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

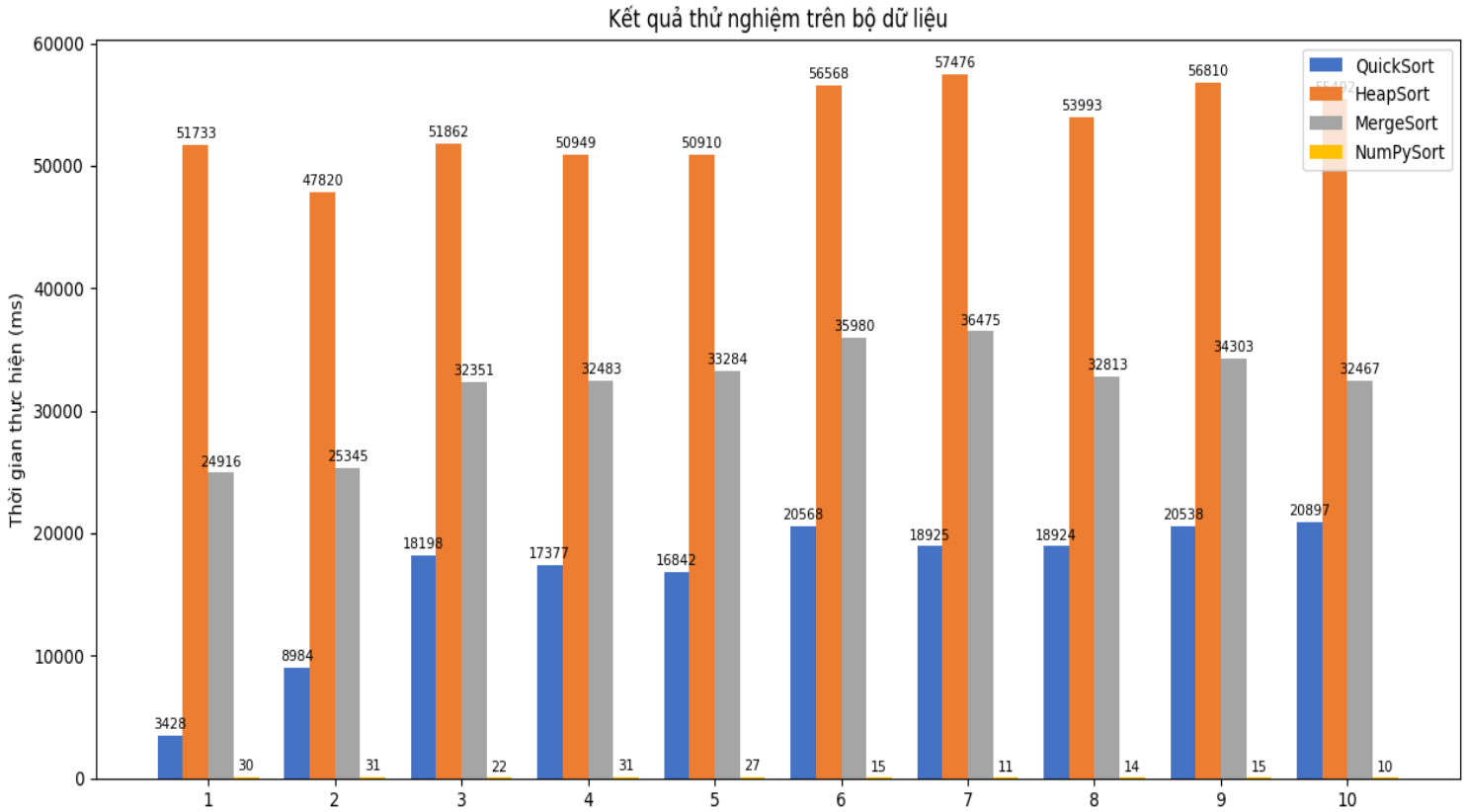
BÁO CÁO KẾT QUẢ THỬ NGHIỆM

I. Kết quả thử nghiệm

1. Bảng thời gian thực hiện

Dữ liệu	Thời gian thực hiện (ms)			
	QuickSort	HeapSort	MergeSort	NumpySort
1	3428	51733	24916	30
2	8984	47820	25345	31
3	18198	51862	32351	22
4	17377	50949	32483	31
5	16842	50910	33284	27
6	20568	56568	35980	15
7	18925	57476	36475	11
8	18924	53993	32813	14
9	20538	56810	34303	15
10	20897	55492	32467	10
Trung bình	16468.1	53361.3	32041.7	20.6

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 20 ms, nhanh hơn khoảng 700–800 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 16.4 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 $O(N^2)$ thường gặp khi chọn chốt là phần tử đầu/cuối trên dữ liệu đã sắp xếp.
- **MergeSort:** Hoạt động rất ổn định với thời gian trung bình khoảng 32s. 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 53 giây), gần gấp 3 lần QuickSort và khoảng 1.5 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