

## BÁO CÁO KẾT QUẢ THỬ NGHIỆM

Sinh viên thực hiện: Nguyễn Lưu Hải Anh

Nội dung báo cáo:

### *I. Kết quả thử nghiệm*

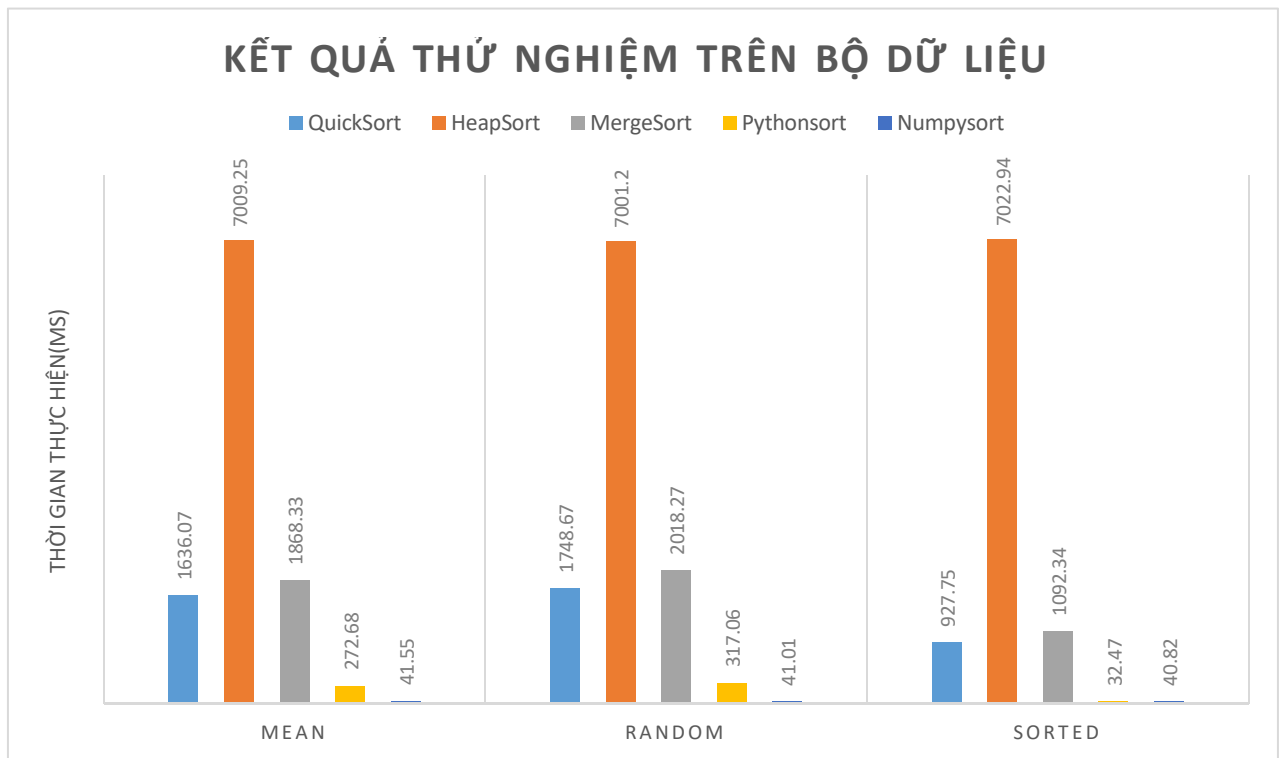
#### *1. Bảng thời gian thực hiện<sup>1</sup>*

Dữ liệu	Thời gian thực hiện (ms)				
	Quicksort	Heapsort	Mergesort	sort (python)	sort (numpy)
1	927.75	7022.94	1092.34	32.47	40.82
2	921.41	6528.25	1098.80	32.13	40.07
3	1748.67	7001.20	2018.27	317.06	41.01
4	1844.61	7109.48	2071.14	331.16	41.74
5	1769.98	7045.96	2060.30	394.29	41.97
6	1863.18	7043.25	2059.88	315.59	40.95
7	1881.26	7072.58	2150.01	328.97	42.02
8	1767.99	7185.93	2063.37	318.44	42.57
9	1886.00	7233.83	2088.06	315.21	41.63
10	1749.84	6849.04	1981.12	340.50	42.72
Trung bình	1636.07	7009.25	1868.33	272.68	41.55

#### *2. Biểu đồ (cột) thời gian thực hiện*

---

<sup>1</sup> Số liệu chỉ mang tính minh họa



## II. Kết luận:

Dựa trên bảng kết quả đo lường thực tế cho 1.000.000 phần tử, chúng ta có thể rút ra những kết luận chuyên sâu về hiệu suất của các thuật toán trong môi trường Python và thư viện NumPy như sau:

### Quick Sort (~1636.07ms) - Hiệu quả nhất trong nhóm tự viết

- **Ưu điểm:** Tốc độ xử lý tốt nhất nhờ tận dụng *List Comprehension* của Python, giúp việc phân đoạn (partitioning) diễn ra rất nhanh.
- **Nhược điểm:** Tốn bộ nhớ hơn bản gốc vì phải tạo ra các danh sách phụ (left, mid, right) liên tục.

### Merge Sort (~1868.33ms) - Độ ổn định cao nhất

- **Đặc điểm:** Thời gian chạy giữa các file ngẫu nhiên và file đã sắp xếp không chênh lệch quá nhiều.
- **Kết luận:** Đây là lựa chọn "an toàn" nếu bạn không biết trước đặc điểm của dữ liệu đầu vào, vì nó luôn đảm bảo độ phức tạp  $O(n \log n)$  mà không sợ bị suy biến như Quick Sort.

### Heap Sort (~7009.25ms) - Điểm yếu của Python

- **Vấn đề:** Mặc dù về lý thuyết Heap Sort có độ phức tạp giống Merge Sort, nhưng việc triển khai heapify bằng đệ quy khiến nó cực kỳ chậm trong Python.
- **Nguyên nhân:** Mỗi lần đổi chỗ và gọi đệ quy trong vòng lặp 1 triệu phần tử tạo ra một gánh nặng khổng lồ cho CPU.

**Numpy\_sort (41.55ms):** Do được viết bằng ngôn ngữ C và tối ưu hóa mức hệ thống, nó nhanh gấp 40 lần Quick Sort và 170 lần Heap Sort. Điều này khẳng định: Với dữ liệu lớn dạng mảng, hãy luôn dùng NumPy.

**Python\_sort (272.68ms):** Mặc dù chậm hơn NumPy, nhưng nó cho thấy khả năng thích nghi cực tốt. Đặc biệt ở dữ liệu đã sắp xếp (file\_01), nó chỉ mất ~33ms, thậm chí nhanh hơn cả NumPy trong một số khoảng khắc nhờ thuật toán Timsort có khả năng phát hiện các đoạn dữ liệu có thứ tự sẵn.

Bảng kết quả cho thấy dữ liệu ảnh hưởng thế nào đến tốc độ:

- **Dữ liệu có thứ tự :** Python\_sort và Quick Sort chạy nhanh nhất ở đây. Đặc biệt Quick Sort giảm từ ~1800ms xuống còn ~900ms.
- **Dữ liệu ngẫu nhiên (Random):** Thời gian tăng lên đáng kể đối với tất cả các thuật toán tự viết do phải thực hiện nhiều phép so sánh và hoán đổi hơn để thiết lập trật tự.

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

1. Dữ liệu thử nghiệm:

[https://drive.google.com/drive/folders/1QW34EbU1cqGrfmCRbK5nwL\\_vsHPCxA4Ms?usp=drive\\_link](https://drive.google.com/drive/folders/1QW34EbU1cqGrfmCRbK5nwL_vsHPCxA4Ms?usp=drive_link)

2. Github bài tập: [https://github.com/HaiAnh08102007/sorting\\_algorithms](https://github.com/HaiAnh08102007/sorting_algorithms)