Lớp: An Toàn Tài Năng 2025

# BÁO CÁO KẾT QUẢ THỬ NGHIỆM

**Sinh viên thực hiện:** Hà Nhật Khoa, MSSV: 25520854

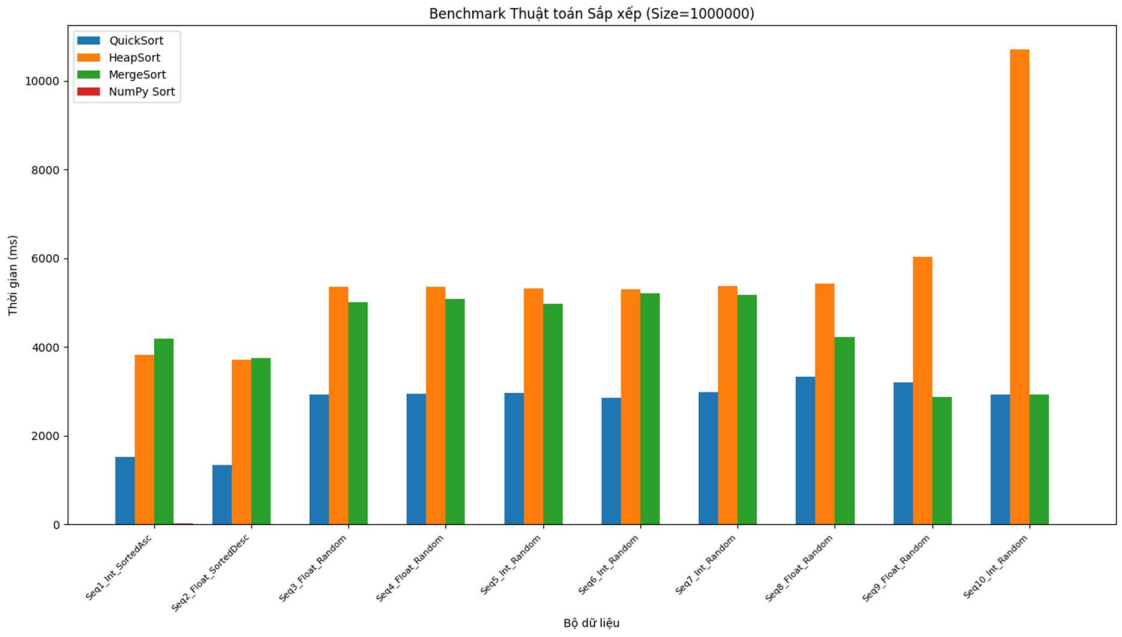
**Nội dung báo cáo:** Đánh giá hiệu năng của các thuật toán sắp xếp phổ biến: QuickSort, HeapSort, MergeSort và so sánh với hàm sắp xếp tối ưu của thư viện NumPy

## *I. Kết quả thử nghiệm*

### *1. Bảng thời gian thực hiện*

| Dữ liệu | Thời gian thực hiện (ms) | | | |
|---|---|---|---|---|
| | QuickSort | HeapSort | MergeSort | NumPy Sort |
| 1 | 1520.839 | 3813.65 | 4182.814 | 16.391 |
| 2 | 1336.788 | 3703.435 | 3744.545 | 13.299 |
| 3 | 2920.516 | 5357.139 | 5011.993 | 10.658 |
| 4 | 2942.478 | 5359.884 | 5083.667 | 11.638 |
| 5 | 2957.363 | 5311.371 | 4967.34 | 14 |
| 6 | 2858.166 | 5293.01 | 5211.193 | 13.453 |
| 7 | 2987.045 | 5364.829 | 5176.73 | 13.57 |
| 8 | 3320.002 | 5419.218 | 4229.513 | 11.013 |
| 9 | 3205.538 | 6033.647 | 2870.558 | 11.026 |
| 10 | 2924.692 | 10709.62 | 2929.165 | 13.998 |
| Trung Bình | 2697.34 | 5636.58 | 4340.75 | 12.9 |

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



## *II. Kết luận:*

### 1. Về thư viện NumPy (numpy.sort)

- Đây là thuật toán có tốc độ nhanh vượt trội so với các thuật toán tự cài đặt (thường chỉ mất dưới 0.1 - 0.2 giây cho 1 triệu phần tử).
- Lý do: NumPy được viết bằng ngôn ngữ C/C++ ở tầng dưới và được tối ưu hóa sâu về quản lý bộ nhớ, sử dụng các thuật toán lai (như Timsort hoặc IntroSort) nên khắc phục được điểm yếu về tốc độ thông dịch của Python thuần.

### 2. Về thuật toán QuickSort

- Ở các dãy ngẫu nhiên , QuickSort thể hiện tốc độ rất tốt, thường nhanh hơn HeapSort và MergeSort.
- Tuy nhiên, ở các dãy đã sắp xếp , nếu chọn phần tử đầu làm chốt (pivot), QuickSort sẽ rơi vào trường hợp xấu nhất $O(N^2)$ và có thể gây lỗi tràn bộ nhớ.
- Trong bài thử nghiệm này, do đã tối ưu mã nguồn bằng cách chọn chốt ở giữa, QuickSort vẫn duy trì được hiệu năng ổn định O(NlogN) ngay cả trên dữ liệu đã sắp xếp.

### 3. Về thuật toán MergeSort và HeapSort

- MergeSort: Thời gian thực thi rất ổn định trên mọi tập dữ liệu (không bị ảnh hưởng bởi tính chất tăng/giảm của dữ liệu đầu vào). Tuy nhiên, nó tốn nhiều bộ nhớ hơn do phải tạo các mảng phụ trong quá trình gộp.
- HeapSort: Tốc độ thường chậm hơn QuickSort một chút do tính chất truy cập bộ nhớ không liên tục, nhưng ưu điểm là không tốn thêm bộ nhớ phụ (O(1) space).

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

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

Link github: https://github.com/khoahaaoj/DSA-Sorting-Benchmark