**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**BÀI TẬP NHÓM**

A blue circle with white text

AI-generated content may be incorrect.

**Môn: Cấu trúc dữ liệu và giải thuật**

**Sinh viên thực hiện:**

**Giảng Văn Hiển – 3123411091**

**NGUYỄN VĂN AN - 3123411007**

Phân công:

Giảng Văn Hiển: trình bày word, bài tập ứng dụng

Nguyễn Văn An: Bài tập cơ sở

Mục lục

[A. Câu hỏi 3](#_Toc192883446)

[1. Trình bày tư tưởng của các thuật toán sắp xếp? 3](#_Toc192883447)

[2. Trong các thuật toán sắp xếp, bạn thích nhất thuật toán nào? Thuật toán nào bạn không thích nhất? Tại sao? 4](#_Toc192883448)

[3. Trình bày và cài đặt tất cả các thuật toán sắp xếp nội, ngoại theo thứ tự giảm dần. Cho nhận xét về các thuật toán này. 4](#_Toc192883449)

[a) Bubble Sort (Sắp xếp nổi bọt) 4](#_Toc192883450)

[b) Selection Sort (Sắp xếp chọn) 5](#_Toc192883451)

[c) Insertion Sort (Sắp xếp chèn) 6](#_Toc192883452)

[d) Merge Sort (Sắp xếp trộn) 7](#_Toc192883453)

[e) Quick Sort (Sắp xếp nhanh) 8](#_Toc192883454)

[f) Heap Sort: 9](#_Toc192883455)

[4. Ưu điểm và nhược điểm của mỗi thuật toán sắp xếp? Cách khắc phục nhược điểm? 11](#_Toc192883456)

[B. Bài tập cơ sở 11](#_Toc192883457)

[bài 1: 11](#_Toc192883458)

[bài 2: 12](#_Toc192883459)

[bài 3: 13](#_Toc192883460)

[C. Bài tập ứng dụng 13](#_Toc192883461)

# Câu hỏi

## Trình bày tư tưởng của các thuật toán sắp xếp?

Thuật toán sắp xếp là các phương pháp sắp xếp một danh sách các phần tử theo thứ tự tăng dần hoặc giảm dần. Có hai loại thuật toán sắp xếp chính:

* Sắp xếp nội (Internal Sorting): Sắp xếp trực tiếp trên bộ nhớ chính (RAM).
* Sắp xếp ngoại (External Sorting): Sắp xếp với dữ liệu quá lớn, cần dùng bộ nhớ ngoài (HDD, SSD).

Các thuật toán sắp xếp phổ biến

* Sắp xếp đổi chỗ (Exchange Sorts):
  + Bubble Sort (Sắp xếp nổi bọt)
  + Quick Sort (Sắp xếp nhanh)
* Sắp xếp chọn lọc (Selection Sorts):
  + Selection Sort (Sắp xếp chọn)
  + Heap Sort (Sắp xếp vun đống)
* Sắp xếp chèn (Insertion Sorts):
  + Insertion Sort (Sắp xếp chèn)
  + Shell Sort
* Sắp xếp theo chia để trị:
  + Merge Sort (Sắp xếp trộn)
  + Quick Sort

Mỗi thuật toán có cách hoạt động khác nhau, dựa trên các nguyên tắc như đổi chỗ, chọn lọc, chèn, hoặc chia để trị.

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

* Thuật toán yêu thích: Quick Sort
  + Có độ phức tạp trung bình là O(n log n), nhanh hơn nhiều thuật toán khác.
  + Hoạt động tốt trên nhiều loại dữ liệu khác nhau.
  + Sử dụng kỹ thuật chia để trị, giúp tối ưu thời gian chạy.
* Thuật toán ít yêu thích: Bubble Sort
  + Có độ phức tạp O(n²) trong trường hợp xấu nhất, rất chậm khi số lượng phần tử lớn.
  + Cần nhiều phép hoán đổi, không hiệu quả.
  + Ít được sử dụng trong thực tế.

## 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.

### Bubble Sort (Sắp xếp nổi bọt)

#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]);

}

}

}

}

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

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

cout << endl;

}

int main() {

int arr[] = {5, 3, 8, 6, 2, 7};

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

bubbleSort(arr, n);

printArray(arr, n);

return 0;

}

Nhận xét: Đơn giản nhưng chậm, không hiệu quả với dữ liệu lớn.

### Selection Sort (Sắp xếp chọn)

#include <iostream>

using namespace std;

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

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

int maxIdx = i;

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

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

maxIdx = j;

}

}

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

}

}

int main() {

int arr[] = {10, 3, 2, 8, 7, 6};

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

selectionSort(arr, n);

printArray(arr, n);

return 0;

}

Nhận xét: Ít hoán đổi nhưng vẫn có O(n²), không tối ưu với dữ liệu lớn.

### Insertion Sort (Sắp xếp chèn)

#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[] = {12, 11, 13, 5, 6};

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

insertionSort(arr, n);

printArray(arr, n);

return 0;

}

Nhận xét: Tốt với dữ liệu gần sắp xếp, nhưng vẫn chậm với dữ liệu lớn.

### Merge Sort (Sắp xếp trộn)

#include <iostream>

using namespace std;

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

int n1 = mid - left + 1, n2 = right - mid;

int L[n1], R[n2];

for (int i = 0; i < n1; i++) L[i] = arr[left + i];

for (int i = 0; i < n2; i++) R[i] = arr[mid + 1 + i];

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

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

if (L[i] >= R[j]) arr[k++] = L[i++];

else arr[k++] = R[j++];

}

while (i < n1) arr[k++] = L[i++];

while (j < n2) arr[k++] = R[j++];

}

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

if (left < right) {

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

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

int main() {

int arr[] = {38, 27, 43, 3, 9, 82, 10};

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

mergeSort(arr, 0, n - 1);

printArray(arr, n);

return 0;

}

Nhận xét: O(n log n), ổn định, nhưng dùng đệ quy nên tốn bộ nhớ.

### Quick Sort (Sắp xếp nhanh)

#include <iostream>

using namespace std;

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) { // Đổi dấu để sắp xếp giảm dần

i++;

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

}

}

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);

}

}

int main() {

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

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

quickSort(arr, 0, n - 1);

printArray(arr, n);

return 0;

}

Nhận xét: Rất nhanh với O(n log n), nhưng có thể chậm nếu chọn pivot không tốt.

### Heap Sort:

#include <iostream>

using namespace std;

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

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

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

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

if (largest != i) {

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

heapify(arr, n, largest);

}

}

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

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

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

swap(arr[0], arr[i]);

heapify(arr, i, 0);

}

}

int main() {

int arr[] = {4, 10, 3, 5, 1};

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

heapSort(arr, n);

printArray(arr, n);

return 0;

}

Nhận xét: O(n log n), không ổn định nhưng ít tốn bộ nhớ.

## Ưu điểm và nhược điểm của mỗi thuật toán sắp xếp? Cách khắc phục nhược điểm?

| **Thuật toán** | **Ưu điểm** | **Nhược điểm** | **Cách khắc phục** |
| --- | --- | --- | --- |
| **Bubble Sort** | Đơn giản, dễ hiểu | Chậm (O(n²)), nhiều phép hoán đổi | Tối ưu bằng biến kiểm tra (Flag Optimization): Nếu trong một lần lặp không có hoán đổi nào, nghĩa là mảng đã được sắp xếp → có thể dừng sớm. |
| **Selection Sort** | Ít hoán đổi | Vẫn có O(n²), không hiệu quả với dữ liệu lớn | Dùng Selection Sort Đệ Quy để tận dụng bộ nhớ đệ quy thay vì vòng lặp. |
| **Insertion Sort** | Tốt cho dữ liệu gần sắp xếp | Chậm trên dữ liệu lớn (O(n²)) | Dùng Tìm Kiếm Nhị Phân để tìm vị trí chèn nhanh hơn.  Thay vì dịch từng phần tử một, có thể di chuyển theo khối để giảm số lần ghi vào bộ nhớ. |
| **Merge Sort** | O(n log n), ổn định | Tốn bộ nhớ do dùng đệ quy | Dùng Merge Sort Không Dùng Bộ Nhớ Phụ (In-place Merge Sort) bằng cách gộp hai phần của mảng ngay trong chính nó. |
| **Quick Sort** | Rất nhanh (O(n log n)) | Trường hợp xấu có thể thành O(n²) | Chọn pivot thông minh hơn (Median-of-Three Pivoting): Chọn trung vị của phần tử đầu, giữa và cuối làm pivot.  Dùng Quick Sort Không Đệ Quy (Iterative Quick Sort) để tránh tràn stack. |
| **Heap Sort** | O(n log n), tốt trên dữ liệu lớn | Không ổn định, phức tạp hơn Merge Sort | Tối ưu Heapify bằng Floyd’s Algorithm, giúp giảm số lần hoán đổi. |

# Bài tập cơ sở

## bài 1:

c.

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

Duyệt từng phần tử trong mảng.

So sánh nó với tất cả phần tử phía sau và đổi chỗ nếu cần.

Vòng lặp ngoài chạy lần.

Vòng lặp trong chạy lần.

Tổng số phép so sánh:

Trường hợp xấu nhất: Dãy giảm dần, cần đổi chỗ nhiều nhất.

Trường hợp tốt nhất: Dãy đã sắp xếp, không cần đổi chỗ, nhưng vẫn có so sánh.

2. Sắp xếp chọn trực tiếp (Selection Sort)

Mỗi lần duyệt tìm phần tử nhỏ nhất trong phần chưa sắp xếp và đổi với vị trí đầu tiên của phần đó.

Vòng lặp ngoài chạy n-1 lần

Mỗi lần tìm phần tử nhỏ nhất chỉ đổi chỗ đúng 1 lần, nên số lần đổi chỗ là 0(n)

3. Sắp xếp chèn trực tiếp (Insertion Sort)

Chèn từng phần tử vào đúng vị trí trong phần đã sắp xếp.

Nếu dãy ban đầu đã sắp xếp, chỉ cần 0(n)so ánh.

Nếu dãy ngược chiều, mỗi phần tử phải đẩy toàn bộ phần đã sắp xếp ra sau → số lần dịch chuyển tối đa

4. Sắp xếp nổi bọt (Bubble Sort)

Liên tục hoán đổi các phần tử nếu chúng không đúng thứ tự

Luôn cần 0(n^2) so sánh, nhưng nếu có tối ưu thì có thể dừng sớm khi không có đổi chỗ.

## bài 2:

a. Mô phỏng các bước sắp xếp

**1. Quick Sort (Sắp xếp nhanh)**  
 Dãy ban đầu: [8, 5, 1, 3, 6, 9, 12, 4, 7, 10]

* Bước 1: Chọn pivot = 10, chia thành hai phần:  
   [8, 5, 1, 3, 6, 9, 4, 7] (bé hơn pivot) và [12] (lớn hơn pivot)
* Bước 2: Đệ quy tiếp tục cho hai phần trên.
* Quá trình tiếp diễn cho đến khi mảng được sắp xếp.

**2. Merge Sort (Sắp xếp trộn trực tiếp)**  
 Dãy ban đầu: [8, 5, 1, 3, 6, 9, 12, 4, 7, 10]

* Chia đôi dãy thành: [8, 5, 1, 3, 6] và [9, 12, 4, 7, 10]
* Tiếp tục chia nhỏ đến khi mỗi phần tử đứng riêng lẻ.
* Ghép các dãy nhỏ lại theo thứ tự tăng dần.

**3. Heap Sort (Sắp xếp cây)**  
 Dãy ban đầu: [8, 5, 1, 3, 6, 9, 12, 4, 7, 10]

* Xây dựng cây heap.
* Lần lượt đưa phần tử lớn nhất lên đầu, hoán đổi và tiếp tục xây dựng heap.

c)

Quick Sort:

Trường hợp tốt nhất và trung bình:

O(nlogn)

Trường hợp xấu nhất (nếu chọn pivot kém):

O(n^2)

Merge Sort:

Độ phức tạp:

O(nlogn)

Luôn ổn định nhưng tốn bộ nhớ do sử dụng mảng tạm.

Heap Sort:

Độ phức tạp:

O(nlogn)

Ít ổn định hơn nhưng dùng ít bộ nhớ hơn Merge Sort.

## bài 3:

-nhận xét từng thuật toán sắp xếp:

* Quick Sort thường chạy nhanh nhất nhưng không ổn định nếu dữ liệu gần như đã sắp xếp.
* Merge Sort có độ phức tạp ổn định O(n log n) nhưng cần bộ nhớ phụ thêm.
* Heap Sort có hiệu năng khá tốt nhưng thường chậm hơn Quick Sort trong thực tế.

- Để biết thời gian thực hiện các thuật toán ta cần:

Tạo mảng chung để đảm bảo tất cả thuật toán có cùng dữ liệu.

Sao chép mảng trước khi chạy mỗi thuật toán.

Đếm số lần so sánh và đổi chỗ trong thuật toán.

Đo thời gian chạy bằng <chrono>.

# Bài tập ứng dụng

FILE CODE ĐÍNH KÈM