## Câu hỏi

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

* Selection Sort: Ở mỗi bước của thuật toán luôn đưa ra phần tử nhỏ nhất và đưa về đầu dãy. Để sắp xếp dãy có n phần tử cần thực hiện n-1 bước
* Bubble Sort: So sánh 2 phần tử liền kề. Nếu 2 phần tử liền kề với nhau là một cặp nghịch thế (phần tử đứng trước > phần tử đứng sau) thì nó sẽ hoán đổi 2 phần tử đó và đưa phần tử lớn nhất về cuối dãy
* Insertion Sort: Ở mỗi bước của thuật toán sẽ cố gắng đưa phần tử ở vị trí hiện tại về đúng vị trí, trong đó các phần tử đứng trước phần tử hiện tại này đều đã được sắp xếp
* Counting Sort: Đếm phần tử trong mảng xuất hiện bao nhiêu lần sau đó nó sẽ duyệt từ phần tử nhỏ nhất đến phần tử lớn nhất và in ra số lần xuất hiện của mỗi phần tử
* Merge Sort: Thuật toán sắp xếp trộn sử dụng phương pháp chia và trị, chia mảng ban đầu thành các dãy con cho tới khi chỉ còn 1 phần tử, sau đó thực hiện trộn 2 dãy con tăng dần thành 1 dãy tăng dần với độ phức tạp tuyến tính
* Heap Sort: Sắp xếp vun đống hoạt động bằng cách hình dung các phần tử của mảng như 1 loại cây nhị phân hoàn chỉnh được gọi là heap
* Quick sort: Thuật toán sắp xếp nhanh tương tự Merge Sort đó là nó cũng sử dụng phương pháp chia và trị

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

Trong các thuật toán sắp xếp, tôi thích nhất thuật toán Selection Sort tại vì nó khá đơn giản và dễ hiểu, tóm lại thì cái thuật toán này khi bạn muốn sắp xếp mảng giảm dần thì nó sẽ đưa phần tử nhỏ nhất lên đầu dãy còn sắp xếp mảng tăng dần thì nó sẽ đưa phần tử lớn nhất về đầu dãy, rồi cứ như thế duyệt tần phần tử còn lại

1. 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: Selection Sort, Bubble Sort, Insertion Sort, Counting Sort, Merge Sort, Heap Sort, Quick sort
* Thuật toán sắp xếp ngoại: Merge Sort
* Nhận xét
* Dễ cài đặt: Bubble Sort, Selection Sort, Insertion Sort, Counting Sort.
* Khó cài đặt hơn: Merge Sort, Quick Sort.
* Độ phức tạp thời gian thấp nhất: Merge Sort, Quick Sort (trong trường hợp trung bình).
* Độ phức tạp bộ nhớ thấp nhất: Bubble Sort, Selection Sort, Insertion Sort.
* Ứng dụng thực tế: Merge Sort, Quick Sort, Counting Sort (với điều kiện phạm vi giá trị nhỏ).

1. Hãy trình bày những ưu điểm và nhược điểm của mỗi thuật toán sắp xếp? Theo bạn, cách khắc phục những nhược điểm này là gì?

|  |  |  |  |
| --- | --- | --- | --- |
| **Các thuật toán sắp xếp** | **Ưu điểm** | **Nhược điểm** | **Cách khắc phục** |
| Selection Sort | -Đơn giản và dễ hiểu. -Không cần bộ nhớ phụ trợ. -Tính ổn định (điều này đúng với phiên bản cải tiến). | - Thời gian chạy chậm với mảng lớn (**O(n²)**). - Hiệu suất kém khi dữ liệu không gần như đã sắp xếp. | - Có thể cải tiến bằng cách giảm số lần tìm kiếm phần tử nhỏ nhất trong mỗi lần lặp. - Sử dụng thuật toán sắp xếp hiệu quả hơn như **Quick Sort** hoặc **Merge Sort** nếu yêu cầu hiệu suất cao. |
| Bubble Sort | -Đơn giản và dễ hiểu. -Không cần bộ nhớ phụ trợ. -Tính ổn định. | - Thời gian chạy chậm với mảng lớn (**O(n²)**). - Tốn nhiều vòng lặp. | - Có thể cải thiện bằng cách thêm một biến kiểm tra nếu không có sự thay đổi nào trong một vòng lặp, khi đó có thể dừng sớm. - Sử dụng thuật toán sắp xếp hiệu quả hơn như **Quick Sort** hoặc **Merge Sort**. |
| Insertion Sort | -Hiệu quả với dãy đã được sắp xếp gần đúng. -Đơn giản và dễ hiểu. -Tính ổn định. Tốt cho các mảng nhỏ. | - Thời gian chạy chậm với mảng lớn (**O(n²)**). - Không hiệu quả với mảng có kích thước lớn. | - Áp dụng **Insertion Sort** cho các mảng nhỏ hoặc các dãy gần như đã sắp xếp. - Kết hợp với thuật toán sắp xếp khác như **Merge Sort** cho các mảng lớn hơn. |
| Counting Sort | -Hiệu quả về thời gian khi dãy có phạm vi giá trị nhỏ. -Tính ổn định. -Không sử dụng so sánh, giúp tiết kiệm chi phí thời gian. -Quá trình sắp xếp có thể rất nhanh với mảng lớn nếu phạm vi giá trị nhỏ.. | - Cần bộ nhớ phụ trợ lớn đối với dãy có phạm vi giá trị lớn. - Giới hạn về kiểu dữ liệu (chỉ làm việc với dữ liệu số nguyên hoặc dữ liệu có giá trị phân biệt rõ ràng). - Không hiệu quả với phạm vi giá trị lớn. | - Chỉ áp dụng khi phạm vi giá trị của dữ liệu nhỏ hoặc biết trước phạm vi của các phần tử. |
| Merge Sort | -Thời gian chạy ổn định với độ phức tạp **O(nlogn)** -Tính ổn định. -Dễ dàng chia nhỏ mảng để thực hiện sắp xếp. | - Cần bộ nhớ phụ trợ (độ phức tạp không gian **O(n)**). - Thời gian xử lý cao hơn các thuật toán sắp xếp tại chỗ. | - Có thể sử dụng **Merge Sort** khi cần sắp xếp dữ liệu lớn và khi bộ nhớ không phải là vấn đề lớn. - Dễ dàng triển khai song song hoặc phân tán. |
| Heap Sort | - Thời gian xử lý ổn định **O(nlogn)**. - Không yêu cầu không gian bổ sung (sắp xếp tại chỗ). - Tính ổn định (tuy nhiên cần thay đổi cách triển khai để đảm bảo tính ổn định). | - Không phải lựa chọn nhanh nhất (so với Quick Sort). - Phức tạp trong việc triển khai. - Không phải là thuật toán ổn định nếu không cải tiến. | - Để cải thiện tính ổn định, có thể sử dụng **Merge Sort** nếu yêu cầu ổn định. - Dễ dàng triển khai trong môi trường không gian bộ nhớ hạn chế. |
| Quick Sort | - Thường nhanh hơn các thuật toán **O(n log n)** khác do hiệu suất bộ nhớ đệm tốt. - Sắp xếp tại chỗ (không cần bộ nhớ phụ trợ). | - Độ phức tạp trong trường hợp xấu nhất là **O(n²)** (khi dữ liệu đã gần như sắp xếp). - Không phải là thuật toán sắp xếp ổn định. | - Chọn pivot tốt |