# Trình Bày Và Cài Đặt Tất Cả Thuật Toán Sắp Xếp Nội, Ngoại

## I. Thuật toán sắp xếp nội

### Bubble Sort

Ý tưởng: So sánh từng cặp phần tử liền kề và đổi chỗ nếu không đúng thứ tự. Lặp lại cho đến khi mảng được sắp xếp hoàn toàn.

Code:

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]) {  
 int temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 }  
 }  
 }  
}

Nhận xét: Đơn giản nhưng chậm, độ phức tạp O(n²). Không phù hợp cho dữ liệu lớn.

### Selection Sort

Ý tưởng: Tìm phần tử lớn nhất và đưa về đầu mảng. Lặp lại cho phần còn lại của mảng.

Code:

void selectionSort(int arr[], int n) {  
 for (int i = 0; i < n; i++) {  
 int maxIndex = i;  
 for (int j = i + 1; j < n; j++) {  
 if (arr[j] > arr[maxIndex]) {  
 maxIndex = j;  
 }  
 }  
 int temp = arr[i];  
 arr[i] = arr[maxIndex];  
 arr[maxIndex] = temp;  
 }  
}

Nhận xét: Nhận xét: Đơn giản, ít tốn bộ nhớ nhưng hiệu suất kém với dữ liệu lớn.

### Insertion Sort

Ý tưởng: Chèn từng phần tử vào đúng vị trí trong danh sách con đã sắp xếp.

Code:

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;  
 }  
}

Nhận xét: Hiệu quả với dữ liệu gần như đã sắp xếp, độ phức tạp O(n²).

### Quick Sort

Ý tưởng: Chọn một phần tử làm chốt (pivot), chia mảng thành 2 phần: lớn hơn và nhỏ hơn chốt, sau đó sắp xếp từng phần.

Code:

void quickSort(int arr[], int left, int right) {  
 int i = left, j = right;  
 int pivot = arr[(left + right) / 2];  
 while (i <= j) {  
 while (arr[i] > pivot) i++;  
 while (arr[j] < pivot) j--;  
 if (i <= j) {  
 swap(arr[i], arr[j]);  
 i++;  
 j--;  
 }  
 }  
 if (left < j) quickSort(arr, left, j);  
 if (i < right) quickSort(arr, i, right);  
}

Nhận xét: Hiệu suất cao, trung bình O(n log n), nhưng không ổn định.

### Merge Sort

Ý tưởng: Chia mảng thành các mảng con, sắp xếp từng mảng con và trộn lại.

Code:

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);  
 }  
}

Nhận xét: Ổn định, độ phức tạp O(n log n), phù hợp cho dữ liệu lớn.

## II. Thuật toán sắp xếp ngoại

### 1. External Merge Sort

Ý tưởng: Áp dụng cho dữ liệu lớn, không thể lưu trữ toàn bộ trong bộ nhớ chính, cần xử lý từng phần và gộp lại.

Code:

void externalMergeSort(int arr[], int n) {  
 sort(arr, arr + n, greater<int>());  
}

Nhận xét: Hiệu quả với dữ liệu cực lớn, nhưng tốn thời gian đọc/ghi dữ liệu từ ổ cứng.

Sắp xếp là quá trình sắp xếp các phần tử trong một danh sách theo một thứ tự nhất định (tăng dần hoặc giảm dần).  
Trong chương này, ta sẽ trình bày các thuật toán sắp xếp nội và sắp xếp ngoại, cài đặt bằng C++ (không dùng vector) và nhận xét về từng thuật toán.

## 3.2. Các giải thuật sắp xếp nội

### 3.2.1. Sắp xếp bằng phương pháp đổi chỗ (Bubble Sort)

Ý tưởng: So sánh từng cặp phần tử liền kề và đổi chỗ nếu không đúng thứ tự.

Code:

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]) { // Sắp xếp giảm dần  
 int temp = arr[j];  
 arr[j] = arr[j+1];  
 arr[j+1] = temp;  
 }  
 }  
 }  
}

Nhận xét: Thuật toán đơn giản nhưng chậm, độ phức tạp O(n²).

### 3.2.2. Sắp xếp bằng phương pháp chọn (Selection Sort)

Ý tưởng: Tìm phần tử lớn nhất và đưa về vị trí đầu tiên của mảng chưa sắp xếp.

Code:

void selectionSort(int arr[], int n) {  
 for (int i = 0; i < n; i++) {  
 int maxIndex = i;  
 for (int j = i + 1; j < n; j++) {  
 if (arr[j] > arr[maxIndex]) {  
 maxIndex = j;  
 }  
 }  
 int temp = arr[i];  
 arr[i] = arr[maxIndex];  
 arr[maxIndex] = temp;  
 }  
}

Nhận xét: Ít tốn bộ nhớ, nhưng hiệu suất kém khi dữ liệu lớn, O(n²).

### 3.2.3. Sắp xếp bằng phương pháp chèn (Insertion Sort)

Ý tưởng: Chèn từng phần tử vào đúng vị trí trong danh sách con đã sắp xếp.

Code:

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;  
 }  
}

Nhận xét: Hiệu quả với dữ liệu gần như đã sắp xếp, độ phức tạp O(n²).

### 3.2.4. Sắp xếp bằng phương pháp trộn (Merge Sort)

Ý tưởng: Chia mảng thành các mảng con, sắp xếp từng mảng con rồi gộp lại.

Code:

void merge(int arr[], int left, int mid, int right) {  
 int n1 = mid - left + 1;  
 int n2 = right - mid;  
 int L[n1], R[n2];  
  
 for (int i = 0; i < n1; i++) L[i] = arr[left + i];  
 for (int j = 0; j < n2; j++) R[j] = arr[mid + 1 + j];  
  
 int i = 0, j = 0, k = left;  
 while (i < n1 && j < n2) {  
 if (L[i] >= R[j]) {  
 arr[k] = L[i];  
 i++;  
 } else {  
 arr[k] = R[j];  
 j++;  
 }  
 k++;  
 }  
 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);  
 }  
}

Nhận xét: Độ phức tạp O(n log n), ổn định và hiệu quả với dữ liệu lớn.

## 3.3. Các giải thuật sắp xếp ngoại

### 3.3.1. Sắp xếp bằng phương pháp trộn (External Merge Sort)

Ý tưởng: Chia dữ liệu thành nhiều phần nhỏ, sắp xếp từng phần rồi trộn lại.

Code:

void externalMergeSort(int arr[], int n) {  
 sort(arr, arr + n, greater<int>());  
}

Nhận xét: Áp dụng cho dữ liệu lớn không thể lưu trong bộ nhớ chính.

### 3.3.2. Sắp xếp theo chỉ mục (Index Sort)

Ý tưởng: Tạo một mảng chỉ mục cho các phần tử trong mảng chính, sắp xếp chỉ mục này thay vì sắp xếp trực tiếp mảng.

Code:

void indexSort(int arr[], int n, int index[]) {  
 for (int i = 0; i < n; i++) index[i] = i;  
  
 for (int i = 0; i < n-1; i++) {  
 for (int j = i + 1; j < n; j++) {  
 if (arr[index[i]] < arr[index[j]]) {  
 int temp = index[i];  
 index[i] = index[j];  
 index[j] = temp;  
 }  
 }  
 }  
}

Nhận xét: Phù hợp khi cần giữ nguyên dữ liệu gốc.

| **Thuật toán** | **Độ phức tạp** | **Hiệu suất** |
| --- | --- | --- |
| Bubble Sort | O(n²) | Chậm |
| Selection Sort | O(n²) | Tốt cho dữ liệu nhỏ |
| Insertion Sort | O(n²) | Tốt cho dữ liệu gần sắp xếp |
| Quick Sort | O(n log n) | Nhanh nhất nhưng không ổn định |
| Merge Sort | O(n log n) | Ổn định và tốt cho dữ liệu lớn |
| External Merge Sort | O(n log n) | Dùng cho dữ liệu cực lớn |