Lớp: IT003.Q21.CTTN

BÁO CÁO KẾT QUẢ THỬ NGHIỆM

Sinh viên thực hiện: Nguyễn Toàn Thắng - 25521679

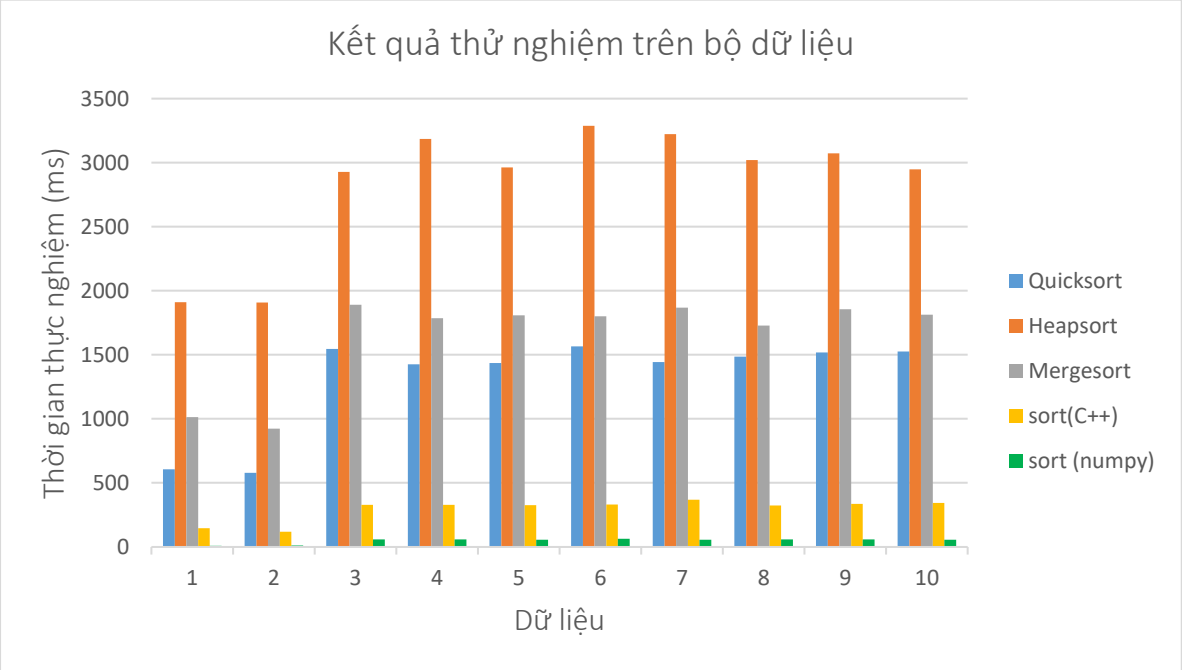
Nội dung báo cáo:

I. Kết quả thử nghiệm

1. Bảng thời gian thực hiện[1]

Dữ liệu	Thời gian thực hiện (ms)				
	Quicksort	**Heapsort**	**Mergesort**	**sort (C++)**	**sort (numpy)**
1	606.03	1910.73	1013.45	146.55	9.49
2	579.91	1907.07	923.15	118.72	12.58
3	1544.82	2928.44	1891.39	328.39	58.8
4	1425.07	3184.08	1785.13	329.13	58.43
5	1435.61	2962.39	1808.49	326.48	55.68
6	1565.02	3288.28	1801.64	330.92	63.82
7	1443.48	3223.76	1867.61	367.62	55.87
8	1486.44	3020.48	1729.28	324.06	59.26
9	1517.37	3073.37	1855.45	336.74	57.82
10	1526.89	2948.79	1813.38	342.74	56.43
Trung bình	1313.06	2844.74	1655.20	295.13	48.82

2. Biểu đồ (cột) thời gian thực hiện



II. Kết luận:

[1] Số liệu chỉ mang tính minh họa

Từ kết quả thực nghiệm, ta rút ra được những nhận xét và kết luận như sau:

- **Sự chênh lệch về tốc độ (thời gian thực hiện):**
    - **Nhóm dẫn đầu (Hiệu suất cao): sort (numpy)** đứng đầu về tốc độ với thời gian trung bình chỉ 48.82 ms, theo sát là **sort (C++)** với 295.13 ms. Sự chênh lệch giữa nhóm này và nhóm chậm nhất lên đến hơn 50 lần.
    - ⇨ Numpy đạt tốc độ nhanh nhất vì thực chất nó gọi các hàm sắp xếp đã được tối ưu hóa cực hạn bằng ngôn ngữ C/Fortran bên dưới. C++ nhanh vì được biên dịch trực tiếp ra mã máy, loại bỏ bước trung gian.
    - **Nhóm trung bình (Tự cài đặt):** Vị trí thứ 3 và thứ 4 lần lượt thuộc về **Quicksort** (1313.06 ms) và **Mergesort** (1655.20). Tuy kết quả khá tương đồng nhưng **Quicksort** vẫn cho thấy ưu thế về tốc độ xử lý nhanh hơn **Mergesort** khoảng 20%.
    - ⇨ Trong khi **Quicksort** hiệu quả nhờ cơ chế phân vùng tốt, đặc biệt nhanh khi dữ liệu đã có sẵn thứ tự, giúp giảm thiểu số phép hoán đổi cần thiết, thì **Mergesort** ổn định nhưng tốn thời gian cho việc cấp phát bộ nhớ để tạo các mảng phụ và thao tác trộn (merge).
    - **Nhóm thấp nhất: Heapsort** duy trì mức thời gian cao nhất trong tất cả các thử nghiệm với trung bình 2844.74 ms, cho thấy đây là thuật toán có chi phí thực hiện lớn nhất trên môi trường Python.
    - ⇨ **Heapsort** chậm nhất do cách truy cập dữ liệu không tuần tự trên cấu trúc cây (Heap), khiến CPU thường xuyên gặp lỗi vì không tìm thấy dữ liệu sẵn trong bộ nhớ đệm, làm tăng thời gian đọc RAM.

- **Biến động theo tính chất dữ liệu:**
    - Mặt khác, ta thấy có sự sụt giảm thời gian đáng kể ở bộ dữ liệu số 1 và số 2 (dãy đã tăng/giảm dần) đối với hầu hết các thuật toán. Điển hình là **Quicksort** giảm từ mức ~1500 ms (ngẫu nhiên) xuống còn khoảng ~600 ms (có thứ tự).
    - Ngược lại, các bộ dữ liệu từ số 3 đến số 10 (ngẫu nhiên) chứng kiến thời gian thực hiện tăng vọt và duy trì ở mức cao ổn định, phản ánh đúng độ phức tạp thực tế khi xử lý các dãy số lộn xộn.

III. Thông tin chi tiết – link github, trong repo gibub cần có

1. Báo cáo
2. Mã nguồn
3. Dữ liệu thử nghiệm

https://github.com/Meow-advocate/25521679_DSA_1.git