CÁC THUẬT TOÁN SẮP XẾP VÀ Ý TƯỞNG TỐI ƯU

1. Interchange Sort

- Trong Interchange Sort, ta lấy một phần tử cố định và so sánh với tất cả các phần tử còn lại, nếu sai thứ tự thì đổi chỗ ngay lập tức.
- Độ phức tạp**:** Luôn là O(n^2) vì phải duyệt qua mọi cặp bất kể dữ liệu đầu vào như thế nào.

2. Quick Sort

- Được cải tiến từ Interchange Sort.
- Cách hoạt động của QuickSort bằng cách chọn một phần tử làm cột mốc (thường chọn phần tử giữa, đầu hoặc cuối mảng). Sau đó đẩy tất cả phần tử nhỏ hơn chốt về bên trái. Đẩy tất cả phần tử lớn hơn chốt về bên phải. Đưa chốt vào giữa. (Lúc này chốt đã đứng đúng vị trí tuyệt đối).
- Đệ quy: Lặp lại y hệt hai bước trên cho hai nửa mảng vừa tạo ra cho đến khi mỗi nửa chỉ còn 1 hoặc 0 phần tử.

Ý tưởng tối ưu:

- Giảm số lần đổi chỗ (Hoán vị): Chỉ đổi chỗ khi tìm thấy một cặp sai vị trí ở hai đầu mảng (một số lớn nằm bên trái và một số nhỏ nằm bên phải).
- Dùng phương pháp chia để trị: Sau mỗi lần phân hoạch, nó chia mảng thành 2 nửa độc lập. Các phần tử ở nửa trái không bao giờ phải so sánh với nửa phải nữa. Việc chia nhỏ này giúp tốc độ tăng theo cấp số nhân.
- Tốc độ: Là một trong những thuật toán nhanh nhất trên thực tế với độ phức tạp trung bình là O(nlogn).

3. Bubble Sort

- Là thuật toán cơ bản nhất, hoạt động dựa trên việc so sánh các cặp phần tử kế cận. Duyệt từ đầu đến cuối mảng, nếu thấy hai phần tử đứng cạnh nhau bị sai thứ tự (số lớn đứng trước số nhỏ) thì đổi chỗ chúng. Sau mỗi lượt duyệt, phần tử lớn nhất sẽ nổi dần về cuối mảng (giống như bọt khí nổi lên mặt nước).
- Độ phức tạp: Rất lớn có thể lên tới O(n^2).

4. Shake Sort

- Được cải tiến từ Bubble sort

- Là bản nâng cấp của Bubble Sort, thay vì chỉ đi một chiều, nó đi theo hai chiều. Lượt đi đẩy phần tử lớn nhất về cuối mảng (giống Bubble Sort). Lượt về ngay lập tức quay ngược lại, đẩy phần tử nhỏ nhất về đầu mảng. Mảng được thu hẹp và sắp xếp từ cả hai đầu cùng lúc.
- Độ phức tạp: Nó nhanh hơn Bubble Sort về mặt thực tế nhưng vẫn thuộc nhóm độ phức tạp O(n^2).

Ý tưởng tối ưu:

- Trong Bubble Sort (chỉ đi từ trái sang phải), một số nhỏ nằm ở cuối mảng chỉ có thể nhích sang trái duy nhất 1 vị trí sau mỗi vòng lặp. Nó di chuyển rất chậm chạp.
- Shake Sort có lượt về từ phải sang trái. Lượt này sẽ lấy số nhỏ đó và đẩy thẳng nó về đầu mảng chỉ trong một lần duyệt.

5. Insertion Sort

- Insertion Sort (Sắp xếp chèn) chia mảng thành hai phần là đã sắp xếp (bên trái) và chưa sắp xếp (bên phải). Lấy từng phần tử từ bên chưa sắp xếp đem đi so sánh ngược về phía bên đã sắp xếp. Tìm vị trí thích hợp và chèn nó vào giữa các phần tử đã có để duy trì thứ tự tăng dần.
- Độ phức tạp: Rất lớn lên tới O(n^2)

6. Binary Insertion Sort

- Là bản nâng cấp của Insertion Sort, tập trung tối ưu hóa bước tìm vị trí chèn. Thay vì duyệt từng phần tử để tìm vị trí chèn. Thì nó dùng Binary Search.

Ý tưởng cải tiến

- Thay vì so sánh tuần tự từng phần tử từ phải sang trái (mất nhiều thời gian), nó sử dụng Tìm kiếm nhị phân (Binary Search) để tìm ngay vị trí cần chèn trong phần mảng đã sắp xếp.
- Độ phức tạp: Thuật toán này nhanh hơn Insertion Sort về mặt so sánh, nhưng tổng thời gian vẫn là O(n^2) vì thao tác dịch chuyển các phần tử trong mảng vẫn tốn nhiều thời gian.

7. Shell Sort

- Cách hoạt động của Shell Sort: Ban đầu, thuật toán không so sánh các phần tử đứng cạnh nhau. Nó chọn một khoảng cách gap lớn. Mảng sẽ được chia thành nhiều nhóm nhỏ dựa trên khoảng cách này. Thuật toán thực hiện Insertion Sort trên từng nhóm này. Các phần tử sẽ nhảy những bước dài để về gần vị trí đúng của chúng. Sau khi xong một lượt, gap sẽ được giảm đi. Vòng lặp cuối cùng: Khi gap = 1, thuật toán thực hiện một lượt Insertion Sort bình thường trên toàn bộ mảng. Vì mảng lúc này đã "gần như sắp xếp xong" sau các bước nhảy xa, lượt cuối này diễn ra cực kỳ nhanh.
- Độ phức tạp: Shell Sort là thuật toán nhanh hơn các thuật toán cơ bản (O(n^2)) nhưng vẫn chậm hơn các thuật toán O(nlogn) trên dữ liệu cực lớn.

Ý tưởng cải tiến

- Cải tiến: Thay thế di chuyển từng phần tử bằng nhảy quãng xa.
- Chiến thuật: So sánh và chèn các phần tử cách xa nhau một khoảng gap trước.
- Lợi ích: Giúp các phần tử nhỏ ở cuối mảng vọt về đầu mảng cực nhanh, giảm tối đa số lần dịch chuyển dữ liệu nặng nề của Insertion Sort truyền thống.

8. Selection Sort

- Selection Sort (Sắp xếp chọn) là thuật toán đi tìm phần tử nhỏ nhất và đưa nó về đúng vị trí đầu tiên. Tìm phần tử nhỏ nhất trong toàn bộ mảng. Hoán đổi phần tử nhỏ nhất đó với phần tử ở vị trí đầu tiên. Xem như vị trí đầu tiên đã xong, tiếp tục tìm phần tử nhỏ nhất trong phần còn lại và đổi chỗ với vị trí thứ hai. Cứ thế tiếp tục cho đến cuối mảng.
- Độ phức tạp: Luôn là O(n^2) trong mọi trường hợp.

9. Heap Sort

- Heap Sort (Sắp xếp vun đống) là bản cải tiến "cao cấp" của Selection Sort, sử dụng cấu trúc dữ liệu Heap (một loại cây nhị phân) để tìm phần tử lớn nhất/nhỏ nhất một cách cực nhanh.
- Độ phức tạp: Rất ổn định O(nlogn)

Ý tưởng tối ưu

- Heap Sort: Nó tổ chức dữ liệu vào một cấu trúc Cây nhị phân (Heap). Trong cấu trúc này, các phần tử được duy trì theo một thứ tự cha-con nghiêm ngặt. Nhờ đó, phần tử lớn nhất luôn nằm ngay ở đỉnh cây, không cần đi tìm.

10.Merge Sort

- Merge Sort hoạt động bằng cách chia đôi mảng liên tục cho đến khi thu được các mảng con chỉ chứa một phần tử (đã được coi là sắp xếp), sau đó thực hiện quá trình trộn ngược lên bằng cách dùng hai con trỏ so sánh từng cặp phần tử ở đầu các mảng con, lấy giá trị nhỏ hơn xếp vào mảng kết quả cho đến khi khôi phục được mảng ban đầu theo đúng thứ tự tăng dần
- Độ phức tạp: trong trường hợp tệ nhất O(nlogn)

11.Natural Merge Sort

- Natural Merge Sort hoạt động bằng cách quét mảng từ trái sang phải để nhận diện các đoạn dữ liệu đã có thứ tự tăng dần tự nhiên (gọi là các Run), sau đó đưa chúng vào một hàng đợi và thực hiện trộn từng cặp Run lại với nhau một cách đệ quy cho đến khi chỉ còn lại một dãy duy nhất, giúp thuật toán đạt hiệu suất tối ưu O(n) nếu mảng đã có nhiều đoạn sắp xếp sẵn và giữ vững mức O(n \log n) trong trường hợp xấu nhất.

Ý tưởng tối ưu

- Ý tưởng tối ưu của Natural Merge Sort so với Merge Sort cơ bản nằm ở tính thích nghi: thay vì chia đôi mảng một cách máy móc và mù quáng, nó tận dụng các đoạn đã có thứ tự sẵn (Runs) để giảm thiểu số lượng mảnh con ban đầu, từ đó rút ngắn đáng kể số tầng trộn và giúp thuật toán đạt tốc độ lý tưởng O(n) khi dữ liệu đã gần như sắp xếp xong thay vì luôn tốn O(nlogn) như phiên bản truyền thống.

12.K-way Merge Sort

- K-way Merge Sort hoạt động bằng cách chia mảng thành k phần bằng nhau thay vì chia đôi, sau đó sử dụng cấu trúc Min-Heap để quản lý các phần tử đầu tiên của k danh sách đã sắp xếp, cho phép lấy ra phần tử nhỏ nhất và bổ sung phần tử mới vào cây một cách liên tục cho đến khi trộn hoàn tất, giúp giảm thiểu đáng kể số tầng đệ quy và tối ưu hóa việc đọc/ghi khi xử lý các tập dữ liệu khổng lồ không thể chứa hết trong bộ nhớ RAM.

Ý tưởng tối ưu

- Ý tưởng tối ưu của K-way Merge Sort so với Merge Sort thông thường là tăng số đường trộn đồng thời từ 2 lên k đường nhằm giảm thiểu chiều sâu của cây đệ quy từ log2(n) xuống còn logk(n).

13.Counting Sort

- Counting Sort là thuật toán sắp xếp tuyến tính không dựa trên so sánh, hoạt động bằng cách đếm số lần xuất hiện của từng giá trị trong mảng và sử dụng thông tin đó để tính toán trực tiếp vị trí của chúng trong mảng kết quả, giúp đạt tốc độ tối ưu O(n) khi phạm vi giá trị của dữ liệu không quá lớn.

14.Radix Sort

- Radix Sort là thuật toán sắp xếp tuyến tính hoạt động bằng cách phân loại các phần tử dựa trên từng chữ số của chúng theo thứ tự từ hàng thấp đến hàng cao, sử dụng một thuật toán sắp xếp ổn định như Counting Sort để xử lý từng vị trí chữ số, giúp đạt hiệu suất cực cao mà không cần so sánh trực tiếp giá trị của các phần tử với nhau.