ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
IT003.Q21.CTTN

SO SÁNH THỜI GIAN CHẠY CỦA CÁC THUẬT TOÁN SẮP XẾP

Sinh viên thực hiện	MSSV
BÙI THÁI SƠN	25521588

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 2 NĂM 2025

1 Giới thiệu

Báo cáo sử dụng chương trình python "gen.py"để tạo ra bộ dữ liệu "dataset.txt"theo yêu cầu:

- Hàng 1 là dãy số nguyên tăng dần
- Hàng 2 là dãy số nguyên giảm dần
- Hàng 3 đến 5 là dãy số nguyên không theo thứ tự
- Hàng 6 đến 10 là dãy số thực không theo thức tự

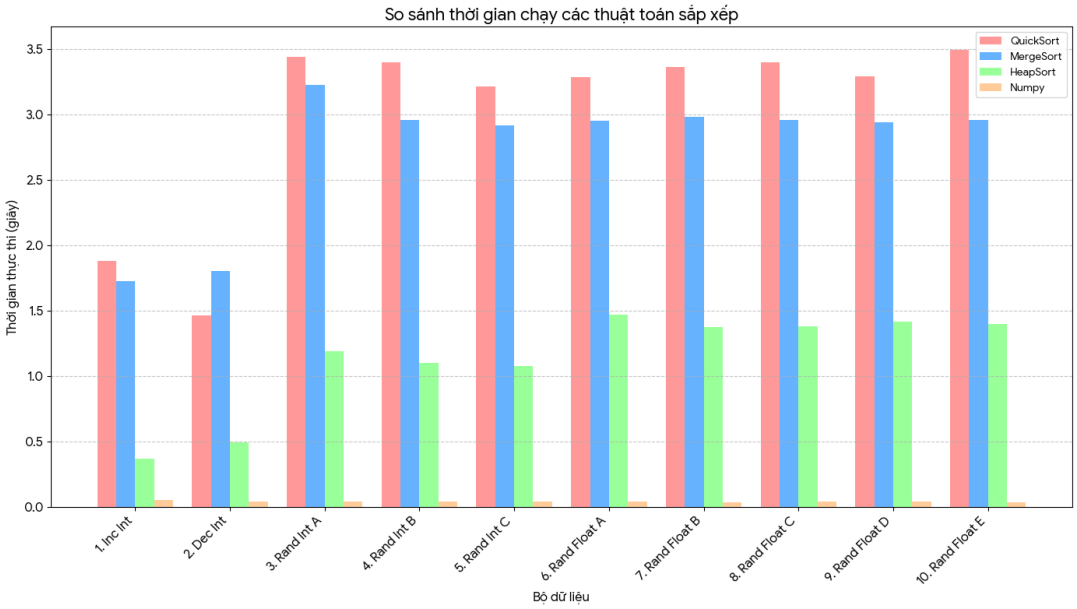
Mỗi dãy số gồm một triệu (1000000) phần tử nằm trong khoảng [0, 1000000000]
Để chạy thuật toán và thống kê lại dữ liệu, báo cáo sử dụng chương trình python "benchmark.py". Chương trình bao gồm mã nguồn cách cài của 4 thuật toán: QuickSort, HeapSort, MergeSort và hàm sort của Numpy, và một hàm để so sánh (benchmark) thời gian hoạt động của các thuật toán. Chương trình sẽ xuất ra một bảng dữ liệu thống kê thời gian chạy của từng thuật toán, tương ứng với từng hàng trong bộ dữ liệu.
Bộ dữ liệu "dataset.txt"được sử dụng để viết báo cáo không được tải lên github do kích thước quá lớn.
Biểu đồ cột "Chart.png"được tạo bởi ứng dụng Gemini từ kết quả chạy của chương trình.
Tất cả các file liên quan được sử dụng trong bài báo cáo đều được đăng tải lên github: `https://github.com/BetterCuckChuck/IT003.Q21.CTTN_LAB_REPORT_1`.

2 Thống kê

Bảng 1: Kết quả thử nghiệm thời gian thực thi các thuật toán sắp xếp (giây)

Dãy Dữ Liệu	QuickSort	MergeSort	HeapSort	Numpy
1. Inc Int	1.8815	1.7271	0.3693	0.0535
2. Dec Int	1.4641	1.8028	0.4903	0.0418
3. Rand Int A	3.4381	3.2272	1.1867	0.0411
4. Rand Int B	3.3995	2.9587	1.0967	0.0399
5. Rand Int C	3.2153	2.9165	1.0744	0.0400
6. Rand Float A	3.2871	2.9507	1.4664	0.0367
7. Rand Float B	3.3636	2.9834	1.3719	0.0358
8. Rand Float C	3.3983	2.9578	1.3768	0.0380
9. Rand Float D	3.2891	2.9379	1.4171	0.0393
10. Rand Float E	3.4959	2.9578	1.3979	0.0357

Hình 1: Biểu đồ cột cho thời gian thực thi các thuật toán sắp xếp



3 Phân tích, Đánh giá

Từ kết quả thực nghiệm được ghi nhận, ta có thể rút ra các phân tích chi tiết sau:

3.1 Về mặt tổng quát

- **Numpy Sort vượt trội nhất:** Thời gian chạy của Numpy luôn nhỏ hơn 0.06 giây trong mọi trường hợp, nhanh hơn gấp hàng chục đến hàng trăm lần so với các thuật toán khác.
  Nguyên nhân: Thư viện Numpy được viết bằng ngôn ngữ C và đã được tối ưu hóa cực tốt ở mức thấp (low-level), tránh được overhead của trình thông dịch Python.

- **HeapSort nhanh nhất trong nhóm Python thuần:** HeapSort có tốc độ ổn định khoảng $1.0 - 1.4$ giây với dữ liệu ngẫu nhiên, nhanh hơn đáng kể so với MergeSort và QuickSort.
  Nguyên nhân: Trong Python, module `heapq` cũng được viết bằng C , giúp giảm thiểu chi phí tính toán so với việc thao tác thủ công như trong QuickSort hay MergeSort.

- **QuickSort có trung bình thời gian chậm nhất:** QuickSort mất khoảng $3.2 - 3.5$ giây cho dữ liệu ngẫu nhiên.
  Nguyên nhân: Việc cài đặt QuickSort bằng Python thuần thường gặp vấn đề về chi phí đệ quy (recursion overhead) và việc tạo ra nhiều list con (list slicing) tốn kém bộ nhớ và thời gian sao chép dữ liệu.

- **MergeSort nằm ở mức giữa HeapSort và QuickSort:** MergeSort mất khoảng trung bình 3 giây cho dữ liệu ngẫu nhiên.
  Nguyên nhân: Việc MergeSort nhanh hơn QuickSort (ngược với các benchmark thông thường) là do cách cài đặt của hàm Quicksort trong chương trình cũng tạo ra nhiều list con, làm gia tăng chi phí đáng kể.

3.2 Về ảnh hưởng của dữ liệu đầu vào

- **Dữ liệu đã sắp xếp (Inc/Dec Int):** Cả QuickSort và MergeSort đều chạy nhanh hơn đáng kể trên dữ liệu đã có thứ tự (hoặc ngược thứ tự) so với dữ liệu ngẫu nhiên (giảm từ ~ 3.4s xuống $\sim 1.5 - 1.8$s). Điều này cho thấy việc chọn Pivot (trong QuickSort) hoặc quá trình Merge hoạt động hiệu quả hơn khi dữ liệu có tính thứ tự cao.

- **Dữ liệu Int vs. Float:** Không có sự khác biệt quá lớn về thời gian xử lý giữa số Nguyên (Int) và số Thực (Float). Các thuật toán chạy trên tập dữ liệu Float (mục 6-10) có thời gian tương đương với tập dữ liệu Int ngẫu nhiên (mục 3-5), trừ HeapSort (xử lý dữ liệu số Nguyên nhanh hơn $\sim 0.2 - 0.3$s so với số Thực).

4 Kết luận

- Nếu cần tốc độ xử lý trong thực tế với Python, luôn ưu tiên sử dụng các thư viện có sẵn như `Numpy` hoặc hàm `sorted()` của Python (TimSort) thay vì tự cài đặt lại thuật toán.
- Đối với mục đích học thuật, HeapSort cho thấy hiệu quả tốt nhất khi cài đặt trong môi trường Python do có sự hỗ trợ của thư viện chuẩn `heapq`.