Sinh viên: Bùi Nguyễn Kim Hải
Mã số sinh viên: 22520377
Lớp học: KHTN2022
Github: https://github.com/whynotkimhari

SORTING METHODS REPORT

1. Generate Data and Collect Data:

- Tổng: 10 bộ, trong đó 1 bộ ascending, 1 bộ descending, 8 bộ random order
- Thực hiện tạo bằng python, mỗi bộ gồm 1 triệu số thực ngẫu nhiên.
- Data được thu bằng cách viết vào file output
- Table được vẽ bằng cách điền số liệu từ output
- Graph được vẽ bằng thư viện matplot – python

- Thông tin tại github repo:
https://github.com/whynotkimhari/report_of_searching_algorithms

2. Sorting Data:

2.1. Bảng thống kê:

SPEED RATE TABLE (s)

	QuickSort	MergeSort	HeapSort	IntroSort	Order
Running 1	0.26	0.874	0.519	0.233	ASC
Running 2	0.371	0.887	0.032	0.263	DSC
Running 3	0.211	0.849	0.031	0.246	RD
Running 4	0.229	0.755	0.016	0.247	RD
Running 5	0.242	0.877	0.031	0.248	RD
Running 6	0.461	0.901	0.024	0.235	RD
Running 7	0.341	0.863	0.031	0.251	RD
Running 8	0.301	0.839	0.03	0.236	RD
Running 9	0.37	0.886	0.031	0.251	RD
Running 10	0.247	0.871	0.031	0.255	RD

Màu vàng: nhanh nhất – Màu xám: Chậm nhất
- Các cột là các loại thuật toán sắp xếp, bao gồm: **QuickSort, MergeSort, HeapSort, và IntroSort**.
- Cột cuối cùng thể hiện tính chất của bộ data, gồm có: **tăng dần(ASC), giảm dần(DSC) và ngẫu nhiên(RD)** như kết cấu ở phần 1.
- Ở mỗi hàng là thời gian chạy của bốn thuật toán sắp xếp khác nhau với cùng một bộ data, **đơn vị tính bằng giây** và tổng cộng có 10 hàng tất cả.

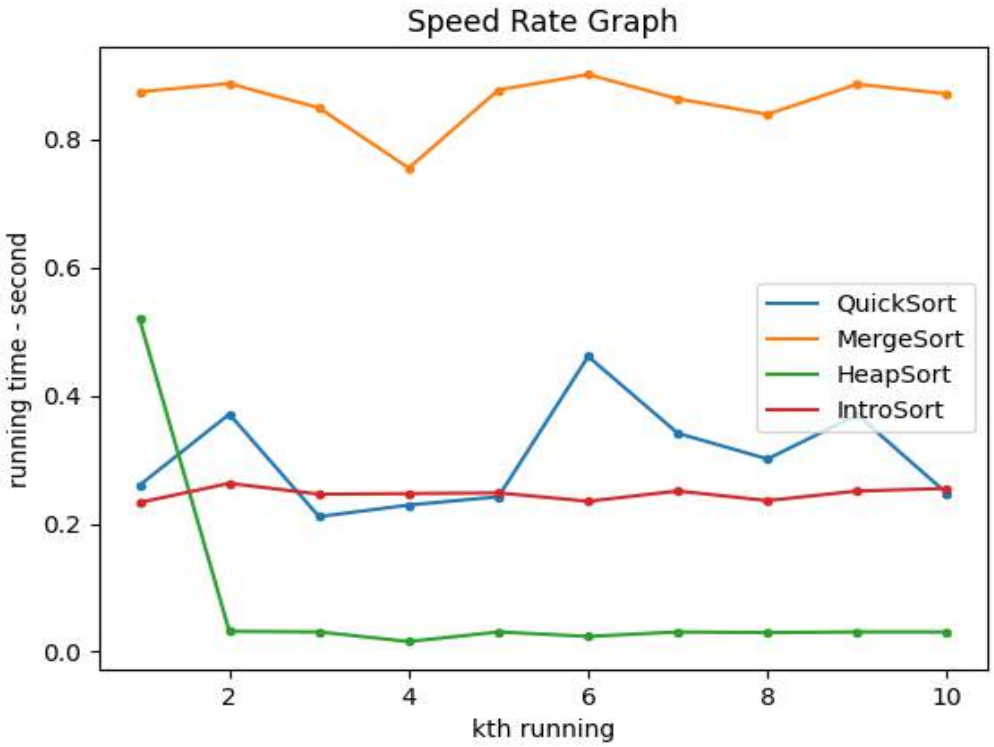
- Từ bảng thống kê thời gian các thuật toán cần để sắp xếp một bộ dữ liệu ngẫu nhiên, em có một vài kết luận như sau:
  + Thứ nhất, sau 10 bộ data, **HeapSort cho hiệu quả cao nhất** (giả sử hiệu quả được xét trên thời gian chạy nhanh hay chậm). Ngược lại, **MergeSort cho** thấy mình đang thua ba thuật toán còn lại trong 10 bộ data này với **thời gian chạy chậm nhất**.

+ Thứ hai, ta thấy được đó là **tính phù hợp với từng tình huống của mỗi loại thuật**. Ví dụ: mặc dù HeapSort cho tổng hiệu quả cao nhất, tuy nhiên HeapSort sẽ tốn rất nhiều thời gian khi xử lí một dãy vốn đã được sort sẵn (bộ 1)

+ Thứ ba, ta thấy được **sự cải tiến** rõ ràng của thuật toán (IntroSort là cải tiến của QuickSort), IntroSort đã cho gần như là các kết quả tốt hơn so với QuickSort.

2.2. Đồ thị:

SPEED RATE GRAPH



- Đồ thị thu được bằng cách biểu diễn các giá trị ở bảng trên vào trục toạ độ với trục hoành là các lần chạy các bộ data thứ k (kth running) và trục tung là giá trị thời gian mà mỗi thuật xử lí bộ data đó.

- Từ đồ thị, em có một vài nhận xét như sau:

+ Thứ nhất, đồ thị **trực quan hoá** sự chênh lệch về thời gian của bốn giải thuật một cách rõ ràng hơn.

+ Ở mỗi nấc k, các giải thuật **có sự thay đổi vị trí** về thứ hạng thành tích, tuy nhiên đó là **không đáng kể** do phần lớn chỉ là sự thay đổi của QuickSort và IntroSort, còn phần còn lại thì không có sự hoán đổi nào.

3. Tổng kết:

- Sau bài báo cáo, em có một vài kết luận như sau:

+ Dựa trên kết quả báo cáo, có lẽ việc sử dụng HeapSort sẽ thường cho hiệu quả hơn. Song, điều này tất nhiên không phải lúc nào cũng đúng, đặc biệt đối với trường hợp dãy đã được sort theo hướng ascending trước.

+ Ngoài HeapSort, em sẽ "hướng" mình dùng hàng thư viện cpp – IntroSort nhiều hơn. Nó đem lại kết quả cũng khá ấn tượng trên tổng số 10 bộ data.