TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

Project 01
Sorting Algorithm

Môn: Cấu trúc dữ liệu và giải thuật

Giáo viên hướng dẫn: Nguyễn Thanh Phương
Bùi Huy Thông
Nguyễn Ngọc Thảo

Lớp: 22CLC04

Nhóm: 08

Sinh viên thực hiện: Giang Gia Bảo – 20127446
Tô Nguyễn Trúc Nghi - 20127254

Tp Hồ Chí Minh, 28 tháng 07 năm 2023

MỤC LỤC

I. INFORMATION PAGE

MSSV	Họ và tên	Lớp	Note
20127446	Giang Gia Bảo	22CLC04	
20127254	Tô Nguyễn Trúc Nghi	22CLC04	Không liên lạc để thực hiện đồ án

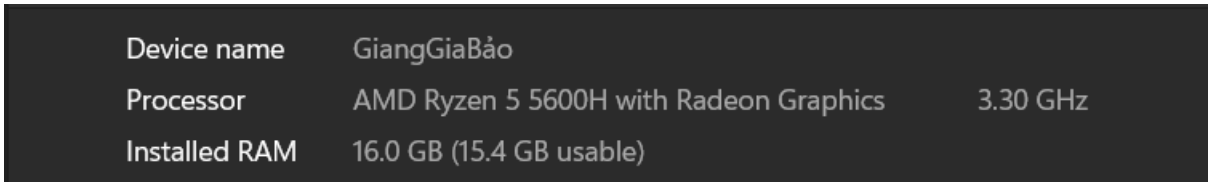
II. INTRODUCTION PAGE

Trong project này nhóm em sẽ cài đặt và chạy (11 algorithms): Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Flash Sort.

Về phần thực nghiệm: đã chạy được 11 thuật toán và đo được thời gian cũng như số phép so sánh.

Về phần output command line: nhóm em đã thực hiện đủ 5/5 commands theo yêu cầu.

Cấu hình máy chạy thực nghiệm:



III. ALGORITHM PRESENTATION

1. Selection Sort

Idea

Thuật toán này sắp xếp dãy bằng cách tìm phần tử nhỏ nhất và đổi chỗ nó với phần tử đầu tiên. Sau đó, tìm phần tử nhỏ nhất trong dãy con chưa được sắp xếp và đổi chỗ nó với phần tử thứ hai, và tiếp tục cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

Bước 1: Ta đặt phần tử đầu tiên làm phần tử nhỏ nhất và gán nó vào một biến đặt tên là min

Bước 2: So sánh min với phần tử sau nó nếu thấp hơn ta sẽ gán giá trị của phần tử đó cho biến min và là tiếp tục với các giá trị tiếp theo.

Bước 3: Sau mỗi lần lặp lại, giá trị nhỏ nhất được đặt ở phía đầu danh sách chưa được sắp xếp.

Bước 4: Lặp đi lặp lại bước 1 đến bước 3 cho tới khi tất cả phần tử đã được sắp xếp.

Complexity evaluations

O(n^2) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

2. Insertion Sort

Idea

Thuật toán này xây dựng dãy đã sắp xếp bằng cách chèn từng phần tử vào dãy con đã sắp xếp trước đó một cách phù hợp.

Step-by-step descriptions

Bước 1: Kiểm tra phần tử đầu tiên xem đã được sắp xếp chưa

Bước 2: Lấy phần tử tiếp theo kiểm tra

Bước 3: So sánh với tất cả các phần tử con đã được sắp xếp trước đó

Bước 4: Dịch chuyển tất cả các phần tử có trong mảng con nếu nó lớn hơn giá trị của phần tử đó

Bước 5: sau khi dịch chuyển ta chèn giá trị đó

Bước 6: Lặp lại cho tới khi phần tử sắp xếp đúng thứ tự.

Complexity evaluations

O(n^2) - Độ phức tạp trung bình và trong trường hợp xấu nhất, nhưng hiệu quả hơn khi dữ liệu gần như đã sắp xếp.

3. Bubble Sort

Idea

Thuật toán này so sánh lần lượt hai phần tử liền kề và đổi chỗ nếu chúng không được sắp xếp đúng thứ tự. Quá trình này được lặp lại cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

Bước 1: Xét phần tử thứ [0] và phần tử thứ [1] nếu không thỏa tiến hành hoán vị hai phần tử này

Bước 2: Lặp lại điều trên sẽ có được phần tử lớn nhất xuống cuối và loại phần tử này khỏi mảng đang xét

Bước 3: Tiếp tục thực hiện các bước trên đối với mảng để đưa dần các phần tử lớp nhất về sau

Bước 4: Thực hiện cho tới khi mảng chỉ còn 1 phần tử.

Complexity evaluations

O(n^2) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

4. Shaker Sort

Idea

Đây là một biến thể của Bubble Sort, ngoài việc duyệt dãy từ đầu đến cuối, nó còn thực hiện duyệt từ cuối đến đầu để đẩy phần tử nhỏ nhất lên đầu và phần tử lớn nhất xuống cuối. Quá trình này lặp lại cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

Bước 1: l = 1; r = n; // từ l đến r là đoạn cần sắp xếp

k = n; // ghi nhận vị trí k xảy ra hoán vị sau cùng

// để làm cơ sở thu hẹp đoạn l đến r.

Bước 2:

Bước 2a: j = r; // đẩy phần tử nhỏ về đầu mảng

Trong khi (j > l) thực hiện

Nếu a[j] < a[j-1]: a[j] ↔ a[j-1];

k = j; // lưu lại nơi xảy ra hoán vị.

j = j – 1;l = k; // loại các phần tử đã có thứ tự ở đầu dãy

Bước 2b: j = l; // đẩy phần tử lớn về cuối mảng

Trong khi (j < r) thực hiện

Nếu a[j] > a[j-1]: a[j] ↔ a[j+1];

k = j; // lưu lại nơi xảy ra hoán vị.

j = j + 1;

r = k; // loại các phần tử đã có thứ tự ở cuối dãy

Bước 3: Nếu l < r : Lặp lại bước 2.

Ngược lại: Dừng.

Complexity evaluations

O(n^2) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

5. Shell Sort

Idea

Thuật toán này sắp xếp dãy bằng cách chia dãy thành các đoạn nhỏ hơn, sau đó sử dụng phương pháp Insertion Sort để sắp xếp từng đoạn. Quá trình này được lặp lại với các đoạn ngày càng lớn cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

Bước 1: Khởi tạo giá trị h

Bước 2: Chia list thành các sublist nhỏ hơn tương ứng với h

Bước 3: Sắp xếp các sublist này bởi sử dụng sắp xếp chèn (Insertion Sort)

Bước 4: Lặp lại cho tới khi list đã được sắp xếp

Complexity evaluations

Tùy thuộc vào khoảng cách giữa các đoạn, nhưng trung bình là O(n log n) đến O(n^2) trong trường hợp xấu nhất.

6. Heap Sort

Idea

Thuật toán này sử dụng cấu trúc dữ liệu heap để sắp xếp dãy. Đầu tiên, dãy ban đầu được chuyển thành một heap. Sau đó, phần tử lớn nhất (trong trường hợp sắp xếp tăng dần) được di chuyển vào cuối dãy và loại bỏ khỏi heap. Quá trình này được lặp lại cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

B1: Build heap tree

Tạo cây heap với phần tử đầu là parent, phần tử child là 2 *parentIndex + 1 và 2 * parentIndex + 2. Và cứ tiếp tục cho đến phần tử cuối của mảng

B2: Sắp xếp cho heap tree về dạng max heap tree Đổi vị trí tất cả phần tử sao cho mỗi parent >= childs của nó bằng hàm updateHeapAt

B3: Đổi vị trí của phần tử đầu và cuối rồi cắt phần tử cuối ra khỏi heap tree không xét đến nó nữa. Rồi quay lại B2 cho đến khi hoàn tất.

Complexity evaluations

O(n log n) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

7. Merge Sort

Idea

Thuật toán này sử dụng phương pháp chia để trị. Dãy ban đầu được chia thành hai phần bằng nhau, sau đó tiếp tục chia đôi đến khi mỗi phần chỉ còn một phần tử. Sau đó, các phần tử được trộn lại theo thứ tự để tạo ra dãy đã sắp xếp.

Step-by-step descriptions

B1: Tách mảng thành 2 nửa trái phải, tiếp tục gọi đệ quy để tách mỗi nửa thành 2 phần trái phải cho đến khi mỗi nửa chỉ còn 1 phần tử

B2: Sau khi tách xong thì sẽ sắp xếp 2 nửa trái phải và gộp lại thành một và cứ tiếp tục đệ quy cho đến khi tạo thành 1 mảng hoàn chỉnh

Complexity evaluations

O(n log n) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

8. Quick Sort

Idea

Thuật toán này sử dụng phương pháp chia để trị. Một phần tử được chọn làm "pivot" và các phần tử nhỏ hơn pivot được đưa vào bên trái, các phần tử lớn hơn pivot được đưa vào

bên phải. Quá trình này được lặp lại cho đến khi các phần tử chỉ còn một phần tử. Sau đó, các dãy con được kết hợp lại để tạo ra dãy đã sắp xếp.

Step-by-step descriptions

B1: Chọn một phần tử làm pivot để tách mảng thành 2 phần

B2: Tách mảng thành 2 nửa, bé hơn hoặc bằng và lớn hơn

B3: Từ mỗi nửa đã tách, tiếp tục chọn pivot và tách tiếp thành 2 nửa và sắp xếp cho đến khi mỗi nửa chỉ còn 1 phần tử thì dừng

Complexity evaluations

O(n log n) - Độ phức tạp trung bình, nhưng trong trường hợp xấu nhất có thể là O(n^2) nếu pivot không được chọn tốt.

9. Counting Sort

Idea

Thuật toán này thích hợp cho dãy số nguyên có giới hạn khoảng giá trị. Nó đếm số lần xuất hiện của từng giá trị và sau đó sử dụng thông tin đó để đặt chính xác các phần tử vào vị trí của chúng trong dãy đã sắp xếp.

Step-by-step descriptions

Bước 1: Tìm ra phần tử lớn nhất (giả sử là max) từ mảng đã cho.

Bước 2: Khởi tạo một mảng count có độ dài là max+1 với tất cả các phần tử 0. Mảng này được sử dụng để lưu trữ số lượng các phần tử trong mảng.

Bước 3: Lưu trữ số lượng của từng phần tử tại chỉ mục tương ứng của chúng trong mảng count.

Bước 4: Lưu trữ tổng số lượng của các phần tử trong mảng count . Nó sẽ giúp ta đặt chính xác các phần tử vào chỉ mục của mảng đã sắp xếp.

Bước 5: Tìm chỉ số của từng phần tử của mảng ban đầu trong mảng count . Đặt phần tử tại chỉ số được tính.

Bước 6: Sau khi đặt mỗi phần tử vào đúng vị trí của nó, ta giảm số lượng của mảng count đi một.

Complexity evaluations

O(n + k) - Độ phức tạp trung bình và trong trường hợp xấu nhất, với k là phạm vi giá trị của dãy.

10. Radix Sort

Idea

Thuật toán này thích hợp cho dãy số nguyên không âm. Nó sắp xếp dãy dựa trên từng chữ số, bắt đầu từ chữ số thấp nhất và tiếp tục cho đến khi dãy được sắp xếp theo chữ số cao nhất.

Step-by-step descriptions

Bước 1 : k cho biết chữ số dùng để phân loại hiện hành

k = 0; (k = 0: hàng đơn vị; k = 1: hàng chục)

Bước 2 : Tạo các lô chứa các loại phần tử khác nhau

Khởi tạo 10 lô B0, B1, .. , B9 rỗng;

Bước 3 :

For i = 1 .. n do

Đặt ai vào lô Bt với t = chữ số thứ k của ai;

Bước 4 :

Nối B0, B1, .. , B9 lại (theo đúng trình tự) thành a.

Bước 5 :

k = k+1;

Nếu k < m thì trở lại bước 2.

Ngược lại: Dừng

Complexity evaluations

O(n * k) - Độ phức tạp trung bình và trong trường hợp xấu nhất, với k là số lượng chữ số.

11. Flash Sort

Idea

Thuật toán này thích hợp cho dãy số nguyên không âm có phạm vi giá trị lớn. Nó sử dụng một bước phân loại để phân chia dãy thành các phần và sau đó đệ quy sắp xếp từng phần riêng biệt. Quá trình này tiếp tục cho đến khi dãy được sắp xếp hoàn chỉnh.

Step-by-step descriptions

Bước 1: Tìm giá trị nhỏ nhất của các phần tử trong mảng (minVal) và vị trí phần tử lớn nhất của các phần tử trong mảng(max).

Bước 2: Khởi tạo 1 vector L có m phần tử (ứng với m lớp, trong source code lần này chọn số lớp bằng 0.45n).

Bước 3: Đếm số lượng phần tử các lớp theo quy luật, phần tử a[i] sẽ thuộc lớp k = int((m - 1) * (a[i]-minVal) / (a[max] - minVal)).

Bước 4: Tính vị trí kết thúc của phân lớp thứ j theo công thức L[j] = L[j] + L[j - 1] (j tăng từ 1 đến m - 1).

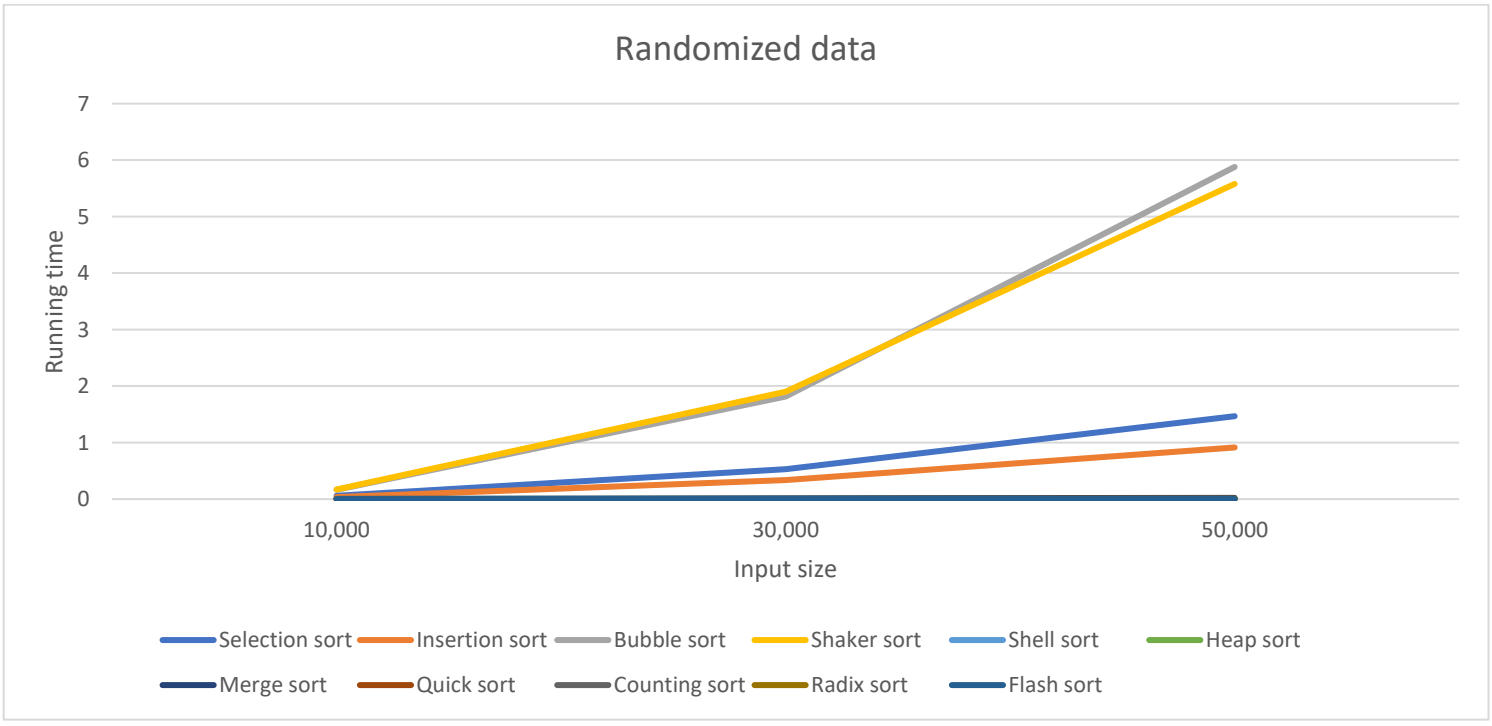
Complexity evaluations

O(n log n) - Độ phức tạp trung bình và trong trường hợp xấu nhất.

IV. EXPERIMENTAL RESULTS AND COMMENTS

1. Randomized data

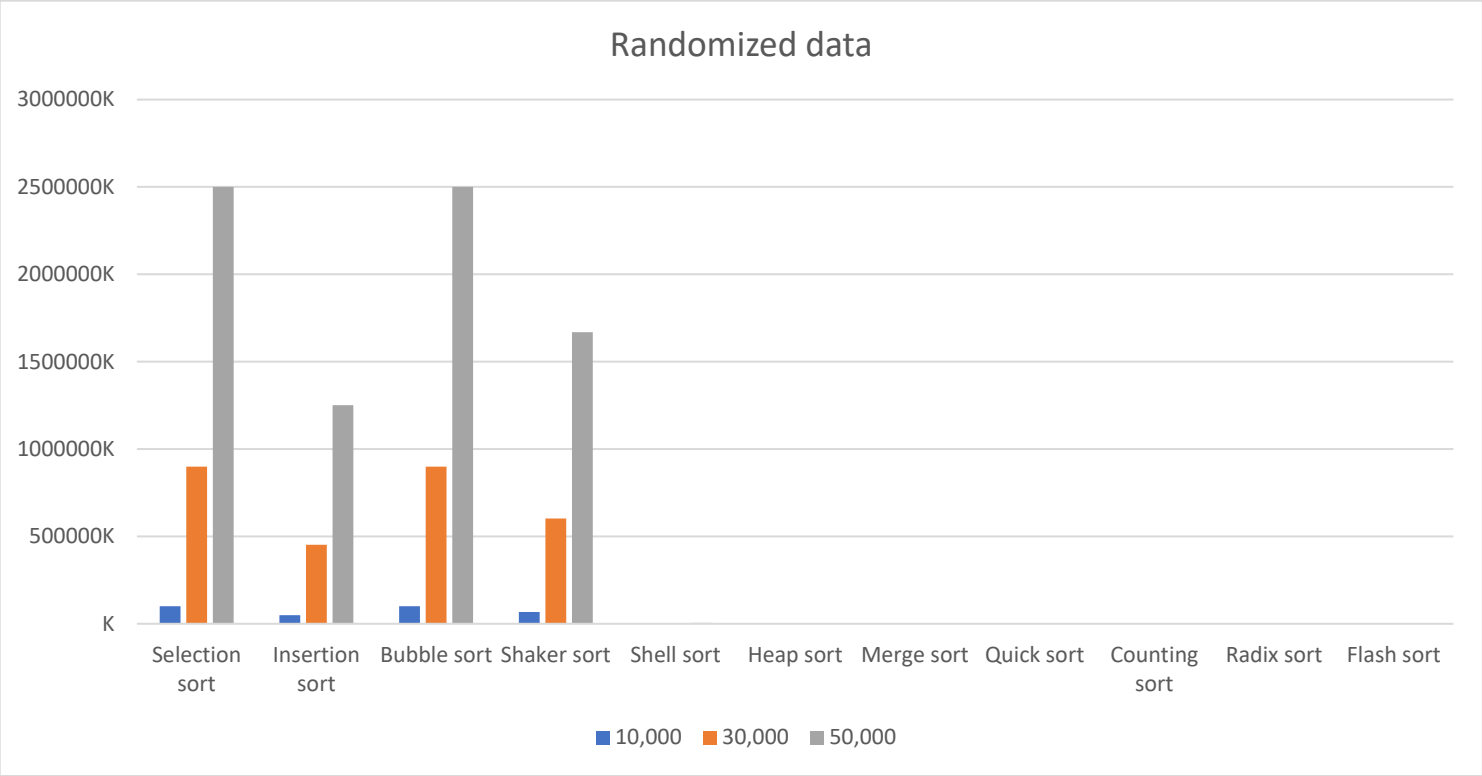
Data order: Randomized						
Data size	10,000		30,000		50,000	
Resulting statics	Running time	Comparision	Running time	Comparision	Running time	Comparision
Selection sort	0.059	100019998	0.529	900059998	1.467	2500099998
Insertion sort	0.036	50096183	0.338	452071438	0.914	1250497473
Bubble sort	0.164	100009999	1.813	900029999	5.879	2500049999
Shaker sort	0.17	66920546	1.901	602416029	5.578	1668740213
Shell sort	0.001	630438	0.005	2331067	0.011	4629303
Heap sort	0.002	89996	0.007	269996	0.012	449996
Merge sort	0.004	337226	0.012	1104458	0.022	1918922
Quick sort	0.001	268649	0.003	918072	0.005	1571763
Counting sort	0	60004	0	179996	0.001	300004
Radix sort	0.001	140058	0.003	501172	0.007	850072
Flash sort	0	98814	0.001	301965	0.001	479127



Nhận xét:

Ở biểu đồ này ta thấy Flash sort là thuật toán có thời gian chạy thấp nhất. Bubble sort là thuật toán có thời gian chạy cao nhất.

Shaker sort và Bubble sort có sự gia tăng thời gian chạy khá nhanh theo số mẫu input đầu vào tăng dần.



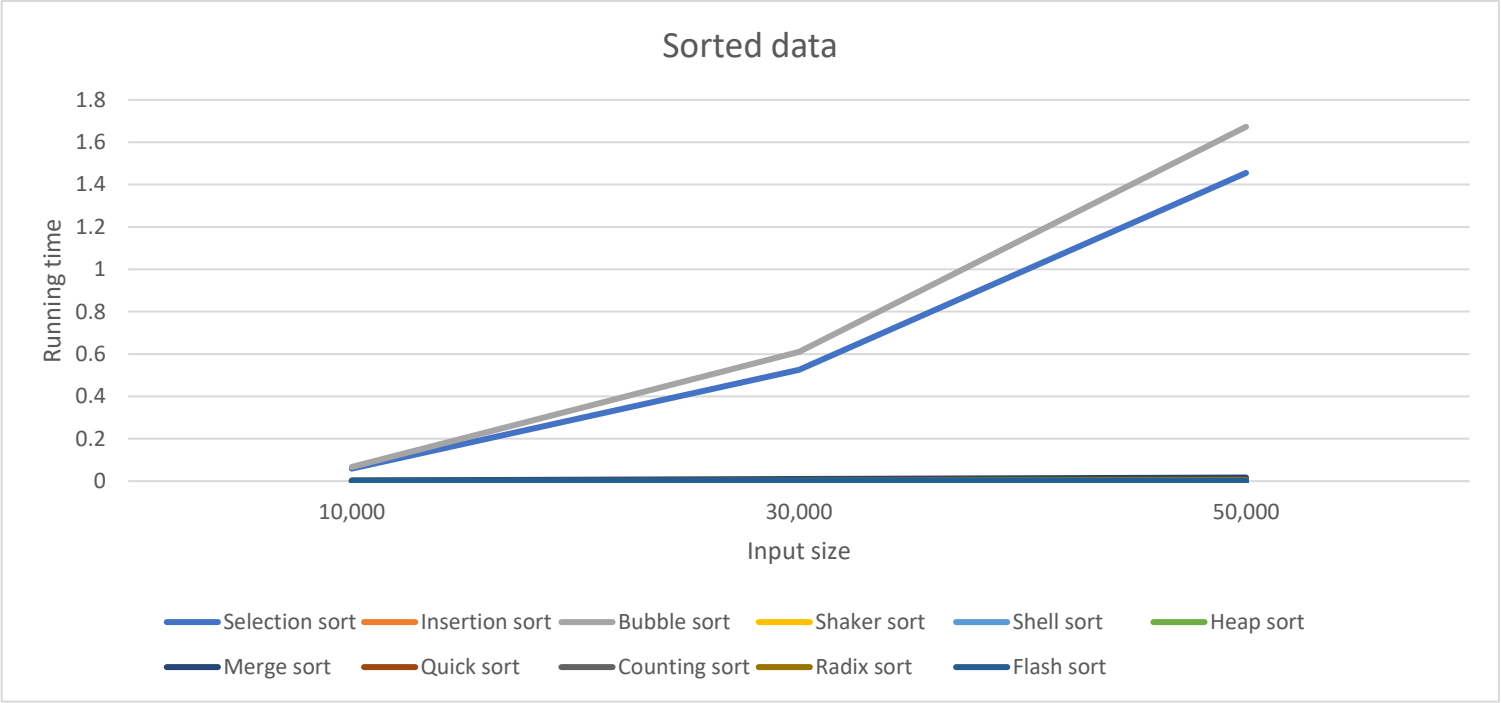
Nhận xét:

Ở biểu đồ này ta thấy Selection sort, Insertion sort, Bubble sort và Shaker sort là các thuật toán có số phép so sánh cao nhất và gia tăng theo số lượng mẫu input. Còn các thuật toán còn lại là thì số phép so sánh khá thấp và ổn định với số lượng mẫu input.

2. Sorted data

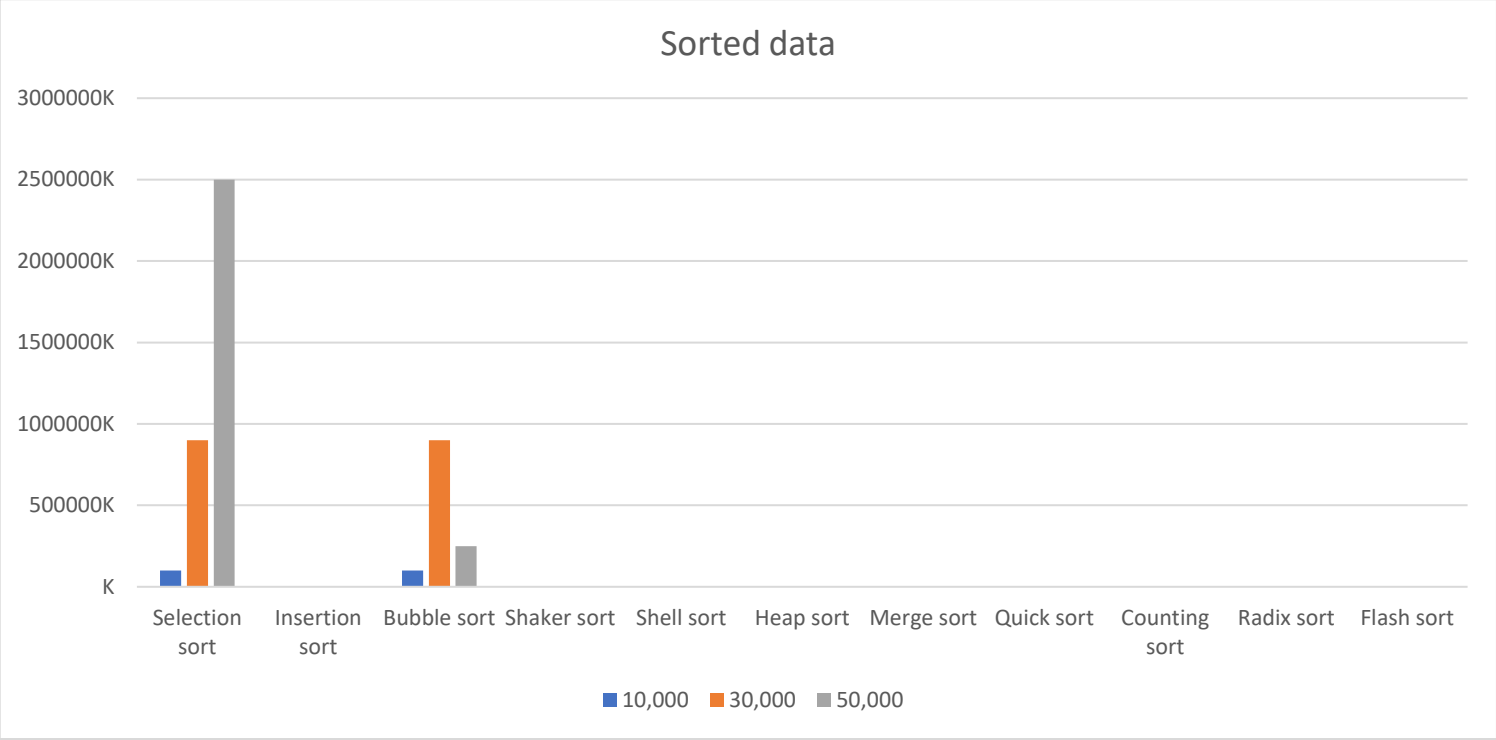
Data order: Sorted						
Data size	10,000		30,000		50,000	
Resulting statics	Running time	Comparision	Running time	Comparision	Running time	Comparision
Selection sort	0.059	100019998	0.525	900059998	1.455	2500099998
Insertion sort	0	29998	0	89998	0	149998
Bubble sort	0.067	100009999	0.61	900029999	1.673	250004999
Shaker sort	0	20002	0	60002	0.001	100002
Shell sort	0	360042	0.001	1170050	0.002	2100049

Heap sort	0.002	89996	0.005	269996	0.009	449996
Merge sort	0.003	337226	0.01	1104458	0.017	1918922
Quick sort	0.001	154959	0	501929	0.001	913850
Counting sort	0	60004	0	180004	0.001	300004
Radix sort	0.001	140058	0.003	510072	0.006	850072
Flash sort	0.001	127992	0.001	383992	0.001	639992



Nhận xét:

Ở biểu đồ này ta thấy Bubble sort và Selection sort là 2 thuật toán có thời gian chạy cao nhất và tăng khá nhanh tương ứng với số lượng đầu vào. Các thuật toán còn lại thì khá ổn định với kích cỡ đầu vào tăng dần.



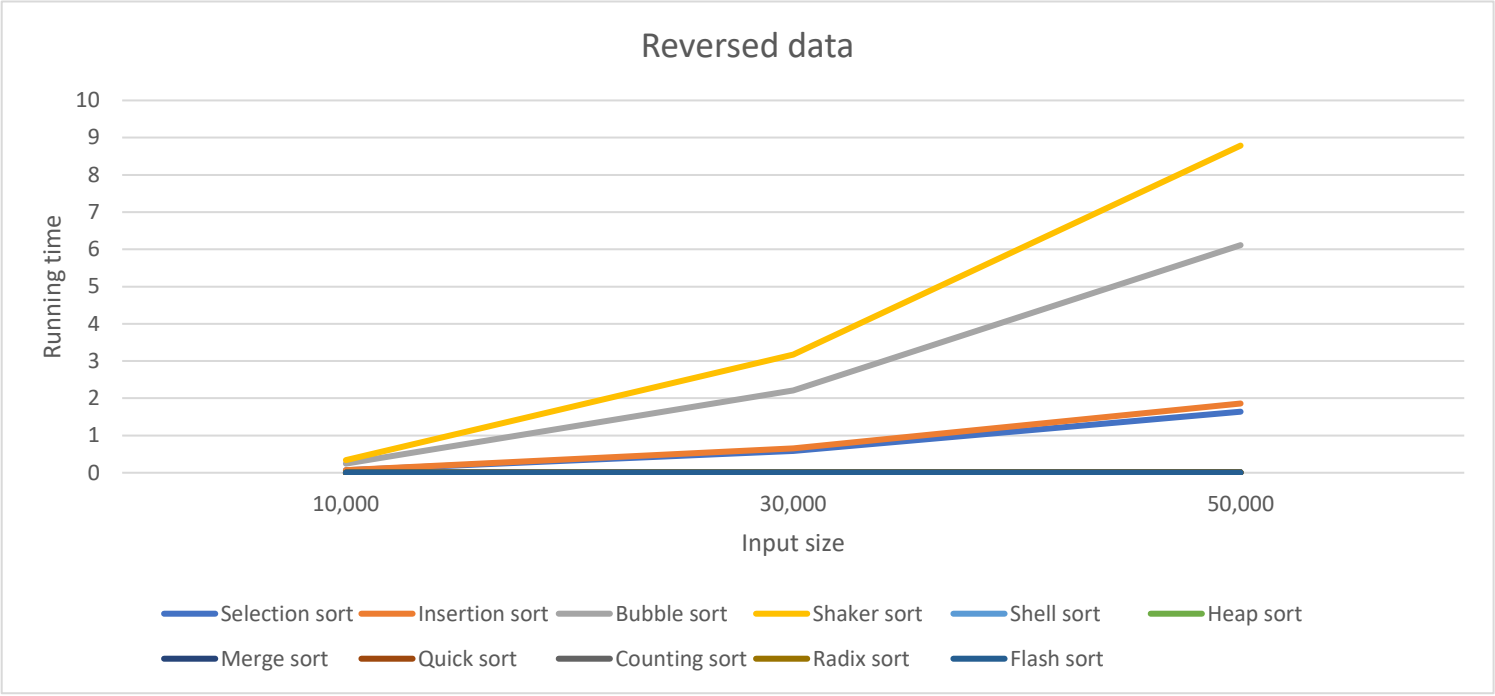
Nhận xét:

Ở biểu đồ này ta thấy Selection sort là thuật toán có số phép so sánh cao nhất và gia tăng nhanh theo số lượng mẫu input. Ngoại trừ bubble sort tăng không ổn định thì các thuật toán còn lại là thì số phép so sánh khá thấp và ổn định với số lượng mẫu input tăng dần.

3. Reversed data

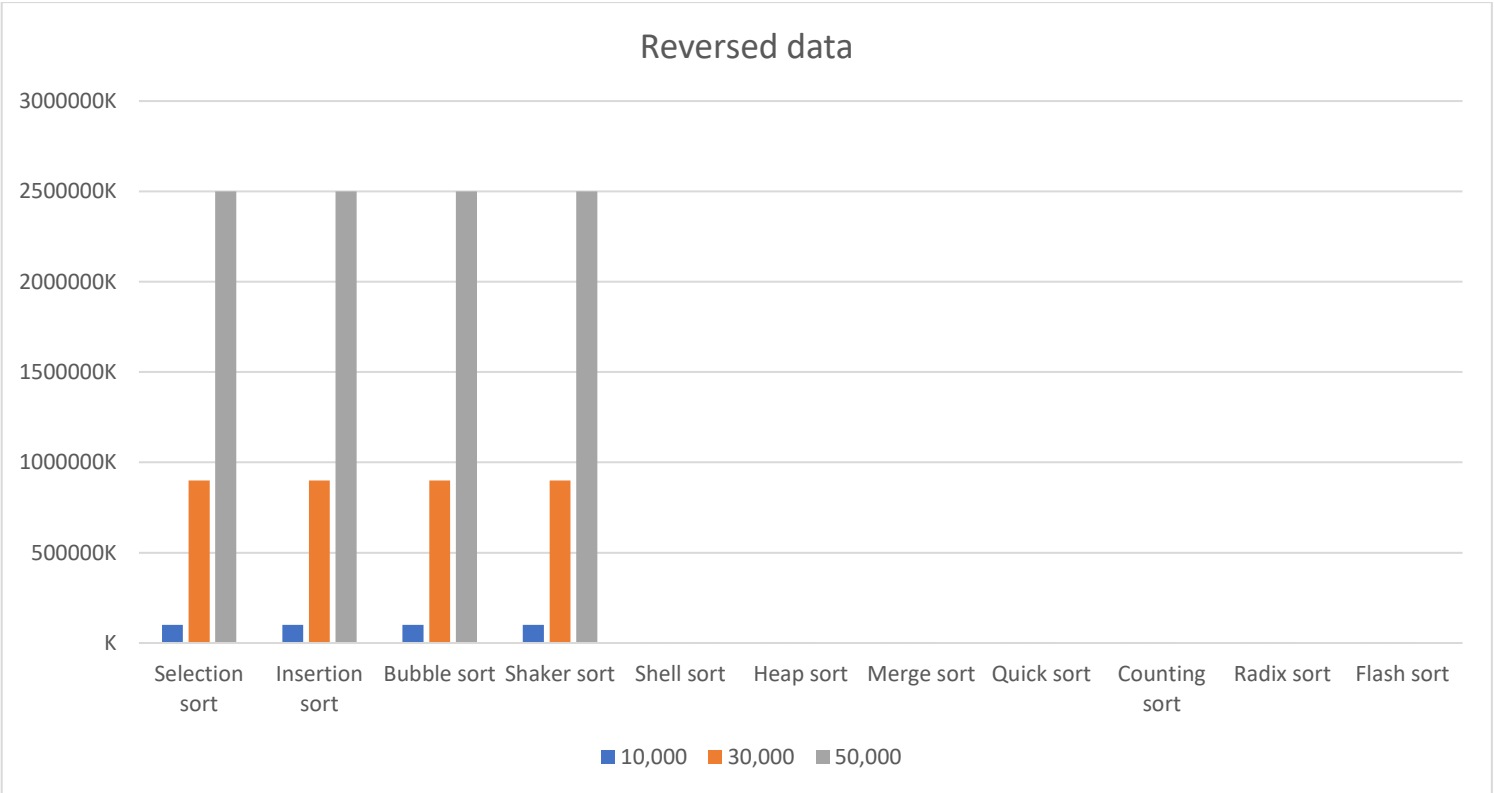
Data order: Reversed						
Data size	10,000		30,000		50,000	
Resulting statics	Running time	Comparision	Running time	Comparision	Running time	Comparision
Selection sort	0.067	100019998	0.586	900059998	1.639	2500099998
Insertion sort	0.07	100009999	0.656	900029999	1.859	2500049999
Bubble sort	0.245	100009999	2.213	900029999	6.113	2500049999
Shaker sort	0.339	100005001	3.17	900015001	8.787	2500025001
Shell sort	0.001	475175	0.001	1554051	0.003	2844628
Heap sort	0.001	89996	0.005	269996	0.009	449996
Merge sort	0.003	337226	0.01	1104458	0.017	1918922
Quick sort	0	164975	0.001	531939	0.001	963861
Counting sort	0	60004	0	180004	0.001	300004
Radix sort	0.001	140058	0.003	510072	0.006	850072

Flash sort	0.001	110501	0.001	331501	0.001	552501



Nhận xét:

Ở biểu đồ này ta thấy Shaker sort là thuật toán có thời gian chạy cao nhất và tăng nhanh tương ứng với số lượng đầu vào. Sau đó là Bubble sort với thời gian chạy khá cao, chỉ sau Shaker sort. Các thuật toán còn lại thì tăng khá chậm với kích cỡ đầu vào tăng dần.

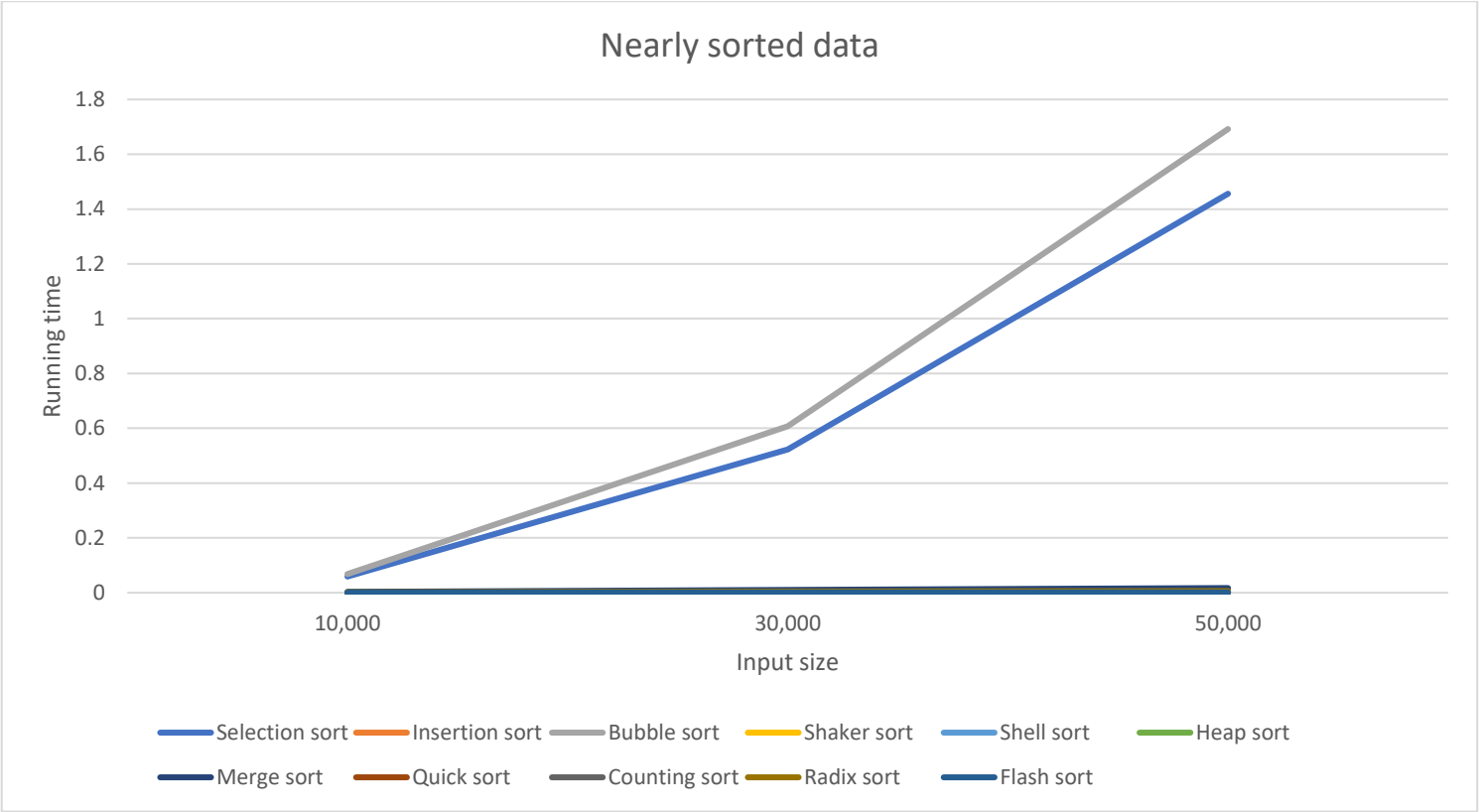


Nhận xét:

Ở biểu đồ này ta thấy Selection sort, Insertion sort, Bubble sort, Shaker sort là các thuật toán có số phép so sánh cao nhất và gia tăng nhanh theo số lượng mẫu input. Còn các thuật toán còn lại là thì số phép so sánh khá thấp và ổn định với số lượng mẫu input tăng dần.

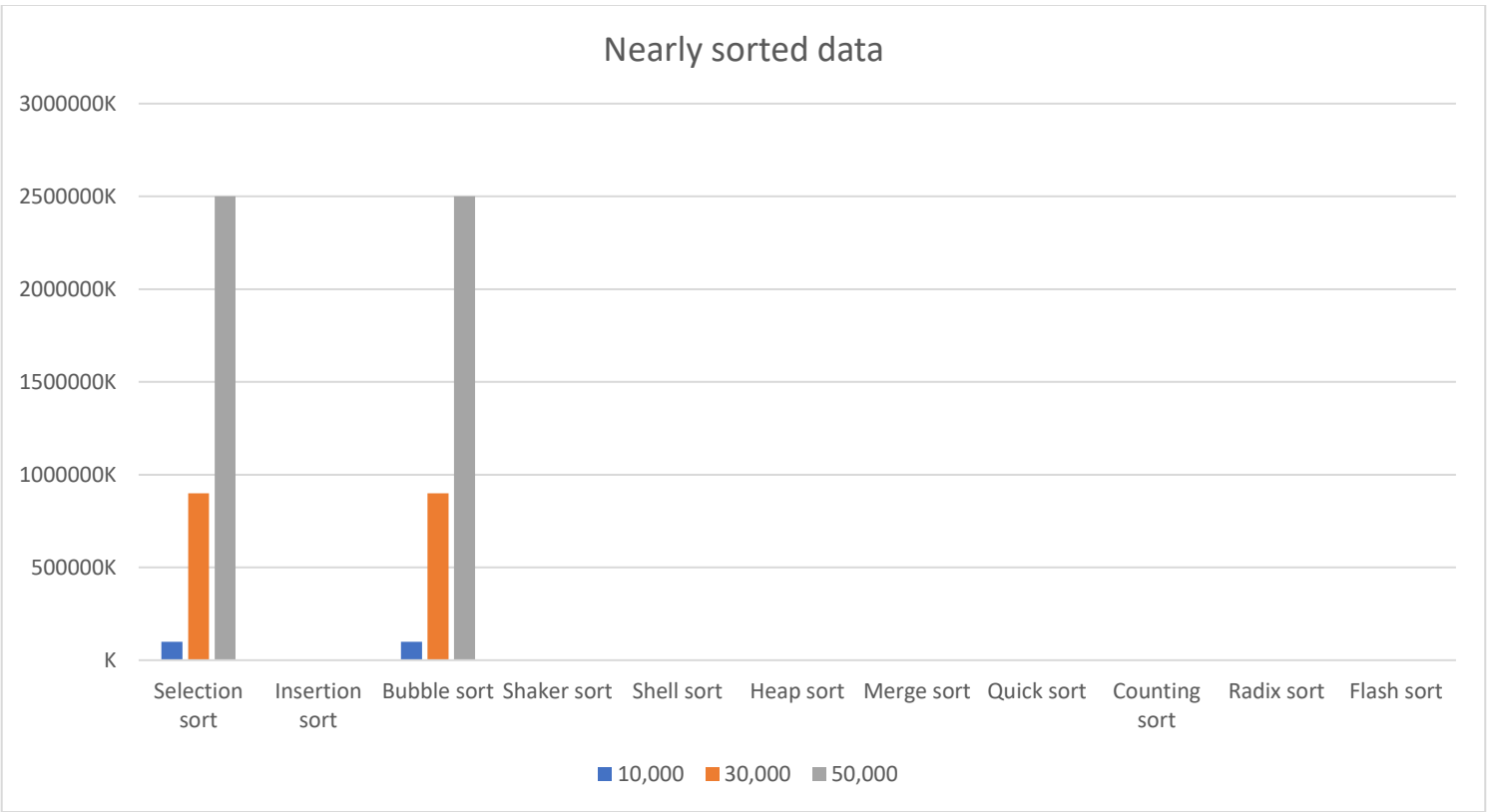
4. Nearly sorted data

Data order: Nearly sorted						
Data size	10,000		30,000		50,000	
Resulting statics	Running time	Comparision	Running time	Comparision	Running time	Comparision
Selection sort	0.059	100019998	0.523	900059998	1.456	2500099998
Insertion sort	0.001	219622	0	595570	0.001	858990
Bubble sort	0.068	100009999	0.607	900009999	1.692	2500049999
Shaker sort	0.001	219726	0.002	603170	0.002	873634
Shell sort	0	416560	0.002	1321108	0.003	2380925
Heap sort	0.002	89996	0.005	269996	0.009	449996
Merge sort	0.003	337226	0.01	1104458	0.017	1918922
Quick sort	0	154999	0.001	501973	0.001	913898
Counting sort	0	60004	0	180004	0.001	300004
Radix sort	0.001	140058	0.004	510072	0.006	850072
Flash sort	0	127968	0.001	383962	0.001	639962



Nhận xét:

Ở biểu đồ này ta thấy Bubble sort và Selection sort là hai thuật toán có thời gian chạy cao nhất và tăng nhanh tương ứng với số lượng đầu vào. Các thuật toán còn lại thì tăng khá chậm với kích cỡ đầu vào tăng dần.



Nhận xét:

Ở biểu đồ này ta thấy Selection sort và Bubble sort là hai thuật toán có số phép so sánh cao nhất và gia tăng nhanh theo số lượng mẫu input. Còn các thuật toán còn lại là thì số phép so sánh khá thấp và ổn định với số lượng mẫu input tăng dần.

5. Nhận xét chung về 11 thuật toán

Qua tổng hợp các biểu đồ tương ứng với 4 kiểu dữ liệu thì em có các nhận xét sau:

1. **Nhanh nhất**

Các thuật toán có độ phức tạp trung bình O(n log n): Heap Sort, Merge Sort, Quick Sort (trong trường hợp trung bình tốt).

Các thuật toán có độ phức tạp tuyến tính O(n): Counting Sort, Radix Sort (cho dữ liệu có phạm vi giá trị hữu hạn).

2. **Chậm nhất**

Các thuật toán có độ phức tạp trung bình và trong trường hợp xấu nhất O(n^2): Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort.

3. **Ổn định**

Các thuật toán ổn định là các thuật toán duy trì thứ tự tương đối của các phần tử có giá trị bằng nhau sau khi sắp xếp. Các thuật toán ổn định là: Insertion Sort, Merge Sort, Counting Sort, Radix Sort.

4. **Không ổn định**

Các thuật toán không ổn định là các thuật toán có thể thay đổi thứ tự của các phần tử có giá trị bằng nhau sau khi sắp xếp. Các thuật toán không ổn định là: Selection Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Quick Sort, Flash Sort.

5. **Hiệu suất tốt cho dữ liệu gần như đã sắp xếp:**

Insertion Sort và Shell Sort là hai thuật toán có hiệu suất tốt cho dữ liệu gần như đã sắp xếp. Vì các thuật toán này chèn hoặc chuyển đổi nhỏ số lần, vì vậy thời gian thực thi sẽ tương đối nhanh.

6. **Hiệu suất tốt cho dữ liệu ngẫu nhiên:**

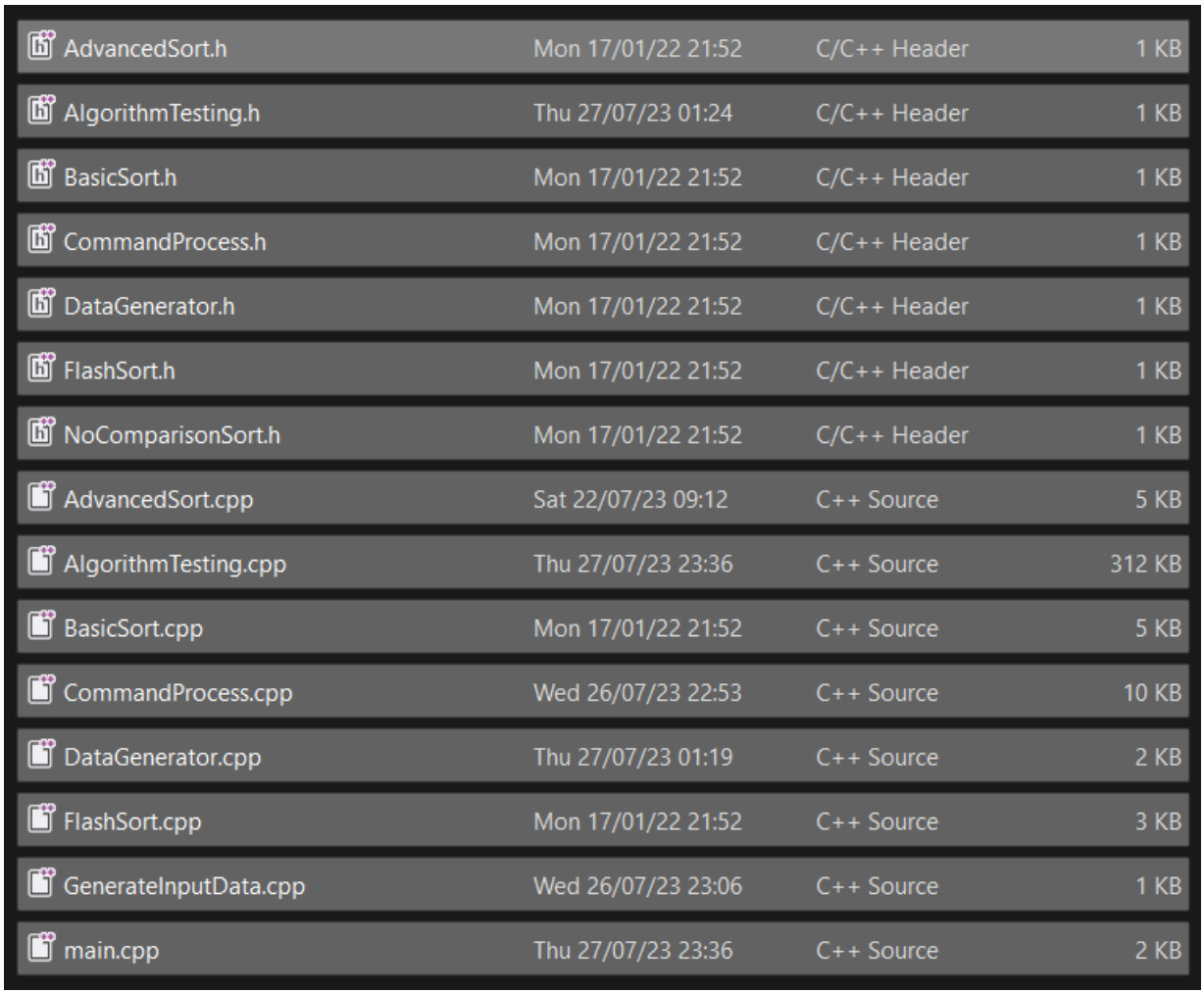
Merge Sort, Quick Sort, Heap Sort và Flash Sort thường có hiệu suất tốt với dữ liệu ngẫu nhiên. Điều này là do chúng thực hiện các phép so sánh và di chuyển các phần tử linh hoạt hơn so với các thuật toán sắp xếp khác.

7. **Hiệu suất tốt cho dữ liệu đã sắp xếp ngược:**

Radix Sort và Counting Sort có hiệu suất tốt khi dữ liệu đã sắp xếp ngược. Điều này là do chúng sử dụng các phép toán không dựa vào so sánh giữa các phần tử.

V. PROJECT ORGANIZATION AND PROGRAMMING NOTES

Cấu trúc file source code:



AdvancedSort.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
AlgorithmTesting.h	Thu 27/07/23 01:24	C/C++ Header	1 KB
BasicSort.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
CommandProcess.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
DataGenerator.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
FlashSort.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
NoComparisonSort.h	Mon 17/01/22 21:52	C/C++ Header	1 KB
AdvancedSort.cpp	Sat 22/07/23 09:12	C++ Source	5 KB
AlgorithmTesting.cpp	Thu 27/07/23 23:36	C++ Source	312 KB
BasicSort.cpp	Mon 17/01/22 21:52	C++ Source	5 KB
CommandProcess.cpp	Wed 26/07/23 22:53	C++ Source	10 KB
DataGenerator.cpp	Thu 27/07/23 01:19	C++ Source	2 KB
FlashSort.cpp	Mon 17/01/22 21:52	C++ Source	3 KB
GenerateInputData.cpp	Wed 26/07/23 23:06	C++ Source	1 KB
main.cpp	Thu 27/07/23 23:36	C++ Source	2 KB

- Trong file BasicSort là các thuật toán cơ bản: Selection sort, Insertion sort, Bubble sort, Shaker Sort, and Shell sort.
- Trong file AdvancedSort là các thuật toán tối ưu: Heap sort, Merge sort, and Quicksort.
- Trong file NoComparisonSort là các thuật toán không so sánh: Counting sort và Radix sort.
- Còn thuật toán Flash sort thì nằm trong file FlashSort.
- Trong đồ án, em không dùng thư viện hay cấu trúc dữ liệu đặc biệt nào.

VI. LIST OF REFERENCES

[1] Lectures from Dr. Nguyen Thanh Phuong

[2] https://iq.opengenus.org/time-complexity-of-selection-sort/

[3] https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques

[4] https://www.geeksforgeeks.org/binary-insertion-sort/

[5] https://www.geeksforgeeks.org/bubble-sort/

[6] https://www.javatpoint.com/cocktail-sort

[7] https://www.geeksforgeeks.org/cocktail-sort/

[8] https://www.tutorialspoint.com/Shell-Sort

[9] https://www.programiz.com/dsa/heap-sort

[10] https://www.educative.io/blog/data-structure-heaps-guide

[11] https://www.programiz.com/dsa/merge-sort

[12] https://www.geeksforgeeks.org/quick-sort/

[13] https://www.geeksforgeeks.org/iterative-quick-sort/

[14] https://www.geeksforgeeks.org/counting-sort/

[15] https://www.interviewcake.com/concept/java/counting-sort

[16] https://www.geeksforgeeks.org/radix-sort/

[17] https://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-12.php

[18] https://www.neubert.net/FSOIntro.html

[19] https://github.com/dungbachviet/SortingAlgorithms

[20] https://github.com/vanloc1808/HCMUS-DSA-SortingAlgorithms/tree/main/sources