Báo cáo kết quả thực nghiệm

Tên : Nguyễn Đặng Nguyên Khang MSSV: 22520617

https://github.com/NgKhang3012/report-sort-Algorithm

Merge sort :

Ý tưởng

Sắp xếp trộn hoạt động kiểu đệ quy:

- Đầu tiên chia dữ liệu thành 2 phần, và sắp xếp từng phần.
- Sau đó gộp 2 phần lại với nhau. Để gộp 2 phần, ta làm như sau:
  - Tạo một dãy A mới để chứa các phần tử đã sắp xếp.
  - So sánh 2 phần tử đầu tiên của 2 phần. Phần tử nhỏ hơn ta cho vào A và xóa khỏi phần tương ứng.
  - Tiếp tục như vậy đến khi ta cho hết các phần tử vào dãy A.

Ưu điểm

- Chạy nhanh, độ phức tạp O(N* logN)
- Ổn định

Nhược điểm

- Cần dùng thêm bộ nhớ để lưu mảng A.

HeapSort

Ý tưởng

Ta lưu mảng vào CTDL Heap.

Ở mỗi bước, ta lấy ra phần tử nhỏ nhất trong heap, cho vào mảng đã sắp xếp.

Ưu điểm

- Cài đặt đơn giản nếu đã có sẵn thư viện Heap.
- Chạy nhanh, độ phức tạp O(N* logN)

Nhược điểm

- Không ổn định

QuickSort

Ý tưởng

- Chia dãy thành 2 phần, một phần "lớn" và một phần "nhỏ".
  - Chọn một khóa **pivot**
  - Những phần tử lớn hơn **pivot** chia vào phần lớn

    - Những phần tử nhỏ hơn hoặc bằng **pivot** chia vào phần nhỏ.
- Gọi đệ quy để sắp xếp 2 phần.

Ưu điểm

- Chạy nhanh (nhanh nhất trong các thuật toán sắp xếp dựa trên việc só sánh các phần tử). Do đó quicksort được sử dụng trong nhiều thư viện của các ngôn ngữ như Java, C++ (hàm sort của C++ dùng Intro sort, là kết hợp của Quicksort và Insertion Sort).

Nhược điểm

- Tùy thuộc vào cách chia thành 2 phần, nếu chia không tốt, độ phức tạp trong trường hợp xấu nhất có thể là O(N^2). Nếu ta chọn pivot ngẫu nhiên, thuật toán chạy với độ phức tạp trung bình là O(N* logN) (trong trường hợp xấu nhất vẫn là O(N^2)nhưng ta sẽ không bao giờ gặp phải trường hợp đó).
- Không ổn định.

std::sort in C++:

Hàm sort có thể sắp xếp các dạng dữ liệu khác nhau như vector, string, array,… và có thể chỉ định phương thức sắp xếp giảm dần hoặc sử dụng custom comparator function.

Ưu điểm của hàm sort trong c++

1. Hiệu quả: Hàm sort trong C++ có thể sắp xếp các dãy dữ liệu lớn một cách nhanh chóng và hiệu quả bằng cách sử dụng các thuật toán sắp xếp như Quick sort, Merge sort.
2. Tiện dụng: Hàm sort rất dễ sử dụng và tiện dụng, chỉ cần truyền vào 2 con trỏ đến vị trí bắt đầu và kết thúc của dãy cần sắp xếp.
3. Tùy chỉnh: Hàm sort có thể chỉ định phương thức sắp xếp giảm dần hoặc sử dụng custom comparator function để sắp xếp theo yêu cầu cụ thể.
4. Tương thích: Hàm sort có thể sắp xếp các dạng dữ liệu khác nhau như vector, string, array,…
5. Tiêu chuẩn: Hàm sort là một phần của thư viện STL (Standard Template Library) của C++, nên được sử dụng rộng rãi và được kiểm soát chặt chẽ về chất lượng và tính nhất quán.

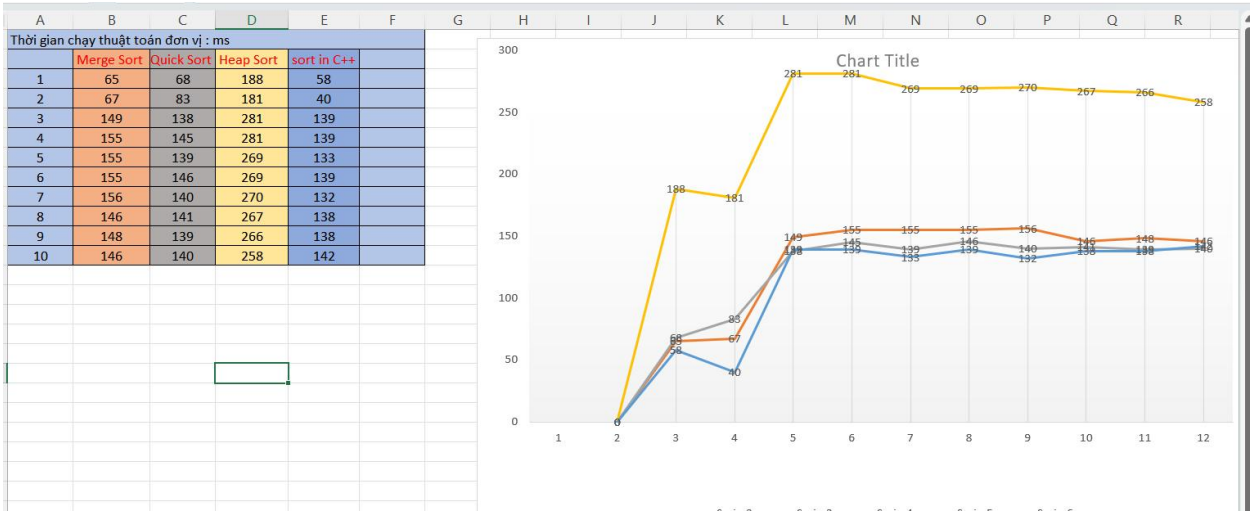
6. ĐPT O(N*logN)

Thực nghiệm đánh giá với 10 bộ test case được sinh ngẫu nhiên: bộ dữ liệu gồm 10 dãy, mỗi dãy khoảng 1 triệu số thực (ngẫu nhiên); dãy thứ nhất đã có thứ tự tăng dần, dãy thứ hai có thứ tự giảm dần, 8 dãy còn lại trật tự ngẫu nhiên

Bằng 1 đoạn code gồm 4 thuật toán: Quick Sort, Heap Sort, Merge Sort, std::sort()

Kết quả:

Biểu đồ:



Thời gian chạy thuật toán đơn vị : ms

	Merge Sort	Quick Sort	Heap Sort	sort in C++
1	65	68	188	58
2	67	83	181	40
3	149	138	281	139
4	155	145	281	139
5	155	139	269	133
6	155	146	269	139
7	156	140	270	132
8	146	141	267	138
9	148	139	266	138
10	146	140	258	142

Nhận xét:

- Thuật toán Heap Sort có thời gian chạy lâu nhất trong 4 thuật toán do chi phí hằng số lớn. Khi mảng sắp xếp tăng dần thì việc xây dừng cấu trúc dữ liệu Heap trở nên tốn thời gian hơn do các phần tử trong Heap liên tục bị đổi gốc. Ngược lại khi dãy sắp xếp giảm dần thì thuật toán chạy hiêu quả và nhanh hơn.

- Hàm sort của thư viện chuẩn C++ có tốc độ chạy nhanh nhất bởi lẽ đã được tối ưu bằng việc kết hợp nhiều thuật toán khác nhau.

-Merge Sort và Quick Sort có thời gian chạy tương đối nhanh, Quick Sort chọn khóa ngẫu nhiên nên không ổn định bằng Merge Sort

Kết luận:

Thông qua việc thực nghiệm ta có thể biết được cơ chế và thời gian chạy của các thuật toán sắp xếp. Qua đó tùy theo mục đích sử dụng ta sẽ cài đặt thuật toán phù hợp với chương trình