Lớp: IT003.Q21.TTNT

BÁO CÁO KẾT QUẢ THỬ NGHIỆM

Sinh viên thực hiện: Lê Gia Huy – 25520693

Nội dung báo cáo:

I. Kết quả thử nghiệm

1. Bảng thời gian thực hiện

- Hiệu năng của các thuật toán sắp xếp được đo thời gian thực thi trên 10 bộ dữ liệu thử nghiệm, mỗi bộ gồm 1.000.000 phần tử. Các bộ dữ liệu bao gồm:
    - 2 bộ dữ liệu tăng dần (1 và 2)
    - 2 bộ dữ liệu giảm dần (3 và 4)
    - 6 bộ dữ liệu ngẫu nhiên (các bộ dữ liệu còn lại)
    - Bao gồm cả kiểu số nguyên (int) và số thực (float)
- Thời gian thực thi được đo bằng đơn vị giây (seconds). Kết quả thu được được trình bày trong bảng dưới đây.

Dữ liệu	Thời gian thực hiện (s)							
	Heap Sort	Shell Sort	Merge Sort	Quick Sort	Couting Sort	Radix Sort	Python Sorted	Numpy Sort
1	17.830365	6.234191	7.412639	3.374786	0.79102	7.21834	0.06859	0.04852
2	16.599470	5.418253	6.610452	3.272434	X	X	0.06889	0.08168
3	17.611144	6.176849	7.213179	3.317055	0.78897	7.06446	0.06928	0.04540
4	16.113183	5.354097	6.584244	3.231875	X	X	0.06843	0.08009
5	17.229435	25.546746	8.962811	5.995857	0.95762	7.20302	0.65258	0.05083
6	15.824230	21.878299	8.381768	5.615647	X	X	0.71740	0.08391
7	17.153262	24.100167	9.101248	5.974849	0.93873	7.07139	0.69757	0.04985
8	15.919061	22.483328	8.301102	5.612605	X	X	0.71874	0.08556
9	17.110714	24.430629	9.187240	6.115813	0.93236	6.92218	0.69492	0.05144
10	15.581897	20.194699	8.141267	5.511600	X	X	0.67311	0.08318

Bảng đo lường hiệu năng thuật toán

!Chú thích:

*Dấu “X” thể hiện cho việc các thuật toán không thể thực hiện trên bộ dữ liệu số thực (float) nên không có dữ liệu

*Mỗi thuật toán được thực hiện độc lập trên cùng một tập dữ liệu để đảm bảo tính khách quan trong quá trình so sánh.

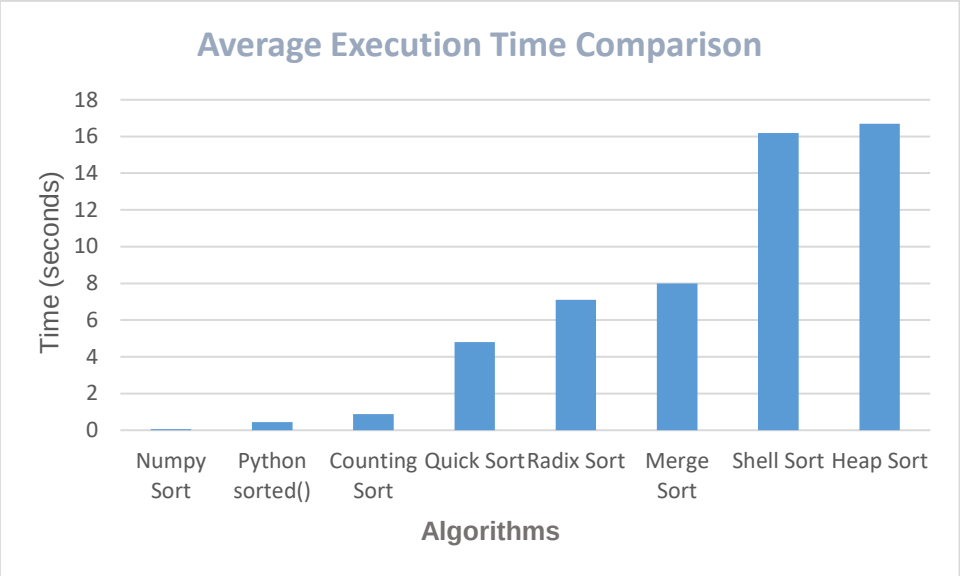
2. Biểu đồ (cột) thời gian thực hiện

- Để trực quan hóa sự khác biệt về hiệu năng giữa các thuật toán sắp xếp, em sử dụng biểu đồ cột thể hiện **thời gian thực thi trung bình (Average Execution Time)** của từng thuật toán trên 10 bộ dữ liệu thử nghiệm.

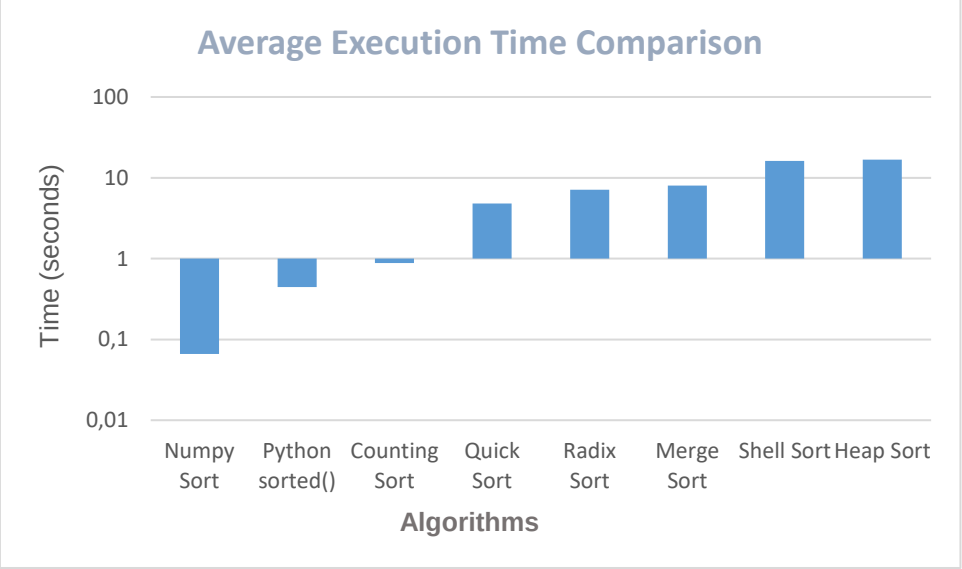
Thuật toán	Heap Sort	Shell Sort	Merge Sort	Quick Sort	Couting Sort	Radi Sort	Python Sorted()	Numpy Sort
Average	16,327237	20,82259545	8,404801	5,51056753	0,907056	7,0090366	0,63243568	0,07156208

Bảng thống kê thời gian chạy trung bình các thuật toán

- Do sự chênh lệch lớn giữa các thuật toán, trục tung (Y-axis) có thể được biểu diễn theo **thang đo logarit** nhằm thể hiện rõ sự khác biệt về mức độ hiệu năng.



Biểu đồ cột biểu diễn không dùng Log Scale.



Biểu đồ cột biểu diễn có dùng Log Scale.

II. Phân tích

Dựa trên kết quả đo lường thời gian thực thi của 8 phương pháp sắp xếp trên 10 bộ dữ liệu thử nghiệm (mỗi bộ 1.000.000 phần tử , bao gồm cả số nguyên và số thực), ta có

thể phân tích hiệu năng từ bức tranh tổng quan đến bản chất chi tiết của từng nhóm thuật toán như sau:

1. Đánh giá bao quát

- Thử nghiệm cho thấy sự phân hóa rõ rệt về hiệu năng giữa ba nhóm: **hàm thư viện, thuật toán phi so sánh và thuật toán dựa trên so sánh**, theo thứ tự giảm dần. Sự khác biệt chủ yếu đến từ độ phức tạp Big O và đặc tính của dữ liệu đầu vào như mức độ ngẫu nhiên hay kiểu dữ liệu (int/float). Bên cạnh đó, hiệu năng thực tế còn chịu ảnh hưởng bởi yếu tố hệ thống như cache locality hoặc chi phí cấp phát bộ nhớ, khiến một số thuật toán có hành vi chậm hơn so với dự đoán lý thuyết.

2. Phân tích chi tiết từng nhóm thuật toán (Theo thứ tự giảm dần hiệu năng)

2.1. Nhóm Built-in:

- **Numpy Sort** (0.04s - 0.08s) và **Python sorted()** (0.06s - 0.71s): Đây là hai thuật toán có hiệu suất tốt nhất

    - *Phân tích:* Hiệu năng của **NumPy** không đến từ một thuật toán sắp xếp đặc biệt, mà từ việc thư viện này được viết bằng C và thao tác trực tiếp trên vùng nhớ, giúp loại bỏ phần lớn overhead của Python. Ngược lại, hàm **sorted()** của **Python** sử dụng **Timsort** — thuật toán lai giữa Merge Sort và Insertion Sort — vốn tối ưu nhờ khả năng nhận diện và khai thác các đoạn dữ liệu đã có tính sắp xếp trước.

2.2. Nhóm Thuật toán Phi so sánh:

- **Counting Sort** (0.7s - 0.9s) và **Radix Sort** (6.9s - 7.2s) : Trên các mảng số nguyên (dữ liệu 1, 3, 5, 7, 9), Counting Sort cho tốc độ vượt trội so với mọi thuật toán tự cài đặt khác do có độ phức tạp tuyến tính $O(N+K)$.

    - *Phân tích:* Mặc dù rất nhanh, điểm hạn chế lớn của nhóm thuật toán này là sự phụ thuộc mạnh vào kiểu dữ liệu. Với các mảng số thực (ví dụ: 2, 4, 6, 8, 10), thuật toán không thể vận hành do không thể sử dụng giá trị thập phân làm chỉ mục cho mảng đếm. Ngoài ra, nếu dải giá trị quá lớn, **Counting Sort** dễ dẫn đến tiêu tốn bộ nhớ vượt mức cho phép, gây lỗi tràn bộ nhớ.

2.3. Nhóm Thuật toán chia để trị:

- **Quick Sort** (3.2s - 6.1s) và **Merge Sort** (6.5s - 9.1s): Về mặt lý thuyết, cả hai đều có độ phức tạp trung bình là $O(N \log N)$. Tuy nhiên, thực tế chứng minh Quick Sort luôn nhanh hơn Merge Sort khoảng 1.5 đến 2 lần.

    - *Phân tích:* Hiệu năng của **Quick Sort** chủ yếu đến từ khả năng tận dụng **cache locality**. Vì là thuật toán *in-place,* **Quick Sort** thao tác trực tiếp trên mảng gốc nên dữ liệu được truy cập liên tục trong bộ nhớ đệm, giúp CPU xử lý rất hiệu quả. Ngược lại, **Merge Sort** phải cấp phát thêm mảng phụ có kích thước O(N), khiến dữ liệu liên tục di chuyển giữa RAM và các vùng nhớ tạm. Việc cấp phát, sao chép và giải phóng bộ nhớ này tạo ra chi phí đáng kể,

khiến **Merge Sort** thường chậm hơn trong thực tế dù có độ phức tạp lý thuyết tối ưu.

2.4. Nhóm Thuật toán bất ổn định

- **Shell Sort** (5.3s - 25.5s): Kết quả thực nghiệm cho thấy hiệu năng của **Shell Sort** biến thiên rất mạnh. Với bốn bộ dữ liệu có trật tự (tăng dần hoặc giảm dần), thời gian xử lý tương đối ổn định, dao động khoảng 5–6 giây. Tuy nhiên, khi chuyển sang sáu bộ dữ liệu ngẫu nhiên, thời gian thực thi tăng đáng kể và đạt mức 25.5 giây – cao nhất trong toàn bộ thí nghiệm.
- **Heap Sort** (15.5s - 17.8s): Mặc dù duy trì độ phức tạp lý thuyết O(N log N) và không bị suy biến, **Heap Sort** lại nằm trong nhóm thuật toán chậm nhất theo kết quả thực nghiệm.
- *Phân tích:* Có thể thấy thuật toán **Shell Sort** có sự dao động lớn về hiệu năng do sự phụ thuộc lớn vào số lượng nghịch thế (inversions) trong mảng. Trong khi đó, hiệu năng của **Heap Sort** bị ảnh hưởng mạnh bởi kiến trúc phần cứng. Quá trình heapify yêu cầu truy cập bộ nhớ theo dạng "nhảy chỉ số" (từ nút cha i sang các nút con 2i + 1), khiến dữ liệu phân tán và liên tục gây ra các lần truy cập cache không hiệu quả (cache misses). Do đó, thời gian xử lý duy trì ở mức cao (15–17 giây), gần như không phụ thuộc vào việc dữ liệu đã có trật tự trước hay chưa.

III. Kết luận

- Qua kết quả thực nghiệm, có thể thấy hiệu năng thuật toán sắp xếp trong thực tế không chỉ phụ thuộc vào độ phức tạp Big O mà còn chịu ảnh hưởng mạnh bởi cách cài đặt, đặc tính dữ liệu và kiến trúc phần cứng.
- Các hàm built-in như **NumPy sort** và **Python sorted()** cho hiệu suất vượt trội nhờ được tối ưu ở mức hệ thống. **Counting Sort** rất nhanh khi điều kiện dữ liệu phù hợp nhưng thiếu tính tổng quát. Trong nhóm O(N log N), **Quick Sort** hoạt động hiệu quả hơn **Merge Sort** và **Heap Sort** nhờ tính in-place và khả năng tận dụng cache.
- Từ đó có thể kết luận rằng việc lựa chọn thuật toán cần dựa trên đặc thù dữ liệu và môi trường thực thi, thay vì chỉ dựa vào độ phức tạp lý thuyết.

IV. Thông tin chi tiết

1. Link Github: https://github.com/Ngaostzy/project_DSA_sortAgorithms_02-2026.git