TRƯỜNG ĐẠI HỌC SÀI GÒN

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue circle with text

Description automatically generated

Tiểu luận học phần Cấu trúc dữ liệu và giải thuật

**Bài tập sắp xếp**

Giảng viên hướng dẫn: T.S Đỗ Như Tài

Thành phố Hồ Chí Minh, tháng 03 năm 2025

TRƯỜNG ĐẠI HỌC SÀI GÒN

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue circle with text

Description automatically generated

**Cấu trúc dữ liêu và giải thuật**

**Bài tập sắp xếp**

Sinh viên thực hiện: Phạm Khánh Duy(3124411058)

Lê Phú Hiếụ (3124411090)

Tôn Trần Thiên Phú (3124411233)

Hoàng Nhật Anh (3124411008)

Thành phố Hồ Chí Minh, tháng 03 năm 2025

# LỜI CAM ĐOAN

Trong cấu trúc dữ liệu và giải thuật, sắp xếp là một bài toán quan trọng, giúp tổ chức dữ liệu một cách có hệ thống, hỗ trợ tối ưu hóa các thuật toán tìm kiếm và cải thiện hiệu suất của nhiều ứng dụng thực tế.

Báo cáo này nhằm trình bày các thuật toán sắp xếp phổ biến, bao gồm sắp xếp chọn, sắp xếp chèn, sắp xếp nổi bọt và một số thuật toán nâng cao khác. Nội dung báo cáo sẽ phân tích nguyên lý hoạt động, đánh giá hiệu suất và so sánh các thuật toán để có cái nhìn tổng quan về ưu nhược điểm của từng phương pháp.

Thông qua báo cáo này, người đọc sẽ hiểu rõ hơn về các thuật toán sắp xếp, cách áp dụng chúng vào thực tế và lựa chọn thuật toán phù hợp cho từng bài toán cụ thể.

# LỜI MỞ ĐẦU

Cấu trúc dữ liệu và giải thuật là nền tảng quan trọng trong khoa học máy tính, đóng vai trò quyết định trong việc tối ưu hóa hiệu suất và xử lý dữ liệu hiệu quả. Việc lựa chọn cấu trúc dữ liệu phù hợp giúp tổ chức dữ liệu có hệ thống, trong khi các giải thuật cung cấp phương pháp tiếp cận để giải quyết bài toán một cách nhanh chóng và tối ưu.

Trong quá trình phát triển phần mềm, việc hiểu và áp dụng đúng các cấu trúc dữ liệu như mảng, danh sách liên kết, cây, đồ thị hay các giải thuật tìm kiếm, sắp xếp không chỉ giúp cải thiện hiệu năng mà còn hỗ trợ giải quyết các bài toán phức tạp trong thực tế. Báo cáo này được thực hiện nhằm trình bày những kiến thức quan trọng về cấu trúc dữ liệu và giải thuật, giúp nâng cao hiểu biết và khả năng ứng dụng của người học vào lập trình và phát triển hệ thống.

Hy vọng rằng nội dung báo cáo sẽ cung cấp cái nhìn tổng quan và hữu ích về chủ đề này, từ đó giúp người đọc áp dụng linh hoạt vào thực tiễn.

**Xin trân trọng cảm ơn!**

# BÀI TẬP SẮP XẾP

©

Các bạn làm việc nhóm tối đa 4 sv/nhóm, thực hiện giải các bài tập.

# Câu hỏi

**1. Trình bày tư tưởng của các thuật toán sắp xếp?**

**2. Trong các thuật toán sắp xếp, bạn thích nhất thuật toán nào? Thuật toán nào bạn không thích nhất? Tại sao?**

**3. Trình bày và cài đặt tất cả các thuật toán sắp xếp nội, ngoại theo thứ tự giảm dần. Cho nhận xét về các thuật toán này.**

**4. Hãy trình bày những ưu điểm và nhược điểm của mỗi thuật toán sắp xếp? Theo bạn, cách khắc phục những nhược điểm này là gì?**

**Bài tập cơ sở Bài tập 1.** Cho dãy *n* số nguyên sau:

39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10

**a. Hãy mô phỏng các bước sắp xếp tăng dần dãy số trên bằng các giải thuật: sắp xếp đổi chỗ trực tiếp, sắp xếp chọn trực tiếp, sắp xếp chèn trực tiếp, Sắp xếp nổi bọt.**

**b. Cài đặt hoàn chỉnh các giải thuật trên.**

**c. Chứng minh độ phức tạp của các giải thuật đã triển khai.**

**Bài làm**

**Mô Phòng :**

1. Sắp xếp đổi chỗ trực tiếp (Interchange Sort):

Quá trình sắp xếp từng bước:

Bước 1: So sánh 39 với các phần tử sau và đổi chỗ nếu cần

39 > 8 → (8, **39**, 5, 1, 3, 6, 9, 12, 4, 7, 10)

39 > 5 → (8, 5, **39**, 1, 3, 6, 9, 12, 4, 7, 10)

39 > 1 → (8, 5, 1, **39**, 3, 6, 9, 12, 4, 7, 10)

39 > 3 → (8, 5, 1, 3, **39**, 6, 9, 12, 4, 7, 10)

39 > 6 → (8, 5, 1, 3, 6, **39**, 9, 12, 4, 7, 10)

39 > 9 → (8, 5, 1, 3, 6, 9, **39**, 12, 4, 7, 10)

39 > 12 → (8, 5, 1, 3, 6, 9, 12, **39**, 4, 7, 10)

39 > 4 → (8, 5, 1, 3, 6, 9, 12, 4, **39**, 7, 10)

39 > 7 → (8, 5, 1, 3, 6, 9, 12, 4, 7, **39**, 10)

39 > 10 → (8, 5, 1, 3, 6, 9, 12, 4, 7, 10, **39**)

Bước 2: So sánh 8 với các phần tử sau và đổi chỗ nếu cần

8 > 5 → **(5, 8, 1, 3, 6, 9, 12, 4, 7, 10, 39)**

8 > 1 → **(5, 1, 8, 3, 6, 9, 12, 4, 7, 10, 39)**

8 > 3 → **(5, 1, 3, 8, 6, 9, 12, 4, 7, 10, 39)**

8 > 6 → **(5, 1, 3, 6, 8, 9, 12, 4, 7, 10, 39)**

8 > 4 → **(5, 1, 3, 6, 4, 9, 12, 8, 7, 10, 39)**

8 > 7 → **(5, 1, 3, 6, 4, 9, 12, 7, 8, 10, 39)**

8 < 10 (không đổi)

Bước 3: So sánh 5 với các phần tử sau và đổi chỗ nếu cần

5 > 1 → **(1, 5, 3, 6, 4, 9, 12, 7, 8, 10, 39)**

5 > 3 → **(1, 3, 5, 6, 4, 9, 12, 7, 8, 10, 39)**

5 > 4 → **(1, 3, 4, 6, 5, 9, 12, 7, 8, 10, 39)**

Bước 4: So sánh 6 với các phần tử sau và đổi chỗ nếu cần

6 > 5 → **(1, 3, 4, 5, 6, 9, 12, 7, 8, 10, 39)**

6 > 9 (không đổi)

Bước 5: So sánh 9 với các phần tử sau và đổi chỗ nếu cần

9 > 7 → **(1, 3, 4, 5, 6, 7, 12, 9, 8, 10, 39)**

9 > 8 → **(1, 3, 4, 5, 6, 7, 8, 9, 12, 10, 39)**

9 > 10 (không đổi)

Bước 6: So sánh 12 với các phần tử sau và đổi chỗ nếu cần

12 > 10 → **(1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39)**

2. Sắp xếp chọn trực tiếp (Selection Sort):

Tìm phần tử nhỏ nhất trong dãy, đổi chỗ với phần tử đầu tiên:

Nhỏ nhất là **1** → đổi với **39** →  
 (1, 8, 5, 39, 3, 6, 9, 12, 4, 7, 10)

Lặp lại với phần còn lại:

Nhỏ nhất là **3** → đổi với **8** →  
 (1, 3, 5, 39, 8, 6, 9, 12, 4, 7, 10)

Nhỏ nhất là **4** → đổi với **39** →  
 (1, 3, 4, 39, 8, 6, 9, 12, 5, 7, 10)

Nhỏ nhất là **5** → đổi với **39** →  
 (1, 3, 4, 5, 8, 6, 9, 12, 39, 7, 10)

Nhỏ nhất là **6** → đổi với **8** →  
 (1, 3, 4, 5, 6, 8, 9, 12, 39, 7, 10)

Nhỏ nhất là **7** → đổi với **12** →  
 (1, 3, 4, 5, 6, 7, 9, 12, 39, 8, 10)

Nhỏ nhất là **8** → đổi với **12** →  
 (1, 3, 4, 5, 6, 7, 8, 12, 39, 9, 10)

Nhỏ nhất là **9** → đổi với **12** →  
 (1, 3, 4, 5, 6, 7, 8, 9, 39, 12, 10)

Nhỏ nhất là **10** → đổi với **39** →  
 (1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39)

**3. Sắp xếp chèn trực tiếp (Insertion Sort)**

Lấy phần tử thứ 2 (8), chèn vào vị trí phù hợp:  
 (8, 39, 5, 1, 3, 6, 9, 12, 4, 7, 10)

Lấy phần tử thứ 3 (5), chèn vào trước 8:  
 (5, 8, 39, 1, 3, 6, 9, 12, 4, 7, 10)

Lấy phần tử thứ 4 (1), chèn vào đầu:  
 (1, 5, 8, 39, 3, 6, 9, 12, 4, 7, 10)

Tiếp tục với các phần tử sau:

|  |
| --- |
| (1, 3, 5, 8, 39, 6, 9, 12, 4, 7, 10)  (1, 3, 5, 6, 8, 39, 9, 12, 4, 7, 10)  (1, 3, 5, 6, 8, 9, 39, 12, 4, 7, 10)  (1, 3, 4, 5, 6, 8, 9, 39, 12, 7, 10)  (1, 3, 4, 5, 6, 7, 8, 9, 39, 12, 10)  (1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39) |

4. Sắp xếp nổi bọt (Bubble Sort):

**Bước 1: Lặp qua dãy, so sánh từng cặp và đổi chỗ nếu cần**

**39 > 8** → Đổi chỗ → **(8, 39, 5, 1, 3, 6, 9, 12, 4, 7, 10)**

**39 > 5** → Đổi chỗ → **(8, 5, 39, 1, 3, 6, 9, 12, 4, 7, 10)**

**39 > 1** → Đổi chỗ → **(8, 5, 1, 39, 3, 6, 9, 12, 4, 7, 10)**

**39 > 3** → Đổi chỗ → **(8, 5, 1, 3, 39, 6, 9, 12, 4, 7, 10)**

**39 > 6** → Đổi chỗ → **(8, 5, 1, 3, 6, 39, 9, 12, 4, 7, 10)**

**39 > 9** → Đổi chỗ → **(8, 5, 1, 3, 6, 9, 39, 12, 4, 7, 10)**

**39 > 12** → Không đổi (vẫn giữ nguyên)

**12 > 4** → Đổi chỗ → **(8, 5, 1, 3, 6, 9, 39, 4, 12, 7, 10)**

**12 > 7** → Đổi chỗ → **(8, 5, 1, 3, 6, 9, 39, 4, 7, 12, 10)**

**12 > 10** → Đổi chỗ → **(8, 5, 1, 3, 6, 9, 39, 4, 7, 10, 12)**  
 **Phần tử lớn nhất (39) đã được đẩy về cuối.**

**Bước 2: Lặp tiếp, không xét phần tử cuối cùng**

**8 > 5** → Đổi chỗ → **(5, 8, 1, 3, 6, 9, 39, 4, 7, 10, 12)**

**8 > 1** → Đổi chỗ → **(5, 1, 8, 3, 6, 9, 39, 4, 7, 10, 12)**

**8 > 3** → Đổi chỗ → **(5, 1, 3, 8, 6, 9, 39, 4, 7, 10, 12)**

**8 > 6** → Đổi chỗ → **(5, 1, 3, 6, 8, 9, 39, 4, 7, 10, 12)**

**8 > 9** → Không đổi

**9 > 39** → Không đổi

**39 > 4** → Đổi chỗ → **(5, 1, 3, 6, 8, 9, 4, 39, 7, 10, 12)**

**39 > 7** → Đổi chỗ → **(5, 1, 3, 6, 8, 9, 4, 7, 39, 10, 12)**

**39 > 10** → Đổi chỗ → **(5, 1, 3, 6, 8, 9, 4, 7, 10, 39, 12)**  
 **Phần tử lớn thứ hai (39) đã đúng vị trí.**

**Bước 3: Lặp tiếp, không xét hai phần tử cuối**

**5 > 1** → Đổi chỗ → **(1, 5, 3, 6, 8, 9, 4, 7, 10, 39, 12)**

**5 > 3** → Đổi chỗ → **(1, 3, 5, 6, 8, 9, 4, 7, 10, 39, 12)**

**5 > 6** → Không đổi

**6 > 8** → Không đổi

**8 > 9** → Không đổi

**9 > 4** → Đổi chỗ → **(1, 3, 5, 6, 8, 4, 9, 7, 10, 39, 12)**

**9 > 7** → Đổi chỗ → **(1, 3, 5, 6, 8, 4, 7, 9, 10, 39, 12)**

**9 > 10** → Không đổi  
 **Phần tử lớn thứ ba (10) đã đúng vị trí.**

**Lặp tiếp tục đến khi dãy hoàn toàn sắp xếp**

**(1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39)**  
 ✅ **Dãy đã được sắp xếp tăng dần hoàn toàn!**

**Bài tập 2.** Cho dãy *n* số nguyên sau:

8, 5, 1, 3, 6, 9, 12, 4, 7, 10

1. Hãy mô phỏng các bước sắp xếp tăng dần dãy số trên bằng các giải thuật: sắp xếp nhanh (Quick Sort); Sắp xếp trộn trực tiếp (Merge Sort); Sắp xếp cây (Heap Sort).
2. Cài đặt hoàn chỉnh các giải thuật trên.
3. Chứng minh độ phức tạp của các giải thuật đã triển khai.

Bài Làm

I.Ý Tưởng:

1. Quick Sort (Sắp xếp nhanh):

- Ý tưởng: Chọn một phần tử làm chốt (pivot), chia mảng thành hai phần: phần nhỏ hơn pivot và phần lớn hơn pivot. Đệ quy sắp xếp hai phần đó.

- Độ phức tạp:

+ Trung bình: O(n log n)

+ Tệ nhất: O(n^2) (nếu chọn pivot không tốt)

+ Tốt nhất: O(n log n)

- Ứng dụng: Thường được sử dụng khi cần sắp xếp nhanh với bộ nhớ hạn chế.

2. Merge Sort (Sắp xếp trộn):

- Ý tưởng: Chia mảng thành các phần nhỏ, sắp xếp từng phần rồi trộn lại.

- Độ phức tạp: O(n log n) trong mọi trường hợp.

- Ứng dụng: Sử dụng khi cần sắp xếp ổn định hoặc khi làm việc với dữ liệu liên kết.

3. Heap Sort (Sắp xếp cây):

- Ý tưởng: Dùng cấu trúc dữ liệu heap (max-heap) để liên tục lấy phần tử lớn nhất và đưa về cuối mảng.

- Độ phức tạp: O(n log n) trong mọi trường hợp.

- Ứng dụng: Sắp xếp tại chỗ không cần bộ nhớ phụ.

II. Ý tưởng triển khai:

1. Quick Sort:

- Chọn phần tử cuối cùng làm pivot.

- Sắp xếp sao cho các phần tử nhỏ hơn pivot nằm bên trái, lớn hơn pivot nằm bên phải.

- Gọi đệ quy sắp xếp hai phần đó.

2. Merge Sort:

- Chia mảng thành hai nửa bằng nhau.

- Sắp xếp đệ quy từng nửa.

- Trộn hai nửa đã sắp xếp thành một mảng hoàn chỉnh

3. Heap Sort:

- Xây dựng cây Heap (dạng Max-Heap).

- Lấy phần tử lớn nhất (gốc của cây) và chuyển nó xuống cuối mảng.

- Giảm kích thước cây và tiếp tục quá trình đến khi hoàn tất.

II.Thuật Toán

1. Quick Sort:

- Chọn một phần tử pivot (ở đây chọn phần tử cuối cùng).

- Chia mảng thành hai phần: phần bên trái chứa các phần tử nhỏ hơn hoặc bằng pivot, phần bên phải chứa các phần tử lớn hơn pivot.

- Gọi đệ quy sắp xếp phần bên trái và phần bên phải.

2. Merge Sort:

- Chia mảng thành hai nửa (nếu mảng chỉ có một phần tử thì xem như đã sắp xếp).

- Sắp xếp đệ quy từng nửa.

- Trộn hai nửa đã sắp xếp thành một mảng hoàn chỉnh bằng cách so sánh từng phần tử nhỏ nhất của mỗi nửa và đưa vào mảng kết quả.

3. Heap Sort:

- Xây dựng cây Heap từ mảng (biến đổi mảng thành một Max-Heap).

- Với mỗi lần sắp xếp: Đưa phần tử lớn nhất (ở gốc cây) xuống cuối mảng.

- Giảm kích thước mảng cần sắp xếp và lặp lại quá trình điều chỉnh Heap để tìm phần tử lớn nhất tiếp theo.

III.TestCase

Dữ liệu đầu vào: {8, 5, 1, 3, 6, 9, 12, 4, 7, 10}

- Quick Sort:

Bước 1: Chọn pivot = 10 (cuối mảng). Sau khi phân chia: {8, 5, 1, 3, 6, 9, 4, 7, 10, 12}

+ Bước 2: Gọi đệ quy tiếp tục cho các phần bên trái và phải.

- Merga Sort:

+ Bước 1: Chia mảng thành hai nửa: {8, 5, 1, 3, 6} và {9, 12, 4, 7, 10}

+ Bước 2: Sắp xếp từng nửa: {1, 3, 5, 6, 8} và {4, 7, 9, 10, 12}

+ Bước 3: Trộn lại: {1, 3, 4, 5, 6, 7, 8, 9, 10, 12}

- Heap Sort:

+ Bước 1: Xây dựng cây Heap từ mảng ban đầu.

+ Bước 2: Trích phần tử lớn nhất (12) và đặt vào cuối.

+ Bước 3: Lặp lại quá trình đến khi mảng được sắp xếp hoàn toàn.

Code:

|  |
| --- |
| #include <iostream>  using namespace std;  void interchangeSort(int arr[], int n)  {      for (int i = 0; i < n - 1; i++)      {          for (int j = i + 1; j < n; j++)          {              if (arr[i] > arr[j])              {                  swap(arr[i], arr[j]);              }          }      }      cout << "Interchange Sort: ";      for (int i = 0; i < n; i++)          cout << arr[i] << " ";      cout << endl;  }  void selectionSort(int arr[], int n)  {      for (int i = 0; i < n - 1; i++)      {          int min\_idx = i;          for (int j = i + 1; j < n; j++)          {              if (arr[j] < arr[min\_idx])              {                  min\_idx = j;              }          }          swap(arr[i], arr[min\_idx]);      }      cout << "Selection Sort: ";      for (int i = 0; i < n; i++)          cout << arr[i] << " ";      cout << endl;  }  void insertionSort(int arr[], int n)  {      for (int i = 1; i < n; i++)      {          int key = arr[i];          int j = i - 1;          while (j >= 0 && arr[j] > key)          {              arr[j + 1] = arr[j];              j--;          }          arr[j + 1] = key;      }      cout << "Insertion Sort: ";      for (int i = 0; i < n; i++)          cout << arr[i] << " ";      cout << endl;  }  void bubbleSort(int arr[], int n)  {      for (int i = 0; i < n - 1; i++)      {          for (int j = 0; j < n - 1 - i; j++)          {              if (arr[j] > arr[j + 1])              {                  swap(arr[j], arr[j + 1]);              }          }      }      cout << "Bubble Sort: ";      for (int i = 0; i < n; i++)          cout << arr[i] << " ";      cout << endl;  }  int main()  {      int n;      cout << "Nhập số phần tử: ";      cin >> n;      int arr[n];      for (int i = 0; i < n; i++)      {          cout << "Nhập phần tử thứ " << i + 1 << ": ";          cin >> arr[i];      }      int arr1[n], arr2[n], arr3[n], arr4[n];      for (int i = 0; i < n; i++)      {          arr1[i] = arr[i];          arr2[i] = arr[i];          arr3[i] = arr[i];          arr4[i] = arr[i];      }      interchangeSort(arr1, n);      selectionSort(arr2, n);      insertionSort(arr3, n);      bubbleSort(arr4, n);      return 0;  } |

**Bài tập 3**. Sử dụng hàm random trong C để tạo ra một dãy M có n số nguyên. Vận dụng các thuật toán sắp xếp để sắp xếp các phần tử của mảng M theo thứ tự tăng dần về mặt giá trị. Với cùng một dữ liệu như nhau, làm thế nào để biết thời gian thực hiện các thuật toán (đếm số lần so sánh, đổi chỗ)? Có nhận xét gì đối với các thuật toán sắp xếp này? Thực nghiệm mỗi trường hợp t lần với kích cỡ mảng n=10, 100, 200, …, 10000.

I.Ý Tưởng

-Viết chương trình tạo mảng ngẫu nhiên gồm n phần tử (số nguyên).

-Thử nghiệm nhiều thuật toán sắp xếp khác nhau trên cùng một mảng dữ liệu.

-Đo thời gian thực hiện, số phép so sánh và số phép đổi chỗ của mỗi thuật toán.

-So sánh hiệu suất của các thuật toán với các kích cỡ mảng khác nhau: n = 10, 100, 200, ..., 10.000.

-Lặp lại quá trình này t lần (ví dụ: t = 10) và tính trung bình kết quả.

II. Thuật Toán

Chương trình sẽ cài đặt các thuật toán sắp xếp sau:

### **2.1. Selection Sort**

* 1. Chọn phần tử nhỏ nhất trong dãy chưa được sắp xếp.
  2. Đổi chỗ với phần tử đầu tiên của dãy chưa được sắp xếp.
  3. Lặp lại cho đến khi mảng được sắp xếp.

### **2.2. Bubble Sort**

* 1. So sánh từng cặp phần tử liên tiếp.
  2. Đổi chỗ nếu phần tử trước lớn hơn phần tử sau.
  3. Lặp lại cho đến khi không còn phép đổi chỗ nào.

### **2.3. Insertion Sort**

* 1. Duyệt qua từng phần tử.
  2. Chèn từng phần tử vào vị trí đúng trong phần mảng đã sắp xếp.

### **2.4. Quick Sort**

* 1. Chọn một phần tử làm pivot (chọn phần tử cuối).
  2. Phân chia mảng thành hai phần: bên trái nhỏ hơn hoặc bằng pivot, bên phải lớn hơn pivot.
  3. Gọi đệ quy sắp xếp hai phần còn lại.

**III.Testcase**

### **Mục tiêu: Thử nghiệm tất cả thuật toán trên các mảng có kích cỡ khác nhau.**

### **3.1. Kích cỡ mảng**

n = 10

n = 100

n = 200

n = 500

n = 1000

n = 2000

n = 5000

n = 10000

### **3.2. Số lần chạy**

t = 10 lần (với mỗi kích thước mảng).

Kết quả là trung bình của 10 lần chạy.

### **3.3. Đo lường**

Thời gian thực hiện (ms).

Số phép so sánh.

Số phép đổi chỗ.

### **3.4. So sánh**

Lập bảng kết quả để so sánh tốc độ và hiệu quả của từng thuật toán.

Nhận xét sự khác nhau giữa các thuật toán theo từng kích cỡ mảng.

Code:

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  #define MAX\_SIZE 10000  long long quick\_cmp = 0, quick\_swap = 0;  long long merge\_cmp = 0, merge\_swap = 0;  long long heap\_cmp = 0, heap\_swap = 0;  void generateArray(int arr[], int n)  {  for (int i = 0; i < n; i++)  {  arr[i] = rand() % MAX\_SIZE;  }  }  void copyArray(int src[], int dest[], int n)  {  for (int i = 0; i < n; i++)  {  dest[i] = src[i];  }  }  void quickSort(int arr[], int left, int right)  {  if (left >= right)  return;  int pivot = arr[right];  int i = left - 1;  for (int j = left; j < right; j++)  {  quick\_cmp++;  if (arr[j] <= pivot)  {  i++;  swap(arr[i], arr[j]);  quick\_swap++;  }  }  swap(arr[i + 1], arr[right]);  quick\_swap++;  quickSort(arr, left, i);  quickSort(arr, i + 2, right);  }  void merge(int arr[], int left, int mid, int right)  {  int n1 = mid - left + 1, n2 = right - mid;  int L[n1], R[n2];  for (int i = 0; i < n1; i++)  L[i] = arr[left + i];  for (int j = 0; j < n2; j++)  R[j] = arr[mid + 1 + j];  int i = 0, j = 0, k = left;  while (i < n1 && j < n2)  {  merge\_cmp++;  if (L[i] <= R[j])  {  arr[k++] = L[i++];  }  else  {  arr[k++] = R[j++];  }  merge\_swap++;  }  while (i < n1)  {  arr[k++] = L[i++];  merge\_swap++;  }  while (j < n2)  {  arr[k++] = R[j++];  merge\_swap++;  }  }  void mergeSort(int arr[], int left, int right)  {  if (left < right)  {  int mid = left + (right - left) / 2;  mergeSort(arr, left, mid);  mergeSort(arr, mid + 1, right);  merge(arr, left, mid, right);  }  }  void heapify(int arr[], int n, int i)  {  int largest = i;  int left = 2 \* i + 1, right = 2 \* i + 2;  heap\_cmp++;  if (left < n && arr[left] > arr[largest])  largest = left;  heap\_cmp++;  if (right < n && arr[right] > arr[largest])  largest = right;  if (largest != i)  {  swap(arr[i], arr[largest]);  heap\_swap++;  heapify(arr, n, largest);  }  }  void heapSort(int arr[], int n)  {  for (int i = n / 2 - 1; i >= 0; i--)  heapify(arr, n, i);  for (int i = n - 1; i > 0; i--)  {  swap(arr[0], arr[i]);  heap\_swap++;  heapify(arr, i, 0);  }  }  int main()  {  srand(time(0));  int sizes[] = {10, 100, 200, 500, 1000, 5000, 10000};  int num\_tests = 10;  cout << "Size, QuickSort Time (ms), QuickSort Comparisons, QuickSort Swaps, "  "MergeSort Time (ms), MergeSort Comparisons, MergeSort Swaps, "  "HeapSort Time (ms), HeapSort Comparisons, HeapSort Swaps\n";  for (int s = 0; s < 7; s++)  {  int n = sizes[s];  long long total\_quick\_time = 0, total\_merge\_time = 0, total\_heap\_time = 0;  long long total\_quick\_cmp = 0, total\_quick\_swap = 0;  long long total\_merge\_cmp = 0, total\_merge\_swap = 0;  long long total\_heap\_cmp = 0, total\_heap\_swap = 0;  for (int t = 0; t < num\_tests; t++)  {  int arr[MAX\_SIZE], temp[MAX\_SIZE];  generateArray(arr, n);  copyArray(arr, temp, n);  quick\_cmp = quick\_swap = 0;  clock\_t start = clock();  quickSort(temp, 0, n - 1);  total\_quick\_time += clock() - start;  total\_quick\_cmp += quick\_cmp;  total\_quick\_swap += quick\_swap;  generateArray(arr, n);  copyArray(arr, temp, n);  merge\_cmp = merge\_swap = 0;  start = clock();  mergeSort(temp, 0, n - 1);  total\_merge\_time += clock() - start;  total\_merge\_cmp += merge\_cmp;  total\_merge\_swap += merge\_swap;  generateArray(arr, n);  copyArray(arr, temp, n);  heap\_cmp = heap\_swap = 0;  start = clock();  heapSort(temp, n);  total\_heap\_time += clock() - start;  total\_heap\_cmp += heap\_cmp;  total\_heap\_swap += heap\_swap;  }  cout << n << ", "  << (total\_quick\_time \* 1000.0 / CLOCKS\_PER\_SEC / num\_tests) << ", " << (total\_quick\_cmp / num\_tests) << ", " << (total\_quick\_swap / num\_tests) << ", "  << (total\_merge\_time \* 1000.0 / CLOCKS\_PER\_SEC / num\_tests) << ", " << (total\_merge\_cmp / num\_tests) << ", " << (total\_merge\_swap / num\_tests) << ", "  << (total\_heap\_time \* 1000.0 / CLOCKS\_PER\_SEC / num\_tests) << ", " << (total\_heap\_cmp / num\_tests) << ", " << (total\_heap\_swap / num\_tests) << "\n";  }  return 0;  } |

**Bài tập 4.** Hãy đo thời gian thi hành của mỗi giải thuật sắp xếp đã học trên **cùng một bộ dữ liệu** chứa khoảng **30.000 số nguyên ngẫu nhiên** và **trên cùng một máy tính của bạn**.

* Dữ liệu đầu vào: Được lưu trên file văn bản.
* Dữ liệu đầu ra: Kết quả sắp xếp được lưu trên file văn bản.

Các thuật toán sắp xếp cần đo thời gian: Sắp xếp đổi chỗ trực tiếp (Interchange Sort); Sắp xếp chọn trực tiếp (Selection Sort); Sắp xếp chèn trực tiếp (Insertion Sort); Sắp xếp nổi bọt (Bubble Sort); Sắp xếp nhanh (Quick Sort); Sắp xếp trộn (Merge Sort); Sắp xếp cây (Heap Sort)

Yêu cầu thực hiện:

* + Đọc dữ liệu từ file văn bản.
  + Đo thời gian chạy từng thuật toán trên cùng bộ dữ liệu.
  + Lưu kết quả sắp xếp và thời gian chạy vào file văn bản.
  + So sánh hiệu suất của các thuật toán sắp xếp.
  1. **I.Ý Tưởng**
  2. Viết chương trình đọc dữ liệu từ file văn bản chứa khoảng 30.000 số nguyên ngẫu nhiên.
  3. Triển khai và đo lường thời gian thực thi của các thuật toán sắp xếp.
  4. Ghi kết quả sắp xếp và thời gian thực thi của từng thuật toán vào file văn bản đầu ra.
  5. So sánh hiệu suất của các thuật toán.

**II.Thuật Toán**

Chương trình sẽ cài đặt các thuật toán sắp xếp sau:

### **2.1. Interchange Sort (Sắp xếp đổi chỗ trực tiếp)**

* 1. So sánh từng cặp phần tử và đổi chỗ nếu cần.
  2. Độ phức tạp: O(n²).

### **2.2. Selection Sort (Sắp xếp chọn trực tiếp)**

* 1. Chọn phần tử nhỏ nhất trong dãy chưa được sắp xếp.
  2. Đổi chỗ với phần tử đầu tiên của dãy chưa được sắp xếp.
  3. Độ phức tạp: O(n²).

### **2.3. Insertion Sort (Sắp xếp chèn trực tiếp)**

* 1. Chèn từng phần tử vào vị trí đúng trong phần đã sắp xếp.
  2. Độ phức tạp: O(n²).

### **2.4. Bubble Sort (Sắp xếp nổi bọt)**

* 1. So sánh từng cặp phần tử liên tiếp và đổi chỗ nếu cần.
  2. Độ phức tạp: O(n²).

### **2.5. Quick Sort (Sắp xếp nhanh)**

* 1. Chọn một phần tử làm pivot và phân chia mảng theo pivot.
  2. Gọi đệ quy cho hai phần mảng còn lại.
  3. Độ phức tạp: Trung bình O(n log n).

### **2.6. Merge Sort (Sắp xếp trộn)**

* 1. Chia mảng thành hai nửa, sắp xếp từng nửa và trộn lại.
  2. Độ phức tạp: O(n log n).

### **2.7. Heap Sort (Sắp xếp cây)**

* 1. Chuyển mảng thành cây nhị phân và thực hiện sắp xếp.
  2. Độ phức tạp: O(n log n).

**III.Testcase**

### **Mục tiêu:**

* 1. Thử nghiệm tất cả các thuật toán trên cùng một bộ dữ liệu.
  2. Dữ liệu được lưu trong file input.txt gồm 30.000 số nguyên ngẫu nhiên, mỗi số trên một dòng.

### **3.1. File Đầu Vào: input.txt**

* 1. Ví dụ:9873  
     123  
     4567  
     ...

### **3.2. File Đầu Ra: output.txt**

* 1. Ghi kết quả sắp xếp theo thứ
  2. tự tăng dần.
  3. Ví dụ:

Thuật toán: Quick Sort  
Thời gian thực hiện: 12.34 ms  
Kết quả sắp xếp:  
123  
4567  
9873  
...

### **3.3. Đo lường:**

* 1. Thời gian thực hiện (ms).
  2. Ghi nhận kết quả sắp xếp sau mỗi thuật toán.

### **3.4. So sánh:**

* 1. Lập bảng thống kê thời gian chạy của từng thuật toán.
  2. Nhận xét sự khác biệt giữa các thuật toán.

Code:

# Bài tập Cơ Sở

**Bài tập 1.** Cho dãy *n* số nguyên: a0, a1,..., an-1

* 1. Hãy cho biết vị trí của *k* phần tử có giá trị lớn nhất trong dãy.
  2. Sắp xếp các phần tử tăng dần theo tổng các chữ số của từng phần tử.
  3. Hãy xóa tất cả các số nguyên tố có trong dãy

Bài làm

I.Ý Tưởng

Viết chương trình thực hiện các yêu cầu sau:

* 1. **Tìm vị trí của k phần tử có giá trị lớn nhất trong dãy.**
  2. **Sắp xếp các phần tử tăng dần theo tổng các chữ số của từng phần tử.**
  3. **Xóa tất cả các số nguyên tố trong dãy.**

**II.Thuật Toán**

### **2.1. Tìm vị trí của k phần tử lớn nhất**

* 1. Duyệt mảng và tìm ra k phần tử có giá trị lớn nhất.
  2. Ghi lại vị trí của các phần tử đó.

### **2.2. Sắp xếp theo tổng chữ số của từng phần tử**

* 1. Tính tổng các chữ số của từng phần tử bằng cách chia lấy dư (n % 10).
  2. Sử dụng một thuật toán sắp xếp (ví dụ: Bubble Sort) để sắp xếp các phần tử theo tổng chữ số.

### **2.3. Xóa số nguyên tố**

* 1. Kiểm tra từng phần tử xem có phải số nguyên tố hay không.
  2. Nếu là số nguyên tố, loại bỏ khỏi dãy.
  3. Thuật toán kiểm tra số nguyên tố: Kiểm tra ước số từ 2 đến √n.

**III.Testcase**

### **Mục tiêu:**

* 1. Kiểm tra đầy đủ 3 yêu cầu của bài toán.

### **3.1. Dữ liệu đầu vào:**

* 1. Dãy số nguyên: 45, 12, 31, 78, 23, 5, 91, 56
  2. k = 3 (Tìm 3 phần tử lớn nhất)

### **3.2. Kết quả mong đợi:**

#### **a. Tìm k phần tử lớn nhất**

* 1. Các phần tử lớn nhất: 91, 78, 56
  2. Vị trí: 6, 3, 7

#### **b. Sắp xếp tăng dần theo tổng các chữ số**

* 1. Tổng chữ số của từng phần tử: 45(9), 12(3), 31(4), 78(15), 23(5), 91(10), 56(11)
  2. Kết quả sau khi sắp xếp: 12, 31, 23, 45, 91, 56, 78

#### **c. Xóa số nguyên tố**

* 1. Các số nguyên tố: 5, 31, 23
  2. Kết quả sau khi xóa: 45, 12, 78, 91, 56

**Code:**

|  |
| --- |
| #include <iostream>  #include <cmath>  using namespace std;  const int MAX\_N = 1000;  int sumOfDigits(int num)  {  int sum = 0;  num = abs(num);  while (num > 0)  {  sum += num % 10;  num /= 10;  }  return sum;  }  bool isPrime(int num)  {  if (num < 2)  return false;  for (int i = 2; i \* i <= num; i++)  {  if (num % i == 0)  return false;  }  return true;  }  void swap(int &a, int &b)  {  int temp = a;  a = b;  b = temp;  }  void sortDescending(int arr[], int n)  {  for (int i = 0; i < n - 1; i++)  {  for (int j = i + 1; j < n; j++)  {  if (arr[i] < arr[j])  {  swap(arr[i], arr[j]);  }  }  }  }  void sortByDigitSum(int arr[], int n)  {  for (int i = 0; i < n - 1; i++)  {  for (int j = i + 1; j < n; j++)  {  if (sumOfDigits(arr[i]) > sumOfDigits(arr[j]))  {  swap(arr[i], arr[j]);  }  }  }  }  int main()  {  int n, k;  int arr[MAX\_N], sortedArr[MAX\_N];  cout << "Nhập số phần tử của dãy: ";  cin >> n;  cout << "Nhập dãy số: ";  for (int i = 0; i < n; i++)  {  cin >> arr[i];  sortedArr[i] = arr[i];  }  cout << "Nhập k: ";  cin >> k;  sortDescending(sortedArr, n);  int maxK = sortedArr[k - 1];  cout << "Vị trí của " << k << " phần tử lớn nhất: ";  for (int i = 0; i < n; i++)  {  if (arr[i] >= maxK)  {  cout << i << " ";  }  }  cout << endl;  sortByDigitSum(arr, n);  cout << "Dãy sau khi sắp xếp theo tổng chữ số: ";  for (int i = 0; i < n; i++)  {  cout << arr[i] << " ";  }  cout << endl;  int newSize = 0;  for (int i = 0; i < n; i++)  {  if (!isPrime(arr[i]))  {  arr[newSize++] = arr[i];  }  }  cout << "Dãy sau khi xóa số nguyên tố: ";  for (int i = 0; i < newSize; i++)  {  cout << arr[i] << " ";  }  cout << endl;  return 0;  } |

**Bài tập 2.** Thông tin về mỗi số hạng của một dãy thức bậc n bao gồm: Hệ số – là một số thực, Bậc – là một số nguyên có giá trị từ 0 đến 100.

* 1. Hãy định nghĩa cấu trúc dữ liệu để lưu trữ các dữ liệu trong bộ nhớ trong của máy tính.
  2. Với cấu trúc dữ liệu đã được định nghĩa, hãy vận dụng một thuật toán sắp xếp và cài đặt chương trình thực hiện việc sắp xếp các số hạng trong dãy theo thứ tự tăng dần của các bậc.

**Bài Làm**

**I.Ý tưởng**

* 1. Chương trình cần quản lý nhiều số hạng của một dãy thức, mỗi số hạng có:
  2. **Hệ số (he\_so): Số thực (float).**
  3. **Bậc (bac): Số nguyên từ 0 đến 100.**
  4. Các số hạng này được lưu trữ trong một mảng (không dùng vector).
  5. Mục tiêu là sắp xếp mảng theo thứ tự tăng dần của bac

**II.Thuật Toán**

Ý tưởng:

* 1. Chọn một phần tử làm chốt (pivot).
  2. Phân chia dãy sao cho các phần tử có bậc nhỏ hơn pivot nằm bên trái, lớn hơn pivot nằm bên phải.
  3. Gọi đệ quy sắp xếp hai phần đó.
  4. Độ phức tạp:
  5. Trung bình: O(n log n)
  6. Tệ nhất: O(n^2)
  7. Tốt nhất: O(n log n)

Ứng dụng: Thích hợp để sắp xếp dãy số hạng theo thứ

III.Testcase:  
Dữ liệu đầu vào: Dãy số hạng với n = 5

|  |
| --- |
| { (3.5, 2), (1.2, 5), (4.0, 1), (2.1, 3), (0.9, 0) } |

* 1. Quick Sort:
  2. Bước 1: Chọn pivot là số hạng cuối cùng (0.9, 0).
  3. Bước 2: Phân chia mảng. Kết quả:  
      { (0.9, 0), (4.0, 1), (3.5, 2), (2.1, 3), (1.2, 5) }
  4. Bước 3: Sắp xếp đệ quy các phần bên trái và phải.
  5. Kết quả cuối cùng:  
      { (0.9, 0), (4.0, 1), (3.5, 2), (2.1, 3), (1.2, 5) }

-Code:

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  struct Term {  double coefficient;  int exponent;  };  struct Polynomial {  int degree;  Term \*terms;  Polynomial(int degree) {  this->degree = degree;  terms = new Term[degree];  }  ~Polynomial() {  delete[] terms;  }  };  void readPolynomial(Polynomial &poly) {  cout << "Nhập bậc của đa thức: ";  cin >> poly.degree;  delete[] poly.terms;  poly.terms = new Term[poly.degree];  for (int i = 0; i < poly.degree; i++) {  cout << "Nhập hệ số: ";  cin >> poly.terms[i].coefficient;  cout << "Nhập bậc của biến số: ";  cin >> poly.terms[i].exponent;  }  }  void sortPolynomialTermsByDegree(Polynomial &poly) {  for (int i = 0; i < poly.degree - 1; i++) {  int minIndex = i;  for (int j = i + 1; j < poly.degree; j++) {  if (poly.terms[j].exponent < poly.terms[minIndex].exponent) {  minIndex = j;  }  }  swap(poly.terms[i], poly.terms[minIndex]);  }  }  void printPolynomial(const Polynomial &poly) {  for (int i = 0; i < poly.degree; i++) {  if (i > 0 && poly.terms[i].coefficient > 0) {  cout << " + ";  } else if (poly.terms[i].coefficient < 0) {  cout << " - ";  }  cout << abs(poly.terms[i].coefficient);  if (poly.terms[i].exponent > 0) {  cout << "x";  if (poly.terms[i].exponent > 1) {  cout << "^" << poly.terms[i].exponent;  }  }  }  cout << endl;  }  void test1() {  cout << "Chạy Test 1: Nhập và xuất đa thức" << endl;  Polynomial poly(3);  poly.terms[0] = {3, 4};  poly.terms[1] = {-5, 3};  poly.terms[2] = {2, 2};  cout << "Trước khi sắp xếp: ";  printPolynomial(poly);  sortPolynomialTermsByDegree(poly);  cout << "Sau khi sắp xếp: ";  printPolynomial(poly);  }  void test2() {  cout << "Chạy Test 2: Kiểm tra sắp xếp số hạng theo bậc tăng dần" << endl;  Polynomial poly(4);  poly.terms[0] = {3, 4};  poly.terms[1] = {-5, 3};  poly.terms[2] = {2, 2};  poly.terms[3] = {7, 0};  cout << "Trước khi sắp xếp: ";  printPolynomial(poly);  sortPolynomialTermsByDegree(poly);  cout << "Sau khi sắp xếp: ";  printPolynomial(poly);  }  int main() {  test1();  test2();  return 0;  } |

**Bài tập 3.** Thông tin về các phòng thi tại một hội đồng thi bao gồm: Số phòng – là một số nguyên có giá trị từ 1 đến 200, Nhà – là một chữ cái in hoa từ A → Z, Khả năng chứa – là một số nguyên có giá trị từ 10 → 250.

1. Hãy định nghĩa cấu trúc dữ liệu để lưu trữ các phòng thi này trong bộ nhớ trong của máy tính.
2. Với cấu trúc dữ liệu đã được định nghĩa, vận dụng các thuật toán sắp xếp và cài đặt chương trình thực hiện việc các công việc sau:

* Sắp xếp và in ra màn hình danh sách các phòng thi theo thứ tự giảm dần về Khả năng chứa.
* Sắp xếp và in ra màn hình danh sách các phòng thi theo thứ tự tăng dần theo Nhà (Từ A

→ Z), các phòng cùng một nhà thì sắp xếp theo thứ tự tăng dần theo Số phòng.

* Sắp xếp và in ra màn hình danh sách các phòng thi theo thứ tự tăng dần theo Nhà (Từ A

→ Z), các phòng cùng một nhà thì sắp xếp theo thứ tự **giảm dần theo Khả năng chứa**.

**Bài Làm**

**I.Ý Tưởng**

* 1. **Định nghĩa cấu trúc:**
  2. Sử dụng struct để lưu trữ thông tin phòng thi.
  3. **Các hàm sắp xếp:**
  4. **Sắp xếp theo Khả năng chứa (giảm dần):**
  5. So sánh khaNangChua của hai phòng.
  6. **Sắp xếp theo Nhà, Số phòng (tăng dần):**
  7. So sánh nha trước, nếu giống nhau thì so sánh soPhong.
  8. **Sắp xếp theo Nhà, Khả năng chứa (giảm dần):**
  9. So sánh nha trước, nếu giống nhau thì so sánh khaNangChua (giảm dần).
  10. **Đầu vào, đầu ra:**
  11. **Đầu vào: Danh sách các phòng thi với n phần tử.**
  12. **Đầu ra: Danh sách các phòng thi đã được sắp xếp theo từng yêu cầu.**

**II.Thuật toán**

#### **1. Sắp xếp theo Khả năng chứa (giảm dần):**

* 1. Dùng thuật toán Quick Sort để sắp xếp mảng theo thứ tự giảm dần của trường khaNangChua.
  2. Ý tưởng: Chọn một phần tử làm chốt, phân chia mảng thành hai phần sao cho các phần tử lớn hơn chốt nằm bên trái và nhỏ hơn chốt nằm bên phải.
  3. So sánh trực tiếp giá trị khaNangChua của các phòng thi.

#### **2. Sắp xếp theo Nhà, Số phòng (tăng dần):**

* 1. Dùng thuật toán Quick Sort để sắp xếp mảng theo thứ tự tăng dần của nha.
  2. Nếu hai phòng thi có cùng nha, thì tiếp tục so sánh soPhong và sắp xếp tăng dần.
  3. Ý tưởng: Chọn một phần tử làm chốt, phân chia mảng theo thứ tự nha. Với các phần tử có nha giống nhau, lại tiếp tục phân chia theo soPhong.

#### **3. Sắp xếp theo Nhà, Khả năng chứa (giảm dần):**

* 1. Dùng thuật toán Quick Sort để sắp xếp mảng theo thứ tự tăng dần của nha.
  2. Nếu hai phòng thi có cùng nha, thì tiếp tục so sánh khaNangChua và sắp xếp giảm dần.
  3. Ý tưởng: Chọn một phần tử làm chốt, phân chia mảng theo thứ tự nha. Với các phần tử có nha giống nhau, lại tiếp tục phân chia theo khaNangChua theo chiều giảm dần.

**II.Testcase:**

**Dữ liệu đầu vào:**

Danh sách các phòng thi tại một hội đồng thi:

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | **Số phòng** | | Nhà | Khả Năng chứa |
| 101 | A | 150 |
| 102 | A | 100 |
| 201 | B | 200 |
| 202 | B | 80 |
| 301 | C | 250 |
| 302 | C | 120 |
| 103 | A | 180 |
| 203 | B | 90 |
| 303 | C | 160 |
| 104 | A | 110 |

#### **Yêu cầu 1: Sắp xếp theo Khả năng chứa (giảm dần)**

|  |  |  |
| --- | --- | --- |
| **Số phòng** | **Nhà** | **Khả năng chứa** |
| 301 | C | 250 |
| 103 | A | 180 |
| 303 | C | 160 |
| 101 | A | 150 |
| 302 | C | 120 |
| 104 | A | 110 |
| 102 | A | 100 |
| 203 | B | 90 |
| 201 | B | 200 |
| 202 | B | 80 |

#### **Yêu cầu 2: Sắp xếp theo Nhà (tăng dần), cùng Nhà thì theo Số phòng (tăng dần)**

|  |  |  |
| --- | --- | --- |
| **Số phòng** | **Nhà** | **Khả năng chứa** |
| 101 | A | 150 |
| 102 | A | 100 |
| 103 | A | 180 |
| 104 | A | 110 |
| 201 | B | 200 |
| 202 | B | 80 |
| 203 | B | 90 |
| 301 | C | 250 |
| 302 | C | 120 |
| 303 | C | 160 |

#### **Yêu cầu 3: Sắp xếp theo Nhà (tăng dần), cùng Nhà thì theo Khả năng chứa (giảm dần)**

|  |  |  |
| --- | --- | --- |
| **Số phòng** | **Nhà** | **Khả năng chứa** |
| 103 | A | 180 |
| 101 | A | 150 |
| 104 | A | 110 |
| 102 | A | 100 |
| 201 | B | 200 |
| 203 | B | 90 |
| 202 | B | 80 |
| 301 | C | 250 |
| 303 | C | 160 |
| 302 | C | 120 |

Code:

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  struct Examroom {  int roomNumber;  int capacity;  char building;  Examroom(int roomNumber = 0, int capacity = 0, char building = 'A') {  this->roomNumber = roomNumber;  this->capacity = capacity;  this->building = building;  }  };  void printRooms(const vector<Examroom>& rooms) {  for (const auto& room : rooms) {  cout << "Phòng " << room.roomNumber  << ", Nhà " << room.building  << ", Khả năng chứa " << room.capacity << endl;  }  cout << endl;  }  int partitionByCapacity(vector<Examroom>& rooms, int low, int high) {  int pivot = rooms[high].capacity;  int left = low;  for (int i = low; i < high; i++) {  if (rooms[i].capacity > pivot) {  swap(rooms[i], rooms[left]);  left++;  }  }  swap(rooms[left], rooms[high]);  return left;  }  void quickSortByCapacity(vector<Examroom>& rooms, int low, int high) {  if (low < high) {  int pivot = partitionByCapacity(rooms, low, high);  quickSortByCapacity(rooms, low, pivot - 1);  quickSortByCapacity(rooms, pivot + 1, high);  }  }  int partitionByBuildingRoom(vector<Examroom>& rooms, int low, int high) {  Examroom pivot = rooms[high];  int left = low;  for (int i = low; i < high; i++) {  if (rooms[i].building < pivot.building ||  (rooms[i].building == pivot.building && rooms[i].roomNumber < pivot.roomNumber)) {  swap(rooms[i], rooms[left]);  left++;  }  }  swap(rooms[left], rooms[high]);  return left;  }  void quickSortByBuildingRoom(vector<Examroom>& rooms, int low, int high) {  if (low < high) {  int pivot = partitionByBuildingRoom(rooms, low, high);  quickSortByBuildingRoom(rooms, low, pivot - 1);  quickSortByBuildingRoom(rooms, pivot + 1, high);  }  }  int partitionByBuildingCapacity(vector<Examroom>& rooms, int low, int high) {  Examroom pivot = rooms[high];  int left = low;  for (int i = low; i < high; i++) {  if (rooms[i].building < pivot.building ||  (rooms[i].building == pivot.building && rooms[i].capacity > pivot.capacity)) {  swap(rooms[i], rooms[left]);  left++;  }  }  swap(rooms[left], rooms[high]);  return left;  }  void quickSortByBuildingCapacity(vector<Examroom>& rooms, int low, int high) {  if (low < high) {  int pivot = partitionByBuildingCapacity(rooms, low, high);  quickSortByBuildingCapacity(rooms, low, pivot - 1);  quickSortByBuildingCapacity(rooms, pivot + 1, high);  }  }  void test1() {  vector<Examroom> rooms = {  {103, 150, 'B'},  {102, 200, 'A'},  {101, 120, 'A'},  {105, 180, 'B'},  {104, 100, 'C'}  };  cout << "Test 1 - Trước khi sắp xếp:\n";  printRooms(rooms);  quickSortByCapacity(rooms, 0, rooms.size() - 1);  cout << "Test 1 - Sắp xếp theo Khả năng chứa (Giảm dần):\n";  printRooms(rooms);  quickSortByBuildingRoom(rooms, 0, rooms.size() - 1);  cout << "Test 1 - Sắp xếp theo Nhà (A → Z) & Số phòng (Tăng dần):\n";  printRooms(rooms);  quickSortByBuildingCapacity(rooms, 0, rooms.size() - 1);  cout << "Test 1 - Sắp xếp theo Nhà (A → Z) & Khả năng chứa (Giảm dần):\n";  printRooms(rooms);  }  void test2() {  vector<Examroom> rooms = {  {201, 100, 'C'},  {105, 250, 'B'},  {103, 180, 'B'},  {101, 130, 'A'},  {102, 220, 'A'}  };  cout << "Test 2 - Trước khi sắp xếp:\n";  printRooms(rooms);  quickSortByCapacity(rooms, 0, rooms.size() - 1);  cout << "Test 2 - Sắp xếp theo Khả năng chứa (Giảm dần):\n";  printRooms(rooms);  quickSortByBuildingRoom(rooms, 0, rooms.size() - 1);  cout << "Test 2 - Sắp xếp theo Nhà (A → Z) & Số phòng (Tăng dần):\n";  printRooms(rooms);  quickSortByBuildingCapacity(rooms, 0, rooms.size() - 1);  cout << "Test 2 - Sắp xếp theo Nhà (A → Z) & Khả năng chứa (Giảm dần):\n";  printRooms(rooms);  }  int main() {  test1();  // test2();  return 0;  } |

**Bài tập 4.** Cho ma trận hai chiều *m* dòng *n* cột, các phần tử là các số nguyên dương.

1. Tìm số nguyên tố lớn nhất trong ma trận.
2. Tìm những dòng của ma trận có chứa giá trị nguyên tố.
3. Tìm những dòng của ma trận chỉ chứa các số nguyên tố.

Bài làm

**I.Ý Tưởng:**

#### **1. Tìm số nguyên tố lớn nhất trong ma trận:**

* 1. Duyệt từng phần tử trong ma trận.
  2. Kiểm tra xem phần tử đó có phải số nguyên tố hay không.
  3. Nếu có, so sánh với số nguyên tố lớn nhất đã tìm thấy.
  4. Cập nhật giá trị lớn nhất nếu tìm thấy số nguyên tố lớn hơn.
  5. Sau khi duyệt hết ma trận, trả về số nguyên tố lớn nhất.

#### **2. Tìm những dòng của ma trận có chứa ít nhất một số nguyên tố:**

* 1. Duyệt từng dòng của ma trận.
  2. Kiểm tra từng phần tử trong dòng, nếu có số nguyên tố thì đánh dấu dòng đó.
  3. Lưu lại danh sách các dòng chứa ít nhất một số nguyên tố.

#### **3. Tìm những dòng của ma trận chỉ chứa số nguyên tố:**

* 1. Duyệt từng dòng của ma trận.
  2. Kiểm tra tất cả phần tử trong dòng:
  3. Nếu toàn bộ phần tử trong dòng đều là số nguyên tố → thêm dòng đó vào danh sách kết quả.
  4. Trả về danh sách các dòng thỏa mãn điều kiện.

**II.Thuật toán:**

#### **1. Tìm số nguyên tố lớn nhất trong ma trận:**

* 1. Duyệt từng phần tử trong ma trận, kiểm tra số đó có phải số nguyên tố không.
  2. Nếu có, so sánh với số nguyên tố lớn nhất đã tìm thấy.
  3. Kết quả là số nguyên tố lớn nhất trong ma trận.

#### **2. Tìm những dòng của ma trận có chứa giá trị nguyên tố:**

* 1. Duyệt từng dòng trong ma trận.
  2. Nếu dòng chứa ít nhất một số nguyên tố, lưu lại chỉ số dòng đó.

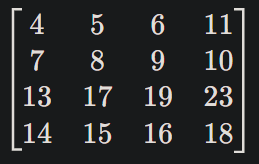
#### **3. Tìm những dòng của ma trận chỉ chứa các số nguyên tố:**

* 1. Duyệt từng dòng trong ma trận.
  2. Nếu tất cả phần tử trong dòng đều là số nguyên tố, lưu lại chỉ số dòng đó.

**III.Testcase:**

#### **Dữ liệu đầu vào:**

Cho ma trận *4×44* :



#### **Kết quả mong đợi:**

* 1. **Số nguyên tố lớn nhất:**
  2. Các số nguyên tố trong ma trận: 5, 11, 7, 13, 17, 19, 23
  3. Số lớn nhất: 23
  4. **Những dòng chứa số nguyên tố:**
  5. Dòng 1: (có 5, 11)
  6. Dòng 2: (có 7)
  7. Dòng 3: (có 13, 17, 19, 23)

**→ Kết quả: Dòng 1, 2, 3**

* 1. **Những dòng chỉ chứa số nguyên tố:**
  2. Dòng 1: (4 không phải số nguyên tố) → ❌
  3. Dòng 2: (8, 9, 10 không phải số nguyên tố) → ❌
  4. Dòng 3: (chỉ chứa 13, 17, 19, 23 – toàn số nguyên tố) → ✅
  5. Dòng 4: (14, 15, 16, 18 không phải số nguyên tố) → ❌

**→ Kết quả: Dòng 3**

**Code:**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  void printMatrix(int\*\* Matrix, int row, int col);  bool Is\_prime(int n);  int Max\_prime\_element(int\*\* Matrix, int row, int col);  vector<int> Row\_only\_have\_prime\_element(int\*\* Matrix, int row, int col);  void test1();  void test2();  int main() {  test1();  test2();  return 0;  }  void printMatrix(int\*\* Matrix, int row, int col) {  cout << "Ma trận:\n";  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  cout << Matrix[i][j] << " ";  }  cout << endl;  }  }  bool Is\_prime(int n) {  if (n < 2) return false;  if (n == 2) return true;  for (int i = 2; i <= sqrt(n); i++) {  if (n % i == 0) return false;  }  return true;  }  int Max\_prime\_element(int\*\* Matrix, int row, int col) {  int Max\_element = -1;  for (int i = 0; i < row; i++) {  for (int j = 0; j < col; j++) {  if (Is\_prime(Matrix[i][j]) && Matrix[i][j] > Max\_element) {  Max\_element = Matrix[i][j];  }  }  }  return Max\_element;  }  vector<int> Row\_only\_have\_prime\_element(int\*\* Matrix, int row, int col) {  vector<int> rows;  for (int i = 0; i < row; i++) {  bool allPrime = true;  for (int j = 0; j < col; j++) {  if (!Is\_prime(Matrix[i][j])) {  allPrime = false;  break;  }  }  if (allPrime) {  rows.push\_back(i);  }  }  return rows;  }  void test1() {  int row = 3, col = 3;  int\*\* Matrix = new int\*[row];  for (int i = 0; i < row; i++) {  Matrix[i] = new int[col];  }  int testData[3][3] = {  {2, 3, 5},  {4, 6, 8},  {11, 13, 17}  };  for (int i = 0; i < row; i++)  for (int j = 0; j < col; j++)  Matrix[i][j] = testData[i][j];  cout << "===== TEST 1 ====="<<endl;  printMatrix(Matrix, row, col);  cout << "Số nguyên tố lớn nhất: " << Max\_prime\_element(Matrix, row, col) << endl;  vector<int> primeRows = Row\_only\_have\_prime\_element(Matrix, row, col);  cout << "Các hàng chỉ chứa số nguyên tố: ";  if (primeRows.empty()) cout << "Không có hàng nào.";  else {  for (int r : primeRows) cout << r << " ";  }  cout << endl;  for (int i = 0; i < row; i++) delete[] Matrix[i];  delete[] Matrix;  }  void test2() {  int row = 4, col = 4;  int\*\* Matrix = new int\*[row];  for (int i = 0; i < row; i++) {  Matrix[i] = new int[col];  }  int testData[4][4] = {  {7, 11, 13, 17},  {10, 12, 14, 16},  {19, 23, 29, 31},  {6, 9, 15, 21}  };  for (int i = 0; i < row; i++)  for (int j = 0; j < col; j++)  Matrix[i][j] = testData[i][j];  cout << "===== TEST 2 ====="<<endl;  printMatrix(Matrix, row, col);  cout << "Số nguyên tố lớn nhất: " << Max\_prime\_element(Matrix, row, col) << endl;  vector<int> primeRows = Row\_only\_have\_prime\_element(Matrix, row, col);  cout << "Các hàng chỉ chứa số nguyên tố: ";  if (primeRows.empty()) cout << "Không có hàng nào.";  else {  for (int r : primeRows) cout << r << " ";  }  cout << endl;  for (int i = 0; i < row; i++) delete[] Matrix[i];  delete[] Matrix;  } |

**Bài tập 5.** Cho ma trận hai chiều *m* dòng *n* cột, các phần tử là các số nguyên dương.

1. Tìm dòng có tổng lớn nhất.

©DNTAI

* 1. Sắp xếp các dòng sao cho dòng có tổng các phần tử lớn hơn sẽ nằm phía trên.

Bài làm

**I.Ý Tưởng:**

#### **1. Tìm dòng có tổng lớn nhất:**

* 1. Khởi tạo biến maxSum = 0 để lưu tổng lớn nhất và maxRow để lưu chỉ số dòng có tổng lớn nhất.
  2. Duyệt từng dòng trong ma trận, tính tổng các phần tử của dòng đó.
  3. So sánh với maxSum, nếu tổng lớn hơn thì cập nhật maxSum và maxRow.
  4. Sau khi duyệt hết ma trận, trả về dòng có tổng lớn nhất.

#### **2. Sắp xếp các dòng theo tổng giảm dần:**

* 1. Tính tổng từng dòng và lưu vào một mảng phụ.
  2. Sử dụng một thuật toán sắp xếp (Bubble Sort, Selection Sort, Quick Sort, v.v.) để sắp xếp các dòng theo tổng giảm dần.
  3. In ra ma trận sau khi sắp xếp.

**II.Thuật Toán:**  
**1. Tìm dòng có tổng lớn nhất:**

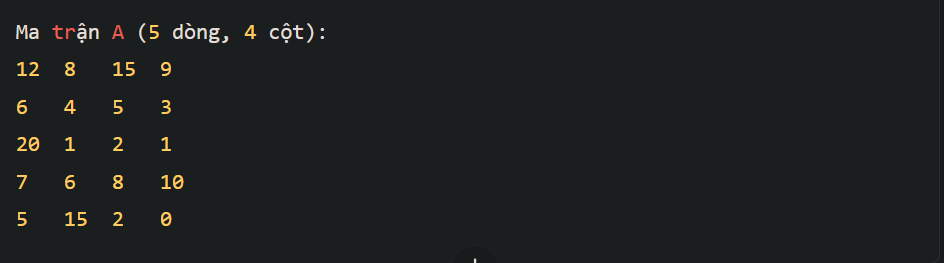
* 1. Duyệt từng dòng i từ 0 đến m - 1:
  2. Tính tổng sum của dòng i.
  3. Nếu sum > maxSum, cập nhật maxSum = sum và maxRow = i.
  4. Kết quả là dòng maxRow có tổng lớn nhất maxSum.

#### **2. Sắp xếp các dòng theo tổng giảm dần:**

* 1. Tạo mảng rowSums chứa tổng các phần tử của từng dòng.
  2. Áp dụng thuật toán Bubble Sort để sắp xếp các dòng theo tổng giảm dần:
  3. Duyệt qua tất cả các dòng i từ 0 đến m - 2:
  4. Duyệt qua tất cả các dòng j từ 0 đến m - 2 - i:

Nếu rowSums[j] < rowSums[j+1], hoán đổi rowSums[j] với rowSums[j+1] và hoán đổi hai dòng trong ma trận.

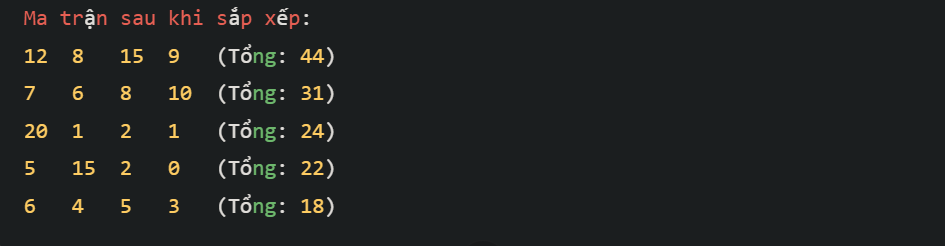
**III.Testcase:**



#### **Kết quả tìm dòng có tổng lớn nhất:**

* 1. Tổng các dòng:
  2. Dòng 0: 12 + 8 + 15 + 9 = 44
  3. Dòng 1: 6 + 4 + 5 + 3 = 18
  4. Dòng 2: 20 + 1 + 2 + 1 = 24
  5. Dòng 3: 7 + 6 + 8 + 10 = 31
  6. Dòng 4: 5 + 15 + 2 + 0 = 22
  7. **Dòng có tổng lớn nhất là dòng 0 với tổng = 44.**

#### **Kết quả sắp xếp các dòng theo tổng giảm dần:**



Code:

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  void ScanArray2D(int \*\*array2d, int &row, int &collum);  vector<int>sumofRows(int \*\*array2d, int row, int collum);  vector<int>RowshaveMaxSum(int \*\*array2d, int row, int collum);  int partition(int \*a, int low, int high);  void quickSort(int \*a, int low, int high);  int main ()  {  return 0;  }  void ScanArray2D(int \*\*array2d, int &row, int &collum)  {  cin >> row >> collum;  array2d = new int \*[row];  for (int i = 0; i < row; i++)  array2d[i] = new int[collum];  for (int i = 0; i < row; i++)  for (int j = 0; j < collum; j++)  cin >> array2d[i][j];  }  vector<int>sumofRows(int \*\*array2d, int row, int collum)  {  vector<int>sum;  for (int i = 0; i < row; i++)  {  int sumRow = 0;  for (int j = 0; j < collum; j++)  sumRow += array2d[i][j];  sum.push\_back(sumRow);  }  return sum;  }  vector<int>RowshaveMaxSum(int \*\*array2d, int row, int collum)  {  vector<int>sum = sumofRows(array2d, row, collum);  vector<int>result;  int maxSum = sum[0];  for (int i = 1; i < row; i++)  if (sum[i] > maxSum)  maxSum = sum[i];  for (int i = 0; i < row; i++)  if (sum[i] == maxSum)  result.push\_back(i);  return result;  } |

**Bài tập 6.** Cho mảng một chiều gồm *n* phần tử là các số nguyên không âm.

* Hãy sắp xếp các số chẵn trong mảng theo thứ tự tăng dần.
* Sắp xếp các số lẻ theo thứ tự giảm dần.
* Các số 0 giữ nguyên vị trí.

Bài làm

**I.Ý Tưởng:**

**1.Phân tích đề bài:**

* 1. Cho một mảng một chiều gồm n phần tử là các số nguyên không âm.
  2. Yêu cầu:
  3. Sắp xếp các số chẵn theo thứ tự tăng dần.
  4. Sắp xếp các số lẻ theo thứ tự giảm dần.
  5. Các số 0 phải giữ nguyên vị trí.
  6. **Phương pháp giải:**
  7. **Bước 1: Tạo hai mảng phụ** evenNumbers và oddNumbers để chứa số chẵn và số lẻ từ mảng gốc.
  8. **Bước 2: Duyệt qua mảng gốc:**
  9. Nếu phần tử là số 0, bỏ qua.
  10. Nếu phần tử là số chẵn khác 0, thêm vào evenNumbers.
  11. Nếu phần tử là số lẻ, thêm vào oddNumbers.
  12. **Bước 3: Sắp xếp:**
  13. evenNumbers theo thứ tự tăng dần.
  14. oddNumbers theo thứ tự giảm dần.
  15. **Bước 4: Tạo một mảng kết quả bằng cách duyệt qua mảng gốc:**
  16. Nếu phần tử là số 0, giữ nguyên.
  17. Nếu phần tử là số chẵn, lấy phần tử đầu tiên trong evenNumbers rồi xoá khỏi mảng đó.
  18. Nếu phần tử là số lẻ, lấy phần tử đầu tiên trong oddNumbers rồi xoá khỏi mảng đó.
  19. **Nhận xét:**
  20. Thao tác phân loại chỉ duyệt qua mảng một lần => Độ phức tạp O(n).
  21. Thao tác sắp xếp phụ thuộc vào số lượng số chẵn và số lẻ:
  22. evenNumbers: O(k1 log k1), với k1 là số phần tử chẵn.
  23. oddNumbers: O(k2 log k2), với k2 là số phần tử lẻ.
  24. Thao tác ghép mảng kết quả lại là O(n).
  25. **Tổng độ phức tạp: O(n log n) trong trường hợp xấu nhất (toàn bộ là số lẻ hoặc số chẵn).**

**II.Thuật Toán:**

#### **1**. Sắp xếp số chẵn tăng dần, số lẻ giảm dần, giữ nguyên số 0:

* 1. Tạo hai mảng phụ evenNumbers và oddNumbers.
  2. Duyệt qua từng phần tử của mảng gốc:
  3. Nếu phần tử là số 0, bỏ qua.
  4. Nếu phần tử là số chẵn khác 0, thêm vào evenNumbers.
  5. Nếu phần tử là số lẻ, thêm vào oddNumbers.
  6. Sắp xếp evenNumbers theo thứ tự tăng dần.
  7. Sắp xếp oddNumbers theo thứ tự giảm dần.
  8. Tạo một mảng kết quả bằng cách duyệt qua mảng gốc:
  9. Nếu phần tử là số 0, thêm số 0 vào mảng kết quả.
  10. Nếu phần tử là số chẵn, lấy số đầu tiên từ evenNumbers.
  11. Nếu phần tử là số lẻ, lấy số đầu tiên từ oddNumbers.

**III.Testcase:**

Ví dụ mảng đầu vào:

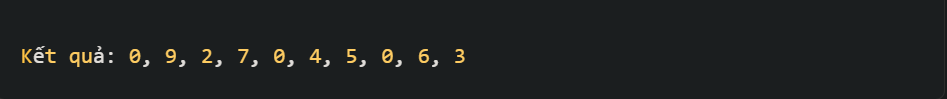


#### **Phân loại:**

* 1. Số chẵn: 4, 6, 2
  2. Số lẻ: 7, 3, 9, 5
  3. Số 0: 0, 0, 0

#### **Sắp xếp:**

* 1. Số chẵn tăng dần: 2, 4, 6
  2. Số lẻ giảm dần: 9, 7, 5, 3



**Code:**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  void readArray(int\* &a, int &n);  void printArray(int\* a, int n);  void swap(int &a, int &b);  void SortEvenOddKeepZero(int\* &array, int n);  int main()  {  int \*Array;  int n;  readArray(Array, n);  cout << "Mang ban dau la: ";  printArray(Array, n);  SortEvenOddKeepZero(Array, n);  cout << "\nMang sau khi sap xep la: ";  printArray(Array, n);  delete[] Array;  return 0;  }  void readArray(int\* &a, int &n)  {  cin >> n;  a = new int[n];  for (int i = 0; i < n; i++)  {  cin >> a[i];  }  }  void printArray(int\* a, int n)  {  for (int i = 0; i < n; i++)  {  cout << a[i] << " ";  }  }  void swap(int &a, int &b)  {  int temp = a;  a = b;  b = temp;  }  void SortEvenOddKeepZero(int\* &array, int n){  vector<int> evenindex, oddindex;  for (int i = 0; i < n; i++)  {  if (array[i] == 0)  {  continue;  }  if (array[i] % 2 == 0)  {  evenindex.push\_back(i);  }  else  {  oddindex.push\_back(i);  }  }  for(int i=0;i<evenindex.size()-1;i++){  for(int j=0;i<evenindex.size()-i-1;i++)  {  if(array[evenindex[j]]>array[evenindex[j+1]])  {  swap(array[evenindex[j]],array[evenindex[j+1]]);  }  }  }  for(int i=0;i<oddindex.size()-1;i++){  for(int j=0;i<oddindex.size()-i-1;i++)  {  if(array[oddindex[j]]>array[oddindex[j+1]])  {  swap(array[oddindex[j]],array[oddindex[j+1]]);  }  }  }  } |

**Bài tập 7.** Cho mảng một chiều gồm *n* phần tử là các số nguyên dương.

* 1. Hãy sắp xếp sao cho các phần tử chẵn ở đầu, các phần tử lẻ về cuối.
  2. Yêu cầu độ phức tạp là O(n).

Bài làm

I.Ý Tưởng:  
Phân tích đề bài:

-Cho một mảng một chiều gồm n phần tử là các số nguyên dương.

-Yêu cầu:

* 1. Sắp xếp sao cho:
  2. Các phần tử chẵn nằm ở đầu mảng.
  3. Các phần tử lẻ nằm ở cuối mảng.
  4. Độ phức tạp yêu cầu: O(n).

Phương pháp giải:

-Đây là bài toán phân loại (Partitioning) với yêu cầu độ phức tạp O(n), có thể giải bằng cách Duyệt hai con trỏ (Two-Pointer Approach).

Thuật toán:

-Khởi tạo hai con trỏ:

* 1. left = 0 (trỏ vào phần tử đầu tiên của mảng).
  2. right = n - 1 (trỏ vào phần tử cuối cùng của mảng).

-Lặp lại đến khi left >= right:

* 1. Nếu phần tử ở left là số chẵn (array[left] % 2 == 0) thì tăng **left lên (left++).**
  2. **Nếu phần tử ở right là số lẻ (array[right] % 2 == 1) thì giảm right xuống (right--).**
  3. **Nếu array[left] là số lẻ và array[right] là số chẵn, hoán đổi hai phần tử và cập nhật left++, right--.**

**Mảng sau khi hoàn thành sẽ có tất cả các số chẵn nằm ở đầu, số lẻ nằm ở cuối.**

**Nhận xét:**

* 1. **Độ phức tạp: O(n) vì mỗi phần tử được duyệt qua đúng một lần.**
  2. **Bộ nhớ: Chỉ cần dùng thêm một vài biến phụ nên không gian sử dụng là O(1).**
  3. **Phù hợp cho mảng có kích thước lớn vì không cần sắp xếp thật sự.**
  4. **II. Thuật Toán:**  
     **Khởi tạo hai con trỏ:**
  5. left = 0 (trỏ đến đầu mảng).
  6. right = n - 1 (trỏ đến cuối mảng).
  7. **Duyệt mảng:**
  8. Lặp lại khi left < right:
  9. Nếu array[left] là số chẵn (array[left] % 2 == 0):
  10. Tăng left lên 1 (left++).
  11. Nếu array[right] là số lẻ (array[right] % 2 == 1):
  12. Giảm right xuống 1 (right--).
  13. Nếu array[left] là số lẻ và array[right] là số chẵn:
  14. Hoán đổi array[left] và array[right].
  15. Tăng left lên 1 (left++) và giảm right xuống 1 (right--).
  16. **Kết thúc:**
  17. Sau khi vòng lặp kết thúc, mảng đã được phân loại thành các phần tử chẵn ở đầu và phần tử lẻ ở cuối.

**III.Testcase:**

|  |
| --- |
|  |

**Code:**

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  void ReadArray(int\* &array ,int &size);  void SwapEvennumandOddnum(int\* &nums ,int size);  void PrintArray(int\* array ,int size);  void test1() {  int\* array = new int[5]{1, 2, 3, 4, 5};  int size = 5;  SwapEvennumandOddnum(array, size);  PrintArray(array, size);  delete[] array;  }  void test2() {  int\* array = new int[10]{10, 21, 32, 43, 54, 65, 76, 87, 98, 109};  int size = 10;  SwapEvennumandOddnum(array, size);  PrintArray(array, size);  delete[] array;  }  void test3() {  int\* array = new int[8]{15, 14, 13, 12, 11, 10, 9, 8};  int size = 8;  SwapEvennumandOddnum(array, size);  PrintArray(array, size);  delete[] array;  }  void test4() {  int\* array = new int[7]{7, 14, 21, 28, 35, 42, 49};  int size = 7;  SwapEvennumandOddnum(array, size);  PrintArray(array, size);  delete[] array;  }  int main() {  test1();  cout << endl;  test2();  cout << endl;  test3();  cout << endl;  test4();  cout << endl;  return 0;  }  void ReadArray(int\* &array ,int &size)  {  cout << "Nhap so phan tu cua mang: ";  cin >> size;  array = new int[size];  for (int i = 0; i < size; i++)  {  cout << "Nhap phan tu thu " << i + 1 << ": ";  cin >> array[i];  }  }  void PrintArray(int\* array ,int size)  {  for (int i = 0; i < size; i++)  {  cout << array[i] << " ";  }  }  void SwapEvennumandOddnum(int\* &nums ,int size)  {  int left = 0, right = size - 1;  while(left < right) {  if(nums[left] % 2 > nums[right] % 2)  swap(nums[left], nums[right]);    if(nums[left] % 2 == 0)  left++;  if(nums[right] % 2 == 1)  right--;  }  } |

**Bài tập 8.** Với mỗi hoán vị A(𝑎𝑎1, 𝑎𝑎2, 𝑎𝑎3, … , 𝑎𝑎𝑁𝑁) của N số tự nhiên đầu tiên, ta ký hiệu: 𝐵𝐵 = (𝑏𝑏1, 𝑏𝑏2, … , 𝑏𝑏𝑁𝑁) là một mảng, trong đó:

𝑏𝑏𝑖𝑖 = số lượng phần tử đứng trước số 𝑖𝑖 và lớn hơn 𝑖𝑖, ∀1 ≤ 𝑖𝑖 ≤ 𝑁𝑁.

Mảng B được gọi là **mảng nghịch thế** của mảng A.

**Ví dụ:** Nếu A là một hoán vị của 9 số tự nhiên đầu tiên: A = (5, 9, 1, 8, 2, 6, 4, 7, 3)

Thì **mảng nghịch thế** sẽ là: B = (2, 3, 6, 4, 0, 2, 2, 1, 0)

**Yêu cầu bài toán**

1. Viết hàm tìm mảng nghịch thế của một hoán vị.
   1. Viết hàm tìm hoán vị của một mảng nghịch thế (nếu có).

Bài Làm

I.Ý Tưởng

* 1. Khởi tạo mảng B có kích thước N với tất cả phần tử bằng 0.
  2. Duyệt qua từng phần tử i trong mảng A:
  3. Với mỗi i, duyệt qua các phần tử j (từ 0 đến i - 1).
  4. Nếu A[j] > i thì tăng B[i - 1] lên 1.
  5. Kết thúc, mảng B sẽ chứa số lượng phần tử lớn hơn i đứng trước i.

### Ý tưởng tìm mảng hoán vị A từ mảng nghịch thế B (nếu có):

* 1. Khởi tạo mảng A có kích thước N với tất cả phần tử bằng 0.
  2. Tạo một danh sách S chứa các số tự nhiên từ 1 đến N.
  3. Duyệt qua từng phần tử i của B:
  4. Nếu B[i] lớn hơn hoặc bằng kích thước của S, trả về thông báo không tồn tại mảng hoán vị A.
  5. Chọn phần tử trong S theo chỉ số B[i] và gán vào A[i].
  6. Xóa phần tử đã chọn khỏi S.
  7. Trả về mảng A.

**II.Thuật Toán:**

### **✅ Thuật toán tìm mảng nghịch thế B từ mảng hoán vị A:**

**Input:** Mảng hoán vị A có N phần tử.

**Output:** Mảng nghịch thế B có N phần tử.

**Các bước thực hiện:**

-Khởi tạo mảng B kích thước N với tất cả phần tử bằng 0.

-Lặp qua từng phần tử i từ 1 đến N (chỉ số từ 0 đến N-1 trong mảng A):

* 1. Khởi tạo B[i] = 0.
  2. Lặp qua từng phần tử j từ 0 đến i - 1:
  3. Nếu A[j] > i + 1 thì tăng B[i] lên 1.

Trả về mảng B.

### **✅ Thuật toán tìm mảng hoán vị A từ mảng nghịch thế B:**

**-Input:** Mảng nghịch thế B có N phần tử.

**-Output:** Mảng hoán vị A có N phần tử hoặc thông báo không tồn tại.

**-Các bước thực hiện:**

* 1. Khởi tạo mảng A kích thước N với tất cả phần tử bằng 0.
  2. Tạo một danh sách S chứa các số tự nhiên từ 1 đến N.
  3. Lặp qua từng phần tử i từ 0 đến N - 1:
  4. Nếu B[i] lớn hơn hoặc bằng kích thước của S:
  5. Trả về thông báo "Không tồn tại mảng hoán vị A tương ứng".
  6. Gán A[i] bằng phần tử ở vị trí B[i] trong danh sách S.
  7. Xóa phần tử đó khỏi S.

-Trả về mảng A.

**III.Testcase:**  
Testcase 1:

* 1. Input: A = (5, 9, 1, 8, 2, 6, 4, 7, 3)
  2. Output: B = (2, 3, 6, 4, 0, 2, 2, 1, 0)

#### Testcase 2:

* 1. Input: A = (1, 2, 3, 4, 5)
  2. Output: B = (0, 0, 0, 0, 0)

#### Testcase 3:

* 1. Input: A = (2, 1, 4, 3)
  2. Output: B = (1, 0, 1, 0)

#### Testcase 4 (Ngược lại):

* 1. Input: B = (2, 3, 6, 4, 0, 2, 2, 1, 0)
  2. Output: A = (5, 9, 1, 8, 2, 6, 4, 7, 3)

Code:

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  void input\_data(int\* &array, int &size);  void output\_data(int\* array, int size);  void Merge(int\* array, int\* B, int left, int mid, int right);  void MergeSort(int\* array, int\* B, int left, int right);  void find\_permutation(int\* B, int size, int\* A);  int main()  {  int \*A, N;  input\_data(A, N);  int \*B = new int[N]();  MergeSort(A, B, 0, N - 1);  output\_data(B, N);  int\* A\_reconstructed = new int[N];  find\_permutation(B, N, A\_reconstructed);  output\_data(A\_reconstructed, N);  delete[] A;  delete[] B;  delete[] A\_reconstructed;  return 0;  }  void input\_data(int\* &array, int &size)  {  cin >> size;  array = new int[size];  for (int i = 0; i < size; i++)  {  cin >> array[i];  }  }  void output\_data(int\* array, int size)  {  for (int i = 0; i < size; i++)  {  cout << array[i] << " ";  }  cout << endl;  }  void Merge(int\* array, int\* B, int left, int mid, int right)  {  int n1 = mid - left + 1;  int n2 = right - mid;  vector<int> L(n1), R(n2);    for (int i = 0; i < n1; i++) L[i] = array[left + i];  for (int i = 0; i < n2; i++) R[i] = array[mid + 1 + i];  int i = 0, j = 0, k = left, count = 0;  while (i < n1 && j < n2)  {  if (L[i] <= R[j])  {  array[k] = L[i];  B[L[i] - 1] += count;  i++;  }  else  {  array[k] = R[j];  count++;  j++;  }  k++;  }  while (i < n1)  {  array[k] = L[i];  B[L[i] - 1] += count;  i++;  k++;  }  while (j < n2)  {  array[k] = R[j];  j++;  k++;  }  }  void MergeSort(int\* array, int\* B, int left, int right)  {  if (left < right)  {  int mid = left + (right - left) / 2;  MergeSort(array, B, left, mid);  MergeSort(array, B, mid + 1, right);  Merge(array, B, left, mid, right);  }  }  void find\_permutation(int\* B, int size, int\* A)  {  vector<int> remaining;  for (int i = 1; i <= size; i++)  remaining.push\_back(i);    for (int i = 0; i < size; i++)  {  A[i] = remaining[B[i]];  remaining.erase(remaining.begin() + B[i]);  }  } |

**Bài tập 9.** Cho thông tin về một sinh viên bao gồm:

* Mã số – là một số nguyên dương.
* Họ và đệm – là một chuỗi có tối đa 20 ký tự.
* Tên sinh viên – là một chuỗi có tối đa 40 ký tự.
* Ngày, tháng, năm sinh – là các số nguyên dương.
* Phái – là “Nam” hoặc “Nữ”.
* Điểm trung bình – là các số thực có giá trị từ 0.00 → 10.00.

1. Viết chương trình nhập vào danh sách sinh viên (ít nhất là 10 sinh viên), không nhập trùng mã giữa các sinh viên với nhau và lưu trữ danh sách này vào một tập tin có tên SINHVIEN.DAT.
2. Sau đó vận dụng các thuật toán sắp xếp đã học để sắp xếp danh sách sinh viên theo thứ tự tăng dần theo Mã sinh viên. In danh sách sinh viên trong tập tin SINHVIEN.DAT sau khi sắp xếp ra màn hình.
3. Tạo các tập tin chỉ mục theo các khóa trong các trường hợp sau:

* Chỉ mục sắp xếp theo Mã sinh viên tăng dần.
* Chỉ mục sắp xếp theo Tên sinh viên từ A → Z, nếu cùng tên thì sắp xếp Họ và đệm theo thứ tự A → Z.
* Chỉ mục sắp xếp theo Điểm trung bình giảm dần.

1. Lưu các tập tin chỉ mục theo các khóa nêu trên vào trong ba tập tin tương ứng là:

* SVMASO.IDX (sắp xếp theo Mã sinh viên)
* SVTH.IDX (sắp xếp theo Tên sinh viên)
* SVDTB.IDX (sắp xếp theo Điểm trung bình)

1. Dựa vào các tập tin chỉ mục, mở và đọc danh sách sinh viên trong tập tin SINHVIEN.DAT

theo đúng thứ tự sắp xếp ứng với từng tập tin chỉ mục.

* 1. Nhận xét về cách thực hiện việc sắp xếp dữ liệu trên tập tin theo chỉ mục.

Bài Làm

**I.Ý Tưởng**

Bài toán yêu cầu sắp xếp danh sách sinh viên lưu trong file nhị phân SINHVIEN.DAT theo nhiều khóa khác nhau và tạo ra các file chỉ mục tương ứng (SVMASO.IDX, SVTH.IDX, SVDTB.IDX).

#### 1. Nhập dữ liệu và lưu file SINHVIEN.DAT:

* 1. Khai báo cấu trúc SinhVien với các thuộc tính cần thiết (MaSo, HoVaDem, Ten, NgaySinh, ThangSinh, NamSinh, Phai, DiemTrungBinh).
  2. Nhập danh sách sinh viên từ bàn phím (ít nhất 10 sinh viên).
  3. Kiểm tra không nhập trùng MaSo bằng cách so sánh với những mã số đã nhập trước đó.
  4. Ghi dữ liệu vào file SINHVIEN.DAT theo định dạng nhị phân.

#### 2. Tạo file chỉ mục (index[]) và sắp xếp:

* 1. Đọc dữ liệu từ SINHVIEN.DAT vào mảng ds[].
  2. Khởi tạo mảng index[] chứa các chỉ số từ 0 đến n-1.
  3. Sắp xếp index[] theo nhiều tiêu chí khác nhau:

-Theo MaSo (tăng dần):

* 1. Sử dụng thuật toán sắp xếp bất kỳ (Bubble Sort, Selection Sort, Merge Sort, v.v.) để sắp xếp index[] dựa theo ds[index[i]].MaSo.
  2. Ghi index[] ra file SVMASO.IDX.
  3. Theo Ten (từ A -> Z), nếu trùng thì theo HoVaDem (A -> Z):
  4. So sánh ds[index[i]].Ten trước, nếu giống nhau thì so sánh ds[index[i]].HoVaDem.
  5. Ghi index[] ra file SVTH.IDX.
  6. Theo DiemTrungBinh (giảm dần):
  7. So sánh ds[index[i]].DiemTrungBinh theo thứ tự từ lớn đến bé.
  8. Ghi index[] ra file SVDTB.IDX.

#### 3. Đọc và hiển thị file theo chỉ mục:

* 1. Mở file SINHVIEN.DAT và các file chỉ mục (SVMASO.IDX, SVTH.IDX, SVDTB.IDX).
  2. Đọc index[] từ các file chỉ mục.
  3. Duyệt index[] và lấy phần tử tương ứng trong ds[] để hiển thị ra màn hình theo đúng thứ tự chỉ mục yêu cầu.

#### 4. Nhận xét về cách thực hiện:

* 1. Việc tạo chỉ mục giúp lưu được nhiều cách sắp xếp khác nhau mà không cần thay đổi file gốc SINHVIEN.DAT.
  2. Chỉ mục giúp tìm kiếm, sắp xếp và hiển thị dữ liệu theo yêu cầu một cách nhanh chóng.
  3. Không cần thay đổi thứ tự lưu trữ vật lý của dữ liệu trong file SINHVIEN.DAT.

**II.Thuật Toán**

* 1. Nhập và lưu dữ liệu sinh viên:
  2. Khai báo cấu trúc SinhVien chứa các trường:
  3. MaSo (int)
  4. HoVaDem (char[21])
  5. Ten (char[41])
  6. NgaySinh, ThangSinh, NamSinh (int)
  7. Phai (char[4])
  8. DiemTrungBinh (float)
  9. Nhập thông tin cho ít nhất 10 sinh viên.
  10. Kiểm tra không nhập trùng MaSo.
  11. Ghi danh sách sinh viên vào tệp SINHVIEN.DAT (dạng nhị phân).
  12. Tạo chỉ mục:
  13. Đọc dữ liệu từ SINHVIEN.DAT vào mảng sinh viên ds[].
  14. Tạo mảng index[] chứa các chỉ số tương ứng với mảng ds[].
  15. Sắp xếp theo các khóa:
  16. Theo MaSo: Sắp xếp index[] bằng cách so sánh ds[index[i]].MaSo.
  17. Theo Ten và HoVaDem: Sắp xếp index[] bằng cách so sánh ds[index[i]].Ten. Nếu trùng thì so sánh ds[index[i]].HoVaDem.
  18. Theo DiemTrungBinh: Sắp xếp index[] bằng cách so sánh ds[index[i]].DiemTrungBinh theo thứ tự giảm dần.
  19. Lưu tệp chỉ mục:
  20. Ghi index[] sau khi sắp xếp vào các tệp:
  21. SVMASO.IDX (sắp xếp theo MaSo)
  22. SVTH.IDX (sắp xếp theo Ten)
  23. SVDTB.IDX (sắp xếp theo DiemTrungBinh)
  24. Đọc dữ liệu theo chỉ mục:
  25. Đọc từng tệp chỉ mục (SVMASO.IDX, SVTH.IDX, SVDTB.IDX) để hiển thị danh sách sinh viên từ SINHVIEN.DAT theo đúng thứ tự sắp xếp.

**III.Testcase:**

|  |
| --- |
| * 1. MaSo: 102, HoVaDem: Le Van, Ten: An, NgaySinh: 10, ThangSinh: 5, NamSinh: 2001, Phai: Nam, DiemTrungBinh: 8.5   2. MaSo: 101, HoVaDem: Tran Thi, Ten: Binh, NgaySinh: 22, ThangSinh: 9, NamSinh: 2000, Phai: Nu, DiemTrungBinh: 9.0   3. MaSo: 103, HoVaDem: Nguyen Van, Ten: Cuong, NgaySinh: 15, ThangSinh: 12, NamSinh: 2002, Phai: Nam, DiemTrungBinh: 7.5   4. MaSo: 105, HoVaDem: Pham Thi, Ten: Dung, NgaySinh: 8, ThangSinh: 8, NamSinh: 2003, Phai: Nu, DiemTrungBinh: 8.0   5. MaSo: 104, HoVaDem: Hoang, Ten: Bao, NgaySinh: 3, ThangSinh: 6, NamSinh: 2001, Phai: Nam, DiemTrungBinh: 7.0   6. MaSo: 106, HoVaDem: Le, Ten: An, NgaySinh: 5, ThangSinh: 10, NamSinh: 2000, Phai: Nam, DiemTrungBinh: 9.2   7. MaSo: 107, HoVaDem: Nguyen, Ten: Bao, NgaySinh: 12, ThangSinh: 11, NamSinh: 1999, Phai: Nam, DiemTrungBinh: 8.8   8. MaSo: 108, HoVaDem: Tran, Ten: Dung, NgaySinh: 14, ThangSinh: 4, NamSinh: 2001, Phai: Nu, DiemTrungBinh: 7.9   9. MaSo: 109, HoVaDem: Pham Van, Ten: Cuong, NgaySinh: 18, ThangSinh: 2, NamSinh: 2002, Phai: Nam, DiemTrungBinh: 7.4   10. MaSo: 110, HoVaDem: Hoang Thi, Ten: Binh, NgaySinh: 9, ThangSinh: 7, NamSinh: 2003, Phai: Nu, DiemTrungBinh: 8.1 |

Code:

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <algorithm>  using namespace std;  struct Dayofbirth {  int day, month, year;  Dayofbirth() : day(0), month(0), year(0) {}  };  struct Student {  int ID;  string firstname\_lastname;  string name;  Dayofbirth dob;  bool gender;  double GPA;  };  void Press\_input(Student\* &A\_student, int \*NumberofStudent);  void Print\_list\_student\_into\_screen(Student\* A\_student, int NumberofStudent);  void Write\_list\_student\_into\_file(Student\* A\_student, int NumberofStudent);  void Read\_data(Student\* &A\_student, int &NumberofStudent);  void sort\_list\_student\_by\_ID(Student\* &A\_student, int NumberofStudent);  void Write\_list\_into\_ID\_file(Student\* A\_student, int NumberofStudent);  void Sort\_list\_student\_by\_name(Student\* &A\_student, int NumberofStudent);  void Write\_list\_into\_name\_file(Student\* A\_student, int NumberofStudent);  int partition(Student\* &A\_student, int low, int high);  void Sort\_list\_student\_by\_GPA(Student\* &A\_student, int low, int high);  void Write\_list\_into\_GPA\_file(Student\* A\_student, int NumberofStudent);  int main() {  Student\* A\_student = nullptr;  int NumberofStudent = 0;  Press\_input(A\_student, &NumberofStudent);  Print\_list\_student\_into\_screen(A\_student, NumberofStudent);  Write\_list\_student\_into\_file(A\_student, NumberofStudent);    Read\_data(A\_student, NumberofStudent);  Write\_list\_into\_ID\_file(A\_student, NumberofStudent);  Write\_list\_into\_name\_file(A\_student, NumberofStudent);  Write\_list\_into\_GPA\_file(A\_student, NumberofStudent);    delete[] A\_student;  return 0;  }  void Press\_input(Student\* &A\_student, int \*NumberofStudent) {  cout << "Enter the number of students: ";  cin >> \*NumberofStudent;  A\_student = new Student[\*NumberofStudent];  for (int i = 0; i < \*NumberofStudent; i++) {  bool idExists;  do {  idExists = false;  cout << "\nEnter the ID of student " << i + 1 << ": ";  cin >> A\_student[i].ID;  // Check for duplicate ID  for (int j = 0; j < i; j++) {  if (A\_student[j].ID == A\_student[i].ID) {  cout << "ID already exists. Please enter a different ID." << endl;  idExists = true;  break;  }  }  } while (idExists);  cin.ignore();  cout << "Enter the first name and last name (ho va dem) of student " << i + 1 << ": ";  getline(cin, A\_student[i].firstname\_lastname);  cout << "Enter the given name of student " << i + 1 << ": ";  getline(cin, A\_student[i].name);  cout << "Enter the day of birth of student " << i + 1 << ": ";  cin >> A\_student[i].dob.day;  cout << "Enter the month of birth of student " << i + 1 << ": ";  cin >> A\_student[i].dob.month;  cout << "Enter the year of birth of student " << i + 1 << ": ";  cin >> A\_student[i].dob.year;  cout << "Enter the gender of student " << i + 1 << " (Male:1, Female:0): ";  cin >> A\_student[i].gender;  cout << "Enter the GPA of student " << i + 1 << ": ";  cin >> A\_student[i].GPA;  }  }  void Print\_list\_student\_into\_screen(Student\* A\_student, int NumberofStudent) {  cout << "\nList of students:" << endl;  for (int i = 0; i < NumberofStudent; i++) {  cout << "Student " << i+1 << ":" << endl;  cout << "ID: " << A\_student[i].ID << endl;  cout << "First Name and Last Name: " << A\_student[i].firstname\_lastname << endl;  cout << "Given Name: " << A\_student[i].name << endl;  cout << "Date of Birth: " << A\_student[i].dob.day << "/" << A\_student[i].dob.month  << "/" << A\_student[i].dob.year << endl;  cout << "Gender: " << (A\_student[i].gender ? "Male" : "Female") << endl;  cout << "GPA: " << A\_student[i].GPA << endl;  cout << "---------------------------" << endl;  }  }  void Write\_list\_student\_into\_file(Student\* A\_student, int NumberofStudent) {  ofstream file("SINHVIEN.DAT");  if (!file) {  cout << "Cannot open file for writing." << endl;  return;  }  file << NumberofStudent << endl;  for (int i = 0; i < NumberofStudent; i++) {  file << A\_student[i].ID << endl;  file << A\_student[i].firstname\_lastname << endl;  file << A\_student[i].name << endl;  file << A\_student[i].dob.day << endl;  file << A\_student[i].dob.month << endl;  file << A\_student[i].dob.year << endl;  file << A\_student[i].gender << endl;  file << A\_student[i].GPA << endl;  }  file.close();  }  void Read\_data(Student\* &A\_student, int &NumberofStudent) {  ifstream file("SINHVIEN.DAT");  if (!file) {  cout << "Cannot open file for reading." << endl;  return;  }  int n;  file >> n;  file.ignore(); // Ignore newline after number  delete[] A\_student;  NumberofStudent = n;  A\_student = new Student[NumberofStudent];  for (int i = 0; i < NumberofStudent; i++) {  file >> A\_student[i].ID;  file.ignore();  getline(file, A\_student[i].firstname\_lastname);  getline(file, A\_student[i].name);  file >> A\_student[i].dob.day;  file >> A\_student[i].dob.month;  file >> A\_student[i].dob.year;  file >> A\_student[i].gender;  file >> A\_student[i].GPA;  file.ignore(); // Ignore newline after GPA  }  file.close();  }  void sort\_list\_student\_by\_ID(Student\* &A\_student, int NumberofStudent) {  for (int i = 0; i < NumberofStudent - 1; i++) {  for (int j = 0; j < NumberofStudent - i - 1; j++) {  if (A\_student[j].ID > A\_student[j+1].ID) {  swap(A\_student[j], A\_student[j+1]);  }  }  }  }  void Write\_list\_into\_ID\_file(Student\* A\_student, int NumberofStudent) {  sort\_list\_student\_by\_ID(A\_student, NumberofStudent);  ofstream file("SVMASO.IDX");  if (!file) {  cout << "Cannot open ID file for writing." << endl;  return;  }  file << NumberofStudent << endl;  for (int i = 0; i < NumberofStudent; i++) {  file << A\_student[i].ID << endl;  file << A\_student[i].firstname\_lastname << endl;  file << A\_student[i].name << endl;  file << A\_student[i].dob.day << endl;  file << A\_student[i].dob.month << endl;  file << A\_student[i].dob.year << endl;  file << A\_student[i].gender << endl;  file << A\_student[i].GPA << endl;  }  file.close();  }  void Sort\_list\_student\_by\_name(Student\* &A\_student, int NumberofStudent) {  for (int i = 0; i < NumberofStudent - 1; i++) {  int Min\_index = i;  for (int j = i + 1; j < NumberofStudent; j++) {  if (A\_student[j].name < A\_student[Min\_index].name ||  (A\_student[j].name == A\_student[Min\_index].name &&  A\_student[j].firstname\_lastname < A\_student[Min\_index].firstname\_lastname)) {  Min\_index = j;  }  }  swap(A\_student[i], A\_student[Min\_index]);  }  }  void Write\_list\_into\_name\_file(Student\* A\_student, int NumberofStudent) {  Sort\_list\_student\_by\_name(A\_student, NumberofStudent);  ofstream file("SVTH.IDX");  if (!file) {  cout << "Cannot open name file for writing." << endl;  return;  }  file << NumberofStudent << endl;  for (int i = 0; i < NumberofStudent; i++) {  file << A\_student[i].ID << " ";  file << A\_student[i].firstname\_lastname << " ";  file << A\_student[i].name << " ";  file << A\_student[i].dob.day << " ";  file << A\_student[i].dob.month << " ";  file << A\_student[i].dob.year << " ";  file << A\_student[i].gender << " ";  file << A\_student[i].GPA << endl;  }  file.close();  }  int partition(Student\* &A\_student, int low, int high) {  double pivot = A\_student[high].GPA;  int i = low - 1;  for (int j = low; j < high; j++) {  if (A\_student[j].GPA > pivot) { // Changed to sort descending  i++;  swap(A\_student[i], A\_student[j]);  }  }  swap(A\_student[i + 1], A\_student[high]);  return i + 1;  }  void Sort\_list\_student\_by\_GPA(Student\* &A\_student, int low, int high) {  if (low < high) {  int pi = partition(A\_student, low, high);  Sort\_list\_student\_by\_GPA(A\_student, low, pi - 1);  Sort\_list\_student\_by\_GPA(A\_student, pi + 1, high);  }  }  void Write\_list\_into\_GPA\_file(Student\* A\_student, int NumberofStudent) {  Sort\_list\_student\_by\_GPA(A\_student, 0, NumberofStudent - 1);  ofstream file("SVDTB.IDX");  if (!file) {  cout << "Cannot open GPA file for writing." << endl;  return;  }  file << NumberofStudent << endl;  for (int i = 0; i < NumberofStudent; i++) {  file << A\_student[i].ID << endl;  file << A\_student[i].firstname\_lastname << endl;  file << A\_student[i].name << endl;  file << A\_student[i].dob.day << endl;  file << A\_student[i].dob.month << endl;  file << A\_student[i].dob.year << endl;  file << A\_student[i].gender << endl;  file << A\_student[i].GPA << endl;  }  file.close();  } |

---o0o---

(Hết)