6: {
    if (arr != nullptr) {
        std::cout << "Array: ";
        displayArray(arr);
        system("pause");
    }
    else {
        std::cout << "Mang chua duoc khoi tao." << std::endl;
    }
    break;
}
default: {
    std::cout << "Gia tri khong hop le. Vui long kiem tra lai!" << std::endl;
    break;
}
}
if (choice > 0 && choice < 6) {
    delete[] arr;
    arr = nullptr;
    //getch();
    return;
}

} while (choice != 0);
}

```

Bảng 3. 7. Bảng mô tả thuật hàm ShowMenu

### 3.2. Các hàm sắp xếp mảng

#### 3.2.1. Hàm sắp xếp mảng trên đồ họa

Để thực hiện các quá trình sắp xếp được diễn ra trên nền đồ họa của C++ ta dùng hàm inplace khởi tạo môi trường và các quá trình sắp xếp được diễn ra trên nền đồ họa.

```
void inplace
t(int* input, int n)
{
    for (int i = 1; i < n; i++)
    {
        int childIndex = i;
        int parentIndex = (childIndex - 1) / 2;

        while (childIndex > 0)
        {
            if (input[childIndex] > input[parentIndex] == ASC)
            {
                swap(input, parentIndex, childIndex, GREEN, MAGENTA);
            }
            else
            {
                break;
            }

            visualize(input, parentIndex, childIndex);
            Sleep(200);

            childIndex = parentIndex;
            parentIndex = (childIndex - 1) / 2;
        }
    }
}
```

```

    }
    for (int heapLast = n - 1; heapLast >= 0; heapLast--)
    {
        int temp = input[0];
        input[0] = input[heapLast];
        input[heapLast] = temp;
        int parentIndex = 0;
        int leftChildIndex = 2 * parentIndex + 1;
        int rightChildIndex = 2 * parentIndex + 2;

        while (leftChildIndex < heapLast == ASC)
        {
            int maxIndex = parentIndex;

            if (input[leftChildIndex] > input[maxIndex] == ASC)
            {
                maxIndex = leftChildIndex;
            }

            if ((rightChildIndex < heapLast == ASC) && (input[rightChildIndex] >
input[maxIndex] == ASC))
            {
                maxIndex = rightChildIndex;
            }

            if (maxIndex == parentIndex)
            {
                break;
            }

            visualize(input, maxIndex, parentIndex, heapLast);
            swap(input, parentIndex, maxIndex, GREEN, MAGENTA);
            //Sleep(200);
        }
    }

```



```

        parentIndex = maxIndex;

        leftChildIndex = 2 * parentIndex + 1;
        rightChildIndex = 2 * parentIndex + 2;

    }

}

}

```

*Bảng 3.2. 1. Bảng mô tả thuật toán của hàm inplac*

### 3.2.2. Hàm Quick Sort

```

int partition_array(int *a, int si, int ei)
{
    int count_small = 0;

    for (int i = (si + 1); i <= ei; i++)
    {
        if (a[i] <= a[si] == ASC)
        {
            count_small++;
        }
    }

    int c = si + count_small;
    visualize(a, c, si);
    swap(a, c, si, GREEN, MAGENTA);
    int i = si, j = ei;

    while (i < c && j > c)
    {
        if (a[i] <= a[c] == ASC)
        {
            i++;
        }
    }
}

```

```

        else if (a[j] > a[c] == ASC)
        {
            j--;
        }
        else
        {
            visualize(a, i, j);
            swap(a, i, j, GREEN, MAGENTA);
            //Sleep(200);

            i++;
            j--;
        }
    }
    return c;
}

void quickSort(int *a, int si, int ei)
{
    if (si >= ei)
    {
        return;
    }

    int c = partition_array(a, si, ei);
    quickSort(a, si, c - 1);
    quickSort(a, c + 1, ei);
}

```

*Bảng 3.2. 2. Hàm Quick Sort*

### 3.2.3. Hàm MergeSort

```
void merge2SortedArrays(int *a, int si, int ei)
{
    int size_output = (ei - si) + 1;
    int* output = new int[size_output];

    int mid = (si + ei) / 2;
    int i = si, j = mid + 1, k = 0;
    while (i <= mid && j <= ei)
    {
        if (a[i] <= a[j] == ASC)
        {
            output[k] = a[i];
            visualize(a, i, j);
            i++;
            k++;
        }
        else
        {
            output[k] = a[j];
            visualize(a, i, j);
            j++;
            k++;
        }
    }

    while (i <= mid)
    {
        output[k] = a[i];
        visualize(a, -1, i);
    }
}
```

```
        i++;
        k++;
    }
    while (j <= ei)
    {
        output[k] = a[j];
        visualize(a, -1, j);
        j++;
        k++;
    }
    int x = 0;
    for (int l = si; l <= ei; l++)
    {
        a[l] = output[x];
        visualize(a, l);
        Sleep(200);
        x++;
    }
    delete[] output;
}

void mergeSort(int *a, int si, int ei)
{
    if (si >= ei)
    {
        return;
    }
    int mid = (si + ei) / 2;

    mergeSort(a, si, mid);
```

```
mergeSort(a, mid + 1, ei);

merge2SortedArrays(a, si, ei);
}
```

*Bảng 3.2. 3.. Hàm MergeSort*

#### 3.2.4. Hàm Bubble Sort

```
void bubbleSort(int* arr)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - 1 - i; j++)
        {
            if (arr[j + 1] < arr[j] == ASC)
            {
                //int temp = arr[j];
                //arr[j] = arr[j + 1];
                //arr[j + 1] = temp;
                visualize(arr, j + 1, j, n - i);
                swap(arr, j, j + 1, GREEN, MAGENTA);
            }
            //Sleep(200);
        }
    }
}
```

*Bảng 3.2. 4. Hàm MergeSort*

#### 3.2.5. Hàm Insertion Sort

```
void insertionSort(int* arr) {
    for (int i = 1; i < n; i++) {
        int j = i - 1;
        int temp = arr[i];
```

```

while (j >= 0 && arr[j] > temp == ASC) {
    arr[j + 1] = arr[j];
    visualize(arr, n, i, j + 1);
    // Thực hiện swap và ngủ 50ms sau mỗi bước di chuyển
    swap(arr, j, j + 1, MAGENTA, GREEN, false);
    Sleep(50);
    j--;
}

arr[j + 1] = temp;

// Hiển thị trạng thái của mảng sau mỗi bước hoàn tất
visualize(arr);
}
}

```

*Bảng 3.2. 5. Hàm Insertion Sort*

### 3.2.6. Hàm Selection Sort

```

void selectionSort(int* arr)
{
    int minIndex;
    for (int i = 0; i < n - 1; i++)
    {
        minIndex = i;
        for (int j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[minIndex] == ASC)
            {
                minIndex = j;
                visualize(arr, n, i, minIndex);
            }
        }
    }
}

```

```
    }  
    Sleep(200);  
    }  
    swap(arr, i, minIndex, GREEN, MAGENTA);  
    }  
}  
  
int* generateRandomArray(int size, int minVal, int maxVal) {  
    srand(time(0));  
    int* arr = new int[size];  
    for (int i = 0; i < size; i++) {  
        arr[i] = rand() % (maxVal - minVal + 1) + minVal;  
    }  
    return arr;  
}
```

*Bảng 3.2. 6. Hàm Selection Sort*

## CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

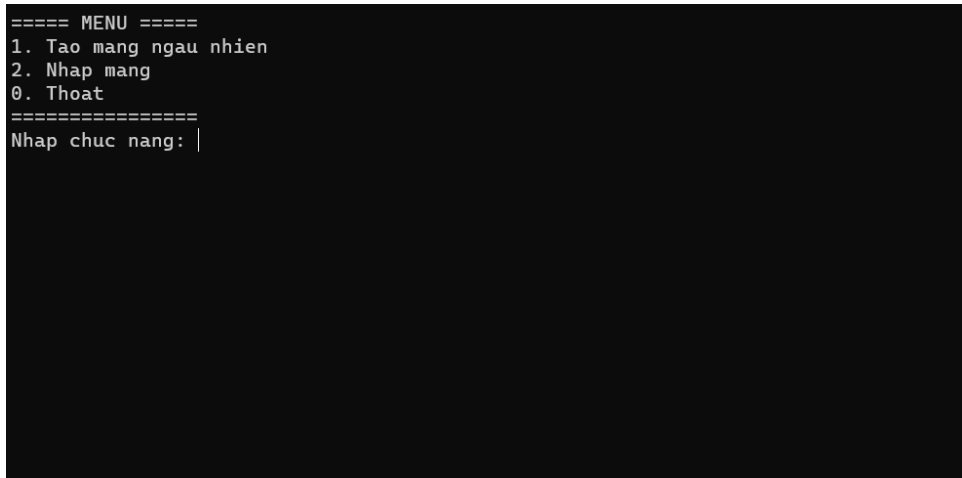
### 4.1 Kết quả đạt được

#### 4.1.1. Giao diện của màn hình chính

Giao diện chính của chương trình bao gồm 3 chức năng chính.

- Tạo mảng ngẫu nhiên
- Nhập mảng từ bàn phím do người dùng nhập
- Và thoát khỏi chương trình

```
===== MENU =====  
1. Tạo mảng ngẫu nhiên  
2. Nhập mảng  
0. Thoát  
=====
```



```
Nhập chức năng: |
```

*Hình 4. 1. Giao diện chính*

#### 4.1.2. Giao diện chọn chọn thuật toán sắp xếp

Giao diện được sẽ hiện ra sau khi người dùng đã chọn một trong hai cách để nhập mảng số nguyên, sau đó để người dùng có thể lựa chọn các thuật toán sắp xếp, hiển thị lại những số vừa được nhập từ bàn phím hoặc được tạo ra từ mảng ngẫu nhiên và thoát khỏi chương trình.



```
==== MENU ====
1. Sáp xếp sử dụng Bubble Sort
2. Sáp xếp sử dụng Insertion Sort
3. Sáp xếp sử dụng Selection Sort
4. Sáp xếp sử dụng Quick Sort
5. Sáp xếp sử dụng Merge Sort
6. Hiện thị mảng
0. Thoát
=====
Nhập chức năng: |
```

Hình 4. 2. Giao diện chọn thuật toán sắp xếp

#### 4.1.3. Giao diện chọn cách sắp xếp

Giao diện này có bao gồm ba chức năng chính sau chính, sắp xếp tăng dần hoặc giảm dần khi người dùng chọn các thuật toán sắp xếp hoàn tất và cuối cùng là thoát khỏi chương trình.

```
==== MENU ====
1. Tăng dần
2. Giảm dần
0. Thoát
=====
Nhập chức năng: |
```

Hình 4. 3. Giao diện chọn chức năng tăng dần hoặc giảm dần

#### 4.1.4. Giao diện đang sắp xếp

Chương trình đang thực hiện quá trình sắp xếp trên nền đồ họa, khi sắp xếp các vòng tròn được chia ra làm ba màu bao gồm trắng, xanh lá cây và màu tím. Để đại diện cho các số đang được sắp xếp và được chọn đang sắp xếp.



*Hình 4. 4. Giao diện đang tiến hành sắp xếp*

#### **4.1.5. Giao diện sắp xếp hoàn tất**

Khi sắp xếp hoàn tất các vòng tròn sẽ chuyển sang màu xanh lá cây để cho người dùng biết rằng quá trình sắp xếp đã hoàn tất.



*Hình 4. 5. Giao diện khi sắp xếp hoàn tất*

## **CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

### **5.1. Kết quả đạt được**

- Thực hiện được quá trình mô phỏng các thuật toán sắp xếp
- Tích hợp được năm thuật toán sắp xếp
- Giao diện dễ dàng sử dụng đối với người dùng

### **5.2. Hạn chế**

- Giao diện đồ họa chưa thực sự mượt mà, đẹp và ấn tượng
- Quá trình sắp xếp vẫn chưa diễn ra đồng đều
- Vẫn còn chưa thích hợp được nhiều thuật toán khó
- Vẫn còn thiếu nhiều chức năng

### **5.3. Hướng phát triển**

- Thêm được nhiều phần tử khi sắp xếp.
- Tăng cường thêm một số hiệu ứng chuyển động
- Làm cho giao diện dễ dàng sử dụng và đẹp hơn.

## DANH MỤC TÀI LIỆU THAM KHẢO

<https://www.youtube.com/watch?app=desktop&v=ogaayYfh984&t=15s&fbclid=IwAR08uMKSDFMYYVSqmbYaUI0a3mYtFRInvVkp61sGZQZ9D639RZLEFiG4j5K8>

<https://blog.luyencode.net/cai-dat-thu-vien-graphics-h/>

<https://www.geeksforgeeks.org/selection-sort/>

<https://www.geeksforgeeks.org/bubble-sort/>

<https://www.geeksforgeeks.org/insertion-sort/>

<https://www.geeksforgeeks.org/merge-sort/>

<https://www.geeksforgeeks.org/quick-sort/>