Báo cáo thực nghiệm các thuật toán sắp xếp

Người thực hiện: Vũ Ngọc Quốc Khánh

Giới thiệu

Báo cáo về thời gian thực thi của các thuật toán sắp xếp như Quicksort, Heapsort, Mergesort và hàm `std::sort()` trong thư viện chuẩn của C++.

Chuẩn bị

- Bộ dữ liệu kiểm thử gồm 10 dãy số thực, trong đó có 1 dãy số tăng dần, 1 dãy số giảm dần và 8 dãy số ngẫu nhiên
- Code cài đặt Quicksort
- Code cài đặt Heapsort
- Code cài đặt Mergesort
- Code sử dụng `std::sort` trong thư viện chuẩn của C++.
- Code thực hiện thuật toán và đo thời gian.

Tất cả các code và tệp tin liên quan đều được để trong **Github này**

Bộ dữ liệu kiểm tra

Được sinh ngẫu nhiên bằng code sử dụng thư viện Testlib của Mike Mirzayanov - Người sáng lập Codeforces

Phân tích các thuật toán sắp xếp

Quicksort

Mô tả

Đây là thuật toán sắp xếp theo hướng tiếp cận Chia để trị với ý tưởng như sau:

1. Chọn 1 phần tử trong mảng cần sắp xếp làm chốt `pivot`.
2. Chia mảng thành 2 phần với điểm chính giữa là `pivot`, các phần tử nhỏ hơn `pivot` sẽ nằm ở một phần và ngược lại, các phần tử lớn hơn `pivot` sẽ nằm ở phần còn lại.
3. Với mỗi phần đã chia, ta thực hiện lại bước 1.

Qua nhiều lần sửa đổi thì đã có nhiều cách chọn `pivot` xuất hiện như: chọn phần tử đầu tiên/cuối cùng, chọn phần tử chính giữa, chọn phần tử trung vị. Nhưng tất cả cách chọn đó đều có thể dẫn đến trường hợp tệ nhất. Nên, cách chọn `pivot` trong bài viết này là chọn ngẫu nhiên

Đánh giá

- **Ưu điểm**
    - Trong đa số trường hợp, thuật toán quicksort có độ phức tạp là $\mathcal{O}(NlogN)$, khá nhanh và cài đặt đơn giản.
- **Nhược điểm**
    - Trong trường hợp tệ nhất, thuật toán quicksort có độ phức tạp là $\mathcal{O}(N^2)$

Mergesort

Mô tả

Đây là thuật toán sắp xếp theo hướng tiếp cận Chia để trị với ý tưởng như sau:

1. Giả sử ta có 2 mảng con A và B đã được sắp xếp, ta tạo thêm một mảng C để "trộn" 2 mảng A và B sao cho sau khi "trộn" thì mảng C cũng được sắp xếp.
2. Nếu mảng A hoặc B chưa được sắp xếp thì ta chia mảng chưa sắp xếp thành 2 phần và quay lại bước 1.

Đánh giá

- **Ưu điểm**
    - Thuật toán chạy ổn định với độ phức tạp là $\mathcal{O}(N \log N)$
- **Nhược điểm**
    - Ta phải sử dụng thêm một mảng trung gian để trộn nên tốn nhiều bộ nhớ nếu dữ liệu lớn

Heapsort

Mô tả

Là một thuật toán sắp xếp được thiết kế trên cấu trúc dữ liệu Heap

Heapsort sẽ thực hiện 2 công việc khác nhau:

- **Tạo heap**
    - Thực hiện việc tạo ra cấu trúc max-heap từ một mảng đã có sẵn
    - Công việc tạo heap sẽ được thực hiện như sau:
    - 1. Ta có đỉnh cha i và 2 đỉnh con $l = i \times 2 + 1$, $r = i \times 2 + 2$.
    - 2. Kiểm tra trong 3 đỉnh có giá trị lớn nhất.
    - 3. Nếu đỉnh có giá trị lớn nhất không phải đỉnh cha, ta thực hiện đổi chỗ 2 đỉnh với nhau và thực hiện việc tạo heap với cây con tại vị trí đỉnh vừa đổi chỗ
- **Sắp xếp**
    - Sau khi đã có được một cấu trúc max-heap như mong muốn, ta thực hiện việc đổi thứ tự các đỉnh nhỏ nhất và tạo heap lại với độ lớn nhỏ hơn ở các đỉnh chưa được lấy ra.

Trong C++, đã có sẵn cấu trúc dữ liệu Heap trong thư viện chuẩn như `priority_queue` sử dụng Fibonacci Heap, ta chỉ cần khai báo để sử dụng.

Đánh giá

- **Ưu điểm**
    - Không tốn bộ nhớ nhiều như Mergesort
    - Độ phức tạp là $\mathcal{O}(N \log N)$
- **Nhược điểm**
    - Không được ổn định

`std::sort` của thư viện chuẩn C++

Mô tả

Đây là thuật toán Introsort sắp xếp được kết hợp bởi 3 thuật toán sắp xếp khác nhau là Quicksort, Heapsort và Selectionsort. Các thuật toán như Heapsort và Selectionsort được sử

dụng để loại bỏ các trường hợp tệ nhất của quicksort.

Cài đặt

Đã có sẵn hàm `std::sort` trong thư viện `algorithm` của C++:

```
std::sort(a, a + n);
```

Thực nghiệm

Code thực hiện thuật toán sắp xếp và đo thời gian:

Code thực hiện những công việc sau:

1. Đọc dữ liệu từ 10 bộ test
2. Với từng bộ test, thực hiện khởi chạy lần lượt các thuật toán
3. Kiểm tra xem mảng đã sắp xếp hay chưa
4. Lấy thông tin thời gian trước và sau khi khởi chạy thuận toán sắp xếp sau đó tính độ chênh lệch
5. In thông báo kết quả ra màn hình

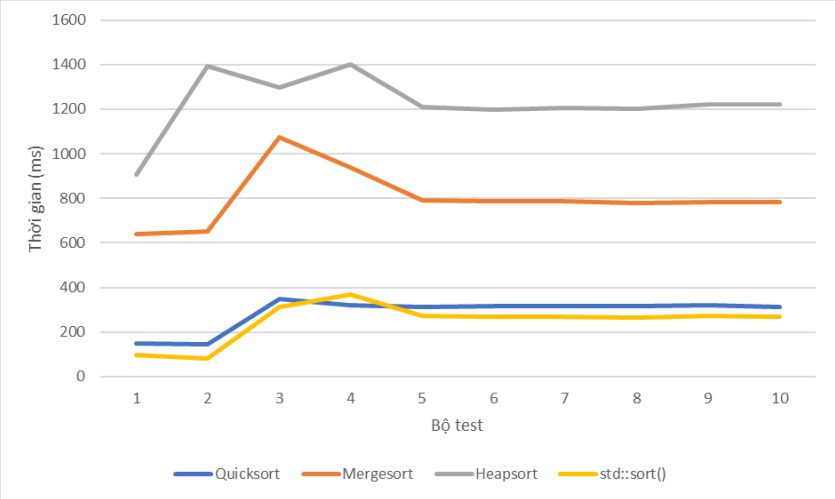
Kết quả thực nghiệm

Khi chạy trên máy tính cá nhân (Intel(R) Core(TM) I5-6300U @ 2.4GHz)

Bảng dữ liệu thời gian thực hiện (Đơn vị ms)

Bộ test	Quicksort	Mergesort	Heapsort	std::sort()
1	149.600	637.335	906.592	97.7447
2	145.610	650.262	1394.27	80.7842
3	350.064	1073.17	1296.54	311.167
4	322.179	936.495	1400.30	367.018
5	311.211	791.842	1210.77	271.275
6	316.155	787.894	1195.81	267.329
7	318.150	785.938	1203.78	270.274
8	317.151	776.965	1201.79	266.290
9	321.142	782.905	1219.70	271.273
10	312.208	781.946	1222.73	268.244

Biểu đồ dữ liệu

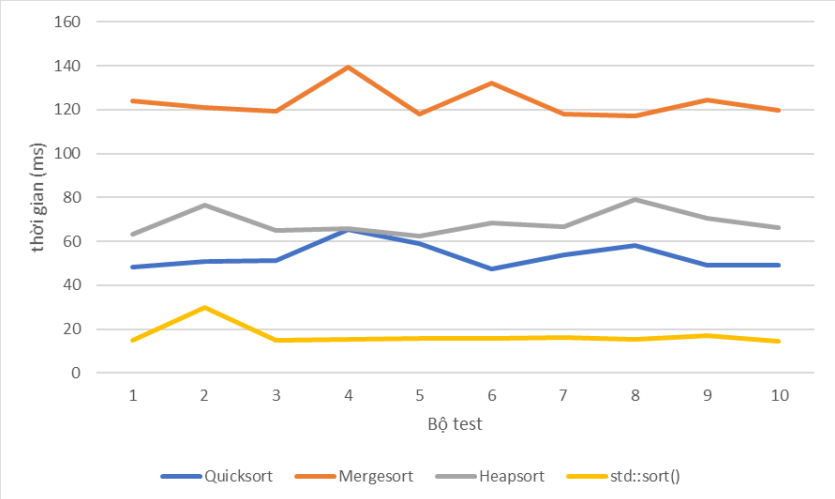


Khi chạy trên Google Colab (Intel(R) Xeon(R) CPU @ 2.20GHz)

Bảng dữ liệu thời gian thực hiện (Đơn vị ms)

Bộ test	Quicksort	Mergesort	Heapsort	std::sort()
1	48.2204	124.061	63.2064	15.0937
2	50.6284	120.826	76.6592	30.0654
3	51.4487	119.376	64.9850	15.0266
4	65.1794	139.575	65.6171	15.1032
5	58.8711	117.969	62.4007	15.7191
6	47.6169	132.112	68.1981	15.6353
7	53.7565	118.071	66.7054	16.3768
8	58.1719	117.169	78.9908	15.2907
9	48.9238	124.349	70.4186	17.1982
10	49.2638	119.653	66.2852	14.2872

Biểu đồ dữ liệu



Tổng kết, nhận xét:

- Trong 4 thuật toán sắp xếp, `std::sort()` trong thư viện chuẩn của C++ thể hiện được khả năng xử lý rất nhanh và ổn định qua các bộ test.
- Quicksort cũng cho thấy một độ ổn định ở mức chấp nhận được.
- Heapsort và Mergesort xử lý chậm và kém ổn định hơn rất nhiều so với Quicksort và `std::sort()`.
- Ngoài ra, việc cài đặt Quicksort, Heapsort và `std::sort()` bằng ngôn ngữ C++ tương đối đơn giản. Dễ sử dụng giúp cho các lập trình có thể dễ dàng cài đặt và sử dụng.