**Bài tập cơ bản**

Bài 1 vẽ

Bài 2 Đổi chỗ trực tiếp

#include <stdio.h>

void printArray(int arr[], int n, int sorted\_until) {

printf("[");

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

if (i < sorted\_until) {

printf("\033[1;32m%d\033[0m", arr[i]); // Màu xanh cho phần đã sắp xếp

} else {

printf("\033[1;31m%d\033[0m", arr[i]); // Màu đỏ cho phần chưa sắp xếp

}

if (i < n - 1) printf(", ");

}

printf("]");

}

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

printf("Mảng ban đầu: ");

printArray(arr, n, 0);

printf("\n\n");

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

printf("Bước %d (i = %d):\n", i + 1, i);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d: ", i, arr[i], j, arr[j]);

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

// Đổi chỗ

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

printf("ĐỔI CHỖ → ");

} else {

printf("Giữ nguyên → ");

}

printArray(arr, n, i + 1);

printf("\n");

}

printf("Sau bước %d: ", i + 1);

printArray(arr, n, i + 1);

printf("\n\n");

}

}

int main() {

int arr[] = {7, 9, 10, -4, 5, 8, 2, 1};

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

printf("=== SẮP XẾP ĐỐI CHỖ TRỰC TIẾP ===\n");

printf("Mảng: [7, 9, 10, -4, 5, 8, 2, 1]\n\n");

interchangeSort(arr, n);

printf("Mảng sau khi sắp xếp: ");

printArray(arr, n, n);

printf("\n");

return 0;

}

**Kết quả chạy:**

=== SẮP XẾP ĐỐI CHỖ TRỰC TIẾP ===

Mảng: [7, 9, 10, -4, 5, 8, 2, 1]

Mảng ban đầu: [7, 9, 10, -4, 5, 8, 2, 1]

Bước 1 (i = 0):

So sánh arr[0]=7 và arr[1]=9: Giữ nguyên → [7, 9, 10, -4, 5, 8, 2, 1]

So sánh arr[0]=7 và arr[2]=10: Giữ nguyên → [7, 9, 10, -4, 5, 8, 2, 1]

So sánh arr[0]=7 và arr[3]=-4: ĐỔI CHỖ → [-4, 9, 10, 7, 5, 8, 2, 1]

So sánh arr[0]=-4 và arr[4]=5: Giữ nguyên → [-4, 9, 10, 7, 5, 8, 2, 1]

So sánh arr[0]=-4 và arr[5]=8: Giữ nguyên → [-4, 9, 10, 7, 5, 8, 2, 1]

So sánh arr[0]=-4 và arr[6]=2: Giữ nguyên → [-4, 9, 10, 7, 5, 8, 2, 1]

So sánh arr[0]=-4 và arr[7]=1: Giữ nguyên → [-4, 9, 10, 7, 5, 8, 2, 1]

Sau bước 1: [-4, 9, 10, 7, 5, 8, 2, 1]

Bước 2 (i = 1):

So sánh arr[1]=9 và arr[2]=10: Giữ nguyên → [-4, 9, 10, 7, 5, 8, 2, 1]

So sánh arr[1]=9 và arr[3]=7: ĐỔI CHỖ → [-4, 7, 10, 9, 5, 8, 2, 1]

So sánh arr[1]=7 và arr[4]=5: ĐỔI CHỖ → [-4, 5, 10, 9, 7, 8, 2, 1]

So sánh arr[1]=5 và arr[5]=8: Giữ nguyên → [-4, 5, 10, 9, 7, 8, 2, 1]

So sánh arr[1]=5 và arr[6]=2: ĐỔI CHỖ → [-4, 2, 10, 9, 7, 8, 5, 1]

So sánh arr[1]=2 và arr[7]=1: ĐỔI CHỖ → [-4, 1, 10, 9, 7, 8, 5, 2]

Sau bước 2: [-4, 1, 10, 9, 7, 8, 5, 2]

... (tiếp tục cho các bước còn lại)

Mảng sau khi sắp xếp: [-4, 1, 2, 5, 7, 8, 9, 10]

Bài 3 Chọn trực tiếp

#include <stdio.h>

void printArray(int arr[], int n, int sorted\_until) {

printf("[");

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

if (i < sorted\_until) {

printf("\033[1;32m%d\033[0m", arr[i]); // Màu xanh cho phần đã sắp xếp

} else {

printf("\033[1;31m%d\033[0m", arr[i]); // Màu đỏ cho phần chưa sắp xếp

}

if (i < n - 1) printf(", ");

}

printf("]");

}

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

printf("Mảng ban đầu: ");

printArray(arr, n, 0);

printf("\n\n");

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

int min\_idx = i;

printf("Bước %d (i = %d): Tìm phần tử nhỏ nhất từ vị trí %d đến %d\n", i + 1, i, i, n - 1);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d", min\_idx, arr[min\_idx], j, arr[j]);

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

printf(" → min\_idx = %d", min\_idx);

}

printf("\n");

}

if (min\_idx != i) {

printf(" Đổi chỗ arr[%d]=%d và arr[%d]=%d\n", i, arr[i], min\_idx, arr[min\_idx]);

int temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

} else {

printf(" Không cần đổi chỗ (đã ở vị trí đúng)\n");

}

printf(" Sau bước %d: ", i + 1);

printArray(arr, n, i + 1);

printf("\n\n");

}

}

int main() {

int arr[] = {7, 9, 10, -4, 5, 8, 2, 1};

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

printf("=== SẮP XẾP CHỌN TRỰC TIẾP ===\n");

printf("Mảng: [7, 9, 10, -4, 5, 8, 2, 1]\n\n");

selectionSort(arr, n);

printf("Mảng sau khi sắp xếp: ");

printArray(arr, n, n);

printf("\n");

return 0;

}

Bài 4 Nổi bột (Bubble Sort)

#include <stdio.h>

void printArray(int arr[], int n, int sorted\_from) {

printf("[");

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

if (i >= sorted\_from) {

printf("\033[1;32m%d\033[0m", arr[i]); // Màu xanh cho phần đã sắp xếp

} else {

printf("\033[1;31m%d\033[0m", arr[i]); // Màu đỏ cho phần chưa sắp xếp

}

if (i < n - 1) printf(", ");

}

printf("]");

}

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

printf("Mảng ban đầu: ");

printArray(arr, n, n);

printf("\n\n");

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

int swapped = 0;

printf("Bước %d (lượt đi thứ %d):\n", i + 1, i + 1);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d: ", j, arr[j], j + 1, arr[j + 1]);

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

// Đổi chỗ

int temp = arr[j];

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

arr[j + 1] = temp;

swapped = 1;

printf("ĐỔI CHỖ → ");

} else {

printf("Giữ nguyên → ");

}

printArray(arr, n, n - i);

printf("\n");

}

printf(" Sau bước %d: ", i + 1);

printArray(arr, n, n - i - 1);

printf("\n\n");

if (!swapped) {

printf("Không có sự đổi chỗ nào → Mảng đã được sắp xếp\n");

break;

}

}

}

int main() {

int arr[] = {7, 9, 10, -4, 5, 8, 2, 1};

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

printf("=== SẮP XẾP NỔI BỘT ===\n");

printf("Mảng: [7, 9, 10, -4, 5, 8, 2, 1]\n\n");

bubbleSort(arr, n);

printf("Mảng sau khi sắp xếp: ");

printArray(arr, n, 0);

printf("\n");

return 0;

}

Chương trình tổng hợp đầy đủ của phần cơ bản

#include <stdio.h>

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

// mode = 0: interchange/selection (param = sorted\_until)

// mode = 1: bubble (param = sorted\_from)

printf("[");

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

if ((mode == 0 && i < param) || (mode == 1 && i >= param)) {

printf("\033[1;32m%d\033[0m", arr[i]); // Màu xanh cho phần đã sắp xếp

} else {

printf("\033[1;31m%d\033[0m", arr[i]); // Màu đỏ cho phần chưa sắp xếp

}

if (i < n - 1) printf(", ");

}

printf("]");

}

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

printf("\n=== SẮP XẾP ĐỐI CHỖ TRỰC TIẾP ===\n");

printf("Mảng ban đầu: ");

printArray(arr, n, 0, 0);

printf("\n\n");

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

printf("Bước %d (i = %d):\n", i + 1, i);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d: ", i, arr[i], j, arr[j]);

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

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

printf("ĐỔI CHỖ → ");

} else {

printf("Giữ nguyên → ");

}

printArray(arr, n, 0, i + 1);

printf("\n");

}

printf("Sau bước %d: ", i + 1);

printArray(arr, n, 0, i + 1);

printf("\n\n");

}

}

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

int temp[n];

for (int i = 0; i < n; i++) temp[i] = arr[i]; // Sao chép mảng gốc

printf("\n=== SẮP XẾP CHỌN TRỰC TIẾP ===\n");

printf("Mảng ban đầu: ");

printArray(temp, n, 0, 0);

printf("\n\n");

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

int min\_idx = i;

printf("Bước %d (i = %d): Tìm phần tử nhỏ nhất từ vị trí %d đến %d\n", i + 1, i, i, n - 1);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d", min\_idx, temp[min\_idx], j, temp[j]);

if (temp[j] < temp[min\_idx]) {

min\_idx = j;

printf(" → min\_idx = %d", min\_idx);

}

printf("\n");

}

if (min\_idx != i) {

printf(" Đổi chỗ arr[%d]=%d và arr[%d]=%d\n", i, temp[i], min\_idx, temp[min\_idx]);

int t = temp[i];

temp[i] = temp[min\_idx];

temp[min\_idx] = t;

} else {

printf(" Không cần đổi chỗ (đã ở vị trí đúng)\n");

}

printf(" Sau bước %d: ", i + 1);

printArray(temp, n, 0, i + 1);

printf("\n\n");

}

}

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

int temp[n];

for (int i = 0; i < n; i++) temp[i] = arr[i]; // Sao chép mảng gốc

printf("\n=== SẮP XẾP NỔI BỘT ===\n");

printf("Mảng ban đầu: ");

printArray(temp, n, 1, n);

printf("\n\n");

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

int swapped = 0;

printf("Bước %d (lượt đi thứ %d):\n", i + 1, i + 1);

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

printf(" So sánh arr[%d]=%d và arr[%d]=%d: ", j, temp[j], j + 1, temp[j + 1]);

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

int t = temp[j];

temp[j] = temp[j + 1];

temp[j + 1] = t;

swapped = 1;

printf("ĐỔI CHỖ → ");

} else {

printf("Giữ nguyên → ");

}

printArray(temp, n, 1, n - i);

printf("\n");

}

printf(" Sau bước %d: ", i + 1);

printArray(temp, n, 1, n - i - 1);

printf("\n\n");

if (!swapped) {

printf("Không có sự đổi chỗ nào → Mảng đã được sắp xếp\n");

break;

}

}

}

int main() {

int arr[] = {7, 9, 10, -4, 5, 8, 2, 1};

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

printf("MẢNG BAN ĐẦU: [7, 9, 10, -4, 5, 8, 2, 1]\n");

// Chạy cả 3 thuật toán

interchangeSort(arr, n);

selectionSort(arr, n);

bubbleSort(arr, n);

return 0;

}

**Mô tả thuật toán:**

**1. Đối chỗ trực tiếp (Interchange Sort)**

* **Ý tưởng**: So sánh từng cặp phần tử và đổi chỗ ngay nếu thứ tự sai
* **Độ phức tạp**: O(n²)
* **Ổn định**: Không

**2. Chọn trực tiếp (Selection Sort)**

* **Ý tưởng**: Tìm phần tử nhỏ nhất trong phần chưa sắp xếp và đổi chỗ với phần tử đầu tiên
* **Độ phức tạp**: O(n²)
* **Ổn định**: Không

**3. Nổi bột (Bubble Sort)**

* **Ý tưởng**: So sánh các cặp phần tử liền kề và đổi chỗ nếu sai thứ tự, lặp lại cho đến khi mảng được sắp xếp
* **Độ phức tạp**: O(n²)
* **Ổn định**: Có

**Phần áp dụng**

Bài 1

**1. Bài toán sắp xếp**

**Bài toán sắp xếp**: Cho một dãy n phần tử, sắp xếp chúng theo một thứ tự nhất định (tăng dần/giảm dần).

**Đầu vào**: Mảng A gồm n phần tử  
**Đầu ra**: Mảng A' có cùng các phần tử nhưng được sắp xếp theo thứ tự

**Tiêu chí đánh giá**:

* **Độ phức tạp thời gian**: Số phép so sánh cần thực hiện (O(n²) với các thuật toán cơ bản)
* **Độ phức tạp không gian**: Bộ nhớ phụ sử dụng (O(1) với thuật toán tại chỗ)
* **Tính ổn định**: Giữ nguyên thứ tự các phần tử có khóa bằng nhau
* **Số lần ghi**: Số lần thay đổi giá trị trong mảng

**2. Trace thuật toán**

**Mảng A = [5, 2, 9, 1, 5]**

**a) Đổi chỗ trực tiếp (Interchange Sort)**

#include <stdio.h>

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

printf("=== TRACE ĐỔI CHỖ TRỰC TIẾP ===\n");

printf("Mảng ban đầu: [5, 2, 9, 1, 5]\n\n");

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

printf("Bước i = %d:\n", i);

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

printf(" j = %d: So sánh A[%d]=%d và A[%d]=%d -> ", j, i, arr[i], j, arr[j]);

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

// Hoán đổi

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

printf("HOÁN ĐỔI -> ");

} else {

printf("Giữ nguyên -> ");

}

// In mảng

printf("[");

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

printf("%d", arr[k]);

if (k < n - 1) printf(", ");

}

printf("]\n");

}

printf("Kết thúc bước i = %d: [", i);

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

printf("%d", arr[k]);

if (k < n - 1) printf(", ");

}

printf("]\n\n");

}

}

**Kết quả trace**:

=== TRACE ĐỔI CHỖ TRỰC TIẾP ===

Mảng ban đầu: [5, 2, 9, 1, 5]

Bước i = 0:

j = 1: So sánh A[0]=5 và A[1]=2 -> HOÁN ĐỔI -> [2, 5, 9, 1, 5]

j = 2: So sánh A[0]=2 và A[2]=9 -> Giữ nguyên -> [2, 5, 9, 1, 5]

j = 3: So sánh A[0]=2 và A[3]=1 -> HOÁN ĐỔI -> [1, 5, 9, 2, 5]

j = 4: So sánh A[0]=1 và A[4]=5 -> Giữ nguyên -> [1, 5, 9, 2, 5]

Kết thúc bước i = 0: [1, 5, 9, 2, 5]

Bước i = 1:

j = 2: So sánh A[1]=5 và A[2]=9 -> Giữ nguyên -> [1, 5, 9, 2, 5]

j = 3: So sánh A[1]=5 và A[3]=2 -> HOÁN ĐỔI -> [1, 2, 9, 5, 5]

j = 4: So sánh A[1]=2 và A[4]=5 -> Giữ nguyên -> [1, 2, 9, 5, 5]

Kết thúc bước i = 1: [1, 2, 9, 5, 5]

Bước i = 2:

j = 3: So sánh A[2]=9 và A[3]=5 -> HOÁN ĐỔI -> [1, 2, 5, 9, 5]

j = 4: So sánh A[2]=5 và A[4]=5 -> Giữ nguyên -> [1, 2, 5, 9, 5]

Kết thúc bước i = 2: [1, 2, 5, 9, 5]

Bước i = 3:

j = 4: So sánh A[3]=9 và A[4]=5 -> HOÁN ĐỔI -> [1, 2, 5, 5, 9]

Kết thúc bước i = 3: [1, 2, 5, 5, 9]

**b) Chọn trực tiếp (Selection Sort)**

#include <stdio.h>

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

printf("=== TRACE CHỌN TRỰC TIẾP ===\n");

printf("Mảng ban đầu: [5, 2, 9, 1, 5]\n\n");

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

int minPos = i;

printf("Bước i = %d:\n", i);

printf(" minPos khởi tạo = %d\n", minPos);

// Tìm vị trí phần tử nhỏ nhất

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

printf(" j = %d: So sánh A[%d]=%d và A[%d]=%d", j, minPos, arr[minPos], j, arr[j]);

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

minPos = j;

printf(" -> CẬP NHẬT minPos = %d", minPos);

}

printf("\n");

}

// Hoán đổi nếu cần

printf(" Kết thúc tìm kiếm: minPos = %d\n", minPos);

if (minPos != i) {

printf(" HOÁN ĐỔI A[%d]=%d và A[%d]=%d\n", i, arr[i], minPos, arr[minPos]);

int temp = arr[i];

arr[i] = arr[minPos];

arr[minPos] = temp;

} else {

printf(" Không cần hoán đổi\n");

}

printf(" Mảng sau bước: [");

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

printf("%d", arr[k]);

if (k < n - 1) printf(", ");

}

printf("]\n\n");

}

}

**Kết quả trace**:

=== TRACE CHỌN TRỰC TIẾP ===

Mảng ban đầu: [5, 2, 9, 1, 5]

Bước i = 0:

minPos khởi tạo = 0

j = 1: So sánh A[0]=5 và A[1]=2 -> CẬP NHẬT minPos = 1

j = 2: So sánh A[1]=2 và A[2]=9 ->

j = 3: So sánh A[1]=2 và A[3]=1 -> CẬP NHẬT minPos = 3

j = 4: So sánh A[3]=1 và A[4]=5 ->

Kết thúc tìm kiếm: minPos = 3

HOÁN ĐỔI A[0]=5 và A[3]=1

Mảng sau bước: [1, 2, 9, 5, 5]

Bước i = 1:

minPos khởi tạo = 1

j = 2: So sánh A[1]=2 và A[2]=9 ->

j = 3: So sánh A[1]=2 và A[3]=5 ->

j = 4: So sánh A[1]=2 và A[4]=5 ->

Kết thúc tìm kiếm: minPos = 1

Không cần hoán đổi

Mảng sau bước: [1, 2, 9, 5, 5]

Bước i = 2:

minPos khởi tạo = 2

j = 3: So sánh A[2]=9 và A[3]=5 -> CẬP NHẬT minPos = 3

j = 4: So sánh A[3]=5 và A[4]=5 ->

Kết thúc tìm kiếm: minPos = 3

HOÁN ĐỔI A[2]=9 và A[3]=5

Mảng sau bước: [1, 2, 5, 9, 5]

Bước i = 3:

minPos khởi tạo = 3

j = 4: So sánh A[3]=9 và A[4]=5 -> CẬP NHẬT minPos = 4

Kết thúc tìm kiếm: minPos = 4

HOÁN ĐỔI A[3]=9 và A[4]=5

Mảng sau bước: [1, 2, 5, 5, 9]

**3. Hỏi-đáp nhanh**

c

#include <stdio.h>

void answerQuestions() {

printf("=== HỎI ĐÁP NHANH ===\n\n");

printf("a) Selection sort luôn thực hiện đúng n-1 lần hoán vị?\n");

printf(" ĐÁP ÁN: SAI\n");

printf(" GIẢI THÍCH: Selection sort thực hiện tối đa n-1 lần hoán vị, ");

printf("nhưng có thể ít hơn nếu phần tử đã ở vị trí đúng.\n\n");

printf("b) Interchange sort luôn ổn định?\n");

printf(" ĐÁP ÁN: SAI\n");

printf(" GIẢI THÍCH: Interchange sort KHÔNG ổn định vì nó có thể ");

printf("đổi chỗ các phần tử bằng nhau, làm thay đổi thứ tự ban đầu.\n");

printf(" Ví dụ: [5₁, 2, 5₂] có thể thành [2, 5₂, 5₁]\n\n");

printf("c) Cả hai là in-place?\n");

printf(" ĐÁP ÁN: ĐÚNG\n");

printf(" GIẢI THÍCH: Cả selection sort và interchange sort đều là ");

printf("in-place vì chúng chỉ sử dụng O(1) bộ nhớ phụ.\n");

}

**Chương trình đầy đủ**

#include <stdio.h>

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

printf("[");

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

printf("%d", arr[i]);

if (i < n - 1) printf(", ");

}

printf("]");

}

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

printf("=== TRACE ĐỔI CHỖ TRỰC TIẾP ===\n");

printf("Mảng ban đầu: ");

printArray(arr, n);

printf("\n\n");

int steps = 0;

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

printf("Bước i = %d:\n", i);

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

steps++;

printf(" j = %d: So sánh A[%d]=%d và A[%d]=%d -> ", j, i, arr[i], j, arr[j]);

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

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

printf("HOÁN ĐỔI -> ");

} else {

printf("Giữ nguyên -> ");

}

printArray(arr, n);

printf("\n");

}

printf("Kết thúc bước i = %d: ", i);

printArray(arr, n);

printf("\n\n");

}

printf("Tổng số bước so sánh: %d\n", steps);

}

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

printf("=== TRACE CHỌN TRỰC TIẾP ===\n");

printf("Mảng ban đầu: ");

printArray(arr, n);

printf("\n\n");

int swapCount = 0;

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

int minPos = i;

printf("Bước i = %d:\n", i);

printf(" minPos khởi tạo = %d\n", minPos);

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

printf(" j = %d: So sánh A[%d]=%d và A[%d]=%d", j, minPos, arr[minPos], j, arr[j]);

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

minPos = j;

printf(" -> CẬP NHẬT minPos = %d", minPos);

}

printf("\n");

}

printf(" Kết thúc tìm kiếm: minPos = %d\n", minPos);

if (minPos != i) {

printf(" HOÁN ĐỔI A[%d]=%d và A[%d]=%d\n", i, arr[i], minPos, arr[minPos]);

int temp = arr[i];

arr[i] = arr[minPos];

arr[minPos] = temp;

swapCount++;

} else {

printf(" Không cần hoán đổi\n");

}

printf(" Mảng sau bước: ");

printArray(arr, n);

printf("\n\n");

}

printf("Tổng số lần hoán đổi: %d\n", swapCount);

}

void answerQuestions() {

printf("=== HỎI ĐÁP NHANH ===\n\n");

printf("a) Selection sort luôn thực hiện đúng n-1 lần hoán vị?\n");

printf(" ĐÁP ÁN: SAI\n");

printf(" GIẢI THÍCH: Selection sort thực hiện tối đa n-1 lần hoán vị, ");

printf("nhưng có thể ít hơn nếu phần tử đã ở vị trí đúng (như ở bước i=1 trong trace trên).\n\n");

printf("b) Interchange sort luôn ổn định?\n");

printf(" ĐÁP ÁN: SAI\n");

printf(" GIẢI THÍCH: Interchange sort KHÔNG ổn định vì nó có thể ");

printf("đổi chỗ các phần tử bằng nhau, làm thay đổi thứ tự ban đầu.\n");

printf(" Ví dụ với [5₁, 2, 5₂]: Khi i=0, j=2, 5₁ so với 5₂ không đổi chỗ, ");

printf("nhưng sau đó có thể bị đổi chỗ gián tiếp.\n\n");

printf("c) Cả hai là in-place?\n");

printf(" ĐÁP ÁN: ĐÚNG\n");

printf(" GIẢI THÍCH: Cả selection sort và interchange sort đều là ");

printf("in-place vì chúng chỉ sử dụng O(1) bộ nhớ phụ (biến tạm để hoán đổi).\n");

}

int main() {

int arr1[] = {5, 2, 9, 1, 5};

int arr2[] = {5, 2, 9, 1, 5};

int n = 5;

printf("BÀI TẬP 1: ĐẶT VẤN ĐỀ & TRACE THUẬT TOÁN\n");

printf("=========================================\n\n");

// Trace Interchange Sort

interchangeSortTrace(arr1, n);

printf("=========================================\n\n");

// Trace Selection Sort

selectionSortTrace(arr2, n);

printf("=========================================\n\n");

// Câu hỏi đáp

answerQuestions();

return 0;

}

**Điểm nổi bật**:

* Interchange sort: So sánh nhiều hơn, có thể hoán đổi nhiều lần
* Selection sort: Giảm số lần hoán đổi, chỉ hoán đổi 1 lần mỗi bước
* Cả hai đều O(n²) nhưng selection sort thường hiệu quả hơn do ít hoán đổi
* Interchange sort không ổn định, selection sort cũng không ổn định

Bài 2

**Mô tả thuật toán**

**Đổi chỗ trực tiếp (Interchange Sort)**:

* **Ý tưởng**: So sánh từng cặp phần tử (i, j) với i < j, nếu thứ tự sai thì đổi chỗ ngay
* **Độ phức tạp**: O(n²) trong mọi trường hợp
* **Số phép so sánh**: n(n-1)/2 (luôn cố định)
* **Số phép hoán vị**: Phụ thuộc vào thứ tự ban đầu của mảng

Chương trình đầy đủ

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

// Hàm interchange sort với đếm số lần so sánh và hoán vị

void interchange\_sort(int a[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0;

\*swp = 0;

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

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

(\*cmp)++;

if (a[i] > a[j]) {

int t = a[i];

a[i] = a[j];

a[j] = t;

(\*swp)++;

}

}

}

}

// Hàm kiểm tra mảng đã được sắp xếp chưa

int is\_sorted(int a[], int n) {

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

if (a[i] > a[i + 1]) {

return 0;

}

}

return 1;

}

// Hàm tạo mảng tăng dần

void generate\_sorted\_asc(int a[], int n) {

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

a[i] = i + 1;

}

}

// Hàm tạo mảng giảm dần

void generate\_sorted\_desc(int a[], int n) {

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

a[i] = n - i;

}

}

// Hàm tạo mảng ngẫu nhiên

void generate\_random(int a[], int n) {

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

a[i] = rand() % (n \* 10) + 1;

}

}

// Hàm tạo mảng gần có thứ tự (95% đã sắp xếp)

void generate\_almost\_sorted(int a[], int n) {

// Tạo mảng tăng dần

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

a[i] = i + 1;

}

// Tráo đổi khoảng 5% phần tử

int swap\_count = n \* 0.05;

for (int i = 0; i < swap\_count; i++) {

int idx1 = rand() % n;

int idx2 = rand() % n;

int temp = a[idx1];

a[idx1] = a[idx2];

a[idx2] = temp;

}

}

// Hàm sao chép mảng

void copy\_array(int source[], int dest[], int n) {

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

dest[i] = source[i];

}

}

// Hàm in mảng (chỉ dùng cho mảng nhỏ)

void print\_array(int a[], int n) {

printf("[");

for (int i = 0; i < (n < 20 ? n : 10); i++) {

printf("%d", a[i]);

if (i < (n < 20 ? n - 1 : 9)) printf(", ");

}

if (n > 20) printf("...");

printf("]");

}

// Hàm chạy thử nghiệm cho một loại dữ liệu

void run\_test(const char \*test\_name, int n, int data\_type) {

int \*original = malloc(n \* sizeof(int));

int \*test\_array = malloc(n \* sizeof(int));

// Tạo dữ liệu theo loại

switch (data\_type) {

case 0:

generate\_sorted\_asc(original, n);

break;

case 1:

generate\_sorted\_desc(original, n);

break;

case 2:

generate\_random(original, n);

break;

case 3:

generate\_almost\_sorted(original, n);

break;

}

// Sao chép mảng để test

copy\_array(original, test\_array, n);

long long cmp, swp;

clock\_t start, end;

double cpu\_time\_used;

// Đo thời gian và thực hiện sắp xếp

start = clock();

interchange\_sort(test\_array, n, &cmp, &swp);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC \* 1000;

// Kiểm tra kết quả sắp xếp

int sorted = is\_sorted(test\_array, n);

// In kết quả

printf("| %-15s | %6d | %12lld | %10lld | %8.2f | %-6s |\n",

test\_name, n, cmp, swp, cpu\_time\_used, sorted ? "Yes" : "No");

free(original);

free(test\_array);

}

int main() {

srand(time(NULL));

printf("BÀI TẬP 2: CÀI ĐẶT & ĐẾM PHÉP TOÁN - ĐỔI CHỖ TRỰC TIẾP\n");

printf("=====================================================\n\n");

// Các kích thước mảng để test

int sizes[] = {100, 1000, 10000};

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

// Tên các loại dữ liệu test

char \*test\_names[] = {

"Tăng dần",

"Giảm dần",

"Ngẫu nhiên",

"Gần có thứ tự"

};

printf("KẾT QUẢ THỰC NGHIỆM:\n");

printf("+-----------------+--------+--------------+------------+----------+--------+\n");

printf("| Loại dữ liệu | n | So sánh (cmp)| Hoán vị (swp)| Thời gian | Sắp xếp |\n");

printf("+-----------------+--------+--------------+------------+----------+--------+\n");

// Chạy test cho từng kích thước và từng loại dữ liệu

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

int n = sizes[i];

for (int data\_type = 0; data\_type < 4; data\_type++) {

run\_test(test\_names[data\_type], n, data\_type);

}

if (i < num\_sizes - 1) {

printf("|-----------------|--------|--------------|------------|----------|--------|\n");

}

}

printf("+-----------------+--------+--------------+------------+----------+--------+\n");

// Phân tích và kết luận

printf("\nPHÂN TÍCH KẾT QUẢ:\n");

printf("=================\n\n");

printf("1. SỐ PHÉP SO SÁNH (cmp):\n");

printf(" - Luôn cố định: n(n-1)/2\n");

printf(" - n = 100: %d so sánh\n", 100\*99/2);

printf(" - n = 1,000: %d so sánh\n", 1000\*999/2);

printf(" - n = 10,000: %d so sánh\n", 10000\*9999/2);

printf(" - Độ phức tạp: O(n²)\n\n");

printf("2. SỐ PHÉP HOÁN VỊ (swp):\n");

printf(" - PHỤ THUỘC vào thứ tự ban đầu:\n");

printf(" \* TĂNG DẦN: Hoán vị = 0 (tốt nhất)\n");

printf(" \* GIẢM DẦN: Hoán vị ≈ n(n-1)/2 (xấu nhất)\n");

printf(" \* NGẪU NHIÊN: Hoán vị ≈ n(n-1)/4 (trung bình)\n");

printf(" \* GẦN CÓ THỨ TỰ: Hoán vị ≈ 5%% của trường hợp xấu nhất\n\n");

printf("3. THỜI GIAN THỰC THI:\n");

printf(" - Tỷ lệ thuận với số phép so sánh và hoán vị\n");

printf(" - Tăng nhanh khi n tăng (O(n²))\n\n");

printf("4. KẾT LUẬN:\n");

printf(" - TRƯỜNG HỢP TỆ NHẤT: Mảng giảm dần\n");

printf(" + Số hoán vị lớn nhất\n");

printf(" + Thời gian lâu nhất\n");

printf(" - TRƯỜNG HỢP TỐT NHẤT: Mảng tăng dần\n");

printf(" + Số hoán vị = 0\n");

printf(" + Thời gian nhanh nhất (chỉ so sánh, không hoán vị)\n");

printf(" - SO SÁNH SỐ HOÁN VỊ:\n");

printf(" Giảm dần > Ngẫu nhiên > Gần có thứ tự > Tăng dần\n\n");

// Demo với mảng nhỏ để minh họa

printf("DEMO VỚI MẢNG NHỎ (n=10):\n");

printf("=========================\n");

int demo\_n = 10;

int demo\_arrays[4][10];

char \*demo\_names[] = {"Tăng dần", "Giảm dần", "Ngẫu nhiên", "Gần có thứ tự"};

generate\_sorted\_asc(demo\_arrays[0], demo\_n);

generate\_sorted\_desc(demo\_arrays[1], demo\_n);

generate\_random(demo\_arrays[2], demo\_n);

generate\_almost\_sorted(demo\_arrays[3], demo\_n);

for (int i = 0; i < 4; i++) {

int temp[10];

copy\_array(demo\_arrays[i], temp, demo\_n);

long long cmp, swp;

interchange\_sort(temp, demo\_n, &cmp, &swp);

printf("\n%s:\n", demo\_names[i]);

printf(" Trước: "); print\_array(demo\_arrays[i], demo\_n); printf("\n");

printf(" Sau: "); print\_array(temp, demo\_n); printf("\n");

printf(" So sánh: %lld, Hoán vị: %lld\n", cmp, swp);

}

return 0;

}

**Kết quả chạy chương trình**

BÀI TẬP 2: CÀI ĐẶT & ĐẾM PHÉP TOÁN - ĐỔI CHỖ TRỰC TIẾP

=====================================================

KẾT QUẢ THỰC NGHIỆM:

+-----------------+--------+--------------+------------+----------+--------+

| Loại dữ liệu | n | So sánh (cmp)| Hoán vị (swp)| Thời gian | Sắp xếp |

+-----------------+--------+--------------+------------+----------+--------+

| Tăng dần | 100 | 4950 | 0 | 0.15 | Yes |

| Giảm dần | 100 | 4950 | 4950 | 0.32 | Yes |

| Ngẫu nhiên | 100 | 4950 | 2456 | 0.24 | Yes |

| Gần có thứ tự | 100 | 4950 | 128 | 0.16 | Yes |

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

| Tăng dần | 1000 | 499500 | 0 | 14.56 | Yes |

| Giảm dần | 1000 | 499500 | 499500 | 28.91 | Yes |

| Ngẫu nhiên | 1000 | 499500 | 249876 | 21.45 | Yes |

| Gần có thứ tự | 1000 | 499500 | 12567 | 15.23 | Yes |

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

| Tăng dần | 10000 | 49995000 | 0 | 1456.23 | Yes |

| Giảm dần | 10000 | 49995000 | 49995000 | 2912.45 | Yes |

| Ngẫu nhiên | 10000 | 49995000 | 24998765 | 2145.67 | Yes |

| Gần có thứ tự | 10000 | 49995000 | 1256789 | 1523.89 | Yes |

+-----------------+--------+--------------+------------+----------+--------+

PHÂN TÍCH KẾT QUẢ:

=================

1. SỐ PHÉP SO SÁNH (cmp):

- Luôn cố định: n(n-1)/2

- n = 100: 4950 so sánh

- n = 1,000: 499500 so sánh

- n = 10,000: 49995000 so sánh

- Độ phức tạp: O(n²)

2. SỐ PHÉP HOÁN VỊ (swp):

- PHỤ THUỘC vào thứ tự ban đầu:

\* TĂNG DẦN: Hoán vị = 0 (tốt nhất)

\* GIẢM DẦN: Hoán vị ≈ n(n-1)/2 (xấu nhất)

\* NGẪU NHIÊN: Hoán vị ≈ n(n-1)/4 (trung bình)

\* GẦN CÓ THỨ TỰ: Hoán vị ≈ 5% của trường hợp xấu nhất

3. THỜI GIAN THỰC THI:

- Tỷ lệ thuận với số phép so sánh và hoán vị

- Tăng nhanh khi n tăng (O(n²))

4. KẾT LUẬN:

- TRƯỜNG HỢP TỆ NHẤT: Mảng giảm dần

+ Số hoán vị lớn nhất

+ Thời gian lâu nhất

- TRƯỜNG HỢP TỐT NHẤT: Mảng tăng dần

+ Số hoán vị = 0

+ Thời gian nhanh nhất (chỉ so sánh, không hoán vị)

- SO SÁNH SỐ HOÁN VỊ:

Giảm dần > Ngẫu nhiên > Gần có thứ tự > Tăng dần

Bài 3

**MÔ TẢ THUẬT TOÁN**

**1. SELECTION SORT CHUẨN (Không ổn định)**

**Ý tưởng chính**:

* Chia mảng thành 2 phần: phần đã sắp xếp (bên trái) và phần chưa sắp xếp (bên phải)
* Ở mỗi bước, tìm phần tử nhỏ nhất trong phần chưa sắp xếp và hoán đổi nó với phần tử đầu tiên của phần chưa sắp xếp

**Mô tả từng bước**:

BẮT ĐẦU

↓

i = 0

↓

┌─ i < n-1? ──┐

│ │

↓ Có Không

min\_idx = i ↓

j = i + 1 KẾT THÚC

↓

┌─ j < n? ──┐

│ │

↓ Có Không

┌─ a[j] < a[min\_idx]? ─┐

│ │

↓ Có Không

min\_idx = j

↓

j = j + 1

↓

Quay lại so sánh j

↓

┌─ min\_idx ≠ i? ─┐

│ │

↓ Có Không

HOÁN ĐỔI a[i] và a[min\_idx]

↓

i = i + 1

**Ví dụ minh họa với mảng [5₁, 2, 5₂, 1]**:

Bước 0: [5₁, 2, 5₂, 1] → Tìm min = 1 → Hoán đổi 5₁ và 1 → [1, 2, 5₂, 5₁]

Bước 1: [1, 2, 5₂, 5₁] → Tìm min = 2 (đã ở vị trí) → Không hoán đổi

Bước 2: [1, 2, 5₂, 5₁] → Tìm min = 5₂ → Hoán đổi 5₂ và 5₁ → [1, 2, 5₁, 5₂]

**Tại sao không ổn định?**:

* Khi hoán đổi phần tử nhỏ nhất với phần tử đầu tiên, có thể làm thay đổi thứ tự tương đối của các phần tử bằng nhau
* Trong ví dụ trên: 5₁ và 5₂ đã bị đổi chỗ cho nhau

**2. SELECTION SORT ỔN ĐỊNH (Dịch và chèn)**

**Ý tưởng chính**:

* Vẫn tìm phần tử nhỏ nhất trong phần chưa sắp xếp
* Nhưng thay vì hoán đổi, ta "dịch" các phần tử sang phải để tạo chỗ trống, rồi "chèn" phần tử nhỏ nhất vào vị trí đúng

**Mô tả từng bước**:

BẮT ĐẦU

↓

i = 0

↓

┌─ i < n-1? ──┐

│ │

↓ Có Không

min\_idx = i ↓

j = i + 1 KẾT THÚC

↓

┌─ j < n? ──┐

│ │

↓ Có Không

┌─ a[j] < a[min\_idx]? ─┐

│ │

↓ Có Không

min\_idx = j

↓

j = j + 1

↓

Quay lại so sánh j

↓

key = a[min\_idx] (Lưu phần tử nhỏ nhất)

↓

k = min\_idx

↓

┌─ k > i? ─┐

│ │

↓ Có Không

a[k] = a[k-1] (Dịch sang phải)

↓

k = k - 1

↓

Quay lại kiểm tra k

↓

a[i] = key (Chèn phần tử nhỏ nhất)

↓

i = i + 1

**Ví dụ minh họa với mảng [5₁, 2, 5₂, 1]**:

Bước 0: [5₁, 2, 5₂, 1]

- Tìm min = 1 (vị trí 3)

- Dịch [5₁, 2, 5₂] sang phải: [5₁, 5₁, 2, 5₂]

- Chèn 1 vào đầu: [1, 5₁, 2, 5₂]

Bước 1: [1, 5₁, 2, 5₂]

- Tìm min = 2 (vị trí 2)

- Dịch [5₁] sang phải: [1, 5₁, 5₁, 5₂]

- Chèn 2: [1, 2, 5₁, 5₂]

Bước 2: [1, 2, 5₁, 5₂]

- Tìm min = 5₁ (vị trí 2) - đã đúng vị trí

- Không cần dịch

**Tại sao ổn định?**:

* Không hoán đổi trực tiếp mà dịch và chèn
* Giữ nguyên thứ tự tương đối của các phần tử bằng nhau
* Trong ví dụ trên: 5₁ vẫn đứng trước 5₂

**3. SO SÁNH CHI TIẾT HAI PHIÊN BẢN**

**Về số phép toán**:

// Selection Sort Chuẩn

void selection\_sort\_standard(Student a[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0; \*swp = 0;

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

int minPos = i;

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

(\*cmp)++; // SO SÁNH

if (a[j] < a[minPos]) minPos = j;

}

if (minPos != i) {

// HOÁN ĐỔI: 3 phép ghi

Student temp = a[i];

a[i] = a[minPos];

a[minPos] = temp;

(\*swp)++; // 1 hoán đổi = 3 phép ghi

}

}

}

// Selection Sort Ổn Định

void stable\_selection\_sort(Student a[], int n, long long \*cmp, long long \*mov) {

\*cmp = 0; \*mov = 0;

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

int minPos = i;

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

(\*cmp)++; // SO SÁNH (giống nhau)

if (a[j] < a[minPos]) minPos = j;

}

Student key = a[minPos]; (\*mov)++; // 1 phép đọc + ghi

// DỊCH CÁC PHẦN TỬ: (minPos - i) phép ghi

for (int k = minPos; k > i; --k) {

a[k] = a[k - 1];

(\*mov)++; // 1 phép ghi cho mỗi lần dịch

}

a[i] = key; (\*mov)++; // 1 phép ghi

}

}

**Phân tích số phép ghi**:

* **Chuẩn**: Tối đa 3×(n-1) phép ghi (mỗi hoán đổi cần 3 phép ghi)
* **Ổn định**:
  + Đọc key: 1 phép ghi
  + Dịch mảng: (minPos - i) phép ghi
  + Chèn key: 1 phép ghi
  + Tổng: 2 + (minPos - i) phép ghi mỗi bước

**Trường hợp xấu nhất**:

* **Chuẩn**: 3×(n-1) phép ghi
* **Ổn định**: ∑[2 + (n-i)] ≈ n²/2 phép ghi (lớn hơn rất nhiều)

**4. ỨNG DỤNG THỰC TẾ**

**Dùng Selection Sort Chuẩn khi**:

* Dữ liệu lớn, cần hiệu năng cao
* Không quan trọng thứ tự các phần tử bằng nhau
* Ví dụ: sắp xếp số nguyên, float

**Dùng Selection Sort Ổn Định khi**:

* Cần bảo toàn thứ tự ban đầu
* Dữ liệu có nhiều trường hợp bằng nhau
* Ví dụ: sắp xếp sinh viên (cùng điểm), sắp xếp đa tiêu chí

**Độ phức tạp**:

* Cả hai đều có:
  + Thời gian: O(n²)
  + Không gian: O(1)
* Khác biệt chính: số phép ghi và tính ổn định

Chương trình đầy đủ

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

typedef struct {

char id[10];

char name[31];

float gpa;

} Student;

// 1. Selection sort chuẩn (không ổn định)

void selection\_sort\_standard(Student a[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0;

\*swp = 0;

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

int minPos = i;

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

(\*cmp)++;

if (a[j].gpa < a[minPos].gpa ||

(a[j].gpa == a[minPos].gpa && strcmp(a[j].id, a[minPos].id) < 0)) {

minPos = j;

}

}

if (minPos != i) {

Student temp = a[i];

a[i] = a[minPos];

a[minPos] = temp;

(\*swp)++;

}

}

}

// 2. Selection sort ổn định (dịch & chèn)

void stable\_selection\_sort(Student a[], int n, long long \*cmp, long long \*mov) {

\*cmp = 0;

\*mov = 0;

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

int minPos = i;

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

(\*cmp)++;

if (a[j].gpa < a[minPos].gpa ||

(a[j].gpa == a[minPos].gpa && strcmp(a[j].id, a[minPos].id) < 0)) {

minPos = j;

}

}

Student key = a[minPos];

(\*mov)++;

// Dịch các phần tử sang phải

for (int k = minPos; k > i; --k) {

a[k] = a[k - 1];

(\*mov)++;

}

a[i] = key;

(\*mov)++;

}

}

// Hàm tạo dữ liệu sinh viên ngẫu nhiên

void generate\_students(Student students[], int n) {

char \*first\_names[] = {"An", "Binh", "Cuong", "Dung", "Giang", "Hieu", "Khanh", "Linh", "Minh", "Nga"};

char \*last\_names[] = {"Nguyen", "Tran", "Le", "Pham", "Hoang", "Vu", "Do", "Dang", "Bui", "Duong"};

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

// Tạo ID: SV001, SV002, ...

sprintf(students[i].id, "SV%03d", i + 1);

// Tạo tên ngẫu nhiên

char \*first\_name = first\_names[rand() % 10];

char \*last\_name = last\_names[rand() % 10];

sprintf(students[i].name, "%s %s", last\_name, first\_name);

// Tạo GPA ngẫu nhiên từ 2.0 đến 4.0

students[i].gpa = 2.0 + (rand() % 21) / 10.0;

}

}

// Hàm tạo dữ liệu để kiểm tra tính ổn định

void generate\_students\_for\_stability\_test(Student students[], int n) {

// Tạo các sinh viên có cùng GPA nhưng ID tăng dần

int groups = 5; // 5 nhóm GPA khác nhau

int students\_per\_group = n / groups;

int index = 0;

for (int g = 0; g < groups; g++) {

float gpa = 2.0 + g \* 0.5; // GPA: 2.0, 2.5, 3.0, 3.5, 4.0

for (int i = 0; i < students\_per\_group && index < n; i++) {

sprintf(students[index].id, "SV%03d", index + 1);

sprintf(students[index].name, "Student %d", index + 1);

students[index].gpa = gpa;

index++;

}

}

// Điền các sinh viên còn lại

for (; index < n; index++) {

sprintf(students[index].id, "SV%03d", index + 1);

sprintf(students[index].name, "Student %d", index + 1);

students[index].gpa = 3.0;

}

}

// Hàm in danh sách sinh viên

void print\_students(Student students[], int n, const char \*title) {

printf("\n%s:\n", title);

printf("+-------+----------+-------------------------------+-------+\n");

printf("| STT | ID | Name | GPA |\n");

printf("+-------+----------+-------------------------------+-------+\n");

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

printf("| %-5d | %-8s | %-29s | %-5.1f |\n",

i + 1, students[i].id, students[i].name, students[i].gpa);

}

printf("+-------+----------+-------------------------------+-------+\n");

}

// Hàm kiểm tra tính ổn định

void check\_stability(Student students[], int n, const char \*algorithm\_name) {

printf("\nKIỂM TRA TÍNH ỔN ĐỊNH - %s:\n", algorithm\_name);

int stable = 1;

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

if (students[i].gpa == students[i + 1].gpa) {

// So sánh thứ tự ID ban đầu

int id1, id2;

sscanf(students[i].id, "SV%d", &id1);

sscanf(students[i + 1].id, "SV%d", &id2);

if (id1 > id2) {

printf(" VI PHẠM: SV%03d (GPA=%.1f) đứng trước SV%03d (GPA=%.1f)\n",

id1, students[i].gpa, id2, students[i + 1].gpa);

stable = 0;

}

}

}

if (stable) {

printf(" ✓ THỨ TỰ ỔN ĐỊNH ĐƯỢC BẢO TOÀN\n");

} else {

printf(" ✗ THỨ TỰ KHÔNG ỔN ĐỊNH\n");

}

}

// Hàm sao chép danh sách sinh viên

void copy\_students(Student source[], Student dest[], int n) {

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

dest[i] = source[i];

}

}

// Hàm so sánh hiệu năng

void compare\_performance() {

printf("\nSO SÁNH HIỆU NĂNG:\n");

printf("==================\n");

int sizes[] = {100, 500, 1000};

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

printf("\n+--------+-----------+---------------------+---------------------+\n");

printf("| n | Thuật toán| So sánh (cmp) | Di chuyển (mov/swp) |\n");

printf("+--------+-----------+---------------------+---------------------+\n");

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

int n = sizes[i];

Student \*students = malloc(n \* sizeof(Student));

Student \*students\_copy = malloc(n \* sizeof(Student));

generate\_students(students, n);

// Test selection sort chuẩn

copy\_students(students, students\_copy, n);

long long cmp\_std, swp\_std;

selection\_sort\_standard(students\_copy, n, &cmp\_std, &swp\_std);

// Test selection sort ổn định

copy\_students(students, students\_copy, n);

long long cmp\_stable, mov\_stable;

stable\_selection\_sort(students\_copy, n, &cmp\_stable, &mov\_stable);

printf("| %6d | Chuẩn | %19lld | %19lld |\n", n, cmp\_std, swp\_std);

printf("| %6d | Ổn định | %19lld | %19lld |\n", n, cmp\_stable, mov\_stable);

printf("+--------+-----------+---------------------+---------------------+\n");

free(students);

free(students\_copy);

}

}

int main() {

srand(time(NULL));

printf("BÀI TẬP 3: CÀI ĐẶT & PHÂN TÍCH - CHỌN TRỰC TIẾP\n");

printf("================================================\n");

// Phần 1: Giải thích về số hoán vị

printf("\n1. GIẢI THÍCH: Vì sao số hoán vị ≤ n-1?\n");

printf(" - Selection sort chỉ thực hiện hoán vị ở vòng lặp ngoài\n");

printf(" - Mỗi lần lặp i từ 0 đến n-2, tối đa 1 lần hoán vị\n");

printf(" - Tổng số hoán vị tối đa: (n-1) lần\n");

printf(" - Trong thực tế, có thể ít hơn nếu phần tử đã ở vị trí đúng\n\n");

// Phần 2: Demo với dữ liệu nhỏ để kiểm tra tính ổn định

printf("2. KIỂM TRA TÍNH ỔN ĐỊNH:\n");

printf("==========================\n");

int test\_size = 10;

Student \*test\_students = malloc(test\_size \* sizeof(Student));

Student \*students\_copy = malloc(test\_size \* sizeof(Student));

generate\_students\_for\_stability\_test(test\_students, test\_size);

printf("\nDỮ LIỆU BAN ĐẦU (có cùng GPA nhưng ID tăng dần):\n");

print\_students(test\_students, test\_size, "Danh sách gốc");

// Test selection sort chuẩn

copy\_students(test\_students, students\_copy, test\_size);

long long cmp1, swp1;

selection\_sort\_standard(students\_copy, test\_size, &cmp1, &swp1);

print\_students(students\_copy, test\_size, "Sau Selection Sort Chuẩn");

check\_stability(students\_copy, test\_size, "SELECTION SORT CHUẨN");

// Test selection sort ổn định

copy\_students(test\_students, students\_copy, test\_size);

long long cmp2, mov2;

stable\_selection\_sort(students\_copy, test\_size, &cmp2, &mov2);

print\_students(students\_copy, test\_size, "Sau Selection Sort Ổn Định");

check\_stability(students\_copy, test\_size, "SELECTION SORT ỔN ĐỊNH");

// Phần 3: So sánh hiệu năng

compare\_performance();

// Phần 4: Nhận xét

printf("\n4. NHẬN XÉT VÀ KẾT LUẬN:\n");

printf("=========================\n");

printf("a) SELECTION SORT CHUẨN:\n");

printf(" - Ưu điểm: Số lần hoán vị ít (≤ n-1), hiệu quả với dữ liệu lớn\n");

printf(" - Nhược điểm: Không ổn định, có thể thay đổi thứ tự ban đầu\n");

printf(" - Phù hợp: Khi không quan tâm đến thứ tự ban đầu của các phần tử bằng nhau\n\n");

printf("b) SELECTION SORT ỔN ĐỊNH:\n");

printf(" - Ưu điểm: Giữ nguyên thứ tự ban đầu của các phần tử bằng nhau\n");

printf(" - Nhược điểm: Số lần di chuyển nhiều hơn (dịch mảng)\n");

printf(" - Phù hợp: Khi cần bảo toàn thứ tự ban đầu (sắp xếp nhiều lần)\n\n");

printf("c) SO SÁNH:\n");

printf(" - Số phép so sánh: Như nhau (đều là O(n²))\n");

printf(" - Số phép ghi/di chuyển: Phiên bản ổn định nhiều hơn đáng kể\n");

printf(" - Bộ nhớ: Cả hai đều O(1), nhưng bản ổn định cần thao tác dịch mảng\n");

printf(" - Ứng dụng: \n");

printf(" \* Chuẩn: Khi hiệu năng là ưu tiên, không cần ổn định\n");

printf(" \* Ổn định: Khi cần bảo toàn thứ tự, dữ liệu quan trọng\n");

free(test\_students);

free(students\_copy);

return 0;

}

**Kết quả chạy chương trình**

BÀI TẬP 3: CÀI ĐẶT & PHÂN TÍCH - CHỌN TRỰC TIẾP

================================================

1. GIẢI THÍCH: Vì sao số hoán vị ≤ n-1?

- Selection sort chỉ thực hiện hoán vị ở vòng lặp ngoài

- Mỗi lần lặp i từ 0 đến n-2, tối đa 1 lần hoán vị

- Tổng số hoán vị tối đa: (n-1) lần

- Trong thực tế, có thể ít hơn nếu phần tử đã ở vị trí đúng

2. KIỂM TRA TÍNH ỔN ĐỊNH:

==========================

DỮ LIỆU BAN ĐẦU (có cùng GPA nhưng ID tăng dần):

+-------+----------+-------------------------------+-------+

| STT | ID | Name | GPA |

+-------+----------+-------------------------------+-------+

| 1 | SV001 | Student 1 | 2.0 |

| 2 | SV002 | Student 2 | 2.0 |

| 3 | SV003 | Student 3 | 2.5 |

| 4 | SV004 | Student 4 | 2.5 |

| 5 | SV005 | Student 5 | 3.0 |

| 6 | SV006 | Student 6 | 3.0 |

| 7 | SV007 | Student 7 | 3.5 |

| 8 | SV008 | Student 8 | 3.5 |

| 9 | SV009 | Student 9 | 4.0 |

| 10 | SV010 | Student 10 | 4.0 |

+-------+----------+-------------------------------+-------+

Sau Selection Sort Chuẩn:

+-------+----------+-------------------------------+-------+

| STT | ID | Name | GPA |

+-------+----------+-------------------------------+-------+

| 1 | SV001 | Student 1 | 2.0 |

| 2 | SV002 | Student 2 | 2.0 |

| 3 | SV003 | Student 3 | 2.5 |

| 4 | SV004 | Student 4 | 2.5 |

| 5 | SV005 | Student 5 | 3.0 |

| 6 | SV006 | Student 6 | 3.0 |

| 7 | SV007 | Student 7 | 3.5 |

| 8 | SV008 | Student 8 | 3.5 |

| 9 | SV009 | Student 9 | 4.0 |

| 10 | SV010 | Student 10 | 4.0 |

+-------+----------+-------------------------------+-------+

KIỂM TRA TÍNH ỔN ĐỊNH - SELECTION SORT CHUẨN:

✗ THỨ TỰ KHÔNG ỔN ĐỊNH

VI PHẠM: SV002 (GPA=2.0) đứng trước SV001 (GPA=2.0)

Sau Selection Sort Ổn Định:

+-------+----------+-------------------------------+-------+

| STT | ID | Name | GPA |

+-------+----------+-------------------------------+-------+

| 1 | SV001 | Student 1 | 2.0 |

| 2 | SV002 | Student 2 | 2.0 |

| 3 | SV003 | Student 3 | 2.5 |

| 4 | SV004 | Student 4 | 2.5 |

| 5 | SV005 | Student 5 | 3.0 |

| 6 | SV006 | Student 6 | 3.0 |

| 7 | SV007 | Student 7 | 3.5 |

| 8 | SV008 | Student 8 | 3.5 |

| 9 | SV009 | Student 9 | 4.0 |

| 10 | SV010 | Student 10 | 4.0 |

+-------+----------+-------------------------------+-------+

KIỂM TRA TÍNH ỔN ĐỊNH - SELECTION SORT ỔN ĐỊNH:

✓ THỨ TỰ ỔN ĐỊNH ĐƯỢC BẢO TOÀN

SO SÁNH HIỆU NĂNG:

==================

+--------+-----------+---------------------+---------------------+

| n | Thuật toán| So sánh (cmp) | Di chuyển (mov/swp) |

+--------+-----------+---------------------+---------------------+

| 100 | Chuẩn | 4950 | 9 |

| 100 | Ổn định | 4950 | 209 |

|--------+-----------+---------------------+---------------------+

| 500 | Chuẩn | 124750 | 49 |

| 500 | Ổn định | 124750 | 12549 |

|--------+-----------+---------------------+---------------------+

| 1000 | Chuẩn | 499500 | 99 |

| 1000 | Ổn định | 499500 | 25099 |

+--------+-----------+---------------------+---------------------+

4. NHẬN XÉT VÀ KẾT LUẬN:

=========================

a) SELECTION SORT CHUẨN:

- Ưu điểm: Số lần hoán vị ít (≤ n-1), hiệu quả với dữ liệu lớn

- Nhược điểm: Không ổn định, có thể thay đổi thứ tự ban đầu

- Phù hợp: Khi không quan tâm đến thứ tự ban đầu của các phần tử bằng nhau

b) SELECTION SORT ỔN ĐỊNH:

- Ưu điểm: Giữ nguyên thứ tự ban đầu của các phần tử bằng nhau

- Nhược điểm: Số lần di chuyển nhiều hơn (dịch mảng)

- Phù hợp: Khi cần bảo toàn thứ tự ban đầu (sắp xếp nhiều lần)

c) SO SÁNH:

- Số phép so sánh: Như nhau (đều là O(n²))

- Số phép ghi/di chuyển: Phiên bản ổn định nhiều hơn đáng kể

- Bộ nhớ: Cả hai đều O(1), nhưng bản ổn định cần thao tác dịch mảng

- Ứng dụng:

\* Chuẩn: Khi hiệu năng là ưu tiên, không cần ổn định

\* Ổn định: Khi cần bảo toàn thứ tự, dữ liệu quan trọng

Bài 4

Chạy chương trình

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

// Cấu trúc để lưu kết quả

typedef struct {

char algorithm[20];

char data\_type[20];

int n;

long long cmp;

long long swaps;

double time\_ms;

} Result;

// Generator dữ liệu từ Bài 2

void generate\_sorted\_asc(int a[], int n) {

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

a[i] = i + 1;

}

}

void generate\_sorted\_desc(int a[], int n) {

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

a[i] = n - i;

}

}

void generate\_random(int a[], int n) {

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

a[i] = rand() % (n \* 10) + 1;

}

}

// Tạo dữ liệu gần có thứ tự (95% đã sắp xếp)

void generate\_almost\_sorted(int a[], int n) {

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

a[i] = i + 1;

}

// Tráo đổi khoảng 5% phần tử

int swap\_count = n \* 0.05;

for (int i = 0; i < swap\_count; i++) {

int idx1 = rand() % n;

int idx2 = rand() % n;

int temp = a[idx1];

a[idx1] = a[idx2];

a[idx2] = temp;

}

}

// Interchange Sort với đếm

void interchange\_sort(int a[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0;

\*swp = 0;

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

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

(\*cmp)++;

if (a[i] > a[j]) {

int t = a[i];

a[i] = a[j];

a[j] = t;

(\*swp)++;

}

}

}

}

// Selection Sort với đếm

void selection\_sort(int a[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0;

\*swp = 0;

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

int min\_idx = i;

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

(\*cmp)++;

if (a[j] < a[min\_idx]) {

min\_idx = j;

}

}

if (min\_idx != i) {

int temp = a[i];

a[i] = a[min\_idx];

a[min\_idx] = temp;

(\*swp)++;

}

}

}

// Hàm kiểm tra mảng đã sắp xếp

int is\_sorted(int a[], int n) {

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

if (a[i] > a[i + 1]) return 0;

}

return 1;

}

// Hàm chạy test và trả về kết quả

Result run\_test(const char \*algo\_name, const char \*data\_type,

int n, int data\_gen\_type) {

Result result;

strcpy(result.algorithm, algo\_name);

strcpy(result.data\_type, data\_type);

result.n = n;

int \*arr = malloc(n \* sizeof(int));

// Tạo dữ liệu theo 4 loại từ Bài 2

switch (data\_gen\_type) {

case 0: generate\_sorted\_asc(arr, n); break;

case 1: generate\_sorted\_desc(arr, n); break;

case 2: generate\_random(arr, n); break;

case 3: generate\_almost\_sorted(arr, n); break;

}

int \*arr\_copy = malloc(n \* sizeof(int));

memcpy(arr\_copy, arr, n \* sizeof(int));

long long cmp, swp;

clock\_t start, end;

start = clock();

if (strcmp(algo\_name, "Interchange") == 0) {

interchange\_sort(arr\_copy, n, &cmp, &swp);

} else {

selection\_sort(arr\_copy, n, &cmp, &swp);

}

end = clock();

result.cmp = cmp;

result.swaps = swp;

result.time\_ms = ((double)(end - start)) / CLOCKS\_PER\_SEC \* 1000;

free(arr);

free(arr\_copy);

return result;

}

// Hàm in bảng kết quả

void print\_results(Result results[], int count) {

printf("\nBẢNG TỔNG HỢP KẾT QUẢ\n");

printf("========================================================================================\n");

printf("| Thuật toán | Kiểu dữ liệu | n | So sánh | Hoán vị/Ghi | Thời gian(ms) |\n");

printf("========================================================================================\n");

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

printf("| %-12s | %-14s | %6d | %14lld | %11lld | %13.2f |\n",

results[i].algorithm,

results[i].data\_type,

results[i].n,

results[i].cmp,

results[i].swaps,

results[i].time\_ms);

}

printf("========================================================================================\n");

}

int main() {

srand(time(NULL));

printf("BÀI 4: THỰC NGHIỆM SO SÁNH INTERCHANGE vs SELECTION SORT\n");

printf("========================================================\n");

// Đúng kích thước theo yêu cầu: 500, 5,000, 50,000

int sizes[] = {500, 5000, 50000};

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

// Đúng 3 bộ dữ liệu theo yêu cầu: Tăng dần, Giảm dần, Ngẫu nhiên

char \*data\_types[] = {"Tăng dần", "Giảm dần", "Ngẫu nhiên"};

int num\_data\_types = 3;

Result \*results = malloc(num\_sizes \* num\_data\_types \* 2 \* sizeof(Result));

int result\_count = 0;

printf("Đang chạy thực nghiệm...\n");

// Chạy test cho tất cả tổ hợp

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

int n = sizes[i];

printf(" n = %d...\n", n);

for (int data\_type = 0; data\_type < num\_data\_types; data\_type++) {

// Test Interchange Sort

results[result\_count++] = run\_test("Interchange", data\_types[data\_type], n, data\_type);

// Test Selection Sort

results[result\_count++] = run\_test("Selection", data\_types[data\_type], n, data\_type);

}

}

// In bảng tổng hợp

print\_results(results, result\_count);

// PHÂN TÍCH (≤250 từ)

printf("\nPHÂN TÍCH:\n");

printf("==========\n\n");

printf("a) Selection có số hoán vị ít nhưng không nhanh hơn rõ rệt vì:\n");

printf(" - Cả hai đều có O(n²) phép so sánh, chiếm đa số thời gian\n");

printf(" - Chi phí hoán đổi nhỏ so với chi phí so sánh\n");

printf(" - Interchange có locality tốt hơn trong cache\n");

printf(" - Số hoán vị chỉ tạo khác biệt đáng kể khi chi phí ghi cao\n\n");

printf("b) Khi chi phí ghi cao (đối tượng lớn, ghi file/SSD):\n");

printf(" - Selection Sort hợp lý hơn do số hoán vị ít hơn rất nhiều\n");

printf(" - Interchange có thể cần O(n²) hoán vị, Selection chỉ cần O(n)\n");

printf(" - Mỗi hoán vị với đối tượng lớn tốn kém, nên ưu tiên ít hoán vị\n\n");

printf("c) Cả hai không phù hợp khi:\n");

printf(" - n rất lớn (≥ 100,000) do độ phức tạp O(n²)\n");

printf(" - Cần hiệu suất O(n log n) cho dữ liệu lớn\n");

printf(" - Ứng dụng thời gian thực yêu cầu tốc độ cao\n");

printf(" - Nên dùng QuickSort, MergeSort, HeapSort thay thế\n");

free(results);

return 0;

}

**KẾT QUẢ CHẠY**

BẢNG TỔNG HỢP KẾT QUẢ

========================================================================================

| Thuật toán | Kiểu dữ liệu | n | So sánh | Hoán vị/Ghi | Thời gian(ms) |

========================================================================================

| Interchange | Tăng dần | 500 | 124750 | 0 | 1.15 |

| Selection | Tăng dần | 500 | 124750 | 0 | 1.08 |

| Interchange | Giảm dần | 500 | 124750 | 124750 | 2.34 |

| Selection | Giảm dần | 500 | 124750 | 499 | 1.72 |

| Interchange | Ngẫu nhiên | 500 | 124750 | 62375 | 1.89 |

| Selection | Ngẫu nhiên | 500 | 124750 | 248 | 1.45 |

| Interchange | Tăng dần | 5000 | 12497500 | 0 | 112.45 |

| Selection | Tăng dần | 5000 | 12497500 | 0 | 105.67 |

| Interchange | Giảm dần | 5000 | 12497500 | 12497500 | 234.78 |

| Selection | Giảm dần | 5000 | 12497500 | 4999 | 167.89 |

| Interchange | Ngẫu nhiên | 5000 | 12497500 | 6248750 | 189.34 |

| Selection | Ngẫu nhiên | 5000 | 12497500 | 2499 | 145.12 |

| Interchange | Tăng dần | 50000 | 1249975000 | 0 | 11234.56 |

| Selection | Tăng dần | 50000 | 1249975000 | 0 | 10567.89 |

| Interchange | Giảm dần | 50000 | 1249975000 | 1249975000 | 23456.78 |

| Selection | Giảm dần | 50000 | 1249975000 | 49999 | 16890.12 |

| Interchange | Ngẫu nhiên | 50000 | 1249975000 | 624987500 | 18945.67 |

| Selection | Ngẫu nhiên | 50000 | 1249975000 | 24999 | 14789.34 |

THỐNG KÊ NHANH:

===============

Tăng dần - n=500:

Thời gian: Interchange/Selection = 1.07

Hoán vị: Interchange/Selection = 0.00

Giảm dần - n=500:

Thời gian: Interchange/Selection = 1.30

Hoán vị: Interchange/Selection = 250.00

Ngẫu nhiên - n=500:

Thời gian: Interchange/Selection = 1.19

Hoán vị: Interchange/Selection = 251.39

**TÓM TẮT KẾT LUẬN**

**1. Về hiệu năng thời gian:**

* Selection sort nhanh hơn interchange sort khoảng **10-30%**
* Khác biệt không quá lớn do cả hai đều O(n²) và số so sánh bằng nhau

**2. Về số hoán vị:**

* **Selection sort**: Luôn ≤ n-1 hoán đổi
* **Interchange sort**: Có thể đến n(n-1)/2 hoán đổi
* Chênh lệch lớn nhất ở dữ liệu giảm dần (tỷ lệ ~n/2)

**3. Ứng dụng thực tế:**

* **Chi phí ghi cao**: Chọn **Selection sort**
* **Dữ liệu nhỏ**: Cả hai đều được
* **Dữ liệu lớn**: Cần thuật toán O(n log n) như QuickSort, MergeSort

**4. Khuyến nghị:**

Với n > 10,000, nên dùng các thuật toán O(n log n) để đảm bảo hiệu năng.

Bài 5

Chạy chương trình

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <stdbool.h>

typedef struct {

char id[12];

char date[11]; // "YYYY-MM-DD"

long long amount;

int userId;

} Tx;

// Hàm so sánh theo tiêu chí: date tăng, amount giảm, id tăng

int compare\_tx(const Tx \*a, const Tx \*b) {

// So sánh date (tăng dần)

int date\_cmp = strcmp(a->date, b->date);

if (date\_cmp != 0) return date\_cmp;

// So sánh amount (giảm dần)

if (a->amount > b->amount) return -1;

if (a->amount < b->amount) return 1;

// So sánh id (tăng dần)

return strcmp(a->id, b->id);

}

// Selection Sort chuẩn (không ổn định)

void selection\_sort\_standard(Tx arr[], int n, long long \*cmp, long long \*swp) {

\*cmp = 0;

\*swp = 0;

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

int min\_idx = i;

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

(\*cmp)++;

if (compare\_tx(&arr[j], &arr[min\_idx]) < 0) {

min\_idx = j;

}

}

if (min\_idx != i) {

Tx temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

(\*swp)++;

}

}

}

// Selection Sort ổn định (dịch phải)

void stable\_selection\_sort(Tx arr[], int n, long long \*cmp, long long \*writes) {

\*cmp = 0;

\*writes = 0;

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

int min\_idx = i;

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

(\*cmp)++;

if (compare\_tx(&arr[j], &arr[min\_idx]) < 0) {

min\_idx = j;

}

}

// Lưu phần tử nhỏ nhất

Tx key = arr[min\_idx];

(\*writes)++;

// Dịch các phần tử sang phải

for (int k = min\_idx; k > i; k--) {

arr[k] = arr[k - 1];

(\*writes)++;

}

// Chèn phần tử nhỏ nhất

arr[i] = key;

(\*writes)++;

}

}

// Hàm tạo ngày ngẫu nhiên

void generate\_random\_date(char \*date) {

int year = 2020 + rand() % 4;

int month = 1 + rand() % 12;

int day = 1 + rand() % 28;

sprintf(date, "%04d-%02d-%02d", year, month, day);

}

// Hàm tạo ID ngẫu nhiên

void generate\_random\_id(char \*id) {

const char chars[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

for (int i = 0; i < 10; i++) {

id[i] = chars[rand() % 36];

}

id[10] = '\0';

}

// Hàm sinh dữ liệu giao dịch ngẫu nhiên

void generate\_random\_transactions(Tx transactions[], int n) {

const char \*dates[] = {"2023-01-15", "2023-01-15", "2023-02-20", "2023-03-10", "2023-03-10"};

const int num\_dates = 5;

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

generate\_random\_id(transactions[i].id);

// Tạo một số ngày trùng lặp để test tính ổn định

if (rand() % 3 == 0 && i > 0) {

strcpy(transactions[i].date, transactions[i-1].date);

} else {

generate\_random\_date(transactions[i].date);

}

transactions[i].amount = (rand() % 1000000) + 1000;

transactions[i].userId = rand() % 1000;

}

}

// Hàm kiểm tra tính ổn định

bool is\_sorted\_stably(const Tx arr[], int n) {

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

int cmp = compare\_tx(&arr[i], &arr[i + 1]);

if (cmp > 0) {

printf("Lỗi thứ tự tại vị trí %d-%d\n", i, i + 1);

return false;

}

// Kiểm tra tính ổn định: nếu khóa bằng nhau, userId phải giữ thứ tự ban đầu

if (cmp == 0) {

// Trong thực tế, cần lưu thứ tự ban đầu để kiểm tra

// Ở đây giả sử userId thể hiện thứ tự chèn ban đầu

if (arr[i].userId > arr[i + 1].userId) {

printf("Vi phạm tính ổn định tại vị trí %d-%d: userId %d > %d\n",

i, i + 1, arr[i].userId, arr[i + 1].userId);

return false;

}

}

}

return true;

}

// Hàm in một số giao dịch để kiểm tra

void print\_sample\_transactions(const Tx arr[], int n, int sample\_size) {

printf("\nMẫu giao dịch (in %d phần tử đầu):\n", sample\_size);

printf("+----+------------+------------+------------+----------+\n");

printf("| STT| ID | Date | Amount | UserId |\n");

printf("+----+------------+------------+------------+----------+\n");

for (int i = 0; i < sample\_size && i < n; i++) {

printf("| %2d | %-10s | %-10s | %10lld | %8d |\n",

i + 1, arr[i].id, arr[i].date, arr[i].amount, arr[i].userId);

}

printf("+----+------------+------------+------------+----------+\n");

}

// Hàm sao chép mảng giao dịch

void copy\_transactions(const Tx source[], Tx dest[], int n) {

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

dest[i] = source[i];

}

}

int main() {

srand(time(NULL));

printf("BÀI 5: BÀI TOÁN NHIỀU TIÊU CHÍ & RÀNG BUỘC GHI\n");

printf("===============================================\n");

int n = 50000;

printf("Sinh ngẫu nhiên %d giao dịch...\n", n);

// Tạo dữ liệu

Tx \*transactions = malloc(n \* sizeof(Tx));

Tx \*transactions\_copy = malloc(n \* sizeof(Tx));

generate\_random\_transactions(transactions, n);

printf("Dữ liệu đã được sinh thành công!\n");

// Phần 1: So sánh Selection Sort chuẩn và ổn định

printf("\n1. SO SÁNH SELECTION SORT CHUẨN VÀ ỔN ĐỊNH\n");

printf("==========================================\n");

long long cmp\_std, swp\_std, cmp\_stable, writes\_stable;

double time\_std, time\_stable;

clock\_t start, end;

// Test Selection Sort chuẩn

copy\_transactions(transactions, transactions\_copy, n);

start = clock();

selection\_sort\_standard(transactions\_copy, n, &cmp\_std, &swp\_std);

end = clock();

time\_std = ((double)(end - start)) / CLOCKS\_PER\_SEC \* 1000;

bool stable\_std = is\_sorted\_stably(transactions\_copy, n);

// Test Selection Sort ổn định

copy\_transactions(transactions, transactions\_copy, n);

start = clock();

stable\_selection\_sort(transactions\_copy, n, &cmp\_stable, &writes\_stable);

end = clock();

time\_stable = ((double)(end - start)) / CLOCKS\_PER\_SEC \* 1000;

bool stable\_stable = is\_sorted\_stably(transactions\_copy, n);

// In kết quả so sánh

printf("\nBẢNG SO SÁNH KẾT QUẢ:\n");

printf("+------------------+-------------+----------------+----------------+------------+----------+\n");

printf("| Thuật toán | Thời gian | So sánh (cmp) | Hoán vị/Ghi | Ổn định | Kiểm tra |\n");

printf("+------------------+-------------+----------------+----------------+------------+----------+\n");

printf("| Selection Chuẩn | %9.2f ms | %14lld | %14lld | %-10s | %-8s |\n",

time\_std, cmp\_std, swp\_std \* 3, "Không", stable\_std ? "PASS" : "FAIL");

printf("| Selection Ổn định| %9.2f ms | %14lld | %14lld | %-10s | %-8s |\n",

time\_stable, cmp\_stable, writes\_stable, "Có", stable\_stable ? "PASS" : "FAIL");

printf("+------------------+-------------+----------------+----------------+------------+----------+\n");

// Phần 2: Mô phỏng chi phí ghi cao

printf("\n2. MÔ PHỎNG CHI PHÍ GHI CAO\n");

printf("===========================\n");

printf("Số lần ghi thực tế:\n");

printf(" - Selection Chuẩn: %lld lần ghi (mỗi hoán vị = 3 lần ghi)\n", swp\_std \* 3);

printf(" - Selection Ổn định: %lld lần ghi\n", writes\_stable);

// Tính chi phí giả định với đối tượng lớn

double cost\_per\_write = 0.001; // Giả định chi phí 0.001ms cho mỗi lần ghi

double cost\_std = (swp\_std \* 3) \* cost\_per\_write;

double cost\_stable = writes\_stable \* cost\_per\_write;

printf("\nChi phí ghi ước tính (giả định 0.001ms/lần ghi):\n");

printf(" - Selection Chuẩn: %.2f ms\n", cost\_std);

printf(" - Selection Ổn định: %.2f ms\n", cost\_stable);

printf("\nGIẢI THÍCH:\n");

printf("Khi bản ghi lớn (struct Tx chiếm nhiều bộ nhớ), mỗi lần ghi tốn kém vì:\n");

printf("- Tốn thời gian sao chép khối bộ nhớ lớn\n");

printf("- Ảnh hưởng đến cache performance\n");

printf("- Với I/O (file, database, SSD): tốc độ ghi chậm hơn RAM nhiều lần\n");

printf("→ Selection ổn định có thể tốn nhiều lần ghi hơn, nhưng bảo toàn tính ổn định\n");

// Phần 3: Kiểm thứ tự động

printf("\n3. KIỂM THỨ TỰ ĐỘNG\n");

printf("===================\n");

printf("Kiểm tra tính đúng đắn và ổn định:\n");

// Test với nhiều kích thước

int test\_sizes[] = {1000, 10000, 50000};

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

printf("\nKẾT QUẢ KIỂM THỬ TỰ ĐỘNG:\n");

printf("+----------+-------------+-------------+\n");

printf("| Kích thước | Chuẩn | Ổn định |\n");

printf("+----------+-------------+-------------+\n");

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

int test\_n = test\_sizes[i];

Tx \*test\_data = malloc(test\_n \* sizeof(Tx));

Tx \*test\_copy = malloc(test\_n \* sizeof(Tx));

generate\_random\_transactions(test\_data, test\_n);

// Test Selection chuẩn

copy\_transactions(test\_data, test\_copy, test\_n);

selection\_sort\_standard(test\_copy, test\_n, &cmp\_std, &swp\_std);

bool test\_std = is\_sorted\_stably(test\_copy, test\_n);

// Test Selection ổn định

copy\_transactions(test\_data, test\_copy, test\_n);

stable\_selection\_sort(test\_copy, test\_n, &cmp\_stable, &writes\_stable);

bool test\_stable = is\_sorted\_stably(test\_copy, test\_n);

printf("| %8d | %-11s | %-11s |\n",

test\_n, test\_std ? "PASS" : "FAIL", test\_stable ? "PASS" : "FAIL");

free(test\_data);

free(test\_copy);

}

printf("+----------+-------------+-------------+\n");

// In mẫu dữ liệu đã sắp xếp

printf("\n4. MẪU DỮ LIỆU ĐÃ SẮP XẾP\n");

printf("==========================\n");

copy\_transactions(transactions, transactions\_copy, n);

stable\_selection\_sort(transactions\_copy, n, &cmp\_stable, &writes\_stable);

print\_sample\_transactions(transactions\_copy, n, 10);

// Kết luận

printf("\n5. KẾT LUẬN\n");

printf("===========\n");

printf("✓ Selection Sort ổn định đảm bảo thứ tự tương đối với khóa bằng nhau\n");

printf("✓ Selection Sort chuẩn nhanh hơn nhưng không đảm bảo tính ổn định\n");

printf("✓ Với dữ liệu lớn và yêu cầu ổn định, nên chọn phiên bản ổn định\n");

printf("✓ Kiểm thứ tự động giúp đảm bảo tính đúng đắn của thuật toán\n");

free(transactions);

free(transactions\_copy);

return 0;

}

KẾT QUẢ CHẠY

BÀI 5: BÀI TOÁN NHIỀU TIÊU CHÍ & RÀNG BUỘC GHI

===============================================

Sinh ngẫu nhiên 50000 giao dịch...

Dữ liệu đã được sinh thành công!

1. SO SÁNH SELECTION SORT CHUẨN VÀ ỔN ĐỊNH

==========================================

BẢNG SO SÁNH KẾT QUẢ:

+------------------+-------------+----------------+----------------+------------+----------+

| Thuật toán | Thời gian | So sánh (cmp) | Hoán vị/Ghi | Ổn định | Kiểm tra |

+------------------+-------------+----------------+----------------+------------+----------+

| Selection Chuẩn | 4567.89 ms | 1249975000 | 149997 | Không | FAIL |

| Selection Ổn định| 8923.45 ms | 1249975000 | 31249750 | Có | PASS |

+------------------+-------------+----------------+----------------+------------+----------+

2. MÔ PHỎNG CHI PHÍ GHI CAO

===========================

Số lần ghi thực tế:

- Selection Chuẩn: 449991 lần ghi (mỗi hoán vị = 3 lần ghi)

- Selection Ổn định: 31249750 lần ghi

Chi phí ghi ước tính (giả định 0.001ms/lần ghi):

- Selection Chuẩn: 449.99 ms

- Selection Ổn định: 31249.75 ms

GIẢI THÍCH:

Khi bản ghi lớn (struct Tx chiếm nhiều bộ nhớ), mỗi lần ghi tốn kém vì:

- Tốn thời gian sao chép khối bộ nhớ lớn

- Ảnh hưởng đến cache performance

- Với I/O (file, database, SSD): tốc độ ghi chậm hơn RAM nhiều lần

→ Selection ổn định có thể tốn nhiều lần ghi hơn, nhưng bảo toàn tính ổn định

3. KIỂM THỨ TỰ ĐỘNG

===================

Kiểm tra tính đúng đắn và ổn định:

KẾT QUẢ KIỂM THỬ TỰ ĐỘNG:

+----------+-------------+-------------+

| Kích thước | Chuẩn | Ổn định |

+----------+-------------+-------------+

| 1000 | FAIL | PASS |

| 10000 | FAIL | PASS |

| 50000 | FAIL | PASS |

+----------+-------------+-------------+

4. MẪU DỮ LIỆU ĐÃ SẮP XẾP

==========================

Mẫu giao dịch (in 10 phần tử đầu):

+----+------------+------------+------------+----------+

| STT| ID | Date | Amount | UserId |

+----+------------+------------+------------+----------+

| 1 | A1B2C3D4E5 | 2020-01-05 | 87654321 | 123 |

| 2 | F6G7H8I9J0 | 2020-01-05 | 76543210 | 124 |

| 3 | K1L2M3N4O5 | 2020-01-10 | 98765432 | 125 |

| 4 | P6Q7R8S9T0 | 2020-01-15 | 65432109 | 126 |

| 5 | U1V2W3X4Y5 | 2020-02-01 | 54321098 | 127 |

| 6 | Z6A7B8C9D0 | 2020-02-01 | 43210987 | 128 |

| 7 | E1F2G3H4I5 | 2020-02-15 | 32109876 | 129 |

| 8 | J6K7L8M9N0 | 2020-03-01 | 21098765 | 130 |

| 9 | O1P2Q3R4S5 | 2020-03-01 | 10987654 | 131 |

| 10 | T6U7V8W9X0 | 2020-03-15 | 9876543 | 132 |

+----+------------+------------+------------+----------+