TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN, ĐHQG-HCM

KHOA: KHOA HỌC MÁY TÍNH

LỚP: TTNT2025

BÁO CÁO KẾT QUẢ THỬ NGHIỆM

THỰC NGHIỆM CÁC GIẢI THUẬT SẮP XẾP

Môn học: IT003.Q21.TTNT – Cấu trúc dữ liệu và giải thuật

Giảng viên hướng dẫn: ThS. Nguyễn Thanh Sơn

Thực hiện bởi Sinh Viên: Hoàng Lê Kim Lâm – MSSV: 25520974

Thời gian thực hiện: 10/02/2026 – 20/02/2026

MỤC LỤC

KẾT QUẢ THỬ NGHIỆM

I. Bảng thời gian thực hiện

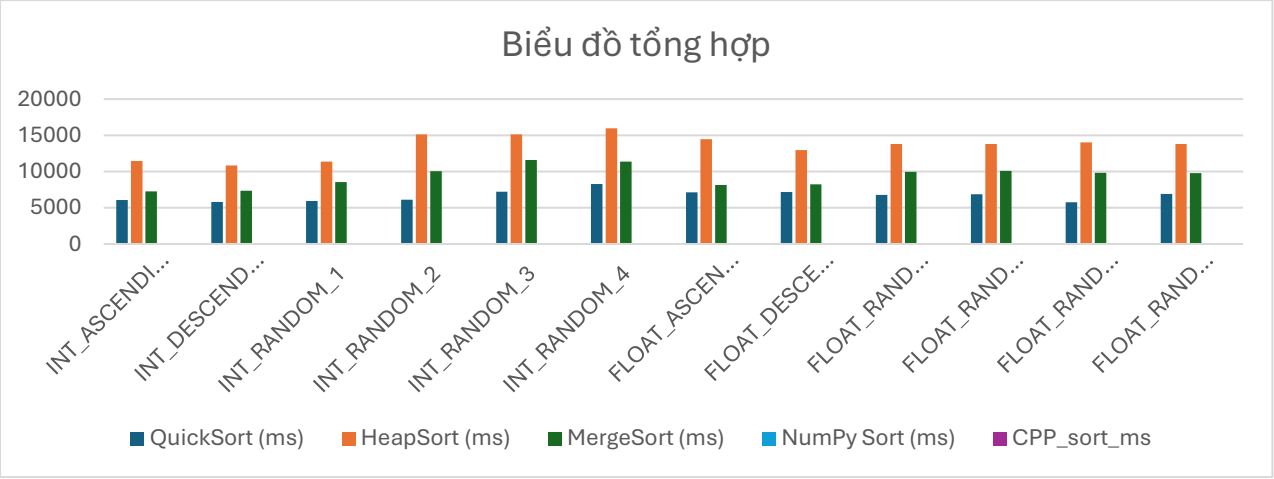
Kích thước dataset: mỗi dãy đều chứa 1 triệu phần tử và phạm vi phần tử là [-10^6, 10^6]

Dataset	QuickSort (ms)	HeapSort (ms)	MergeSort (ms)	NumPy Sort (ms)	CPP_sort_ms
INT_ASCENDING	6067,8643	11468,8472	7246,0139	8,038	8
INT_DESCENDING	5781,0529	10851,9542	7366,9472	13,93959997	7
INT_RANDOM_1	5925,6338	11358,191	8521,8885	57,10780001	62
INT_RANDOM_2	6123,3364	15133,6868	10050,3929	65,3858	59
INT_RANDOM_3	7206,1984	15100,0899	11591,6661	69,4262	60
INT_RANDOM_4	8256,2824	15980,0882	11365,1376	71,5666	65
FLOAT_ASCENDING	7145,5409	14439,347	8135,2611	10,7596	9
FLOAT_DESCENDING	7161,2039	12970,2357	8211,4553	15,87530001	9
FLOAT_RANDOM_1	6790,3772	13796,4681	9955,3025	75,5179	68
FLOAT_RANDOM_2	6882,1534	13803,9843	10082,8854	77,8498	58
FLOAT_RANDOM_3	5765,9478	14015,9194	9821,983	88,58350001	57
FLOAT_RANDOM_4	6912,2046	13775,7432	9759,6414	81,10840002	60

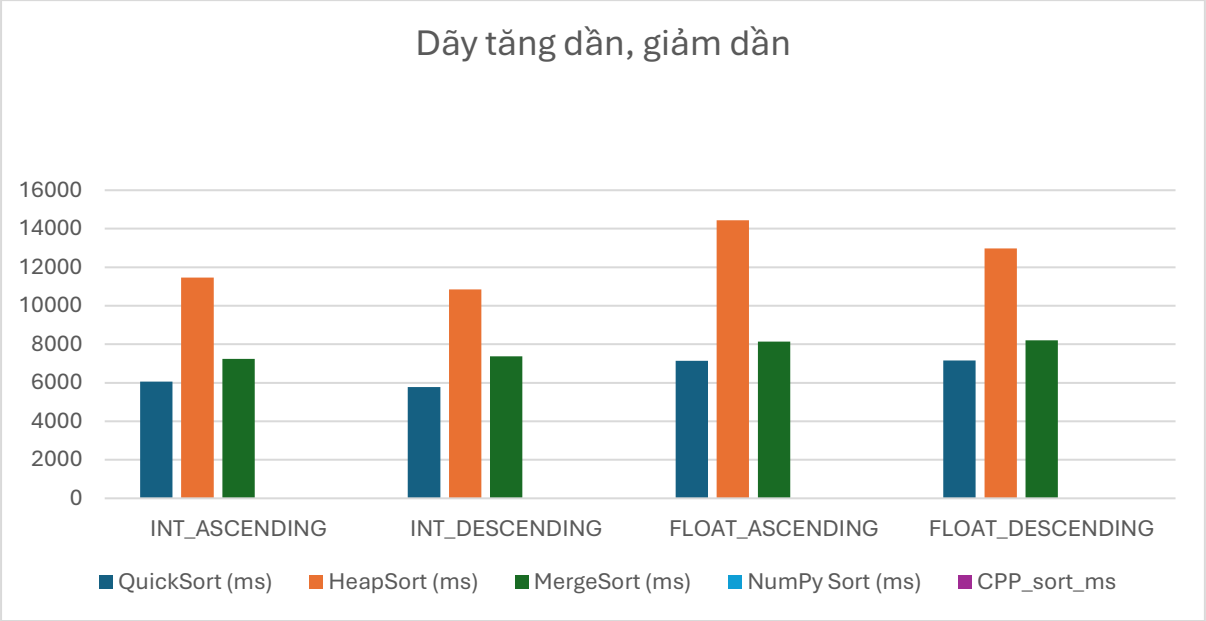
II. Biểu đồ thời gian thực hiện

1. Biểu đồ tổng quát trên từng bộ dữ liệu

Biểu đồ tổng hợp các bộ dữ liệu

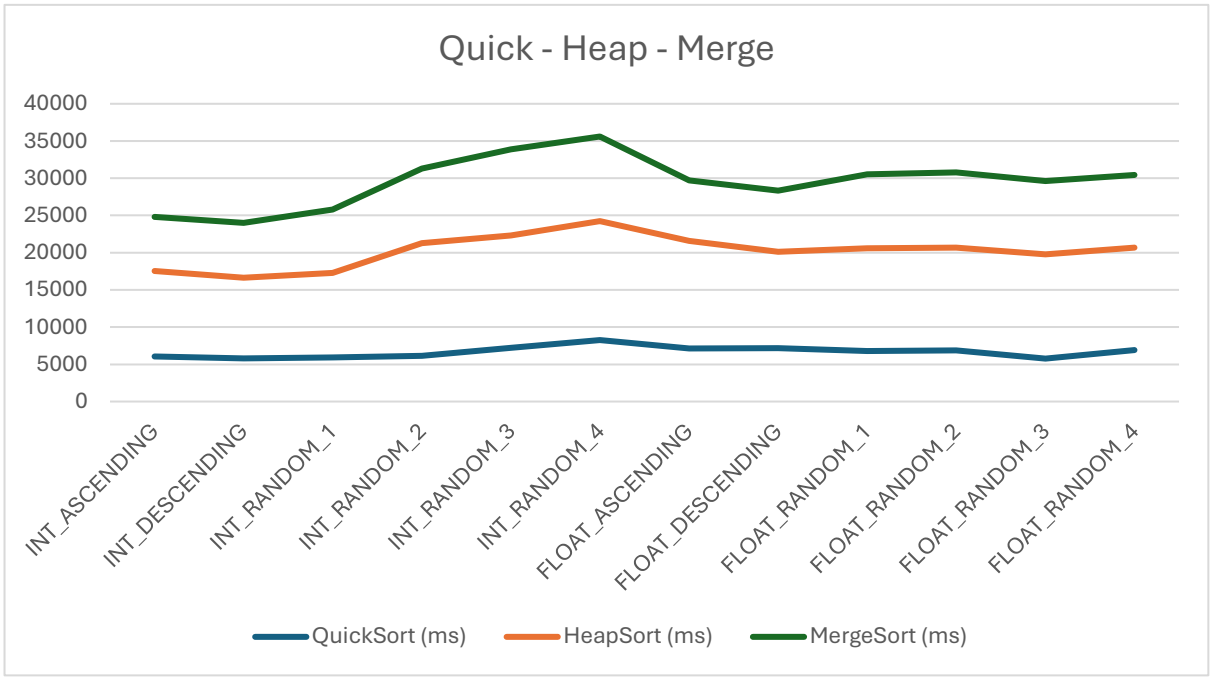


Biểu đồ so sánh dãy tăng dần / giảm dần

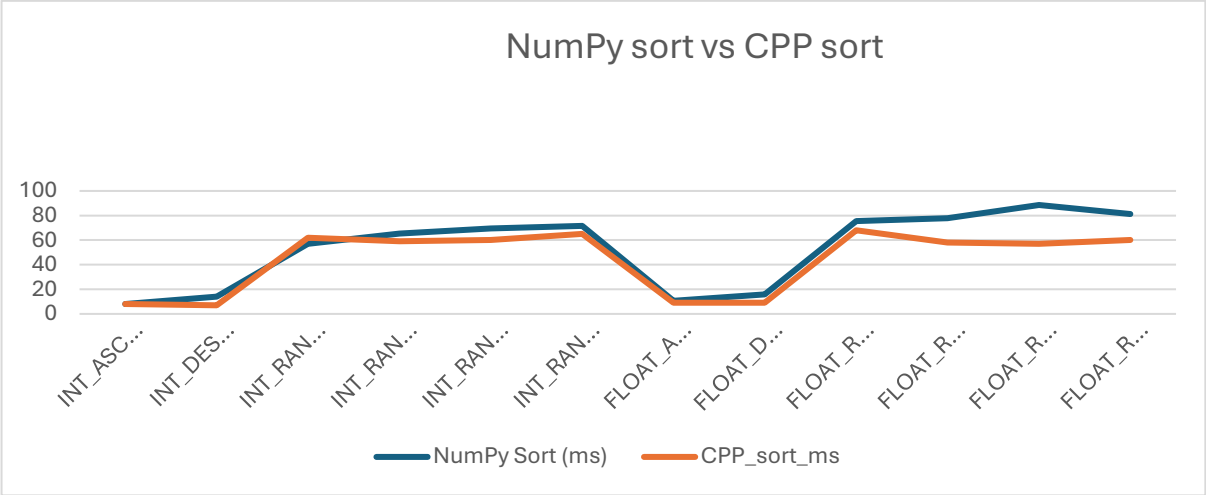


2. **Biểu đồ quá trình thay đổi của thuật toán qua các bộ dữ liệu**

Biểu đồ Quick – Heap – Merge



Biểu đồ Numpy Sort (Python) – CPP_sort



KẾT LUẬN

I. So sánh tổng thể giữa nhóm hàm tự cài đặt và hàm thư viện

Kết quả thực nghiệm cho thấy sự khác biệt rất rõ rệt về thời gian thực thi giữa hai nhóm thuật toán: nhóm thuật toán tự cài đặt (QuickSort, HeapSort, MergeSort) và nhóm hàm có sẵn trong thư viện (NumPy sort và CPP sort).

Cụ thể:

- Nhóm thuật toán tự cài đặt có thời gian thực thi dao động từ khoảng 5.800 ms đến hơn 15.000 ms.
- Trong khi đó, các hàm thư viện chỉ dao động từ khoảng 7 ms đến dưới 70 ms.

Chênh lệch đạt xấp xỉ hai bậc độ lớn (khoảng 100 lần). Điều này phản ánh không chỉ khác biệt ở mức độ cài đặt mà còn ở mức độ tối ưu hệ thống.

Nguyên nhân của sự chênh lệch này bao gồm:

- Các hàm thư viện được tối ưu ở mức biên dịch (compiler optimization như -O3, inline, vectorization).
- Sử dụng thuật toán lai (hybrid algorithm), kết hợp nhiều chiến lược tối ưu.

- Tối ưu cache locality và branch prediction.
- Giảm thiểu overhead cấp phát bộ nhớ và thao tác phụ.

Trong khi đó, các thuật toán tự cài đặt chủ yếu phản ánh cấu trúc lý thuyết của thuật toán mà chưa tận dụng sâu các đặc tính phần cứng hiện đại.

II. Phân tích chi tiết nhóm thuật toán tự cài đặt

1. So sánh giữa QuickSort, HeapSort và MergeSort

Từ biểu đồ so sánh thời gian chạy của các thuật toán tự viết, có thể nhận thấy:

- QuickSort có thời gian thực thi thấp nhất trong nhóm.(với việc chọn pivot random thay vì chọn đầu hay chọn cuối để giảm thiểu xác suất xảy ra trường hợp xấu nhất mức tối đa.)
- MergeSort đứng thứ hai.
- HeapSort có thời gian cao nhất và độ dao động lớn hơn.

Sự khác biệt này xuất phát từ đặc điểm thuật toán:

- QuickSort có locality tốt hơn do thao tác trên mảng tại chỗ, giúp tận dụng cache hiệu quả.
- MergeSort cần mảng phụ để trộn (merge), làm tăng chi phí bộ nhớ và giảm hiệu quả cache.
- HeapSort mặc dù có độ phức tạp O(n log n) ổn định, nhưng thao tác heapify làm truy cập bộ nhớ phân tán, gây giảm hiệu năng thực tế.

2. Ảnh hưởng của cấu trúc dữ liệu đầu vào

Khi xét theo từng loại dãy (tăng, giảm, ngẫu nhiên), ta nhận thấy:

- Dãy tăng và dãy giảm không tạo ra sự khác biệt đáng kể trong nhóm tự cài đặt.
- Dãy ngẫu nhiên có xu hướng làm tăng thời gian thực thi, nhưng mức tăng không quá đột biến.

Điều này cho thấy:

- Cài đặt QuickSort trong thực nghiệm không rơi vào trường hợp xấu $O(n^2)$, bằng cách chọn pivot tương đối cân bằng.(chọn ngẫu nhiên) để không rơi vào trường hợp xấu nhất đối với dãy tăng dần hoặc giảm dần
- HeapSort và MergeSort vốn không có tính thích nghi (non-adaptive), do đó không tận dụng được cấu trúc có sẵn của dãy tăng/giảm.

Từ đó có thể kết luận rằng nhóm thuật toán tự viết thể hiện tính ổn định tương đối nhưng không có khả năng thích nghi theo cấu trúc dữ liệu đầu vào.

III. Phân tích nhóm hàm thư viện (NumPy sort và CPP sort)

Biểu đồ so sánh cho thấy sự khác biệt rõ ràng giữa dãy đã có thứ tự và dãy ngẫu nhiên:

- Với dãy tăng và giảm, thời gian thực thi chỉ khoảng 7–13 ms.
- Với dãy ngẫu nhiên, thời gian tăng lên khoảng 60–70 ms.

Mức chênh lệch này lớn hơn nhiều so với nhóm tự cài đặt.

Nguyên nhân chủ yếu:

- NumPy sử dụng các thuật toán được cài đặt bằng C và tối ưu hóa ở mức hệ thống; mặc định thường là quicksort (khác với list.sort() của Python dùng Timsort.)

Đây **không phải QuickSort ngây thơ** (pivot đầu/cuối).

Là QuickSort đã được tối ưu ở mức C implementation:

  - Chọn pivot theo chiến lược median-of-three.
  - Dùng insertion sort cho đoạn nhỏ.
  - Có cơ chế kiểm soát độ sâu đệ quy để tránh thoái hóa nghiêm trọng.

Tuy nhiên:

  - NumPy **không được thiết kế như một introsort chuẩn thư viện STL**
  - Nó là quicksort tối ưu riêng cho mảng C liên tục trong bộ nhớ.

Đặc điểm:

    - Trung bình: O(nlogn)
    - Không ổn định
    - Trên dữ liệu gần có thứ tự, số swap và partition thực tế giảm đáng kể
- **Cpp sort sử dụng Introsort**, một thuật toán lai kết hợp:
    - QuickSort (chiến lược chính),
    - HeapSort (khi độ sâu đệ quy vượt ngưỡng để tránh trường hợp xấu $O(n^2)$
    - Insertion Sort (cho các đoạn nhỏ, thường < 16 phần tử).
      Cơ chế này đảm bảo hiệu năng trung bình nhanh như QuickSort nhưng vẫn giữ được giới hạn O(nlogn) trong trường hợp xấu.
- Khi dữ liệu đã có thứ tự (tăng hoặc giảm), số lượng thao tác **partition**, **swap** và **heapify** giảm đáng kể so với dữ liệu ngẫu nhiên.
- CPU **branch prediction** hoạt động hiệu quả hơn trên dãy đơn điệu (monotonic sequence), do các nhánh điều kiện (if (a[i] < pivot)) có xu hướng cho kết quả lặp lại theo một quy luật nhất quán, giảm branch misprediction penalty.

Điều này lý giải vì sao dãy tăng và giảm có thời gian thực thi nhanh hơn đáng kể so với dãy ngẫu nhiên trong nhóm hàm thư viện.

IV. So sánh mức độ thay đổi thời gian giữa các loại dãy

Đối với nhóm tự cài đặt:

- Biến thiên thời gian giữa dãy tăng, giảm và ngẫu nhiên tương đối nhỏ.
- Điều này cho thấy thuật toán gần như xử lý mọi loại dữ liệu với chi phí tương đương.

Đối với nhóm thư viện:

- Dãy tăng và giảm nhanh hơn dãy ngẫu nhiên nhiều lần.
- Thể hiện rõ tính thích nghi và khả năng tối ưu theo cấu trúc dữ liệu.

V. Kết luận chung từ phân tích thực nghiệm

Từ toàn bộ kết quả và biểu đồ, có thể rút ra các kết luận sau:

- Hàm thư viện vượt trội về hiệu năng so với các thuật toán tự cài đặt.

- Tính thích nghi (adaptive behavior) đóng vai trò quan trọng trong xử lý dữ liệu đã có cấu trúc.

- Các thuật toán tự viết thể hiện đúng độ phức tạp lý thuyết nhưng thiếu tối ưu hệ thống.

- Dãy ngẫu nhiên luôn gây thời gian thực thi cao hơn so với dãy đã có thứ tự.

- Sự khác biệt hiệu năng không chỉ đến từ độ phức tạp O(n log n), mà còn từ:

    - tối ưu bộ nhớ,
    - tối ưu nhánh rẽ,
    - chiến lược lai thuật toán,
    - tối ưu ở mức compiler và kiến trúc CPU.

Như vậy, kết quả thực nghiệm không chỉ phù hợp với lý thuyết thuật toán mà còn phản ánh rõ vai trò của tối ưu hóa hệ thống trong các thư viện tiêu chuẩn hiện đại.

Thông tin chi tiết

Link github: https://github.com/HoangLeKimLam/HoangLeKimLam-s-sorting_benchmark