**8.13. ĐÁNH GIÁ, NHẬN XÉT**

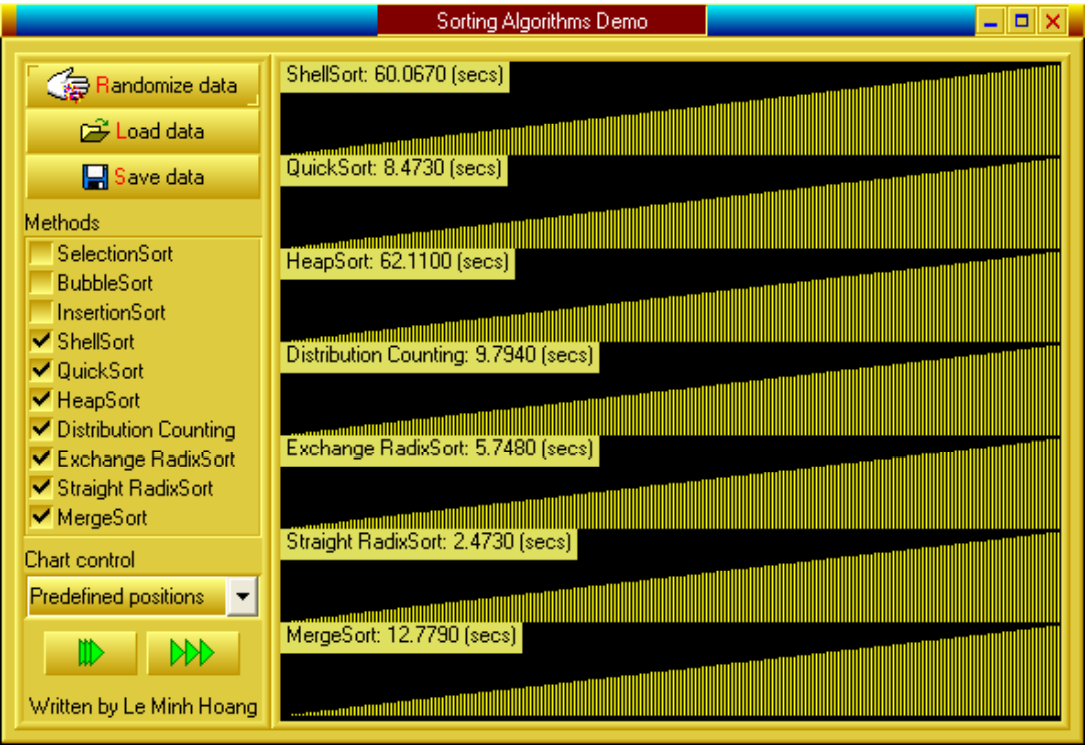
Những con số về thời gian và tốc độ chương trình đo được là qua thử nghiệm trên một bộ dữ liệu cụ thể, với một máy tính cụ thể và một công cụ lập trình cụ thể. Với bộ dữ liệu khác, máy tính và công cụ lập trình khác, kết quả có thể khác. Tuy vậy, việc đo thời gian thực thi của từng thuật toán sắp xếp vẫn cần thiết nếu ta muốn so sánh tốc độ của các thuật toán cùng cấp phức tạp bởi các tính toán trên lý thuyết đôi khi bị lệch so với thực tế vì nhiều lý do khác nhau.

Có một vấn đề đặt ra là ngoài những thuật toán sắp xếp cấp O(n2), rất khó có thể đo được tốc độ trung bình của những thuật toán sắp xếp còn lại khi mà chúng đều chạy không tới một nhịp đồng hồ thời gian thực (đều cho thời gian chạy bằng 0 do không kịp đo thời gian). Một cách giải quyết là cho mỗi thuật toán QuickSort, RadixSort, … thực hiện c lần (c là một số nguyên đủ lớn) trên các bộ dữ liệu ngẫu nhiên rồi lấy thời gian tổng chia cho c, hay có thể tăng kích thước dữ liệu (điều này có thể dẫn đến việc phải sửa lại một vài chỗ trong chương trình hoặc thậm chí phải thay đổi môi trường lập trình).

Tôi đã viết lại chương trình này trên Borland Delphi để đưa vào một số cải tiến:

* Có thể chạy với kích thước dữ liệu lớn hơn rất nhiều (hàng triệu khóa)
* Thiết kế dựa trên kiến trúc đa luồng (MultiThreads) cho phép chạy đồng thời hai hay nhiều thuật toán sắp xếp để so sánh tốc độ, hiển thị quá trình sắp xếp trực quan trên màn hình.
* Cũng cho phép chạy tuần tự các thuật toán sắp xếp để đo thời gian thực hiện chính xác của chúng.

Chú ý: Để chương trình không bị ảnh hưởng bởi các phần mềm khác đang chạy, khi bấm hoặc khởi động các threads, bàn phím, chuột và tất cả các phần mềm khác sẽ bị treo tạm thời đến khi các threads thực hiện xong. Vì vậy không nên chạy các thuật toán sắp xếp chậm với dữ liệu lớn, sẽ không thể đợi đến khi các threads kết thúc và sẽ phải tắt máy khởi động lại. Hình dưới đây là giao diện của chương trình, bạn có thể tham khảo mã nguồn chương trình kèm theo:



**Cài đặt các thuật toán sắp xếp với dữ liệu lớn**

Cùng một mục đích sắp xếp như nhau, nhưng có nhiều phương pháp giải quyết khác nhau. Nếu chỉ dựa vào thời gian đo được trong một ví dụ cụ thể mà đánh giá thuật toán này tốt hơn thuật toán kia về mọi mặt là điều không nên. Việc chọn một thuật toán sắp xếp thích hợp cho phù hợp với từng yêu cầu, từng điều kiện cụ thể là kỹ năng của người lập trình.

Những thuật toán có độ phức tạp O(n2) thì chỉ nên áp dụng trong chương trình có ít lần sắp xếp và với kích thước n nhỏ. Về tốc độ, BubbleSort luôn luôn đứng bét, nhưng mã lệnh của nó lại hết sức đơn giản mà người mới học lập trình nào cũng có thể cài đặt được, tính ổn định của BubbleSort cũng rất đáng chú ý. Trong những thuật toán có độ phức tạp O(n2), InsertionSort tỏ ra nhanh hơn những phương pháp còn lại và cũng có tính ổn định, mã lệnh cũng tương đối đơn giản, dễ nhớ. SelectionSort thì không ổn định nhưng với n nhỏ, việc chọn ra m phần tử nhỏ nhất có thể thực hiện dễ dàng chứ không cần phải sắp xếp lại toàn bộ như sắp xếp chèn.

Thuật toán đếm phân phối và thuật toán sắp xếp bằng cơ số nên được tận dụng trong trường hợp các khoá sắp xếp là số tự nhiên (hay là một kiểu dữ liệu có thể quy ra thành các số tự nhiên) bởi những thuật toán này có tốc độ rất cao. Thuật toán sắp xếp bằng cơ số cũng có thể sắp xếp dãy khoá có số thực hay số âm nhưng ta phải biết được cách thức lưu trữ các kiểu dữ liệu đó trên máy tính thì mới có thể làm được.

QuickSort, HeapSort, MergeSort và ShellSort là những thuật toán sắp xếp tổng quát, dãy khoá thuộc kiểu dữ liệu có thứ tự nào cũng có thể áp dụng được chứ không nhất thiết phải là các số.

QuickSort gặp nhược điểm trong trường hợp suy biến nhưng xác suất xảy ra trường hợp này rất nhỏ. HeapSort thì mã lệnh hơi phức tạp và khó nhớ, nhưng nếu cần chọn ra m phần tử lớn nhất trong dãy khoá thì dùng HeapSort sẽ không phải sắp xếp lại toàn bộ dãy. MergeSort phải đòi hỏi thêm một không gian nhớ phụ, nên áp dụng nó trong trường hợp sắp xếp trên file. Còn ShellSort thì hơi khó trong việc đánh giá về thời gian thực thi, nó là sửa đổi của thuật toán sắp xếp chèn nhưng lại có tốc độ tốt, mã lệnh đơn giản và lượng bộ nhớ cần huy động rất ít. Tuy nhiên, những nhược điểm của bốn phương pháp này quá nhỏ so với ưu điểm chung của chúng là nhanh. Hơn nữa, chúng được đánh giá cao không chỉ vì tính tổng quát và tốc độ nhanh, mà còn là kết quả của những cách tiếp cận khoa học đối với bài toán sắp xếp.

Những thuật toán trên không chỉ đơn thuần là cho ta hiểu thêm về một cách sắp xếp mới, mà kỹ thuật cài đặt chúng (với mã lệnh tối ưu) cũng dạy cho chúng ta nhiều điều: Kỹ thuật sử dụng số ngẫu nhiên, kỹ thuật "chia để trị", kỹ thuật dùng các biến với vai trò luân phiên v.v…Vậy nên nắm vững nội dung của những thuật toán đó, mà cách thuộc tốt nhất chính là cài đặt chúng vài lần với các ràng buộc dữ liệu khác nhau (nếu có thể thử được trên hai ngôn ngữ lập trình thì rất tốt) và cũng đừng quên kỹ thuật sắp xếp bằng chỉ số.