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…………….***

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 bộ môn**

(Ký tên và ghi rõ họ tên )

**Nhận xét của thành viên hội Đồng**

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

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

**Thành viên hội Đồng**

(Ký tên và ghi rõ họ tên)

**LỜI CẢM ƠN**

Em cám ơn thầy Lê Minh Tự đã hướng dẫn em làm bài báo đồ án cơ sở ngành.

Cám ôn thầy (cô) đã đọc nếu có thiếu xót rất mong thầy (cô) bỏ qua.

Xin chân thành cảm ơn thầy (cô).

**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 hoạ 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, đồ họa ta cần tạo ra 1 mảng để chứa các phần tử. Và cho các phần tử di chuyển một cách trực quan, đồng bộ trên đồ họa C++ theo thuật toán cần sắp xếp.

Ngoài ra các biến trong thuật toán còn được biễu diễn trên đồ họa của C++ một cách trực quan sao cho phù hợp với vị trí của phần tử và có giá trị tương ứng với giá trị của biến trong thuật toán

Bao gồm 5 thuật toán sắp xếp tiêu biểu nhất trong cấu trúc dữ liệu và giải thuật.Tạo mới ngẫu nhiên một dãy số các phần tử theo từng thuật toán sắp xếp tương ứng.

Xóa mảng: Khi thực hiện xóa mảng tất cả các phần tử trên màn hình chính sẽ không còn xuất hiện nữa

Tạo ngẫu nhiên với số phần tử tương ứng

Nhập bằng tay: nhập từng phần tử với số phần tử tương ứng khi nhập vào ô số phần tử

Dãy số chưa sắp xếp: Hiển thị dãy số trước khi sắp xếp.

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

**CHƯƠNG 1: TỔNG QUAN**

* 1. **Thuật toán**

**1.1.1 Khái niệm thuật toán sắp xếp**

 - Thuật toán là một tập hữu hạn các chỉ thị hay phương cách được định nghĩa rõ

ràng cho việc hoàn tất một sự việc từ một trạng thái ban đầu cho trước; khi các chỉ thị

này được áp dụng triệt để thì sẽ dẫn đến kết quả sau cùng như đã dự đoán.

 - Sắp xếp là một quá trình biến đổi một danh sách các đối tượng thành một danh

sách thỏa mãn một thứ tự xác định nào đó.

**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:

- Tính Xác Định: Thuật toán phải được định nghĩa rõ ràng và thực hiện một cách dễ hiểu để đạt được một kết quả nhất định.

- Tính Hữu Hạn: Thuật toán phải có một số bước xác định và phải kết thúc sau một số lượng bước hữu hạn.

- Tính Kết Quả: Với dữ liệu đầu vào phù hợp, thuật toán phải tạo ra kết quả đúng theo yêu cầu đã đặt ra.

- Tính Phổ Dụng: Thuật toán có khả năng áp dụng cho nhiều bài toán khác nhau có cùng cấu trúc, nhưng với dữ liệu đầu vào khác nhau.

- Tính Hiệu Quả: Thuật toán nên được thiết kế đơn giản, dễ hiểu, và tối ưu hóa về mặt bộ nhớ và thời gian thực hiện.

- Tính Hình Thức: Mỗi bước trong thuật toán phải được thực hiện đúng theo kịch bản đã được định sẵn (chương trình) mà không cần phải suy đoán hay giả định về mục tiêu cuối cùng.

* 1. **Lập trình trên Code Block**

**1.2.1. Khái niệm**

- Code::Blocks là một môi trường phát triển tích hợp (IDE) dùng cho việc phát triển phần mềm bằng nhiều ngôn ngữ lập trình như C, C++, và Fortran.

**1.2.2. Kiến trúc ứng dụng CodeBlocks**

- Kiến trúc của Code::Blocks 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.

**2.1.4 Lý thuyết của Megre 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.

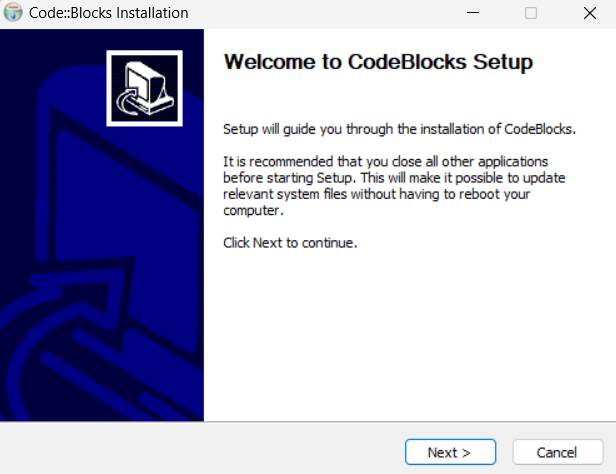
- Thuật toán Merge Sort thường được coi là một trong những ý tưởng quan trọng và sáng tạo trong lĩnh vực thuật toán và nó đã đặt nền tảng cho nhiều ứng dụng và nghiên cứu trong thời gian tiếp theo.

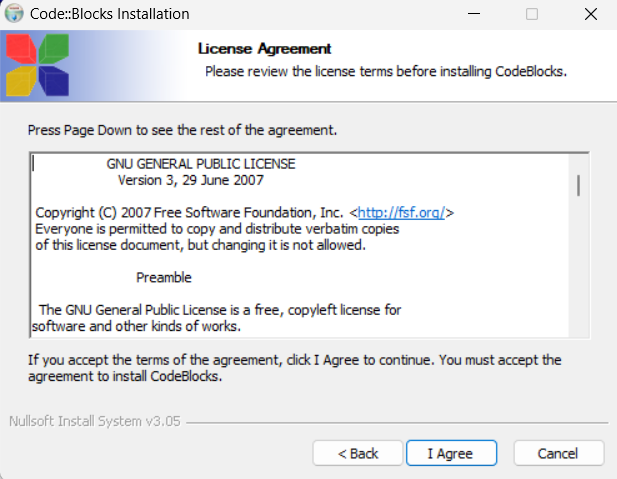
**2.2 Giới thiệu CodeBlocks và cách cài đặt:**

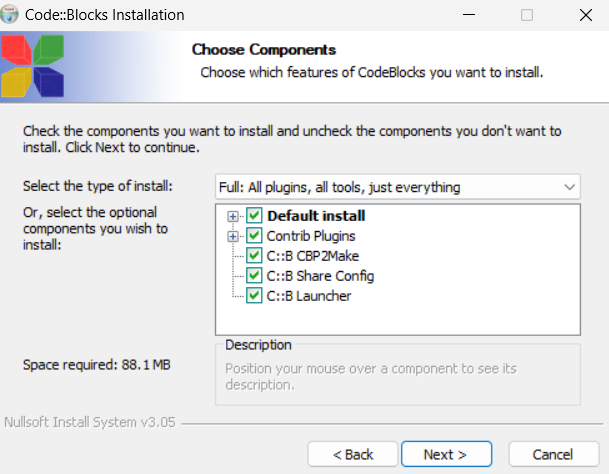
**2.1.1 Giới thiệu CodeBlocks**

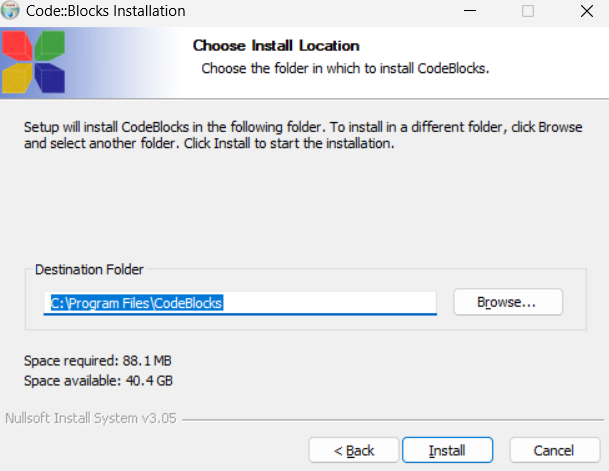
- Code::Blocks là một môi trường phát triển tích hợp (IDE) mạnh mẽ, đa nền tảng và mã nguồn mở, được thiết kế để hỗ trợ người phát triển trong quá trình viết và biên dịch mã nguồn. Với sự linh hoạt và đa chức năng, Code::Blocks là sự chọn lựa ưu việt cho các dự án lập trình C, C++, và Fortran.

**2.1.2 Các bước cài đặt CodeBlocks**









**CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU**

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

- Để tạo mảng sắp xếp thuật toán bằng đồ họa ta dùng thư viện đồ họa của graphics.h để thể hiện rõ quá trình sắp xếp bằng đồ họa. Đầu tiên vẽ bảng xác định kích thước, vật thể và văn bảng trong nền đồ họa.

+ 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 văn bản lên màn hình  } |

-Để tạo khoảng cách giũa 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; // \*2 để tạo khoảng cách giữa các hình tròn  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);  }  } |

- 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;  } |

- Để nhập các giái trị từ bán phím của người dùng ta dùng hàm inputArray

|  |
| --- |
| 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;  } |

- Hàm hiển thị của mãng vừa nhập ta dùng hàm displayArray

|  |
| --- |
| // Hàm hiển thị mảng  void displayArray(int\* arr) {  visualize(arr); |

- Để hiển thị các mneu ta dùng hàm ShowMenu để hiện các menu để 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: ";  } |

- 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);  } |

**3.2 Các hàm xếp mảng**

**3.2.1 Hàm thự hiện các sắp xếp trên đồ họa**

**-** Để thực hiện quá trính sắp xếp trên đồ họa diễn ra được chúng ta dùng hàm inplace để thực hiện các sắp trên đồ 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;  }  }  } |

**3.2.2 Hàm Qick 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);  } |

**3.2.3 Hàm Merge Sort**

|  |
| --- |
| 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);  } |

**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);  }  }  } |

**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);  }  } |

**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; |

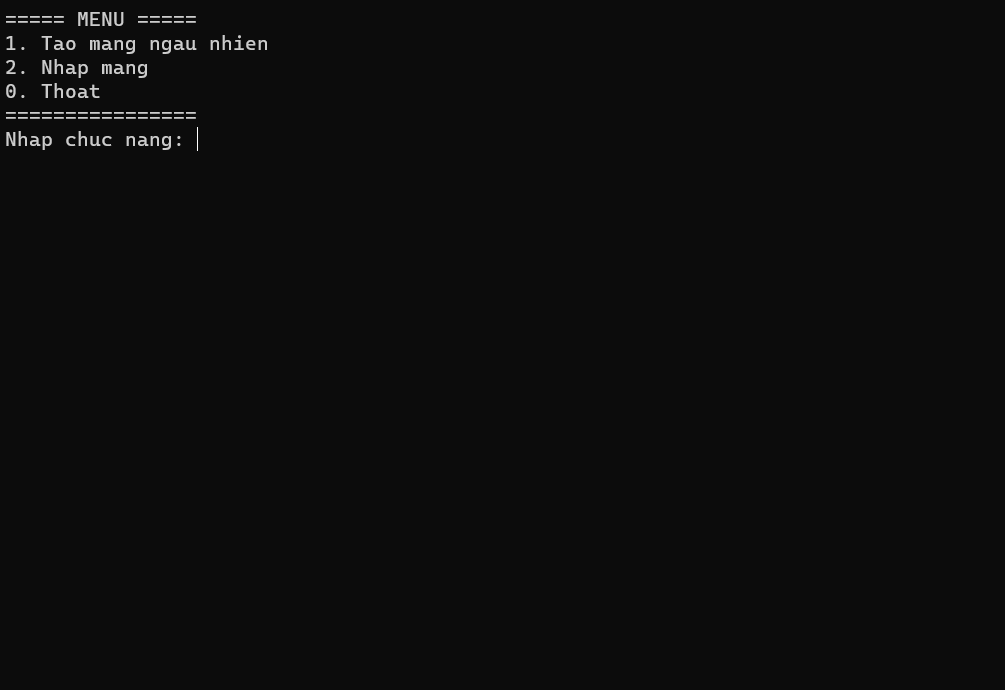
**3.3 Hàm đóng đồ họa**

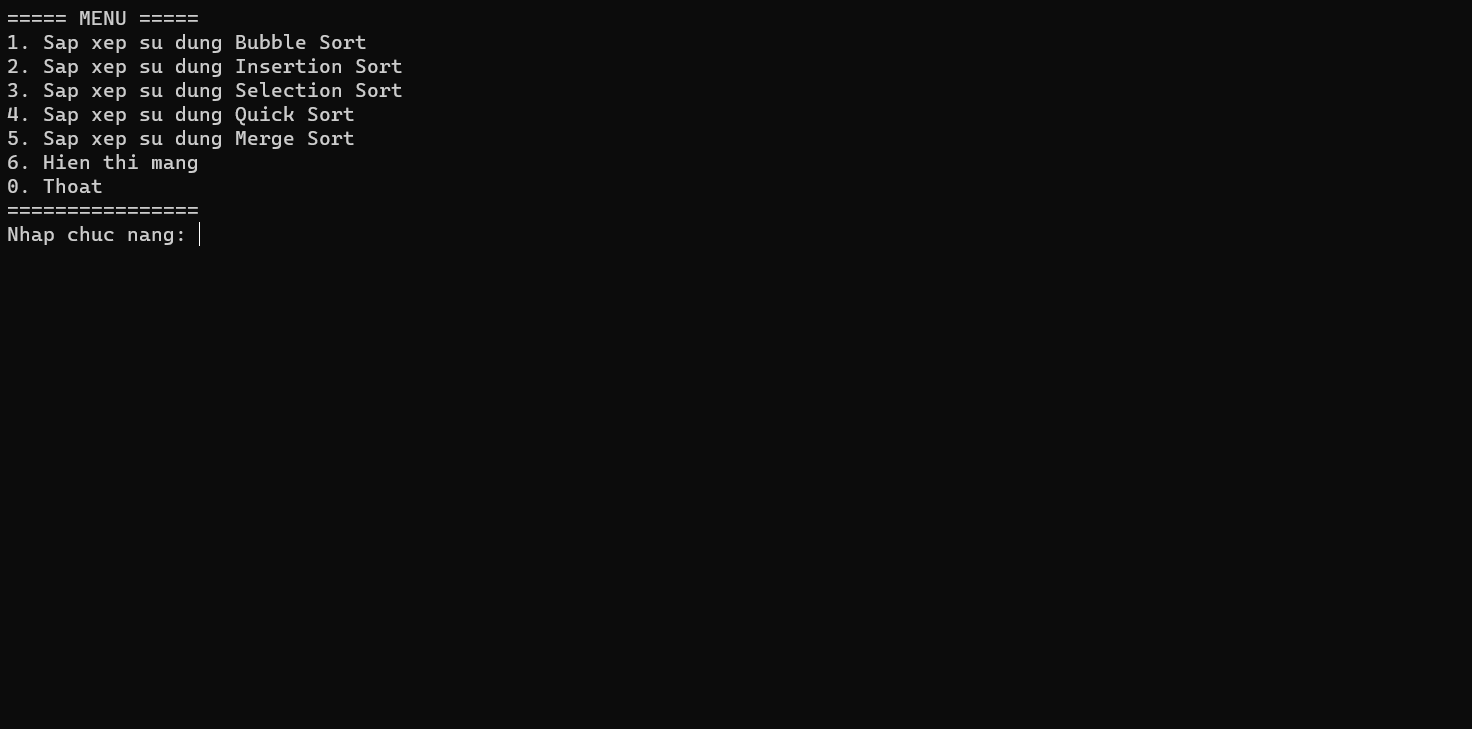
|  |
| --- |
| int main() {  initwindow(screenWidth, screenHeight, "Do hoa", 50, 30);  int\* arr = nullptr;  do {  ShowMenu(0);  std::cin >> choice;  switch (choice) {  case 0: {  std::cout << "\_\_\_\_\_\_\_EXIT\_\_\_\_\_\_\_\_";  return 0;  }  case 1: {  arr = generateRandomArray(n, 0, MAX\_VALUE);  std::cout << "Tao mang thanh cong." << std::endl;  system("pause");  Show(arr);  break;  }  case 2: {  arr = inputArray(n);  Show(arr);  break;  }  default: {  std::cout << "Gia tri khong hop le. Vui long kiem tra lai!" << std::endl;  break;  }  }  } while (true);  delete[] arr;  closegraph(); // Đóng cửa sổ đồ họa  return 0;  } |

**CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU**

**4.1 Minh họa giải thuật sắp xếp Bubble Sort:**

- Khởi tạo







+ Bước 1: chọn tạo mảng ngẫu nhiên hoặc số phân tử

+ Bước 2: chọn thuật toán Bubble Sort

+ Bước 3: chọn hướng sắp xếp tăng dần

- Quá trình sắp xếp:



- Sắp xếp hoàn thành



**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 thân thiên với người dùng

**5.2 Hạn chế**

**-** Giao diện đồ hoạ 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 thân thiện và đẹp hơn.

**DANH MỤC TÀI LIỆU THAM KHẢO**

<https://www.youtube.com/watch?app=desktop&v=ogaayYfh984&t=15s&fbclid=IwAR08uMKSDFMYVSqmbYaUI0a3mYtFRInvVkp61sGZQZ9D639RZLEFiG4j5K8>

<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/

**PHỤ LỤC**

TỐM TẮT ĐỀ TÀI…...………………………………………………………………….6

MỞ ĐẦU…………………………………………………………………...……………7

CHƯƠNG 1. TỔNG QUÁT……………………………………………………….…….8

* 1. Thuật Toán………………………………….………………………………………..8

1.2 Lập trình trên CodeBlocks…………………..……………………………….………8

CHƯƠNG 2. NGHIÊN CỨU LÝ THUYẾT…………………………………………….10

2.1 Lý thuyết của các thuật toán sắp xếp…………………………….…………………10

2.2 Giới thiệu CodeBlocks và cách cài đặt…………………………………………......11

CHƯƠNG 3: HIỆN THỰC HÓA NGHIÊN CỨU……………………………………14

3.1 Phần tử và các hàm có trong phần tử………………………………………………..14

3.2 Các hàm xếp mảng…………………………………………………………………24

3.3 Hàm đóng đồ họa…………………………………………………………………..35

CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU………………………………………………….37

CHƯƠNG 5: HƯỚNG PHÁT TRIỂN…………………………………………………..…40

TÀI LIỆU KHAM KHẢO…………………………………………………………………41

Phụ Lục...…………………………………………………………………………….……..42