TRƯỜNG ĐẠI HỌC SÀI GÒN

**KHOA CÔNG NGHỆ THÔNG TIN**

Bài tập nhóm học phần Cấu trúc dữ liệu và Giải thuật

**THUẬT TOÁN SẮP XẾP**

**Sinh viên thực hiện:**

Đoàn Dương Thùy Linh – 3123411175

Trương Huy Bảo - 3124411039

**Giảng viên hướng dẫn:** Đỗ Như Tài

Thành phố Hồ Chí Minh, tháng 12 năm 2024

Mục lục

[Phân công 3](#_Toc192878643)

[Chương 1: Câu hỏi 4](#_Toc192878644)

[1.1. Trình bày tư tưởng của các thuật toán sắp xếp? 4](#_Toc192878645)

[1.2. Trong các thuật toán sắp xếp, bạn thích nhất thuật toán nào? Thuật toán nào bạn không thích nhất? Tại sao? 6](#_Toc192878646)

[1.3. 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. 6](#_Toc192878647)

[1.4. Hãy trình bày những ưu điểm và nhược điểm của mỗi thuật toánsắp xếp? Theo bạn, cách khắc phục những nhược điểm này là gì? 11](#_Toc192878648)

[Chương 2: Bài tập cơ sở 13](#_Toc192878649)

[2.1. 13](#_Toc192878650)

[a) Mô phỏng các bước sắp xếp tăng dần dãy số 13](#_Toc192878651)

[b) Cài đặt giải thuật 15](#_Toc192878652)

[**c) Độ phức tạp** 18](#_Toc192878653)

[2.2. 19](#_Toc192878654)

[**a) Mô phỏng các giải thuật** 19](#_Toc192878655)

[**b) cài đặt các giải thuật** 24](#_Toc192878656)

[**c) Độ phức tạp của các giải thuật đã triển khai** 29](#_Toc192878657)

[2.3. 33](#_Toc192878658)

[2.4. 38](#_Toc192878659)

[Chương 3: Bài tập ứng dụng 54](#_Toc192878660)

[3.1. 54](#_Toc192878661)

[3.2. 59](#_Toc192878662)

[3.3. 62](#_Toc192878663)

[3.4. 65](#_Toc192878664)

[3.5. 67](#_Toc192878665)

[3.6. 70](#_Toc192878666)

[3.7. 72](#_Toc192878667)

[3.8. 73](#_Toc192878668)

[3.9. 75](#_Toc192878669)

# Phân công

[1] Đoàn Dương Thùy Linh – 3123411175 đảm nhiệm:

- Phần câu hỏi: câu 2 và 4

- Phần bài tập cơ sở: câu 2 và 4

- Phần bài tập ứng dụng: câu 6, 7, 8, 9

[2] Trương Huy Bảo – 3124411039 đảm nhiệm

- Phần câu hỏi: câu 1 và 3

- Phần bài tập cơ sở: câu 1 và 3

- Phần bài tập ứng dụng: câu 1, 2, 3, 4, 5

# Chương 1: Câu hỏi

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

Chia thành các nhóm:

**Nhóm sắp xếp đơn giản**

Insertion Sort

* Bắt đầu với phần tử thứ hai (vì phần tử đầu tiên được xem là đã sắp xếp).
* So sánh phần tử hiện tại với phần tử phía trước nó.
* Nếu phần tử hiện tại nhỏ hơn phần tử phía trước, đổi chỗ chúng.
* Tiếp tục so sánh với các phần tử trước đó và đổi chỗ cho đến khi phần tử hiện tại được đặt ở vị trí đúng trong phần đã sắp xếp.
* Lặp lại các bước trên cho tất cả các phần tử trong mảng.

Selection Sort

* Duyệt qua từng phần tử của mảng.
* Tìm phần tử nhỏ nhất trong mảng chưa được sắp xếp.
* Đổi chỗ phần tử nhỏ nhất với phần tử đầu tiên của mảng chưa được sắp xếp.
* Lặp lại các bước trên cho phần còn lại của mảng.

Bubble Sort:

Chia thành hai loại:

* **Bubble Sort Không Cờ (Basic Bubble Sort)**
* Mỗi lần duyệt, so sánh từng cặp phần tử liền kề.
* Duyệt qua mảng nhiều lần.
* Nếu phần tử trước lớn hơn phần tử sau, hoán đổi chúng.
* Sau mỗi vòng lặp, phần tử lớn nhất sẽ "nổi" lên cuối mảng.
* Tiếp tục lặp lại quá trình này cho đến khi mảng được sắp xếp.
* Bubble Sort Có Cờ (Optimized Bubble Sort)
* Giống Bubble Sort cơ bản, nhưng thêm biến cờ (flag) để theo dõi xem có lần hoán đổi nào xảy ra trong vòng lặp hay không.
* Nếu sau một vòng lặp không có hoán đổi nào, nghĩa là mảng đã được sắp xếp, có thể dừng thuật toán sớm.

Interchange Sort

* Duyệt từng phần tử trong mảng.
* Với mỗi phần tử, so sánh nó với tất cả các phần tử đứng sau.
* Nếu phần tử sau nhỏ hơn phần tử hiện tại, hoán đổi chúng.
* Lặp lại quá trình cho đến khi mảng được sắp xếp.

**Nhóm sắp xếp chia để trị**

Quick sort

* Chọn một phần tử làm pivot.
* Chia danh sách thành hai phần: một bên nhỏ hơn pivot và một bên lớn hơn pivot.
* Gọi đệ quy để sắp xếp hai phần còn lại.
* Hoạt động nhanh nhưng không ổn định do phụ thuộc vào cách chọn pivot.

Merge sort

* Chia danh sách thành hai nửa cho đến khi còn các danh sách con có một phần tử.
* Gộp (merge) các danh sách con theo thứ tự đúng để tạo danh sách lớn hơn.
* Duy trì độ ổn định của thứ tự phần tử.

**Nhóm sắp xếp tối ưu hoá**

Heap Sort

* Xây dựng một heap (cấu trúc dữ liệu cây nhị phân).
* Liên tục trích xuất phần tử lớn nhất (hoặc nhỏ nhất) ra khỏi heap và đưa vào danh sách đã sắp xếp.
* Hiệu quả trong các tình huống yêu cầu sắp xếp liên tục.

**Nhóm sắp xếp đặc biệt**

Radix Sort (Sắp xếp theo chữ số)

* Xác định chữ số lớn nhất (số chữ số của phần tử lớn nhất trong mảng).
* Sắp xếp các phần tử dựa trên từng chữ số, bắt đầu từ chữ số ít quan trọng nhất (chữ số hàng đơn vị) đến chữ số quan trọng nhất (chữ số hàng cao nhất).
* Lặp lại quá trình sắp xếp cho mỗi chữ số, sử dụng một thuật toán sắp xếp ổn định như Counting Sort.

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

Nếu xét về hiệu quả và ứng dụng thực tế, Quick Sort là thuật toán mình thích nhất. Lý do là:

* Nhanh trên thực tế: Dù có độ phức tạp tệ nhất là O(n^2), nhưng nếu chọn pivot tốt, Quick Sort hoạt động trung bình với O(n log n), nhanh hơn so với các thuật toán sắp xếp O(n^2) như Bubble Sort hay Insertion Sort.
* Ít tốn bộ nhớ: Khác với Merge Sort (cần thêm bộ nhớ để lưu mảng tạm), Quick Sort chỉ sử dụng đệ quy mà không cần cấu trúc lưu trữ bổ sung.
* Được sử dụng rộng rãi: Nhiều thư viện lập trình như C++, Java, Python đều tối ưu hóa Quick Sort trong các hàm sắp xếp chuẩn (thường kết hợp với Insertion Sort để xử lý mảng nhỏ).

Ngược lại, thuật toán em không thích nhất là Bubble Sort, vì:

* Quá chậm: Nó có độ phức tạp O((n^2) ngay cả khi mảng gần như đã sắp xếp, trong khi Insertion Sort có thể chạy O(n) trong trường hợp đó.
* Không hiệu quả trên dữ liệu lớn: Khi số lượng phần tử tăng, thuật toán trở nên cực kỳ chậm so với Quick Sort, Merge Sort hay Heap Sort.
* Ít được sử dụng thực tế: Gần như không có ứng dụng thực tế nào dùng Bubble Sort vì có nhiều thuật toán tốt hơn.

## **1.3. 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.**

**Thuật toán sắp xếp nội**

**Interchange Sort (Sắp xếp đổi chỗ)**

1. Trình bày

* Duyệt từng phần tử.
* So sánh với tất cả phần tử phía sau.
* Nếu phần tử sau lớn hơn phần tử hiện tại, hoán đổi chúng.

2. Cài đặt

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]);

}

}

}

}

3. Nhận xét

Thuật toán này thích hợp cho các mảng có kích thước nhỏ và dễ hiểu, nhưng không hiệu quả đối với các mảng lớn do độ phức tạp thời gian là O(n2).

**Bubble Sort (Sắp xếp nổi bọt)**

1. Trình bày

* Lặp qua mảng nhiều lần.
* Hoán đổi nếu phần tử trước nhỏ hơn phần tử sau.
* Lặp lại đến khi không còn hoán đổi.

2. Cài đặt

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] < arr[j+1]) {

std::swap(arr[j], arr[j+1]);

}

}

}

}

3. Nhận xét

Bubble Sort là một thuật toán sắp xếp đơn giản nhưng không hiệu quả đối với các mảng lớn vì độ phức tạp thời gian là O(n2).

**Selection Sort (Sắp xếp chọn)**

1. Trình bày

* Tìm phần tử lớn nhất và đặt nó vào đúng vị trí.
* Lặp lại với phần còn lại của mảng.

2. Cài đặt

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int maxIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] > arr[maxIndex]) {

maxIndex = j;

}

}

swap(arr[i], arr[maxIndex]);

}

}

3 .Nhận xét

Selection Sort tìm phần tử lớn nhất trong mảng chưa sắp xếp và đổi chỗ với phần tử đầu tiên. Độ phức tạp thời gian là O(n2).

**Insertion Sort (Sắp xếp chèn)**

1. Trình bày

Duyệt từng phần tử, chèn nó vào đúng vị trí trong danh sách đã sắp xếp (bằng cách dịch các phần tử lớn hơn sang phải).

2. Cài đặt

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;

}

}

3.Nhận xét

Insertion Sort duyệt qua các phần tử của mảng và sắp xếp từng phần tử vào đúng vị trí của nó trong phần đã sắp xếp. Độ phức tạp thời gian là O(n2).

**Merge Sort (Sắp xếp trộn)**

1. Trình bày

* Chia mảng thành hai nửa.
* Sắp xếp từng nửa rồi gộp lại.

2. Cài đặt

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) {

L[i] = arr[l + i];

}

for (int i = 0; i < n2; i++) {

R[i] = arr[m + 1 + i];

}

int i = 0, j = 0, k = l;

while (i < n1 && j < n2) {

if (L[i] >= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

3. Nhận xét

Merge Sort là một thuật toán sắp xếp chia để trị với độ phức tạp thời gian là O(nlogn).

**Quick Sort (Sắp xếp nhanh)**

1. Trình bày

* Chọn **pivot**, chia mảng thành hai phần.
* Đệ quy sắp xếp từng phần.

2. Cài đặt

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] >= pivot) {

i++;

std::swap(arr[i], arr[j]);

}

}

std::swap(arr[i + 1], arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

3.Nhận xét

Quick Sort là một thuật toán sắp xếp hiệu quả với độ phức tạp thời gian trung bình là O(nlogn) và nhanh hơn Merge Sort.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuật toán** | **Ưu điểm** | **Nhược điểm** | **Cách khắc phục** |
| **Bubble Sort** | Dễ cài đặt, có thể tối ưu nếu mảng đã sắp xếp trước. | Chậm O(n2), không hiệu quả với dữ liệu lớn. | Thay thế bằng Insertion Sort hoặc Quick Sort. |
| **Selection Sort** | Ít hoán đổi hơn Bubble Sort, đơn giản. | Chậm O(n2), không ổn định. | Dùng Heap Sort để tối ưu hóa tìm phần tử nhỏ nhất. |
| **Insertion Sort** | Hiệu quả với mảng nhỏ hoặc gần sắp xếp (O(n)), ổn định. | Chậm O(n2)với dữ liệu ngẫu nhiên. | Dùng Binary Insertion Sort hoặc Quick Sort cho mảng lớn. |
| **Merge Sort** | Luôn O(n log n), ổn định, phù hợp dữ liệu lớn. | Cần bộ nhớ phụ O(n), chậm hơn Quick Sort trong thực tế. | Dùng Iterative Merge Sort hoặc Quick Sort để tiết kiệm bộ nhớ. |
| **Quick Sort** | Nhanh trung bình O(n log n), tiết kiệm bộ nhớ hơn Merge Sort. | Trường hợp xấu nhất O(n2), không ổn định. | Dùng Randomized Quick Sort hoặc Hybrid Quick Sort với Insertion Sort. |
| **Heap Sort** | Luôn O(n log n), không cần bộ nhớ phụ. | Không ổn định, hoán đổi nhiều hơn Quick Sort. | Dùng Quick Sort nếu muốn tốc độ nhanh hơn. |
| **Counting Sort** | Nhanh O(n+k) nếu k nhỏ, ổn định. | Tốn bộ nhớ nếu k lớn, chỉ áp dụng với số nguyên nhỏ. | Dùng Radix Sort hoặc Bucket Sort khi phạm vi dữ liệu rộng. |
| **Radix Sort** | Nhanh với số nguyên/chuỗi ngắn, ổn định. | Không hiệu quả với số thực hoặc dữ liệu không đồng nhất, tốn bộ nhớ. | Dùng Quick Sort hoặc Merge Sort nếu dữ liệu không phù hợp. |
| **Bucket Sort** | Nhanh O(n+k) khi dữ liệu phân bố đều. | Tốn bộ nhớ, không hiệu quả nếu dữ liệu không phân bố đều. | Dùng Quick Sort hoặc Merge Sort nếu dữ liệu không phù hợp. |

# **Chương 2: Bài tập cơ sở**

## **2.1.**

**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 giải:**

## **a) Mô phỏng các bước sắp xếp tăng dần dãy số**

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

|  |  |
| --- | --- |
| **Bước** | **Dãy số** |
| **Bước 1** | **8**, 39, 5, 1, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 2** | **5**, 8, 39, 1, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 3** | **1**, 5, 8, 39, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 4** | **1, 3**, 5, 8, 39, 6, 9, 12, 4, 7, 10 |
| **Bước 5** | **1, 3, 4**, 5, 8, 39, 6, 9, 12, 7, 10 |
| **Bước 6** | **1, 3, 4, 5**, 8, 6, 9, 12, 7, 10, 39 |
| **Bước 7** | **1, 3, 4, 5, 6**, 8, 9, 12, 7, 10, 39 |
| **Bước 8** | **1, 3, 4, 5, 6, 7**, 8, 9, 12, 10, 39 |
| **Bước 9** | **1, 3, 4, 5, 6, 7, 8**, 9, 12, 10, 39 |
| **Bước 10** | **1, 3, 4, 5, 6, 7, 8, 9**, 10, 12, 39 |
| **Bước 11** | **1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39** |

2. Sắp xếp chọn trực tiếp (Selection Sort)

|  |  |
| --- | --- |
| **Bước** | **Dãy số** |
| **Bước 1** | **1**, 8, 5, 39, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 2** | **1, 3**, 5, 39, 8, 6, 9, 12, 4, 7, 10 |
| **Bước 3** | **1, 3, 4**, 39, 8, 6, 9, 12, 5, 7, 10 |
| **Bước 4** | **1, 3, 4, 5**, 8, 6, 9, 12, 39, 7, 10 |
| **Bước 5** | **1, 3, 4, 5, 6**, 8, 9, 12, 39, 7, 10 |
| **Bước 6** | **1, 3, 4, 5, 6, 7**, 9, 12, 39, 8, 10 |
| **Bước 7** | **1, 3, 4, 5, 6, 7, 8**, 12, 39, 9, 10 |
| **Bước 8** | **1, 3, 4, 5, 6, 7, 8, 9**, 39, 12, 10 |
| **Bước 9** | **1, 3, 4, 5, 6, 7, 8, 9, 10**, 12, 39 |
| **Bước 10** | **1, 3, 4, 5, 6, 7, 8, 9, 10, 12**, 39 |

3. Sắp xếp chèn trực tiếp (Insertion Sort)

|  |  |
| --- | --- |
| **Bước** | **Dãy số** |
| **Bước 1** | **8, 39**, 5, 1, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 2** | **5, 8, 39**, 1, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 3** | **1, 5, 8, 39**, 3, 6, 9, 12, 4, 7, 10 |
| **Bước 4** | **1, 3, 5, 8, 39**, 6, 9, 12, 4, 7, 10 |
| **Bước 5** | **1, 3, 5, 6, 8, 39**, 9, 12, 4, 7, 10 |
| **Bước 6** | **1, 3, 5, 6, 8, 9, 39**, 12, 4, 7,10 |
| **Bước 7** | **1, 3, 5, 6, 8, 9, 12, 39**, 4, 7, 10 |
| **Bước 8** | **1, 3, 4, 5, 6, 8, 9, 12, 39**, 7,10 |
| **Bước 9** | **1, 3, 4, 5, 6, 7, 8, 9, 12, 39**, 10 |
| **Bước 10** | 1, 3, 4, 5, 6, 7, 8, 9, 10, 12, 39 |

4. Sắp xếp nổi bọt (Bubble Sort)

4.1. Không cờ

|  |  |
| --- | --- |
| **Lần lặp** | **Trạng thái mảng sau khi duyệt** |
| **Lần 1** | 8, 5, 1, 3, 6, 9, 12, 4, 7, 10, **39** |
| **Lần 2** | 5, 1, 3, 6, 8, 9, 4, 7, 10, **12, 39** |
| **Lần 3** | 1, 3, 5, 6, 8, 4, 7, 9, **10, 12, 39** |
| **Lần 4** | 1, 3, 5, 6, 4, 7, 8, **9, 10, 12, 39** |
| **Lần 5** | 1, 3, 5, 4, 6, 7, **8, 9, 10, 12, 39** |
| **Lần 6** | 1, 3, 4, 5, 6, **7, 8, 9, 10, 12, 39** |
| **Lần 7** | 1, 3, 4, 5, **6, 7, 8, 9, 10, 12, 39** |
| **Lần 8** | 1, 3, 4, **5, 6, 7, 8, 9, 10, 12, 39** |
| **Lần 9** | 1, 3, **4, 5, 6, 7, 8, 9, 10, 12, 39** |
| **Lần 10** | 1, **3, 4, 5, 6, 7, 8, 9, 10, 12, 39** |

4.2. Có cờ

|  |  |
| --- | --- |
| **Lần lặp** | **Trạng thái mảng sau khi duyệt** |
| **Lần 1** | 8, 5, 1, 3, 6, 9, 12, 4, 7, 10, **39** |
| **Lần 2** | 5, 1, 3, 6, 8, 9, 4, 7, 10, **12, 39** |
| **Lần 3** | 1, 3, 5, 6, 8, 4, 7, 9, **10, 12, 39** |
| **Lần 4** | 1, 3, 5, 6, 4, 7, 8, **9, 10, 12, 39** |
| **Lần 5** | 1, 3, 5, 4, 6, 7, **8, 9, 10, 12, 39** |
| **Lần 6** | 1, 3, 4, 5, 6, **7, 8, 9, 10, 12, 39** |
| **Lần 7** | 1, 3, 4, 5, **6, 7, 8, 9, 10, 12, 39** |

# **b) Cài đặt giải thuật**

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

#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]);

}

}

}

}

int main() {

int arr[] = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = sizeof(arr) / sizeof(arr[0]);

interchangeSort(arr, n);

for (int i = 0; i < n; i++) cout << arr[i] << " ";

return 0;

}

2. Sắp xếp chọn trực tiếp (Selection Sort)

#include <iostream>

using namespace std;

void selectionSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

int minIdx = i;

for (int j = i+1; j < n; j++) {

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

swap(arr[i], arr[minIdx]);

}

}

int main() {

int arr[] = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = sizeof(arr) / sizeof(arr[0]);

selectionSort(arr, n);

cout << "Selection Sort: ";

for (int i = 0; i < n; i++) cout << arr[i] << " ";

return 0;

}

3. Sắp xếp chèn trực tiếp (Insertion Sort)

#include <iostream>

using namespace std;

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;

}

}

int main() {

int arr[] = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = sizeof(arr) / sizeof(arr[0]);

insertionSort(arr, n);

cout << "Insertion Sort: ";

for (int i = 0; i < n; i++) cout << arr[i] << " ";

return 0;

}

4. Sắp xếp nổi bọt (Bubble Sort)

4.1. Không cờ

#include <iostream>

using namespace std;

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

swap(arr[j], arr[j+1]);

}

}

}

}

int main() {

int arr[] = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSort(arr, n);

cout << "Bubble Sort không cờ: ";

for (int i = 0; i < n; i++) cout << arr[i] << " ";

return 0;

}

4.2. Có cờ

#include <iostream>

using namespace std;

void bubbleSortFlagged(int arr[], int n) {

for (int i = 0; i < n-1; i++) {

bool swapped = false;

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > arr[j+1]) {

swap(arr[j], arr[j+1]);

swapped = true;

}

}

// Nếu không có phần tử nào được hoán đổi, mảng đã được sắp xếp

if (!swapped) break;

}

}

int main() {

int arr[] = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = sizeof(arr) / sizeof(arr[0]);

bubbleSortFlagged(arr, n);

cout << "Bubble Sort có cờ: ";

for (int i = 0; i < n; i++) cout << arr[i] << " ";

return 0;

}

### **c) Độ phức tạp**

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

**Độ phức tạp:**

* **Thời gian:**
  + Trường hợp tốt nhất: O(n^2)
  + Trường hợp trung bình: O(n^2)
  + Trường hợp xấu nhất: O(n^2)

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

**Độ phức tạp:**

* **Thời gian:**
  + Trường hợp tốt nhất: O(n^2)
  + Trường hợp trung bình: O(n^2)
  + Trường hợp xấu nhất: O(n^2)

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

**Độ phức tạp:**

* **Thời gian:**
  + Trường hợp tốt nhất: O(n)
  + Trường hợp trung bình: O(n^2)
  + Trường hợp xấu nhất: O(n^2)

**4. Sắp xếp nổi bọt (Bubble Sort)**

**Độ phức tạp:**

* **Thời gian:**
  + Trường hợp tốt nhất: O(n)
  + Trường hợp trung bình: O(n^2)
  + Trường hợp xấu nhất: O(n^2)

## **2.2.**

**Cho dãy n số nguyên sau: 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 nhanh (Quick Sort); Sắp xếp trộn trực tiếp (Merge Sort); Sắp xếp cây (Heap Sort).**

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

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

### **a) Mô phỏng các giải thuật**

MÔ PHỎNG QUICK SORT:

bước 1: chọn pivot = 6, chia thành [5, 1, 3, 4] | 6 | [8, 9, 12, 7, 10]

bước 2: sắp xếp phần [5, 1, 3, 4] với pivot = 4, ta được [1, 3] | 4 | [5]

bước 3: sắp xếp phần [8, 9, 12, 7, 10] với pivot = 10, ta được [7, 8, 9] | 10 | [12]

bước 4: kết hợp lại: [1, 3, 4, 5, 7, 8, 9, 10, 12]

MÔ PHỎNG MERGE SORT:

**bước 1: chia nhỏ dãy số. chia dãy thành hai phần bằng nhau đến khi mỗi phần chỉ có một phần tử.**

lần chia thứ nhất: (8, 5, 1, 3, 6) | (9, 12, 4, 7, 10)

lần chia thứ hai: (8, 5) | (1, 3, 6) và (9, 12) | (4, 7, 10)

lần chia thứ ba: (8) | (5) và (1) | (3, 6) và (9) | (12) và (4) | (7, 10)

lần chia thứ tư: (3) | (6) và (7) | (10)

**bước 2: trộn dần các mảng con**

sau khi chia đến mức tối thiểu, ta bắt đầu trộn lại theo đúng thứ tự

lần trộn thứ nhất:

(8) + (5) -> (5, 8)

(1) + (3, 6) -> (1, 3, 6)

(9) + (12) -> (9, 12)

(4) + (7, 10) -> (4, 7, 10)

dãy sau khi trộn: (5, 8) | (1, 3, 6) | (9, 12) | (4, 7, 10)

lần trộn thứ hai:

(5, 8) + (1, 3, 6) -> (1, 3, 5, 6, 8)

(9, 12) + (4, 7, 10) -> (4, 7, 9, 10, 12)

dãy sau khi trộn: (1, 3, 5, 6, 8) | (4, 7, 9, 10, 12)

lần trộn thứ ba:

(1, 3, 5, 6, 8) + (4, 7, 9, 10, 12) -> (1, 3, 4, 5, 6, 7, 8, 9, 10, 12)

MÔ PHỎNG HEAP SORT

**bước 1: xây dựng Max-Heap:**

**A screenshot of a computer

AI-generated content may be incorrect.**

bắt đầu từ các nút cha gần cuối và áp dụng heapify từ dưới lên.

1. nút 4, 7, 10 -> OK
2. nút 3 có con (4, 7) -> chuyển 7 lên
3. nút 5 có con (7, 6) -> chuyển 7 lên
4. nút 1 có con (9, 12) -> chuyển 12 lên
5. nút 8 có con (7, 12) -> chuyển 12 lên

**bước 2: sắp xếp bằng Heap Sort:** thực hiện hoán đổi phần tử đầu (lớn nhất) với phần tử cuối, rồi điều chỉnh Heap lại

**lần 1:**

Hoán đổi 12 với phần tử cuối 6.

Loại bỏ 12 và Heapify lại.

Heap mới sau khi điều chỉnh:

A screenshot of a computer

AI-generated content may be incorrect.

Mảng: [10, 6, 9, 4, 7, 8, 1, 3, 5, 12]

**lần 2:**

Hoán đổi 10 với 5, loại bỏ 10, Heapify lại.

A black background with white text

AI-generated content may be incorrect.

Mảng: [9, 6, 8, 4, 7, 5, 1, 3, 10, 12]

**lần 3:**

Hoán đổi 9 với 3, loại bỏ 9, Heapify lại.

A screenshot of a computer

AI-generated content may be incorrect.

Mảng: [8, 6, 5, 4, 7, 3, 1, 9, 10, 12]

**lần 4:**

Hoán đổi 8 với 1, loại bỏ 8, Heapify lại.

A black background with white numbers

AI-generated content may be incorrect.

Mảng: [7, 6, 5, 4, 1, 3, 8, 9, 10, 12]

**lần 5:**

Hoán đổi 7 với 3, loại bỏ 7, Heapify lại.

A black rectangle with white dots

AI-generated content may be incorrect.

Mảng: [6, 4, 5, 3, 1, 7, 8, 9, 10, 12]

**lần 6:**

Hoán đổi 6 với 1, loại bỏ 6, Heapify lại

A black rectangle with white dots

AI-generated content may be incorrect.

Mảng: [5, 4, 1, 3, 6, 7, 8, 9, 10, 12]

**lần 7:**

Hoán đổi 5 với 3, loại bỏ 5, Heapify lại.

A black background with white text

AI-generated content may be incorrect.

**lần 8:**

Hoán đổi 4 với 1, loại bỏ 4, Heapify lại.

A black rectangle with white dots

AI-generated content may be incorrect.

Mảng: [3, 1, 4, 5, 6, 7, 8, 9, 10, 12]

**lần 9:**

Hoán đổi 3 với 1, loại bỏ 3.

Mảng cuối cùng (đã sắp xếp): 1, 3, 4, 5, 6, 7, 8, 9, 10, 12

### **b) cài đặt các giải thuật**

**giải thuật Quick sort:**

input: mảng chưa được sắp xếp

output: mảng đã được sắp xếp

Quick Sort là thuật toán đệ quy dựa trên nguyên tắc chia để trị (Divide & Conquer):

Chọn pivot (một phần tử làm điểm chia mảng).

Phân hoạch (Partition):

Đưa các phần tử nhỏ hơn pivot về bên trái.

Đưa các phần tử lớn hơn pivot về bên phải.

Đệ quy sắp xếp hai phần còn lại.

-> Điều kiện dừng: Khi mảng con có 0 hoặc 1 phần tử, ta không cần sắp xếp nữa.

#include <iostream>

#include <cstdlib>//thư viện hỗ trợ rand()

#include <ctime> //thư viện hỗ trợ time()

using namespace std;

int Partition (int arr[], int low, int high){

// Chọn pivot ngẫu nhiên

srand(time(0));  // Khởi tạo seed cho số ngẫu nhiên

int randomIndex = low + rand() % (high - low + 1);

// Hoán đổi pivot ngẫu nhiên với phần tử cuối cùng

swap(arr[randomIndex], arr[high]);

int pivot = arr[high];  // Pivot luôn là phần tử cuối

int i = low - 1;        // Con trỏ cho phần tử nhỏ hơn pivot

for (int j = low; j < high; j++) {

    if (arr[j] < pivot) { // Nếu phần tử nhỏ hơn pivot

        i++;

        swap(arr[i], arr[j]); // Đưa về bên trái

    }

}

swap(arr[i + 1], arr[high]); // Đưa pivot về đúng vị trí

return i + 1;  // Trả về vị trí mới của pivot

}

void QuickSort(int arr[], int low, int high) {

    if (low < high) {

        int pivotIndex = Partition(arr, low, high);

        QuickSort(arr, low, pivotIndex - 1); // Sắp xếp bên trái pivot

        QuickSort(arr, pivotIndex + 1, high); // Sắp xếp bên phải pivot

    }

}

void PrintArray(int arr[], int n) {

    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    cout << endl;

}

int main() {

    int arr[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Mảng ban đầu: ";

    PrintArray(arr, n);

    QuickSort(arr, 0, n - 1);

    cout << "Mảng sau khi sắp xếp: ";

    PrintArray(arr, n);

    return 0;

}

**giải thuật Merge sort:**

Merge Sort là một thuật toán chia để trị (Divide & Conquer):

1. Chia (Divide): Chia mảng thành hai nửa bằng nhau cho đến khi mỗi nửa chỉ còn một phần tử.
2. Trộn (Merge): Ghép hai nửa đã sắp xếp thành một dãy có thứ tự.
3. Gọi đệ quy: Thực hiện lại bước trên cho từng phần mảng.

Điều kiện dừng: Khi mảng con chỉ còn 1 phần tử (low >= high), ta không cần sắp xếp nữa.

Chương trình gồm 3 phần chính:

Hàm Merge(): Hợp nhất hai mảng con đã sắp xếp.

Hàm MergeSort(): Chia nhỏ mảng và gọi Merge().

Hàm main(): Kiểm tra thuật toán.

#include <iostream>

using namespace std;

// Hàm trộn hai mảng con đã sắp xếp

void Merge(int arr[], int left, int mid, int right) {

    int n1 = mid - left + 1; // Số phần tử của mảng trái

    int n2 = right - mid;    // Số phần tử của mảng phải

    // Tạo hai mảng tạm thời

    int leftArr[n1], rightArr[n2];

    // Sao chép dữ liệu vào hai mảng tạm

    for (int i = 0; i < n1; i++)

        leftArr[i] = arr[left + i];

    for (int j = 0; j < n2; j++)

        rightArr[j] = arr[mid + 1 + j];

    // Gộp hai mảng đã sắp xếp thành một mảng

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (leftArr[i] <= rightArr[j]) {

            arr[k] = leftArr[i];

            i++;

        } else {

            arr[k] = rightArr[j];

            j++;

        }

        k++;

    }

    // Sao chép các phần tử còn lại của leftArr (nếu có)

    while (i < n1) {

        arr[k] = leftArr[i];

        i++;

        k++;

    }

    // Sao chép các phần tử còn lại của rightArr (nếu có)

    while (j < n2) {

        arr[k] = rightArr[j];

        j++;

        k++;

    }

}

// Hàm Merge Sort

void MergeSort(int arr[], int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2; // Chia đôi mảng

        // Đệ quy sắp xếp hai nửa

        MergeSort(arr, left, mid);

        MergeSort(arr, mid + 1, right);

        // Trộn hai nửa lại

        Merge(arr, left, mid, right);

    }

}

// Hàm in mảng

void PrintArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// Hàm main để chạy chương trình

int main() {

    int arr[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Mảng ban đầu: ";

    PrintArray(arr, n);

    MergeSort(arr, 0, n - 1);

    cout << "Mảng sau khi sắp xếp: ";

    PrintArray(arr, n);

    return 0;

}

**giải thuật Merge sort:**

Heap Sort gồm 2 giai đoạn chính:

1. Xây dựng Max-Heap: Biến mảng thành một Max-Heap (nút cha luôn lớn hơn hai nút con).
2. Sắp xếp bằng cách trích xuất phần tử lớn nhất:

* Hoán đổi phần tử đầu (lớn nhất) với phần tử cuối.
* Giảm kích thước Heap.
* Thực hiện Heapify để duy trì Max-Heap.
* Lặp lại cho đến khi Heap còn 1 phần tử.

Điều kiện dừng: Khi Heap chỉ còn 1 phần tử, ta không cần tiếp tục sắp xếp nữa.

Chương trình gồm 3 phần chính:

* Hàm Heapify(): Điều chỉnh lại cây để giữ tính chất Max-Heap.
* Hàm HeapSort(): Xây dựng Max-Heap và trích xuất phần tử lớn nhất dần dần.
* Hàm main(): Kiểm tra thuật toán.

#include <iostream>

using namespace std;

// Hàm Heapify để duy trì tính chất Max-Heap

void Heapify(int arr[], int n, int i) {

    int largest = i;      // Giả sử nút cha là lớn nhất

    int left = 2 \* i + 1; // Chỉ số con trái

    int right = 2 \* i + 2; // Chỉ số con phải

    // Nếu con trái lớn hơn nút cha

    if (left < n && arr[left] > arr[largest])

        largest = left;

    // Nếu con phải lớn hơn nút cha hoặc con trái

    if (right < n && arr[right] > arr[largest])

        largest = right;

    // Nếu largest thay đổi, hoán đổi và Heapify tiếp

    if (largest != i) {

        swap(arr[i], arr[largest]);

        Heapify(arr, n, largest);

    }

}

// Hàm Heap Sort

void HeapSort(int arr[], int n) {

    // Bước 1: Xây dựng Max-Heap từ mảng ban đầu

    for (int i = n / 2 - 1; i >= 0; i--)

        Heapify(arr, n, i);

    // Bước 2: Trích xuất từng phần tử lớn nhất và Heapify lại

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]); // Đưa phần tử lớn nhất về cuối mảng

        Heapify(arr, i, 0);   // Điều chỉnh Heap với phần còn lại

    }

}

// Hàm in mảng

void PrintArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// Hàm main để kiểm tra chương trình

int main() {

    int arr[] = {8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Mảng ban đầu: ";

    PrintArray(arr, n);

    HeapSort(arr, n);

    cout << "Mảng sau khi sắp xếp: ";

    PrintArray(arr, n);

    return 0;

}

### **c) Độ phức tạp của các giải thuật đã triển khai**

**QUICK SORT:**

Thuật toán **Quick Sort** hoạt động theo nguyên tắc **chia để trị** (Divide and Conquer), gồm ba bước chính:

1. **Chia (Partitioning)**: Chọn một phần tử **pivot** và phân chia mảng thành hai phần: một phần chứa các phần tử nhỏ hơn pivot và một phần chứa các phần tử lớn hơn pivot.
2. **Trị (Recursion)**: Gọi đệ quy sắp xếp hai phần đã chia.
3. **Kết hợp**: Do quá trình chia đã sắp xếp các phần tử xung quanh pivot, ta không cần bước hợp nhất như trong Merge Sort.

**Độ phức tạp của Quick Sort trong trường hợp trung bình và tốt nhất**

Trong trường hợp trung bình và tốt nhất, pivot được chọn một cách tương đối cân bằng, tức là mỗi lần chia, mảng được chia thành hai phần có kích thước gần bằng nhau.

**Giả sử mảng có n phần tử**, nếu mỗi lần chọn pivot đều chia mảng thành hai phần gần bằng nhau, ta có công thức truy hồi:

A math symbols with numbers and symbols

AI-generated content may be incorrect.

Trong đó:

* T(n)T(n) là thời gian chạy của thuật toán Quick Sort trên mảng có nn phần tử.
* Hai lời gọi đệ quy xử lý hai nửa mảng.
* O(n)O(n) là thời gian thực hiện bước phân hoạch (partition).

Sử dụng **định lý T** (Master Theorem) cho công thức truy hồi:

A number and a square

AI-generated content may be incorrect.

So sánh với dạng tổng quát của định lý T:

A number of mathematical equations

AI-generated content may be incorrect.

Ở đây:

a = 2,

b = 2,

d = 1.

Vì d = logba = log22 = 1, theo định lý, độ phức tạp là: O (n log n)

➡ **Trong trường hợp trung bình và tốt nhất, Quick Sort có độ phức tạp là O(n log n)**

**Độ phức tạp của Quick Sort trong trường hợp xấu nhất**

Trường hợp xấu nhất xảy ra khi pivot được chọn không cân bằng, chẳng hạn như pivot luôn là phần tử nhỏ nhất hoặc lớn nhất của mảng. Khi đó, mỗi lần phân hoạch chỉ tách được một phần tử ra khỏi mảng, phần còn lại vẫn chứa (n−1)(n-1) phần tử.

* Khi đó, công thức truy hồi trở thành:

T(n)=T(n−1)+O(n)T(n) = T(n - 1) + O(n)

Triển khai đệ quy:

A math equations on a white background

AI-generated content may be incorrect.Tổng của dãy số:



➡ **Trong trường hợp xấu nhất, Quick Sort có độ phức tạp là O(n2)**

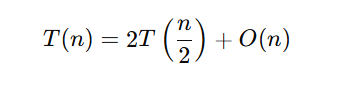
**MERGE SORT**

Thuật toán Merge Sort hoạt động theo nguyên tắc chia để trị (Divide and Conquer) với ba bước chính:

1. Chia (Divide): Chia mảng thành hai nửa bằng nhau.
2. Trị (Conquer): Gọi đệ quy để sắp xếp từng nửa của mảng.
3. Kết hợp (Merge): Gộp hai mảng con đã sắp xếp thành một mảng đã sắp xếp.

**Công thức truy hồi của Merge Sort**

Giả sử chúng ta cần sắp xếp một mảng có nn phần tử. Quá trình thực hiện có thể được mô tả bằng công thức truy hồi:



Giải thích:

* 2T(n/2) Hai lời gọi đệ quy xử lý hai nửa của mảng.
* O(n) Thời gian để gộp hai mảng con lại thành một mảng đã sắp xếp.

**Giải công thức truy hồi bằng phương pháp cây đệ quy**

Ta có công thức:

A black text on a white background

AI-generated content may be incorrect.

**Xây dựng cây đệ quy**

Mỗi lần chia, số phần tử trong mỗi nhánh giảm một nửa, và số tầng của cây sẽ là log2 n (vì n giảm dần đến 1).

A screenshot of a computer

AI-generated content may be incorrect.

Mỗi tầng của cây đệ quy có tổng chi phí là O(n), và số tầng của cây là log2 n. Do đó, tổng độ phức tạp là: O (n log n)

1. **3. Trường hợp xấu nhất, trung bình và tốt nhất**

Thuật toán Merge Sort luôn thực hiện theo cùng một mô hình chia và gộp, bất kể dữ liệu đầu vào:

* Trường hợp tốt nhất: Khi mảng đã được sắp xếp trước.
* Trường hợp trung bình: Dữ liệu ngẫu nhiên.
* Trường hợp xấu nhất: Mảng bị sắp xếp theo thứ tự ngược lại.

Trong mọi trường hợp, công thức truy hồi vẫn giữ nguyên, dẫn đến độ phức tạp:

O(n log n)

**HEAP SORT**

Thuật toán Heap Sort hoạt động dựa trên cấu trúc dữ liệu Heap (đống), với hai bước chính:

1. Xây dựng Heap: Biến đổi mảng thành một Max-Heap (cây nhị phân có tính chất heap, trong đó mỗi node cha có giá trị lớn hơn hoặc bằng các node con).
2. Sắp xếp: Liên tục hoán đổi phần tử lớn nhất (gốc của Heap) với phần tử cuối cùng trong mảng, sau đó giảm kích thước của Heap và khôi phục tính chất Max-Heap.

**Độ phức tạp của Heap sort:**

**Bước 1: Xây dựng Heap**

Mảng ban đầu được biến đổi thành một Max-Heap bằng cách gọi heapify từ các node không phải lá trở lên.

* Số node trong cây nhị phân: nnn.
* Số node lá: n/2 (vì cây nhị phân đầy có nửa số node ở lá).
* Số node cha (cần thực hiện heapify): n/2.

Mỗi node cha ở tầng **h** của cây có thể phải đẩy giá trị xuống tối đa **h** tầng để duy trì tính chất Heap. Tổng số thao tác của quá trình heapify là:

A mathematical equation with numbers and symbols

AI-generated content may be incorrect.

Tổng này có giá trị xấp xỉ O (n), do đó bước xây dựng Heap có độ phức tạp là O(n).

**Bước 2: Sắp xếp (Heap Sort)**

Sau khi xây dựng Heap, ta thực hiện n - 1 lần loại bỏ phần tử lớn nhất (gốc của Heap), mỗi lần phải gọi heapify để khôi phục tính chất Max-Heap.

* Mỗi lần trích xuất phần tử lớn nhất (swap gốc với phần tử cuối) mất O(1)
* Mỗi lần gọi heapify để điều chỉnh lại Heap có độ phức tạp O(log n)

Vì có n−1 lần trích xuất và gọi heapify, tổng thời gian cần thiết là: O(n log n)

## **2.3.**

**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.**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Hàm hoán đổi

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Sắp xếp đổi chỗ trực tiếp

void interchangeSort(int arr[], int n, int \*compareCount, int \*swapCount) {

\*compareCount = 0;

\*swapCount = 0;

for (int i = 0; i < n-1; i++) {

int minIdx = i; // Tìm phần tử nhỏ nhất

for (int j = i+1; j < n; j++) {

(\*compareCount)++;

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

if (minIdx != i) { // Chỉ hoán đổi khi cần

swap(&arr[i], &arr[minIdx]);

(\*swapCount)++;

}

}

}

// Sắp xếp chọn trực tiếp

void selectionSort(int arr[], int n, int \*compareCount, int \*swapCount) {

\*compareCount = 0;

\*swapCount = 0;

for (int i = 0; i < n-1; i++) {

int minIdx = i;

for (int j = i+1; j < n; j++) {

(\*compareCount)++;

if (arr[j] < arr[minIdx]) {

minIdx = j;

}

}

if (minIdx != i) {

swap(&arr[i], &arr[minIdx]);

(\*swapCount)++;

}

}

}

// Sắp xếp chèn trực tiếp

void insertionSort(int arr[], int n, int \*compareCount, int \*swapCount) {

\*compareCount = 0;

\*swapCount = 0;

for (int i = 1; i < n; i++) {

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j]> key) {

(\*compareCount)++;

if (arr[j] > key) {

arr[j + 1] = arr[j];

(\*swapCount)++;

} else {

break;

}

j--;

}

arr[j + 1] = key;

}

}

// Bubble Sort có cờ

void bubbleSortFlagged(int arr[], int n, int \*compareCount, int\*swapCount) {

\*compareCount = 0;

\*swapCount = 0;

for (int i = 0; i < n-1; i++) {

int swapped = 0;

for (int j = 0; j < n-i-1; j++) {

(\*compareCount)++;

if (arr[j] > arr[j+1]) {

swap(&arr[j], &arr[j+1]);

(\*swapCount)++;

swapped = 1;

}

}

if (!swapped) break;

}

}

// Bubble Sort không cờ

void bubbleSort(int arr[], int n, int \*compareCount, int \*swapCount) {

\*compareCount = 0;

\*swapCount = 0;

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

(\*compareCount)++;

if (arr[j] > arr[j+1]) {

swap(&arr[j], &arr[j+1]);

(\*swapCount)++;

}

}

}

}

// Phát sinh mảng ngẫu nhiên

void generateRandomArray(int arr[], int n, int max\_value) {

for (int i = 0; i < n; i++) {

arr[i] = rand() % (max\_value + 1);

}

}

// Thực nghiệm thuật toán

void experimentSortingAlgorithm(void (\*sortFunc)(int[], int, int\*, int\*), int max\_value, const char \*sortName) {

int sizes[] = {10, 100, 200, 500, 1000, 5000, 10000};

int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

int t = 10; // Số lần thực nghiệm mỗi trường hợp

for (int i = 0; i < num\_sizes; i++) {

int n = sizes[i];

double total\_time = 0.0;

int total\_compare = 0;

int total\_swap = 0;

for (int j = 0; j < t; j++) {

int arr[n];

generateRandomArray(arr, n, max\_value);

int compareCount = 0, swapCount = 0;

clock\_t start\_time = clock();

sortFunc(arr, n, &compareCount, &swapCount);

clock\_t end\_time = clock();

double time\_taken = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC;

total\_time += time\_taken;

total\_compare += compareCount;

total\_swap += swapCount;

}

double average\_time = total\_time / t;

double average\_compare = (double)total\_compare / t;

double average\_swap = (double)total\_swap / t;

printf("Thuật toán: %s, Kích thước mảng: %d, Thời gian trung bình: %f giây, Số lần so sánh trung bình: %f, Số lần đổi chỗ trung bình: %f\n",

sortName, n, average\_time, average\_compare, average\_swap);

}

}

int main() {

srand(time(0)); // Khởi tạo seed cho hàm rand()

printf("Thực nghiệm interchangeSort:\n");

experimentSortingAlgorithm(interchangeSort, 10000, "Interchange Sort");

printf("Thực nghiệm selectionSort:\n");

experimentSortingAlgorithm(selectionSort, 10000, "Selection Sort");

printf("Thực nghiệm insertionSort:\n");

experimentSortingAlgorithm(insertionSort, 10000, "Insertion Sort");

printf("Thực nghiệm bubbleSortFlagged:\n");

experimentSortingAlgorithm(bubbleSortFlagged, 10000, "Bubble Sort (có cờ)");

printf("Thực nghiệm bubbleSort:\n");

experimentSortingAlgorithm(bubbleSort, 10000, "Bubble Sort (không cờ)");

return 0;

}

## **2.4.**

**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 nhanh (Quick Sort); Sắp xếp trộn (Merge Sort); Sắp xếp cây (Heap Sort)**

**QUICK SORT:**

Input: 30,000 số nguyên ngẫu nhiên

Output: Thời gian thực thi 2416 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm phân hoạch cho Quick Sort

int Partition(int arr[], int low, int high) {

    int pivot = arr[high];

    int i = low - 1;

    for (int j = low; j < high; j++) {

        if (arr[j] < pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    swap(arr[i + 1], arr[high]);

    return i + 1;

}

// Hàm Quick Sort

void QuickSort(int arr[], int low, int high) {

    if (low < high) {

        int pivotIndex = Partition(arr, low, high);

        QuickSort(arr, low, pivotIndex - 1);

        QuickSort(arr, pivotIndex + 1, high);

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file input.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Quick Sort

    QuickSort(arr, 0, n - 1);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

**SELECTION SORT:**

Thời gian thực thi: 340807 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm Selection Sort

void SelectionSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        int minIndex = i;

        for (int j = i + 1; j < n; j++) {

            if (arr[j] < arr[minIndex]) {

                minIndex = j;

            }

        }

        swap(arr[i], arr[minIndex]);

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file random\_numbers.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Selection Sort

    SelectionSort(arr, n);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file ran\_num\_output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

**INSERTION SORT:**

Thời gian thực thi: 233179 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm Insertion Sort

void InsertionSort(int arr[], int n) {

    for (int i = 1; i < n; i++) {

        int key = arr[i];

        int j = i - 1;

        // Di chuyển các phần tử lớn hơn key lên một vị trí

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j--;

        }

        arr[j + 1] = key;

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file random\_numbers.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Insertion Sort

    InsertionSort(arr, n);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file ran\_num\_output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

**BUBLLE SORT:**

Thời gian thực thi: 1980626 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm Bubble Sort

void BubbleSort(int arr[], int n) {

    for (int i = 0; i < n - 1; i++) {

        bool swapped = false; // Kiểm tra xem có swap không

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

                swapped = true;

            }

        }

        // Nếu không có swap nào, mảng đã sắp xếp xong

        if (!swapped) break;

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file random\_numbers.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Bubble Sort

    BubbleSort(arr, n);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file ran\_num\_output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

**MERGE SORT:**

Thời gian thực thi: 3098 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm trộn hai mảng con đã sắp xếp

void Merge(int arr[], int left, int mid, int right) {

    int n1 = mid - left + 1; // Số phần tử của mảng trái

    int n2 = right - mid;    // Số phần tử của mảng phải

    // Tạo hai mảng tạm thời

    int leftArr[n1], rightArr[n2];

    // Sao chép dữ liệu vào hai mảng tạm

    for (int i = 0; i < n1; i++)

        leftArr[i] = arr[left + i];

    for (int j = 0; j < n2; j++)

        rightArr[j] = arr[mid + 1 + j];

    // Gộp hai mảng đã sắp xếp thành một mảng

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (leftArr[i] <= rightArr[j]) {

            arr[k] = leftArr[i];

            i++;

        } else {

            arr[k] = rightArr[j];

            j++;

        }

        k++;

    }

    // Sao chép các phần tử còn lại của leftArr (nếu có)

    while (i < n1) {

        arr[k] = leftArr[i];

        i++;

        k++;

    }

    // Sao chép các phần tử còn lại của rightArr (nếu có)

    while (j < n2) {

        arr[k] = rightArr[j];

        j++;

        k++;

    }

}

// Hàm Merge Sort

void MergeSort(int arr[], int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2; // Chia đôi mảng

        // Đệ quy sắp xếp hai nửa

        MergeSort(arr, left, mid);

        MergeSort(arr, mid + 1, right);

        // Trộn hai nửa lại

        Merge(arr, left, mid, right);

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file random\_numbers.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Merge Sort

    MergeSort(arr, 0, n - 1);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file ran\_num\_output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

**HEAP SORT:**

Thời gian thực thi: 5438 ms

#include <iostream>

#include <fstream>

#include <chrono>

using namespace std;

using namespace std::chrono;

// Hàm Heapify để duy trì tính chất Max-Heap

void Heapify(int arr[], int n, int i) {

    int largest = i;      // Giả sử nút cha là lớn nhất

    int left = 2 \* i + 1; // Chỉ số con trái

    int right = 2 \* i + 2; // Chỉ số con phải

    // Nếu con trái lớn hơn nút cha

    if (left < n && arr[left] > arr[largest])

        largest = left;

    // Nếu con phải lớn hơn nút cha hoặc con trái

    if (right < n && arr[right] > arr[largest])

        largest = right;

    // Nếu largest thay đổi, hoán đổi và Heapify tiếp

    if (largest != i) {

        swap(arr[i], arr[largest]);

        Heapify(arr, n, largest);

    }

}

// Hàm Heap Sort

void HeapSort(int arr[], int n) {

    // Bước 1: Xây dựng Max-Heap từ mảng ban đầu

    for (int i = n / 2 - 1; i >= 0; i--)

        Heapify(arr, n, i);

    // Bước 2: Trích xuất từng phần tử lớn nhất và Heapify lại

    for (int i = n - 1; i > 0; i--) {

        swap(arr[0], arr[i]); // Đưa phần tử lớn nhất về cuối mảng

        Heapify(arr, i, 0);   // Điều chỉnh Heap với phần còn lại

    }

}

// Hàm đếm số phần tử trong file

int CountElements(const string& filename) {

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return -1;

    }

    int count = 0, num;

    while (inputFile >> num) { // Đọc từng số và đếm

        count++;

    }

    inputFile.close();

    return count;

}

// Hàm đọc dữ liệu từ file vào mảng động

bool ReadFromFile(const string& filename, int\*& arr, int& n) {

    n = CountElements(filename);

    if (n <= 0) return false; // Trả về false nếu file rỗng

    arr = new int[n]; // Cấp phát động mảng

    ifstream inputFile(filename);

    if (!inputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return false;

    }

    for (int i = 0; i < n; i++) {

        inputFile >> arr[i]; // Đọc từng số vào mảng

    }

    inputFile.close();

    return true;

}

// Hàm ghi kết quả vào file

void WriteToFile(const string& filename, int arr[], int n, long long timeElapsed) {

    ofstream outputFile(filename);

    if (!outputFile) {

        cerr << "Không thể mở file " << filename << endl;

        return;

    }

    outputFile << "Thời gian thực thi: " << timeElapsed << " microseconds\n";

    outputFile << "Mảng sau khi sắp xếp:\n";

    for (int i = 0; i < n; i++) {

        outputFile << arr[i] << " ";

    }

    outputFile.close();

}

int main() {

    int\* arr;

    int n;

    // Đọc dữ liệu từ file random\_numbers.txt

    if (!ReadFromFile("random\_numbers.txt", arr, n)) {

        return 1; // Thoát nếu không thể đọc file

    }

    // Bắt đầu đo thời gian

    auto start = high\_resolution\_clock::now();

    // Gọi Heap Sort

    HeapSort(arr, n);

    // Kết thúc đo thời gian

    auto stop = high\_resolution\_clock::now();

    auto duration = duration\_cast<microseconds>(stop - start);

    cout << "Thời gian thực thi: " << duration.count() << " microseconds" << endl;

    // Ghi kết quả vào file ran\_num\_output.txt

    WriteToFile("ran\_num\_output.txt", arr, n, duration.count());

    delete[] arr; // Giải phóng bộ nhớ động

    return 0;

}

# **Chương 3: Bài tập ứng dụng**

## **3.1.**

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

**a. Hãy cho biết vị trí của k phần tử có giá trị lớn nhất trong dãy.**

**b. Sắp xếp các phần tử tăng dần theo tổng các chữ số của từng phần tử.**

**c. Hãy xóa tất cả các số nguyên tố có trong dãy.**

**Bài giải**

**1. Phân tích bài toán**

Cho dãy số nguyên gồm n phần tử:  
 **-** Xác định vị trí của k phần tử lớn nhất.  
 **-** Sắp xếp dãy theo tổng các chữ số của từng phần tử.  
 **-** Xóa tất cả số nguyên tố khỏi dãy.

**2. Thiết kế giải thuật**

a) Xác định vị trí của k phần tử có giá trị lớn nhất trong dãy

- Khởi tạo dãy số và tạo một mảng chỉ số.

- Thực hiện Quick Sort để sắp xếp dãy số theo thứ tự giảm dần.

- Lấy vị trí của k phần tử có giá trị lớn nhất.

b) Sắp xếp dãy theo tổng các chữ số của từng phần tử

- Tính tổng chữ số của từng phần tử trong mảng.

- Dùng Radix Sort, nhưng thay vì sắp xếp theo từng chữ số thì ta sắp xếp theo tổng chữ số.

- Sắp xếp tăng dần theo tổng chữ số.

c) Xoá tất cả số nguyên tố ra khỏi dãy

- Viết hàm kiểm tra số nguyên tố.

- Lọc ra danh sách mới không chứa số nguyên tố.

**3. Cài đặt**

**a**) Xác định vị trí của k phần tử có giá trị lớn nhất trong dãy

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Hàm hoán đổi

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Hàm phân chia cho Quick Sort

int partition(vector<int> &arr, vector<int> &indices, int low, int high) {

int pivot = arr[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (arr[j] > pivot) { // Sắp xếp giảm dần

i++;

swap(&arr[i], &arr[j]);

swap(&indices[i], &indices[j]);

}

}

swap(&arr[i + 1], &arr[high]);

swap(&indices[i + 1], &indices[high]);

return i + 1;

}

// Hàm Quick Sort

void quickSort(vector<int> &arr, vector<int> &indices, int low, int high) {

if (low < high) {

int pi = partition(arr, indices, low, high);

quickSort(arr, indices, low, pi - 1);

quickSort(arr, indices, pi + 1, high);

}

}

int main() {

vector<int> arr = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

int n = arr.size();

int k = 3; // Số phần tử lớn nhất cần lấy

// Tạo mảng chỉ số ban đầu

vector<int> indices(n);

for (int i = 0; i < n; i++) {

indices[i] = i;

}

// Sắp xếp mảng và chỉ số theo thứ tự giảm dần

quickSort(arr, indices, 0, n - 1);

// Lấy vị trí của k phần tử lớn nhất

cout << "Vị trí của " << k << " phần tử lớn nhất: ";

for (int i = 0; i < k; i++) {

cout << indices[i] << " ";

}

cout << endl;

return 0;

}

b) Sắp xếp dãy theo tổng các chữ số của từng phần tử

#include <iostream>

#include <vector>

#include <cmath>

#include <algorithm>

using namespace std;

// Hàm tính tổng các chữ số của một số

int sumOfDigits(int num) {

int sum = 0;

while (num != 0) {

sum += num % 10;

num /= 10;

}

return sum;

}

// Hàm tìm giá trị lớn nhất trong mảng

int getMaxSum(const vector<int>& digitSums) {

int maxSum = 0;

for (int sum : digitSums) {

maxSum = max(maxSum, sum);

}

return maxSum;

}

// Hàm sắp xếp Counting Sort

void countingSort(vector<int>& arr, const vector<int>& digitSums, int exp) {

int n = arr.size();

vector<int> output(n);

vector<int> count(10, 0);

for (int i = 0; i < n; i++) {

int index = (digitSums[i] / exp) % 10;

count[index]++;

}

for (int i = 1; i < 10; i++) {

count[i] += count[i - 1];

}

for (int i = n - 1; i >= 0; i--) {

int index = (digitSums[i] / exp) % 10;

output[count[index] - 1] = arr[i];

count[index]--;

}

for (int i = 0; i < n; i++) {

arr[i] = output[i];

}

}

// Hàm Radix Sort tùy chỉnh để sắp xếp theo tổng các chữ số

void radixSort(vector<int>& arr) {

int n = arr.size();

vector<int> digitSums(n);

for (int i = 0; i < n; i++) {

digitSums[i] = sumOfDigits(arr[i]);

}

int maxSum = getMaxSum(digitSums);

for (int exp = 1; maxSum / exp > 0; exp \*= 10) {

countingSort(arr, digitSums, exp);

}

}

int main() {

vector<int> arr = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

cout << "Mảng ban đầu: ";

for (int num : arr) {

cout << num << " ";

}

cout << endl;

radixSort(arr);

cout << "Mảng sau khi sắp xếp theo tổng các chữ số: ";

for (int num : arr) {

cout << num << " ";

}

cout << endl;

return 0;

}

c) Xoá tất cả số nguyên tố ra khỏi dãy

#include <vector>

#include <iostream>

using namespace std;

// Hàm kiểm tra số nguyên tố

bool isPrime(int n) {

if (n <= 1) return false;

if (n == 2 || n == 3) return true;

if (n % 2 == 0 || n % 3 == 0) return false;

for (int i = 5; i <= sqrt(n); i += 6) {

if (n % i == 0 || n % (i + 2) == 0) return false;

}

return true;

}

// Hàm để xóa các số nguyên tố khỏi dãy số

void removePrimes(vector<int>& arr) {

auto it = arr.begin();

while (it != arr.end()) {

if (isPrime(\*it)) {

it = arr.erase(it);

} else {

++it;

}

}

}

int main() {

vector<int> arr = {39, 8, 5, 1, 3, 6, 9, 12, 4, 7, 10};

cout << "Dãy số trước khi xóa số nguyên tố: ";

for (int num : arr) {

cout << num << " ";

}

cout << endl;

removePrimes(arr);

cout << "Dãy số sau khi xóa số nguyên tố: ";

for (int num : arr) {

cout << num << " ";

}

cout << endl; return 0;

}

return 0;

}

## **3.2.**

**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.**

**a. Hãy định nghĩa cấu trúc dữ liệu để lưu trữ các dữ liệu trong bộ nhớ trong của máy tính.**

**b. 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 giải**

a) Định nghĩa cấu trúc dữ liệu

* Hệ số : Số thực (float hoặc double)
* Bậc: số nguyên có giá trị từ 0 đến 100
* Khai báo:

struct SoHang {

double heSo; // Hệ số của số hạng

int bac; // Bậc của số hạng (0 → 100)

};

b) Cài đặt thuật toán

#include <stdio.h>

struct SoHang {

double heSo; // Hệ số của số hạng

int bac; // Bậc của số hạng (0 → 100)

};

void countingSort(SoHang\* danhSach, int n, int exp) {

SoHang output[n];

int count[10] = {0};

for (int i = 0; i < n; i++) {

count[(danhSach[i].bac / exp) % 10]++;

}

for (int i = 1; i < 10; i++) {

count[i] += count[i - 1];

}

for (int i = n - 1; i >= 0; i--) {

output[count[(danhSach[i].bac / exp) % 10] - 1] = danhSach[i];

count[(danhSach[i].bac / exp) % 10]--;

}

for (int i = 0; i < n; i++) {

danhSach[i] = output[i];

}

}

void radixSort(SoHang\* danhSach, int n) {

int maxBac = danhSach[0].bac;

for (int i = 1; i < n; i++) {

if (danhSach[i].bac > maxBac) {

maxBac = danhSach[i].bac;

}

}

for (int exp = 1; maxBac / exp > 0; exp \*= 10) {

countingSort(danhSach, n, exp);

}

}

void inDanhSachSoHang(SoHang\* danhSach, int soLuong) {

for (int i = 0; i < soLuong; i++) {

printf("He so: %lf, Bac: %d\n", danhSach[i].heSo, danhSach[i].bac);

}

}

int main() {

SoHang danhSach[] = {

{3.0, 4},

{1.0, 5},

{4.8, 3},

{5.6, 1},

{1.2, 0}

};

int soLuong = 5;

printf("Danh sach sap xep theo bac tang dan:\n");

radixSort(danhSach, soLuong);

inDanhSachSoHang(danhSach, soLuong);

return 0;

}

## **3.3.**

**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.**

**a. Hãy định nghĩa cấu trúc dữ liệu để lưu trữ các phòng thi này trong bộ nhớ trong của máy tính.**

**b. 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 giải**

a) Định nghĩa cấu trúc dữ liệu

Số phòng: Số nguyên từ 1 đến 200

Nhà: Kí tự in hoa từ A đến z

Khả năng chứa: Một số nguyên có giá trị từ 10 đến 250

Khai báo:

struct PhongThi {

int soPhong; // Số phòng (1 → 200)

char nha; // Nhà (A → Z)

int khaNangChua; // Khả năng chứa (10 → 250)

};

b) Cài đặt thuật toán

#include <stdio.h>

struct PhongThi {

int soPhong; // Số phòng (1 → 200)

char nha; // Nhà (A → Z)

int khaNangChua; // Khả năng chứa (10 → 250)

};

// Giảm dần về Khả năng chứa

void sapXepTheoKhaNangChuaGiamDan(PhongThi\* danhSach, int soLuongPhong) {

for (int i = 0; i < soLuongPhong - 1; i++) {

for (int j = i + 1; j < soLuongPhong; j++) {

if (danhSach[i].khaNangChua < danhSach[j].khaNangChua) {

PhongThi temp = danhSach[i];

danhSach[i] = danhSach[j];

danhSach[j] = temp;

}

}

}

}

// Tăng dần theo Nhà, các phong cùng Nhà tăng dần theo số Phòng

void sapXepTheoNhaVaSoPhongTangDan(PhongThi\* danhSach, int soLuongPhong) {

for (int i = 0; i < soLuongPhong - 1; i++) {

for (int j = i + 1; j < soLuongPhong; j++) {

if (danhSach[i].nha > danhSach[j].nha ||

(danhSach[i].nha == danhSach[j].nha && danhSach[i].soPhong > danhSach[j].soPhong)) {

PhongThi temp = danhSach[i];

danhSach[i] = danhSach[j];

danhSach[j] = temp;

}

}

}

}

//Tăng dần theo Nhà, các phòng cùng Nhà giảm dần theo Khả năng chứa

void sapXepTheoNhaVaKhaNangChuaGiamDan(PhongThi\* danhSach, int soLuongPhong) {

for (int i = 0; i < soLuongPhong - 1; i++) {

for (int j = i + 1; j < soLuongPhong; j++) {

if (danhSach[i].nha > danhSach[j].nha ||

(danhSach[i].nha == danhSach[j].nha && danhSach[i].khaNangChua < danhSach[j].khaNangChua)) {

PhongThi temp = danhSach[i];

danhSach[i] = danhSach[j];

}

}

}

}

void inDanhSachPhongThi(PhongThi\* danhSach, int soLuongPhong) {

for (int i = 0; i < soLuongPhong; i++) {

printf("Phong %d - Nha %c - Kha nang chua %d\n", danhSach[i].soPhong, danhSach[i].nha, danhSach[i].khaNangChua);

}

}

int main() {

PhongThi danhSach[] = {

{1, 'A', 100},

{2, 'B', 150},

{3, 'A', 80},

{4, 'C', 200},

{5, 'B', 50},

{6, 'D', 30},

{7, 'A', 70},

{8, 'D', 120},

{9, 'C', 40},

{10, 'B', 15}

};

int soLuongPhong = 10;

printf("Danh sach sap xep theo kha nang chua giam dan:\n");

sapXepTheoKhaNangChuaGiamDan(danhSach, soLuongPhong);

inDanhSachPhongThi(danhSach, soLuongPhong);

printf("\nDanh sach sap xep theo nha va so phong tang dan:\n");

sapXepTheoNhaVaSoPhongTangDan(danhSach, soLuongPhong);

inDanhSachPhongThi(danhSach, soLuongPhong);

printf("\nDanh sach sap xep theo nha va kha nang chua giam dan:\n");

sapXepTheoNhaVaKhaNangChuaGiamDan(danhSach, soLuongPhong);

inDanhSachPhongThi(danhSach, soLuongPhong);

return 0;

}

## **3.4.**

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

**a. Tìm số nguyên tố lớn nhất trong ma trận.**

**b. Tìm những dòng của ma trận có chứa giá trị nguyên tố.**

**c. Tìm những dòng của ma trận chỉ chứa các số nguyên tố.**

**Bài giải**

Phân tích bài toán:

a) Tìm số nguyên tố lớn nhất trong ma trận.

Input: 4 6 8 9

5 7 11 13

15 16 17 19

2 3 5 7

Output: 19

b) Tìm những dòng của ma trận có chứa giá trị nguyên tố.

Input: 4 6 8 9

5 7 11 13

15 16 17 19

2 3 5 7

Output: 1,2,3

c) Tìm những dòng của ma trận chỉ chứa các số nguyên tố.

Input: 4 6 8 9

5 7 11 13

15 16 17 19

2 3 5 7

Output: 1,3

Cài đặt thuật toán:

#include <iostream>

#include <vector>

#include <cmath>

using namespace std;

// Kiểm tra số nguyên tố

bool isPrime(int num) {

if (num < 2) return false;

for (int i = 2; i <= sqrt(num); i++) {

if (num % i == 0) return false;

}

return true;

}

// Tìm số nguyên tố lớn nhất

int findLargestPrime(const vector<vector<int>>& matrix) {

int largestPrime = -1;

for (const auto& row : matrix) {

for (int element : row) {

if (isPrime(element) && element > largestPrime) {

largestPrime = element;

}

}

}

return largestPrime;

}

// Tìm dòng ít nhất có 1 số nguyên tố

vector<int> findRowsWithPrime(const vector<vector<int>>& matrix) {

vector<int> rowsWithPrime;

for (size\_t i = 0; i < matrix.size(); ++i) {

for (int element : matrix[i]) {

if (isPrime(element)) {

rowsWithPrime.push\_back(i);

break;

}

}

}

return rowsWithPrime;

}

// Tìm dòng tất cả đều là số nguyên tố

vector<int> findRowsWithAllPrimes(const vector<vector<int>>& matrix) {

vector<int> rowsWithAllPrimes;

for (size\_t i = 0; i < matrix.size(); ++i) {

bool allPrimes = true;

for (int element : matrix[i]) {

if (!isPrime(element)) {

allPrimes = false;

break;

}

}

if (allPrimes) {

rowsWithAllPrimes.push\_back(i);

}

}

return rowsWithAllPrimes;

}

int main() {

vector<vector<int>> matrix = {

{4, 6, 8, 9},

{5, 7, 11, 13},

{15, 16, 17, 19},

{2, 3, 5, 7}

};

int largestPrime = findLargestPrime(matrix);

vector<int> rowsWithPrime = findRowsWithPrime(matrix);

vector<int> rowsWithAllPrimes = findRowsWithAllPrimes(matrix);

cout << "Số nguyên tố lớn nhất trong ma trận: " << largestPrime << endl;

cout << "Những dòng có chứa số nguyên tố: ";

for (int row : rowsWithPrime) {

cout << row << " ";

}

cout << endl;

cout << "Những dòng chỉ chứa số nguyên tố: ";

for (int row : rowsWithAllPrimes) {

cout << row << " ";

}

cout << endl;

return 0;

}

## **3.5.**

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

**a. Tìm dòng có tổng lớn nhất.**

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

Phân tích bài toán:

a) Tìm dòng có tổng lớn nhất:

* Tính tổng các phần tử của từng dòng.
* Tìm dòng có tổng lớn nhất.

Input: 4 6 8 9

5 7 11 13

15 16 17 19

2 3 5 7

Output: 2

b) Sắp xếp các dòng sao cho dòng có tổng các phần tử lớn hơn sẽ nằm phía trên:

* + Tính tổng các phần tử của từng dòng.
  + Sắp xếp các dòng dựa trên tổng các phần tử giảm dần.

Input: 4 6 8 9

5 7 11 13

15 16 17 19

2 3 5 7

Output: 15 16 17 19

5 7 11 13

4 6 8 9

2 3 5 7

**Cài đặt thuật toán:**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Tính tổng các phần tử trong dòng

int sumRow(const vector<int>& row) {

int sum = 0;

for (int element : row) {

sum += element;

}

return sum;

}

// Tìm dòng có tổng lớn nhất

int findRowWithMaxSum(const vector<vector<int>>& matrix) {

int maxSum = -1;

int rowIndex = -1;

for (size\_t i = 0; i < matrix.size(); ++i) {

int currentSum = sumRow(matrix[i]);

if (currentSum > maxSum) {

maxSum = currentSum;

rowIndex = i;

}

}

return rowIndex;

}

// Sắp xếp các dòng dựa trên tổng các phần tử giảm dần

void sortRowsBySum(vector<vector<int>>& matrix) {

sort(matrix.begin(), matrix.end(), [](const vector<int>& a, const vector<int>& b) {

return sumRow(a) > sumRow(b);

});

}

// Kiểm thử

void printMatrix(const vector<vector<int>>& matrix) {

for (const auto& row : matrix) {

for (int element : row) {

cout << element << " ";

}

cout << endl;

}

}

int main() {

vector<vector<int>> matrix = {

{4, 6, 8, 9},

{5, 7, 11, 13},

{15, 16, 17, 19},

{2, 3, 5, 7}

};

// Tìm dòng có tổng lớn nhất

int rowIndex = findRowWithMaxSum(matrix);

cout << "Dòng có tổng lớn nhất: " << rowIndex << endl;

// Sắp xếp các dòng dựa trên tổng các phần tử giảm dần

sortRowsBySum(matrix);

cout << "Ma trận sau khi sắp xếp:\n";

printMatrix(matrix);

return 0;

}

## **3.6.**

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.

Ý TƯỞNG:

* Tạo mảng động để lưu số chẵn, lẻ và chỉ số của chúng trong mảng ban đầu.
* Duyệt qua mảng và phân loại số chẵn, lẻ, bỏ qua số 0.
* Sắp xếp số chẵn tăng dần và số lẻ giảm dần.
* Ghi lại các số đã sắp xếp vào đúng vị trí ban đầu.
* Giải phóng bộ nhớ động để tránh rò rỉ.

#include <iostream>

#include <algorithm>

using namespace std;

// Hàm sắp xếp mảng theo yêu cầu

void CustomSort(int arr[], int n) {

    // Tạo mảng động lưu số chẵn và số lẻ

    int\* even = new int[n];

    int\* odd = new int[n];

    int\* evenIndices = new int[n];

    int\* oddIndices = new int[n];

    int evenCount = 0, oddCount = 0;

    // Phân loại số chẵn, lẻ và lưu chỉ số

    for (int i = 0; i < n; i++) {

        if (arr[i] != 0) {

            if (arr[i] % 2 == 0) {

                even[evenCount] = arr[i];

                evenIndices[evenCount] = i;

                evenCount++;

            } else {

                odd[oddCount] = arr[i];

                oddIndices[oddCount] = i;

                oddCount++;

            }

        }

    }

    // Sắp xếp số chẵn tăng dần

    sort(even, even + evenCount);

    // Sắp xếp số lẻ giảm dần

    sort(odd, odd + oddCount, greater<int>());

    // Đưa các số đã sắp xếp vào mảng gốc

    for (int i = 0; i < evenCount; i++) {

        arr[evenIndices[i]] = even[i];

    }

    for (int i = 0; i < oddCount; i++) {

        arr[oddIndices[i]] = odd[i];

    }

    // Giải phóng bộ nhớ động

    delete[] even;

    delete[] odd;

    delete[] evenIndices;

    delete[] oddIndices;

}

// Hàm in mảng

void PrintArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// Hàm main để kiểm tra chương trình

int main() {

    int arr[] = {8, 5, 0, 3, 6, 9, 12, 4, 7, 10, 0, 1};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Mảng ban đầu: ";

    PrintArray(arr, n);

    CustomSort(arr, n);

    cout << "Mảng sau khi sắp xếp: ";

    PrintArray(arr, n);

    return 0;

}

## **3.7.**

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

**• Hãy sắp xếp sao cho các phần tử chẵn ở đầu, các phần tử lẻ về cuối.**

**• Yêu cầu độ phức tạp là O(n).**

Ý TƯỞNG:

sử dụng thuật toán hai con trỏ (Two Pointers), không cần sắp xếp lại từng phần:

* Dùng hai biến left và right để hoán đổi số chẵn về đầu và số lẻ về cuối.
* Duyệt một lần toàn bộ mảng O(n)

#include <iostream>

using namespace std;

// Hàm sắp xếp số chẵn lên đầu, số lẻ xuống cuối

void ArrangeEvenOdd(int arr[], int n) {

    int left = 0, right = n - 1;

    while (left < right) {

        // Tìm số lẻ bên trái

        while (left < right && arr[left] % 2 == 0) left++;

        // Tìm số chẵn bên phải

        while (left < right && arr[right] % 2 == 1) right--;

        // Hoán đổi nếu cần

        if (left < right) {

            swap(arr[left], arr[right]);

            left++;

            right--;

        }

    }

}

// Hàm in mảng

void PrintArray(int arr[], int n) {

    for (int i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

// Hàm main để kiểm tra chương trình

int main() {

    int arr[] = {8, 5, 2, 3, 6, 9, 12, 4, 7, 10};

    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Mảng ban đầu: ";

    PrintArray(arr, n);

    ArrangeEvenOdd(arr, n);

    cout << "Mảng sau khi sắp xếp: ";

    PrintArray(arr, n);

    return 0;

}

## **3.8.**

**A white paper with black text

AI-generated content may be incorrect.**

**Ý TƯỞNG:**

Tìm mảng nghịch thế từ hoán vị A:

* Duyệt từng phần tử ai​, đếm số phần tử trước đó lớn hơn aia\_iai​.
* Độ phức tạp O(n2), có thể tối ưu về O(n log n) bằng cây Fenwick hoặc Merge Sort.

Tìm hoán vị từ mảng nghịch thế B:

* Xây dựng lại hoán vị bằng cách đặt các số vào đúng vị trí theo B.
* Dùng cấu trúc dữ liệu hỗ trợ chèn nhanh (ví dụ: cây tìm kiếm hoặc danh sách liên kết).

#include <iostream>

using namespace std;

// Hàm tìm mảng nghịch thế từ hoán vị A

void findInversionArray(int A[], int B[], int n) {

    for (int i = 0; i < n; i++) {

        B[i] = 0;

        for (int j = 0; j < i; j++) {

            if (A[j] > A[i]) B[i]++;

        }

    }

}

// Hàm tìm hoán vị từ mảng nghịch thế B

void findPermutationFromInversionArray(int B[], int A[], int n) {

    bool used[n + 1] = {false};  // Đánh dấu số đã dùng

    for (int i = n - 1; i >= 0; i--) {

        int count = B[i];

        for (int j = 1; j <= n; j++) {

            if (!used[j]) {

                if (count == 0) {

                    A[i] = j;

                    used[j] = true;

                    break;

                }

                count--;

            }

        }

    }

}

// Hàm in mảng

void printArray(int arr[], int n) {

    for (int i = 0; i < n; i++) cout << arr[i] << " ";

    cout << endl;

}

// Hàm main

int main() {

    int A[] = {5, 9, 1, 8, 2, 6, 4, 7, 3};

    int n = sizeof(A) / sizeof(A[0]);

    int B[n];

    // Tìm mảng nghịch thế

    findInversionArray(A, B, n);

    cout << "Mang nghich the: ";

    printArray(B, n);

    // Tìm lại hoán vị từ mảng nghịch thế

    int A\_recovered[n];

    findPermutationFromInversionArray(B, A\_recovered, n);

    cout << "Hoan vi tim lai: ";

    printArray(A\_recovered, n);

    return 0;

}

## **3.9.**

**A white background with black text

AI-generated content may be incorrect.**

A white paper with black text

AI-generated content may be incorrect.

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

#define MAX 100  // Số lượng sinh viên tối đa

// Cấu trúc dữ liệu sinh viên

struct SinhVien {

    int maSo;

    char hoDem[21];  // Chuỗi tối đa 20 ký tự + '\0'

    char ten[41];    // Chuỗi tối đa 40 ký tự + '\0'

    int ngay, thang, nam;

    char phai[4];    // "Nam" hoặc "Nữ"

    float diemTB;

};

// Hàm kiểm tra mã số sinh viên có trùng hay không

bool kiemTraTrungMa(SinhVien ds[], int n, int maSo) {

    for (int i = 0; i < n; i++) {

        if (ds[i].maSo == maSo) return true;

    }

    return false;

}

// Hàm nhập danh sách sinh viên

void nhapDanhSach(SinhVien ds[], int &n) {

    cout << "Nhập số lượng sinh viên: ";

    cin >> n;

    while (n < 10) {

        cout << "Phải nhập ít nhất 10 sinh viên. Nhập lại: ";

        cin >> n;

    }

    for (int i = 0; i < n; i++) {

        SinhVien sv;

        cout << "Nhập thông tin sinh viên thứ " << i + 1 << ":\n";

        do {

            cout << "  Mã số: ";

            cin >> sv.maSo;

            if (kiemTraTrungMa(ds, i, sv.maSo)) {

                cout << "  Mã số đã tồn tại, nhập lại!\n";

            }

        } while (kiemTraTrungMa(ds, i, sv.maSo));

        cin.ignore();

        cout << "  Họ và đệm: "; cin.getline(sv.hoDem, 21);

        cout << "  Tên: "; cin.getline(sv.ten, 41);

        cout << "  Ngày sinh (dd mm yyyy): "; cin >> sv.ngay >> sv.thang >> sv.nam;

        cout << "  Phái (Nam/Nữ): "; cin >> sv.phai;

        cout << "  Điểm trung bình: "; cin >> sv.diemTB;

        ds[i] = sv;

    }

}

// Hàm lưu danh sách sinh viên vào file nhị phân

void luuFile(const char \*fileName, SinhVien ds[], int n) {

    ofstream outFile(fileName, ios::binary);

    outFile.write(reinterpret\_cast<char\*>(ds), n \* sizeof(SinhVien));

    outFile.close();

}

// Hàm đọc danh sách sinh viên từ file nhị phân

void docFile(const char \*fileName, SinhVien ds[], int &n) {

    ifstream inFile(fileName, ios::binary);

    if (!inFile) {

        cout << "Không thể mở file " << fileName << "!\n";

        return;

    }

    n = 0;

    while (inFile.read(reinterpret\_cast<char\*>(&ds[n]), sizeof(SinhVien))) {

        n++;

    }

    inFile.close();

}

// Hàm sắp xếp theo mã số tăng dần

void sapXepTheoMaSo(SinhVien ds[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (ds[i].maSo > ds[j].maSo) {

                swap(ds[i], ds[j]);

            }

        }

    }

}

// Hàm sắp xếp theo tên, nếu trùng tên thì xét họ và đệm

void sapXepTheoTen(SinhVien ds[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (strcmp(ds[i].ten, ds[j].ten) > 0 ||

                (strcmp(ds[i].ten, ds[j].ten) == 0 && strcmp(ds[i].hoDem, ds[j].hoDem) > 0)) {

                swap(ds[i], ds[j]);

            }

        }

    }

}

// Hàm sắp xếp theo điểm trung bình giảm dần

void sapXepTheoDiemTB(SinhVien ds[], int n) {

    for (int i = 0; i < n - 1; i++) {

        for (int j = i + 1; j < n; j++) {

            if (ds[i].diemTB < ds[j].diemTB) {

                swap(ds[i], ds[j]);

            }

        }

    }

}

// Hàm tạo file chỉ mục

void taoFileChiMuc(const char \*fileName, SinhVien ds[], int n) {

    ofstream outFile(fileName, ios::binary);

    for (int i = 0; i < n; i++) {

        outFile.write(reinterpret\_cast<char\*>(&ds[i].maSo), sizeof(int));

    }

    outFile.close();

}

// Hàm hiển thị danh sách sinh viên

void hienThiDanhSach(SinhVien ds[], int n) {

    cout << "\nDanh sách sinh viên:\n";

    for (int i = 0; i < n; i++) {

        cout << ds[i].maSo << " - " << ds[i].hoDem << " " << ds[i].ten

             << " - " << ds[i].ngay << "/" << ds[i].thang << "/" << ds[i].nam

             << " - " << ds[i].phai << " - Điểm TB: " << ds[i].diemTB << endl;

    }

}

int main() {

    SinhVien ds[MAX];

    int n;

    // Nhập và lưu danh sách sinh viên

    nhapDanhSach(ds, n);

    luuFile("SINHVIEN.DAT", ds, n);

    // Đọc lại danh sách từ file

    docFile("SINHVIEN.DAT", ds, n);

    // Sắp xếp theo mã số và lưu file chỉ mục

    sapXepTheoMaSo(ds, n);

    luuFile("SVMASO.IDX", ds, n);

    // Sắp xếp theo tên và lưu file chỉ mục

    sapXepTheoTen(ds, n);

    luuFile("SVTH.IDX", ds, n);

    // Sắp xếp theo điểm trung bình và lưu file chỉ mục

    sapXepTheoDiemTB(ds, n);

    luuFile("SVDTB.IDX", ds, n);

    // Hiển thị danh sách sinh viên đã sắp xếp

    hienThiDanhSach(ds, n);

    return 0;

}