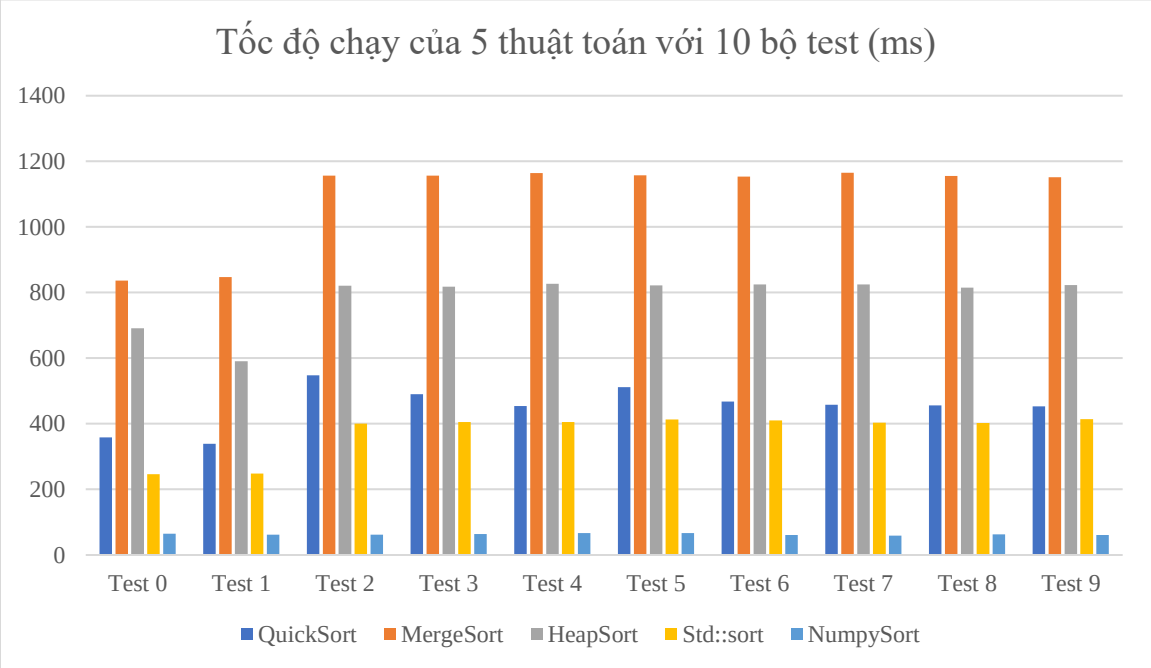
BÁO CÁO THỰC NGHIỆM CÁC GIẢI THUẬT SẮP XẾP NỘI

I. BÁO CÁO

1. Kết quả thực nghiệm

	QuickSort	MergeSort	HeapSort	Std::sort	NumpySort
Test 0	358	836	691	246	65
Test 1	339	847	591	248	62
Test 2	548	1156	821	400	62
Test 3	490	1156	818	405	64
Test 4	454	1164	827	405	67
Test 5	512	1157	822	413	67
Test 6	468	1153	825	410	61
Test 7	458	1165	825	403	59
Test 8	456	1155	815	402	63
Test 9	453	1152	823	414	61

2. Biểu đồ



II. NHẬN XÉT

1. QuickSort

- Trong 3 thuật toán đầu, Quicksort có tốc độ nhanh nhất.
- Pivot được chọn là ngẫu nhiên nên việc xảy ra worst-case là rất thấp, thêm việc thuật toán có O(1) không gian lưu trữ phát sinh.

2. MergeSort

- Mergesort có tốc độ hoạt động chậm nhất do có O(n) bộ nhớ phát sinh.
- Bù lại, Mergesort có tốc độ rất ổn định, worse-case của thuật toán là O(nlogn).

3. HeapSort

- Heapsort nhanh hơn Mergesort nhưng chậm hơn Quicksort.

- Heapsort chạy nhanh hơn so với Mergesort do không phải dùng thêm bộ nhớ, ctdl Heap được viết lên chính mảng ban đầu.
- Heapsort chậm hơn quicksort phần nhiều là do việc truy cập các node của cây không liên tục, thuật toán Quicksort tìm pivot ngẫu nhiên và việc phân mảng theo pivot được thực hiện tuyến tính trên mảng nên dẫn đến tốc độ hơn hẳn Heapsort, thêm việc phải swap các phần tử liên tục.

4. Std::sort
   - Hàm sort của C++ sử dụng introsort (một thuật toán sắp xếp kết hợp) nên hiệu quả đạt mức tối đa, std::sort cải thiện tốc độ đáng kể so với 3 thuật toán trên nhờ vào việc kếp hợp QuickSort + HeapSort + InsertionSort.
5. NumpySort
   - Hàm sort của Numpy sử dụng Timsort(chia thành các đoạn nhỏ, mỗi đoạn sử dụng InsertionSort và kết hợp sử dụng MergeSort
   - Sở dĩ hàm sort của numpy hoạt động rất nhanh là do:
     (1) Thuật toán được implement bằng C, cải thiện tốc độ so với hàm sort của python.
     (2) Numpy tối ưu hóa phần cứng, tận dụng CPU đa luồng.

Link code: [tnisl/Sorting_algorithm]