

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



BÀI BÁO CÁO
MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ
GIẢI THUẬT
ĐỀ TÀI: THỰC NGHIỆM CÁC GIẢI
THUẬT SẮP XẾP NỘI

Giảng viên môn học: ThS. Nguyễn Thanh Sơn

Sinh viên thực hiện: Nguyễn Ngọc Thuận

Mã số sinh viên: 25521819

Lớp: IT003.Q21.TTNT

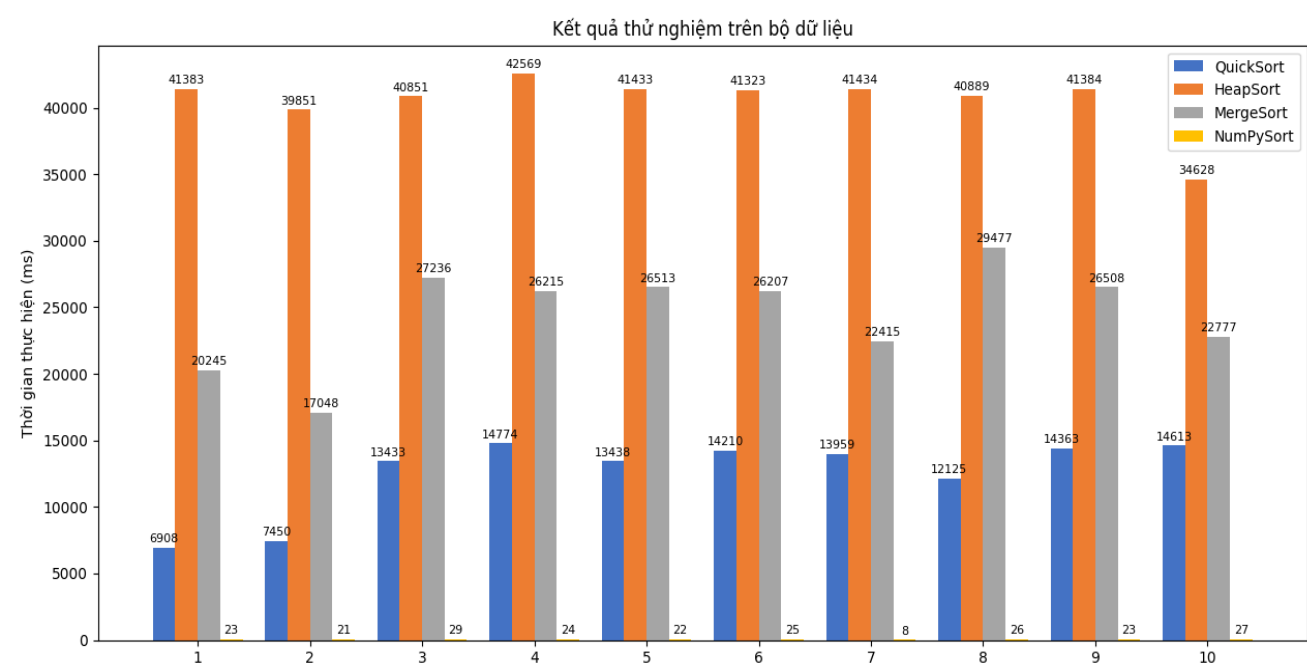
BÁO CÁO KẾT QUẢ THỬ NGHIỆM

I. Kết quả thử nghiệm

1. Bảng thời gian thực hiện

Dữ liệu	Thời gian thực hiện (ms)			
	QuickSort	HeapSort	MergeSort	NumpySort
1	6908	41383	20245	23
2	7450	39851	17048	21
3	13433	40851	27236	29
4	14774	42569	26215	24
5	13438	41433	26513	22
6	14210	41323	26207	25
7	13959	41434	22415	8
8	12125	40889	29477	26
9	14363	41384	26508	23
10	14613	34628	22777	27
Trung bình	12527.3	40574.5	24464.1	22.8

2. Biểu đồ (cột) thời gian thực hiện



II. Kết luận:

1. Về hiệu năng tổng thể

- Kết quả thực nghiệm cho thấy hàm `sort()` của NumPy có hiệu năng vượt trội, với thời gian trung bình chỉ khoảng 22.8 ms, nhanh hơn khoảng 500–1800 lần so với các thuật toán tự cài đặt bằng Python. Nguyên nhân là do NumPy được triển khai ở tầng thấp bằng ngôn ngữ C, tối ưu bộ nhớ liên tục, tận dụng tốt cache CPU và giảm thiểu overhead của Python.
- Trong ba thuật toán tự cài đặt, thứ tự tốc độ từ nhanh đến chậm là: QuickSort > MergeSort > HeapSort.

2. Đánh giá các thuật toán

- **QuickSort:** Có thời gian trung bình khoảng 12.5 giây và là thuật toán nhanh nhất trong nhóm tự cài đặt. Đặc biệt với dữ liệu đã có thứ tự tăng dần hoặc giảm dần, thời gian thực thi thấp hơn đáng kể. Điều này chứng tỏ việc chọn chốt (pivot) phù hợp sẽ giúp thuật toán hoạt động hiệu quả và tránh được trường hợp xấu nhất $O(N^2)$.
- **MergeSort:** Hoạt động rất ổn định với thời gian trung bình khoảng 24.5s. Thời gian thực hiện không thay đổi nhiều giữa các bộ dữ liệu khác nhau, đúng với tính chất lý thuyết là luôn đảm bảo độ phức tạp $O(N \log N)$ trong mọi trường hợp và không phụ thuộc vào thứ tự ban đầu của dữ liệu.
- **HeapSort:** Có thời gian thực thi chậm nhất trong thực nghiệm (khoảng 40.6 giây), gần gấp 3.2 lần QuickSort và khoảng 1.7 lần MergeSort. Nguyên nhân là do số lượng phép hoán đổi lớn, hàm `heapify` được gọi nhiều lần và các thao tác chỉ số/đệ quy gây overhead cao trong môi trường Python.

3. Tổng kết

- Trong thực tế phát triển phần mềm với Python, luôn ưu tiên sử dụng các hàm có sẵn (`list.sort()` hoặc `numpy.sort()`) để đạt hiệu năng tối đa.
- Khi cần tự cài đặt, QuickSort là lựa chọn tốt cho tốc độ, nhưng cần cẩn trọng trong việc chọn pivot. MergeSort là lựa chọn an toàn nếu cần sự ổn định thời gian và không quan tâm đến bộ nhớ phụ.

III. Thông tin chi tiết

Link Github: <https://github.com/ngocthuan-uit/SortingReport>