PHÂN TÍCH CÁC THUẬT TOÁN SẮP XẾP

Sinh viên: Ngô Đình Duy Quang – 1312458

Phân tích các trường hợp tốt nhất và xấu nhất cho các thuật toán:

1. Selection Sort
2. Bubble Sort
3. Shaker Sort
4. Quick Sort
5. Radix Sort

# Selection Sort

Thuật toán Selection sort là cứ mỗi vị trí i chạy từ 0 đến n – 2, ta duyệt từ i + 1 đến cuối mảng để tìm giá trị tốt nhất sau đó đổi chỗ i và giá trị tốt nhất này (nếu tìm ra). Như vậy, số phép so sánh là = , số phép hoán vị cũng bằng cho mọi trường hợp.

Trường hợp tốt nhất chính là mảng đã được sắp xếp và ta không tìm ra được bất cứ vị trí tốt nhất nào từ i + 1 đến cuối mảng nên số lần gán cho biến imin bằng 0. Độ phức tạp trong trường hợp này là O(.

Trường hợp xấu nhất xảy ra khi mảng không hề có 1 phần tử nào đứng đúng vị trí, như vậy mỗi lần duyệt ta luôn phải cập nhật biến imin, như vậy ta phải tốn số phép gán để thay đổi giá trị cho biến imin = n(n – 1) / 2.

Vậy, độ phức tạp của giải thuật là O(n2) cho mọi trường hợp.

# Bubble Sort

Thuật toán này ta duyệt i từ 0 đến n – 2. Sau đó ta cho nổi lên từ từ phần tử tốt nhất từ i đến cuối mảng lên i. Để làm được điều này, ta xét các cặp phần tử từ cuối mảng đến i + 1, nếu hai cặp này nghịch thế (tức phần tử tốt nhất nằm dưới phần tử còn lại), thì ta đổi chỗ hai phần tử này. Quá trình này tựa như bong bóng khí nổi từ đáy hồ lên mặt nước nên được đặt tên là Bubble. Số phép so sánh trong thuật toán này là:

Trường hợp tốt nhất là mảng đã được sắp xếp, ta không phải tốn chi phí cho việc hoán vị, chỉ tốn số phép so sánh như trên.

Trường hợp xấu nhất là mảng sắp xếp ngược (phần tử nhẹ nhất lại ở dưới, còn nặng nhất nằm ở trên), ta thấy với mỗi lần so sánh ta đều phải hoán vị hai phần tử, nên số phép hoán vị tiêu tốn là .

Tổng kết lại, ta có độ phức tạp của Bubble Sort là O(n2) trong mọi trường hợp.

# Shaker Sort

Ý tưởng thuật toán : trong mỗi lần sắp xếp, duyệt mảng theo hai lượt từ hai phía khác nhau, lượt đi: đẩy phần tử nhỏ về đầu mảng, lượt về đẩy phần tử lớn về cuối mảng, đồng thời lưu vị trí có sự hoán vị cuối cùng để lần duyệt sau không thực hiện những so sánh dư thừa.

Trường hợp tốt nhất: lượt đi đầu tiên không tìm ra vị trí nghịch thế, chứng tỏ mảng đã được sắp xếp, ta chỉ tốn n – 1 phép so sánh và thuật toán kết thúc, không hề tốn chi phí hoán vị. Độ phức tạp trong trường hợp này là O(n).

Trường hợp xấu nhất: sau một bước so sánh hai cặp kề lại phát hiện nghịch thế, và sau một lượt đi thì biến left tăng lên 1 đơn vị, sau một lượt về biến right giảm một đơn vị, chi phí so sánh của thuật toán này là chi phí hoán vị của trường hợp này cũng bằng chi phí so sánh. Độ phức tạp trong trường hợp này là O(n2).

# Quick Sort

Ta nhận thấy hiệu quả của thuật toán phụ thuộc vào việc chọn giá trị mốc (phần tử chốt).

* Trường hợp tốt nhất: mỗi lần phân hoạch ta đều chọn được phần tử trung vị (phần tử lớn hơn hay bằng nửa số phần tử và nhỏ hơn hay bằng nửa số phần tử còn lại) làm mốc. Khi đó dãy được phân hoạch thành hai phần bằng nhau, và ta cần log2(n) lần phân hoạch thì sắp xếp xong. Ta cũng dễ nhận thấy trong mỗi lần phân hoạch ta cần duyệt qua n phần tử. Vậy độ phức tạp trong trường hợp tốt nhất thuộc O(nlogn).
* Trường hợp xấu nhất: mỗi lần phần hoạch ta chọn phải phần tử có giá trị cực đại hoặc cực tiểu làm mốc. Khi đó dãy bị phân hoạch thành hai phần không đều: một phần chỉ có một phần tử, phần còn lại có n - 1 phần tử. Do đó, ta cần tới n lần phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất thuộc O(n2).

# Radix Sort

Radix Sort chỉ thực hiện được trên số nguyên vì bản chất của giải thuật này là lấy từng chữ số của mỗi phần tử rồi sắp xếp các chữ số này.

Giải thuật chạy nhanh hay chậm còn tùy vào số chữ số của số lớn nhất, hàm thể hiện độ phức tạp của giải thuật là f(n) = (số chữ số)\*(3n + 9), hay độ phức tạp là O(n) trong mọi trường hợp.