Report: Properties and Time Complexities

Bộ môn: Cấu trúc Dữ liệu và Giải thuật.

Giáo viên hướng dẫn: Nguyễn Thanh Phương, Nguyễn Ngọc Thảo, Nguyễn Thanh Tình.

Tên học sinh: Phùng Quốc Tuấn.  
Mã số sinh viên: 19127616.

# Bảng tự đánh giá

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Tên** | **Người đảm nhiệm** | **Phần trăm hoàn thành** | **Score** |
| 1 | Implement correctly | Tuấn | 100% | 50% |
| 2 | Report | Tuấn | 100% | 20% |
| 3 | Slide | Tuấn | 0% | 10% |
| 4 | Video | Tuấn | 0% | 20% |
|  | Tổng | | 100% x 50% + 100% x 20% = 70% | |

# Thuộc tính thuật toán

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thuật toán** | **Tính ổn định** | **In-place** | **Parsimony** | **Note** |
| Bubble sort | ✓ | ✓ | ✓ | Giữ nguyên vị trí phần tử bằng nhau |
| Selection sort | 🗶 | ✓ | ✓ | Không ổn định do hoán đổi không giữ vị trí ban đầu |
| Insertion sort | ✓ | ✓ | ✓ | Thích hợp với mảng “gần như đã sắp xếp” |
| Heap sort | 🗶 | ✓ | ✓ | Không ổn định do cấu trúc heap thay đổi thứ tự |
| Quick sort | 🗶 | ✓ | ✓ | Có phiên bản ổn định nhưng không phổ biến |
| Merge sort | ✓ | 🗶 | 🗶 | Cần mảng phụ (không in-place) |
| Counting sort | ✓ | 🗶 | Skip | Không dựa trêm so sánh, dùng mảng đếm |
| Radix sort | ✓ | 🗶 | Skip | Dựa vào Counting sort nên không in-place |

Ghi chú các thuộc tính:

* **Stability:** Nếu hai phần tử bằng nhau thì vị trí tương đối của chúng không bị thay đổi sau khi sắp xếp.
* **In-place:** Thuật toán không sử dụng bộ nhớ bổ sung đáng kể, thường chỉ vài biến tạm.
* **Parsimony (Tiết kiệm bộ nhớ):** Giống In-place, nhưng nhấn mạnh không dùng mảng/phần tử phụ thừa thãi.
* ***Không xét "Parsimony" cho thuật toán không dựa trên so sánh như Counting sort, Radix sort.***

# Độ phức tạp thời gian (Big-O)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Thuật toán** | **Best case** | **Average case** | **Worst case** | **Ghi chú** |
| Bubble sort | O(n) | O(n²) | O(n²) | Dừng sớm nếu đã sắp xếp |
| Selection sort | O(n²) | O(n²) | O(n²) | Không phụ thuộc vào dữ liệu |
| Insertion sort | O(n) | O(n²) | O(n²) | Hiệu quả với mảng gần sắp |
| Heap sort | O(nlogn) | O(nlogn) | O(nlogn) | Ổn định về độ phức tạp |
| Quick sort | O(nlogn) | O(nlogn) | O(n²) | Chia không đều gây xấu nhất |
| Merge sort | O(nlogn) | O(nlogn) | O(nlogn) | Phải sử dụng bộ nhớ phụ |
| Counting sort | O(n + k) | O(n + k) | O(n + k) | k là giá trị lớn nhất |
| Radix sort | O(d(n + k)) | O(d(n + k)) | O(d(n + k)) | d là số chữ số tối đa, k cơ số (hệ thập phân thì k = 10, hệ thập lục phân thì k = 16...) |

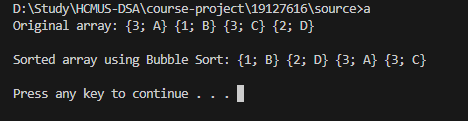
# Xem xét chi tiết thuộc tính của thuật toán

**1. Tính ổn định (Stability)**

**Định nghĩa:**  
Một thuật toán sắp xếp **ổn định** nếu **vị trí tương đối** giữa các phần tử **có giá trị bằng nhau** **không thay đổi** sau khi sắp xếp.

**Cách kiểm tra:**

* Kiểm tra xem **thuật toán có bao giờ hoán đổi hai phần tử bằng nhau không**.
* Nếu **chỉ hoán đổi khi key > key**, không phải >=, thì có khả năng **ổn định**.
* Ví dụ:



Trong ví dụ này: 3-A đứng trước 3-C và sau khi được sắp xếp bằng **bubble sort** thì 3-A vẫn đứng trước 3-C nên ta nói thuật toán này **có ổn định.**

| **Thuật toán** | **Ổn định** | **Giải thích** |
| --- | --- | --- |
| **Bubble Sort** | Yes | Hoán đổi chỉ khi arr[j] > arr[j+1]. Nếu bằng thì giữ nguyên vị trí. |
| **Selection Sort** | No | Có thể hoán đổi phần tử ở xa lên đầu — không giữ thứ tự ban đầu. |
| **Insertion Sort** | Yes | Di chuyển phần tử lớn hơn sang phải, phần tử bằng không bị ảnh hưởng. |
| **Heap Sort** | No | Khi tạo heap và hoán đổi, các phần tử bằng nhau có thể thay đổi thứ tự. |
| **Quick Sort** | No | Các phần tử bằng nhau vẫn có thể bị chia và sắp xếp lại khác thứ tự. |
| **Merge Sort** | Yes | So sánh <= nên phần tử bên trái được ưu tiên giữ trước. |
| **Counting Sort** | Yes | Khi đổ ngược lại mảng output, dùng vòng lặp từ phải qua trái, nên giữ đúng thứ tự. |
| **Radix Sort** | Yes | Dựa vào Counting Sort nên kế thừa tính ổn định. |

**2. In-place (Tại chỗ)**

**Định nghĩa:** Thuật toán là *in-place* nếu **không sử dụng thêm bộ nhớ phụ đáng kể**, thường chỉ là vài biến tạm.

**Cách xác định trong code:**

* Nếu không tạo mảng mới (như vector<Element> output, L, R...), mà chỉ thao tác trong arr, thì là in-place.

| **Thuật toán** | **In-place** | **Giải thích** |
| --- | --- | --- |
| **Bubble Sort** | Yes | Chỉ dùng các biến tạm và swap trực tiếp trong arr. |
| **Selection Sort** | Yes | Tương tự, không dùng thêm mảng nào. |
| **Insertion Sort** | Yes | Dùng biến key để tạm giữ phần tử cần chèn. |
| **Heap Sort** | Yes | Sử dụng các thao tác hoán đổi và heapify trên mảng gốc. |
| **Quick Sort** | Yes | Đệ quy, nhưng không cấp phát mảng mới. Dùng chính mảng arr. |
| **Merge Sort** | No | Dùng mảng phụ L và R nên không in-place. |
| **Counting Sort** | No | Dùng mảng count và output. |
| **Radix Sort** | No | Mỗi lần theo chữ số tạo output và count → dùng bộ nhớ phụ. |

**3. Parsimony (Tiết kiệm bộ nhớ)**

**Định nghĩa:** Thuật toán là *parsimony* nếu **ngoài việc là in-place, còn cực kỳ tiết kiệm bộ nhớ**, không tạo mảng/phần tử phụ trừ trường hợp tối thiểu.

**Cách xác định trong code:**

* Thuật toán in-place gần như luôn có parsimony, trừ khi dùng thêm nhiều biến/phần tử tạm không cần thiết.
* Không áp dụng cho Counting Sort, Radix Sort vì chúng không dựa trên so sánh.

| **Thuật toán** | **Parsimony** | **Giải thích** |
| --- | --- | --- |
| **Bubble Sort** | Yes | Rất ít biến phụ, không tạo mảng tạm. |
| **Selection Sort** | Yes | Tương tự Bubble Sort. |
| **Insertion Sort** | Yes | Dùng 1 biến key để lưu tạm — tối thiểu. |
| **Heap Sort** | Yes | Không cấp phát thêm, các thao tác đều trên mảng. |
| **Quick Sort** | Yes | Nếu đệ quy dùng stack hệ thống → không thêm bộ nhớ đáng kể. |
| **Merge Sort** | No | Dùng nhiều mảng phụ rõ ràng. |
| **Counting Sort** | (skip) | Không áp dụng — không phải sorting by comparison. |
| **Radix Sort** | (skip) | Không áp dụng — dựa trên Counting Sort. |

# Xem xét chi tiết độ phức tạp thời gian của thuật toán

**Nguyên tắc tính độ phức tạp thời gian**

* **Đếm số vòng lặp** (for/while).
* **Tính chi phí mỗi vòng lặp** (gán, so sánh, swap...).
* **Tổng hợp lại thành biểu thức theo n** (số phần tử).
* **Lấy hàm chi phối lớn nhất (Big-O)**.

1. **Bubble Sort – O(n²)**

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

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

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

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

}

}

}

**Phân tích:**

* **Vòng lặp ngoài** chạy n-1 lần ≈ O(n)
* **Vòng lặp trong**: với mỗi i, lặp (n - i - 1) lần → tổng cộng:

1. **Selection Sort – O(n²)**

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

int min\_idx = i;

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

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

min\_idx = j;

}

}

swap(arr[min\_idx], arr[i]);

}

**Phân tích:**

* **Vòng lặp ngoài** chạy n-1 lần.
* **Vòng lặp trong** chạy từ i+1 đến n → tổng:

Lưu ý: chỉ **swap 1 lần mỗi vòng ngoài**, nên số lần hoán đổi ít hơn Bubble Sort → tốt hơn về thực tế, nhưng **Big-O vẫn như nhau**.

1. **Insertion Sort – O(n²)**

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

key = arr[i];

j = i - 1;

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

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

j--;

}

arr[j + 1] = key;

}

**Phân tích:**

* **Best Case**: mảng đã sắp xếp → không vào vòng while → O(n)
* **Worst Case**: mảng ngược → mỗi phần tử phải dời j lần → tổng:

1. **Heap Sort – O(n log n)**

**Gồm 2 phần:**

1. **Xây heap** (heapify từ giữa về đầu): O(n)
2. **Sắp xếp** (lặp n lần, mỗi lần gọi heapify O(log n)): O(n log n)

**Tổng:**

1. **Quick Sort – O(n log n) trung bình, O(n²) worst**

**Gồm:**

1. **Chia mảng** bằng partition → O(n)
2. **Gọi đệ quy** 2 phần nhỏ hơn

**Best case: Chia đều:**

**Worst case: chia lệch (n-1, 0):**

1. **Merge Sort – O(n log n)**

**Mỗi lần gọi:**

* **Chia đôi mảng**: O(1)
* **Gọi đệ quy 2 nửa**
* **Merge** lại: O(n)

**Công thức:**

(Hàm chia + hợp = tổng log n tầng, mỗi tầng n phần tử.)

1. **Counting Sort – O(n + k)**

* **Đếm** từng phần tử: O(n)
* **Cộng dồn**: O(k)
* **Tạo output**: O(n)

Với k là giá trị lớn nhất trong arr.key

1. **Radix Sort – O(d(n + k))**

* Duyệt từng chữ số (digit), mỗi lần dùng Counting Sort.

Với d là số chữ số tối đa, k là base (thường 10).

**Tổng kết so sánh bằng tư duy tính toán:**

| **Thuật toán** | **Cách tính chi tiết** | **Kết luận Big-O** |
| --- | --- | --- |
| Bubble Sort | O(n) + O(n-1) + ... + O(1) | O(n²) |
| Selection Sort | Tìm min O(n) × n lần | O(n²) |
| Insertion Sort | Dời phần tử: O(n) × n lần | O(n²) |
| Heap Sort | Build Heap O(n) + Sort n×log n | O(n log n) |
| Quick Sort | Tốt: 2T(n/2)+O(n) ; Xấu: T(n-1)+O(n) | O(n log n) / O(n²) |
| Merge Sort | 2T(n/2) + O(n) | O(n log n) |
| Counting Sort | O(n) + O(k) + O(n) | O(n + k) |
| Radix Sort | d × Counting Sort | O(d(n + k)) |

# Link video Thuyết trình

* Link:

# Tham khảo

* Learn Cpp from Beginner to Advanced ✅ Practice 🎯 Code 💻 Repeat 🔁 One step solution for c++ beginners and cp enthusiasts: <https://github.com/Lakhankumawat/LearnCPP>
* "Sorting Algorithms Implemented in Go": <https://github.com/ayoubzulfiqar/Sorting-Algorithms>
* Sorting Algorithms Time Complexity Analysis: <https://github.com/MohEsmail143/sorting-algorithms-time-complexity-analysis>
* C task exercises on Sorting Algorithms and Big O notation: <https://github.com/El-gibbor/sorting_algorithms>