Lớp: IT003.O21.CTTN

BÁO CÁO KẾT QUẢ THỬ NGHIỆM

Thời gian thực hiện: 011/03 – 16/03/2022

Sinh viên thực hiện: Trần Vạn Tấn

Mã số sinh viên: 23521407

Ngày sinh: 05/09/2005

Nội dung báo cáo:

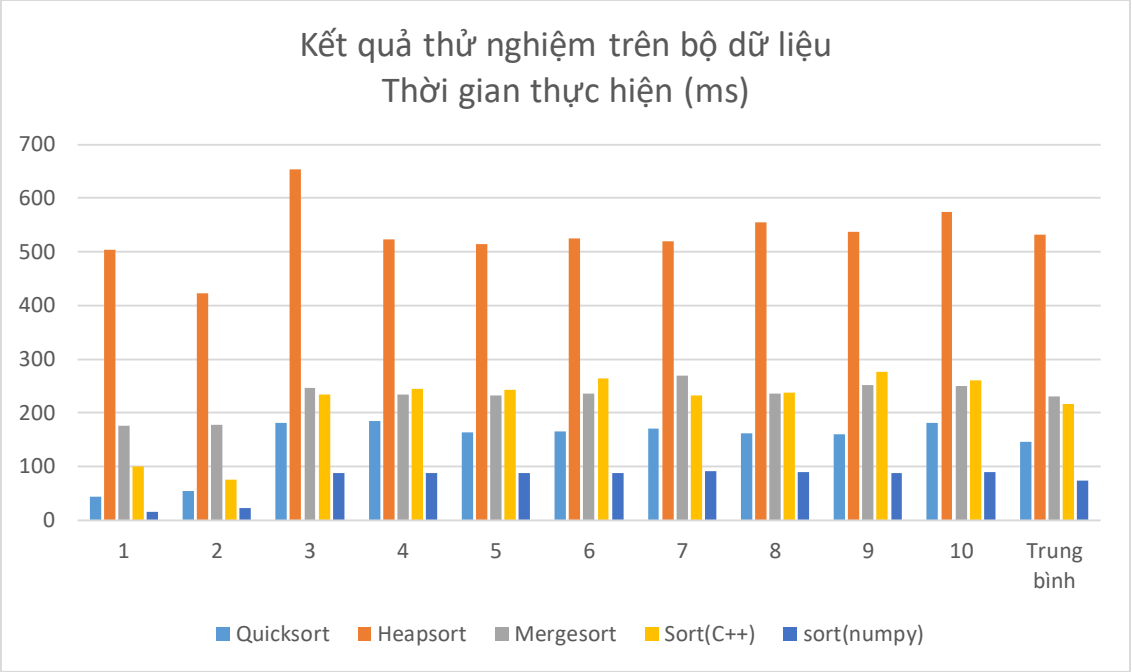
I. Kết quả thử nghiệm

1. Bảng thời gian thực hiện[1]

Dữ liệu	Thời gian thực hiện (ms)				
	Quicksort	Heapsort	Mergesort	sort (C++)	sort (numpy)
1	44	504	175	100	15
2	54	422	178	75	23
3	181	654	247	234	87
4	185	524	234	244	88
5	163	514	233	243	87
6	165	525	235	264	87
7	170	519	269	233	91
8	162	555	236	237	89
9	160	538	251	277	87
10	181	574	250	261	89
Trung bình	146	532	230	216	74

[1] Số liệu chỉ mang tính minh họa

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



II. *Kết luận:*

1. *Quicksort:*

Quicksort có thời gian thực hiện chậm trong một số trường hợp. Trong trường hợp nó đã được sắp xếp rồi thì quá trình sắp xếp không cần thiết vẫn sẽ tiếp tục.

Việc chọn số lớn nhất hoặc nhỏ nhất làm trung tâm sẽ khiến chương trình bị chậm và không hiệu quả. Vì vậy giải pháp là chọn ngẫu nhiên một số tương đương xác suất gặp số lớn nhất hoặc số nhỏ nhất rất thấp nhưng không ổn định.

Tuy không ổn định nhưng vẫn là một trong những lựa chọn tốt khi sắp xếp. Độ phức tạp tốt nhất lên đến $O(n)$, tệ nhất $O(n^2)$, trung bình $O(n \log n)$

2. *Heapsort*

Heapsort có thời gian thực hiện chậm nhất trong hầu hết các trường hợp. Xây dựng và sắp xếp cấu trúc cây nhị phân với số lượng lớn lên đến 10^6 phần tử

Trong trường hợp nó đã được sắp xếp rồi thì quá trình sắp xếp không cần thiết vẫn sẽ tiếp tục. Yếu tố bình đẳng không được lấy ra cùng một lúc mà lấy ra lần lượt.

Thích hợp để sắp xếp một số lượng nhỏ các phần tử. Độ phức tạp tốt nhất

3. *Mergesort*

Mergesort có thời gian thực hiện tương đối ổn định nhưng vẫn còn hạn chế.

Trong trường hợp nó đã được sắp xếp rồi thì quá trình sắp xếp không cần thiết vẫn sẽ tiếp tục.

Khi gặp trường hợp gộp hai mảng theo thứ tự xen kẽ nhau, ví dụ: {1, 3, 5, 7} và {2, 4, 6, 8} thì mỗi phần tử cần được so sánh ít nhất một lần.

4. ***Sort(C++)***

Dễ dàng sử dụng, thời gian thực hiện nhanh chóng.

Sử dụng kết hợp Quicksort, HeapSort và InsertionSort. Cụ thể, nó sử dụng QuickSort, nhưng nếu QuickSort đang thực hiện phân vùng không đồng đều và mất nhiều thời gian hơn $O(n \log n)$, thì nó sẽ chuyển sang HeapSort và khi kích thước mảng trở nên thực sự nhỏ, nó sẽ chuyển sang InsertionSort.

5. ***Sort(numpy)***

Hiệu quả, dễ sử dụng và nhanh nhất.

Sort in NumPy là một hàm được viết bằng C, một ngôn ngữ lập trình hiệu suất cao. Sử dụng C sẽ tối ưu hóa hiệu suất và loại bỏ chi phí liên quan đến việc diễn giải mã Python.

NumPy sử dụng các thuật toán và kỹ thuật chuyên dụng để tận dụng tối đa phần cứng hiện đại, chẳng hạn như SIMD (Single Manual, Multiple Data). Vì vậy thư viện NumPy được tối ưu hóa cho các hoạt động mảng lớn, bao gồm cả việc sắp xếp.

NumPy sử dụng các kiểu dữ liệu theo mảng cụ thể được tối ưu hóa cho các hoạt động được vector hóa. Việc sử dụng các kiểu dữ liệu này có thể tăng hiệu suất so với các kiểu dữ liệu C++ truyền thống.

Link github:

https://github.com/tanprodium/IT003.O21.CTTN.git