Đại học Quốc gia TP Hồ Chí Minh
Đại học Khoa học tự nhiên TP Hồ Chí Minh
Khoa Công nghệ thông tin
Bộ môn Công nghệ phần mềm

Môn học: Kỹ Thuật Lập Trình
Năm học: 2017-2018
Học kì: II
GVLT: Phạm Minh Tuấn

BÁO CÁO ĐỒ ÁN

Thực hiện việc phát sinh ngẫu nhiên N số nguyên, sắp xếp N số nguyên trên tăng dần theo các thuật toán: insertion sort, bubble sort, shaker sort, shell sort, quick sort, merge sort, heap sort và radix sort. Với mỗi thuật toán, hãy thống kê: số phép so sánh, số phép gán, thời gian sắp xếp.

Hãy thực hiện với các trường hợp N = 100, 1000, 10000, 100000 và 1000000.

Thông tin sinh viên thực hiện

STT	MSSV	Họ và tên	Email	Điện thoại	Kí tên
1	1712834	Trần Minh Trí	ilovepeace123st@gmail.com	01215067718	**Trí**

Link github: https://github.com/StrugVN/DoAn2_1712834

(Bao gồm mã nguồn, file .exe đã biên dịch và các kết quả theo từng mục tại phần **I**)

Mục lục

I. Các chức năng đã thực hiện được

1. Thực hiện yêu cầu với kiểu dữ liệu int có sẵn:

- Danh sách các chức năng và chi tiết:

a) Phát sinh ngẫu nhiên mảng N phần tử:

- Mảng được cấp phát động theo số N được nhập.
- Mỗi phần tử được phát sinh ngẫu nhiên bằng công thức "***rand()*(RAND_MAX) +rand()***" (tỉ lệ phát sinh bằng nhau với các số trong khoảng ***[0;RAND_MAX^2)***) vì ***rand()*** trùng quá nhiều với N lớn như "1000000".
- Mảng đã phát sinh (và số lượng phần tử N) được in ra file "***0. Sample test.txt***" để đối chứng sau khi thực hiện chương trình.

b) Tải lại mảng (và số phần tử N) nếu phát hiện tồn tại file chứa:

- Với số lượng phần tử lớn (như 1000000), thực hiện toàn bộ các thuật toán tốn trong một lần chạy tốn nhiều thời gian nên chương trình có lựa chọn chỉ chạy 1 thuật toán trong một lần chạy.
- Lần chạy đầu tiên phát sinh ra mảng N phần tử và lưu vào file "***0. Sample test.txt***", ở các lần chạy tiếp theo nếu tồn tại file đó có thể tải lại toàn mảng ban đầu để chạy tiếp các thuật toán tiếp theo để các thuật toán đều chạy trên một mảng chưa sắp xếp giống nhau (chức năng ***d***).

c) Cài đặt mã nguồn các thuật toán sắp xếp theo yêu cầu

d) Cài đặt chạy và thống kê các thuật toán sắp xếp:

- Có thể lựa chọn chạy toàn bộ thuật hoặc chạy 1 thuật trong lần chạy hiện tại của chương trình.
- Mảng được sao chép thành 1 mảng phụ rồi chạy thuật sắp xếp lên mảng phụ đó nhằm mục đích toàn bộ thuật toán đều chạy trên 1 mảng chưa sắp xếp giống nhau.

- Các thông tin: Thời gian, số phép gán, số phép so sánh, mảng phụ đã sắp xếp được in toàn bộ ra file "***[tên kiểu sort].txt***" (để đối chiếu lại sau này) và màn hình console.

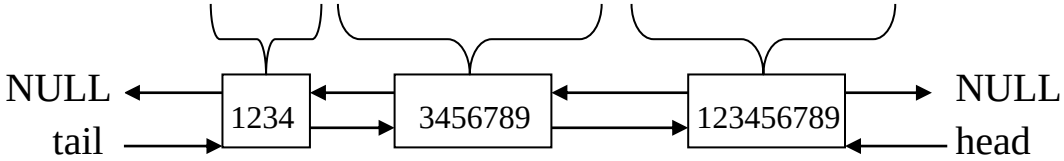
2. Thực hiện yêu cầu với số cực lớn chứa trong kiểu dữ liệu tự định nghĩa:

- Danh sách các chức năng và chi tiết:

a) Cài đặt struct chứa dữ liệu là số cực lớn

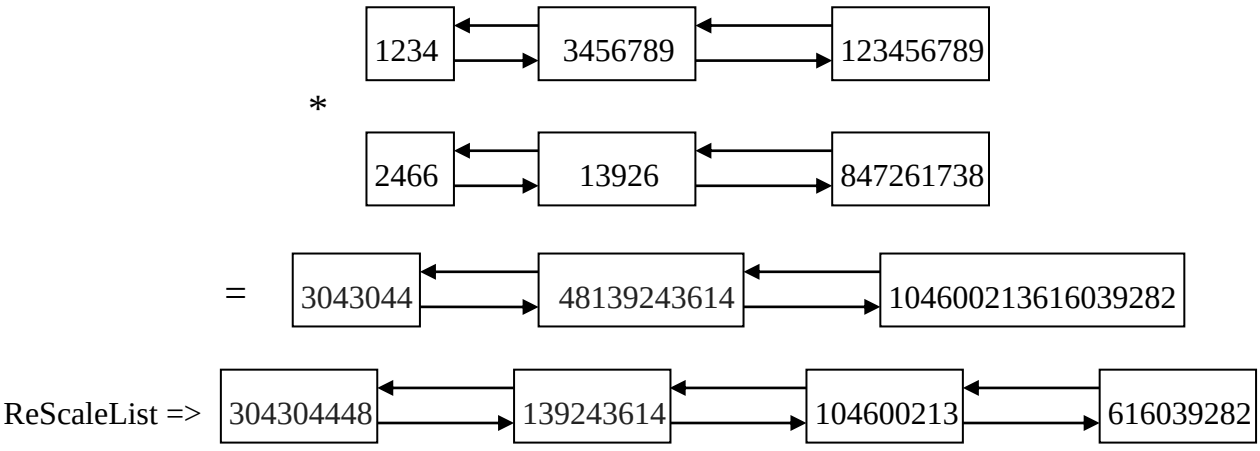
- Ý tưởng: Sử dụng danh sách liên kết kép (doubly linked list) với ***data*** mỗi ***node*** là 1 số ***__int64*** (***signed long long***). Số lớn sẽ được tách ra từ chuỗi sao cho mỗi node của danh sách chứa tối đa ***9*** chữ số, lấp dần từ node đầu tiên tính từ chữ số hàng đơn vị trở lên.

VD: 1 2 3 4 0 0 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9



- ***__int64*** chứa tối đa khoảng 10^{19}, data mỗi node chứa tối đa 9 chữ số $\Leftrightarrow 10^9$. Vậy kể cả thực hiện phép nhân, data của node vẫn chứa đủ kết quả. Sử dụng cách này cũng giúp giảm 9 lần số lượng vòng lặp so với sử dụng chuỗi.
- Hàm ***ReScaleList()*** giúp đưa danh sách về dạng ban đầu sau các phép tính:

VD: 1234003456789123456789 * 2466000013926847261738



b) Phát sinh ngẫu nhiên mảng N phần tử số cực lớn:

- Phát sinh chuỗi chứa số lớn ngẫu nhiên:

  CT cho phép nhập số lượng chữ số của số cực lớn mong muốn (***numOfDigit***), từ đó với mỗi phần tử của mảng tạo chuỗi động có ***m*** phần tử (***m*** được tạo ngẫu nhiên và có giá trị xấp xỉ ***numOfDigit***). Mỗi phần tử của chuỗi được gán ngẫu nhiên kí tự từ '0' đến '9' (trừ phần tử cuối là '\0' và phần tử đầu ngẫu nhiên từ '1' đến '9')
- Từ chuỗi trên nạp vào phần tử qua hàm ***getFromStr***.
- Lặp lại quá trình trên (hàm ***BigNumberGenerator***) cho toàn mảng.

c) Mã nguồn các thuật sắp xếp được chỉnh lại để thực hiện với kiểu dữ liệu này và kiểm tra kĩ để không gây rò rĩ bộ nhớ.

d) Cài đặt chạy và thống kê các thuật toán như ở trên (sao chép mảng phụ, sắp xếp trên mảng phụ, in thông tin và mảng phụ đã sắp xếp), được chỉnh lại để thực hiện với kiểu dữ liệu này.

3. Level 2 (Yêu cầu cũ):

- Danh sách các chức năng và chi tiết:

  a) Cài đặt Stack, Queue cần thiết

  b) Tính biểu thức:

  - Chuyển chuỗi chứa biểu thức thành 1 Queue là biểu thức dưới dạng Postfix.
  - Tính biểu thức từ Queue trên.

  c) Đọc từng dòng (từng biểu thức) từ file rồi tính biểu thức để nạp vào phần tử. Gặp các lỗi như giá trị vượt quá INT_MAX, biểu thức sai (không thể tính kết quả),…
- Các chức năng chạy thuật, thống kê còn lại tương tự như phần **1**.

4. Level 3 (Yêu cầu cũ):

- Danh sách các chức năng và chi tiết:
  - ***a)*** Cài đặt các toán tử cần thiết cho kiểu dữ liệu số lớn.
  - ***b)*** Cài đặt Stack, Queue cần thiết. Node và Stack của số lớn được cài đặt để tự động giải phóng bộ nhớ data khi hàm giải phóng của Node hoặc Stack được gọi.
  - ***c)*** Các hàm thực hiện tính biểu thức được chỉnh để phù hợp kiểu dữ liệu.
- Các chức năng chạy thuật, thống kê còn lại tương tự như phần **2.**

II. Kết quả thống kê (Kiểu dữ liệu int có sẵn)

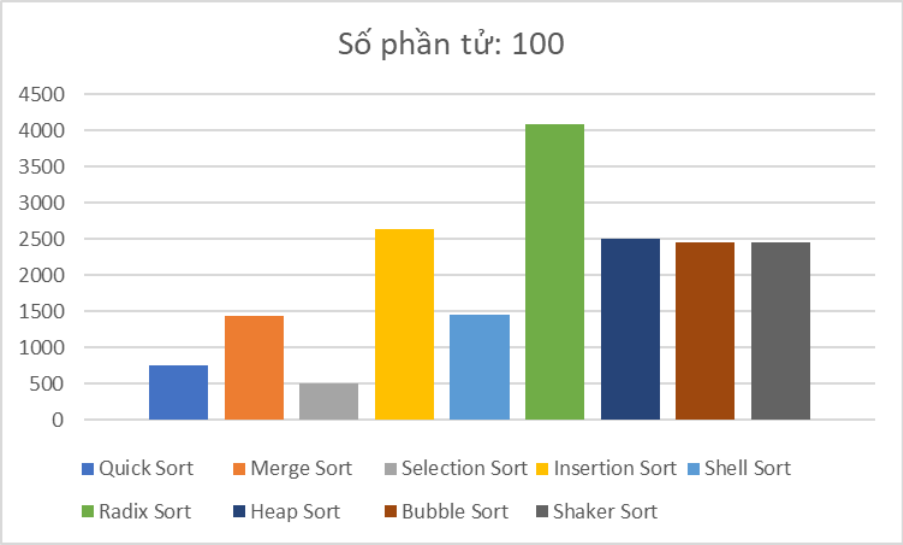
1. So sánh số lượng phép gán

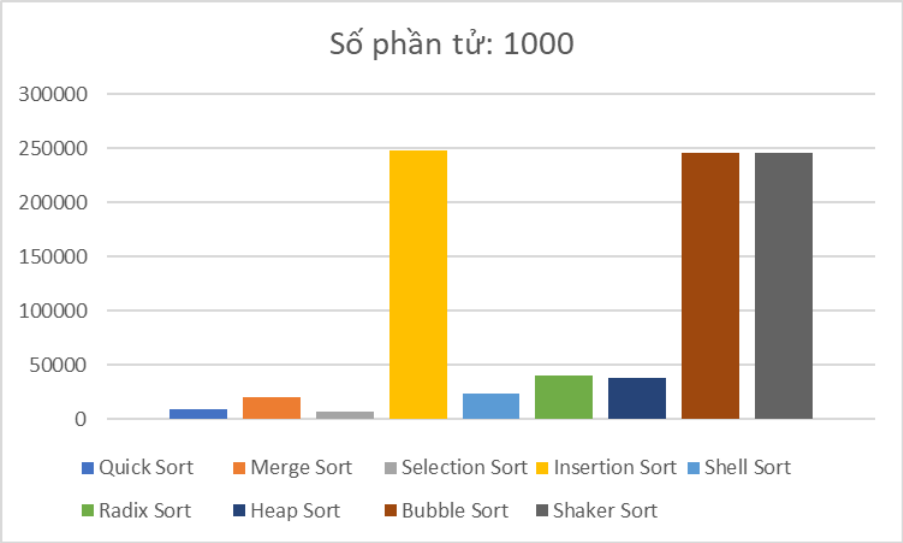
Kết quả số lượng phép gán của các thuật toán với số phần tử theo yêu cầu:

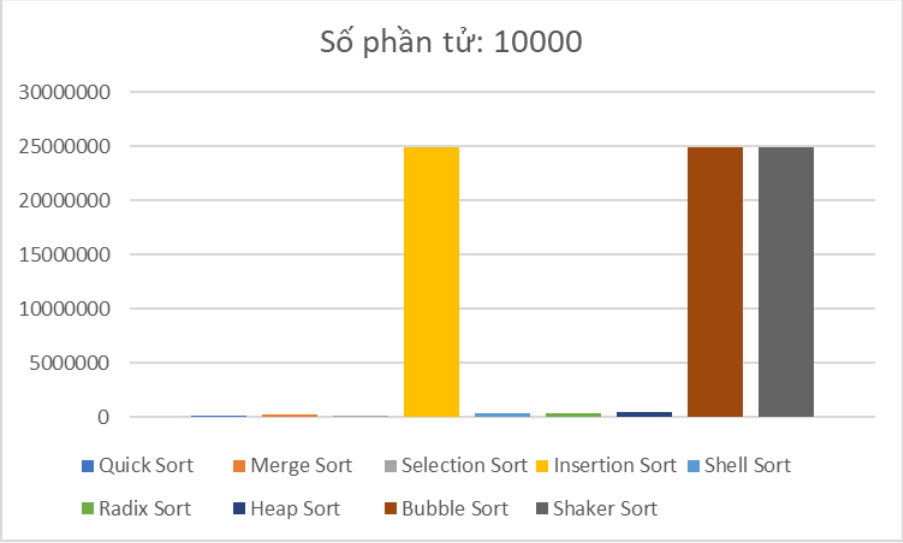
Tên thuật	*Số phần tử*				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	754	9672	114534	1511694	15992160
Merge Sort	1443	20951	277231	3437855	40902847
Selection Sort	504	7529	98479	1202782	14386201
Insertion Sort	2646	248389	24892454	2497408094	250175747014
Shell Sort	1448	23543	386918	5772743	85178932
Radix Sort	4096	40098	400108	4000108	40000104
Heap Sort	2506	37708	511860	6449712	77694828
Bubble Sort	2448	246391	24872456	2497208096	250173747016
Shaker Sort	2448	246391	24872456	2497208096	250173747016

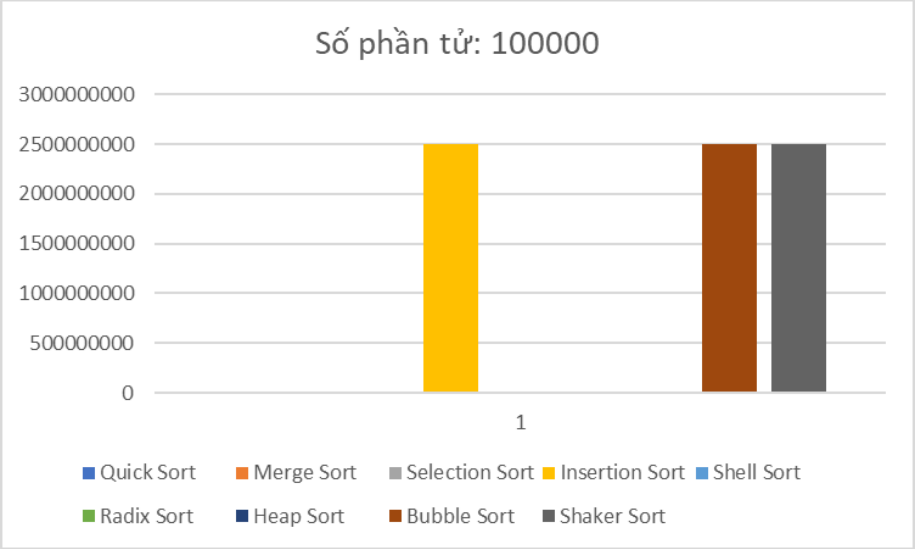
- Biểu đồ dựa vào bảng trên:

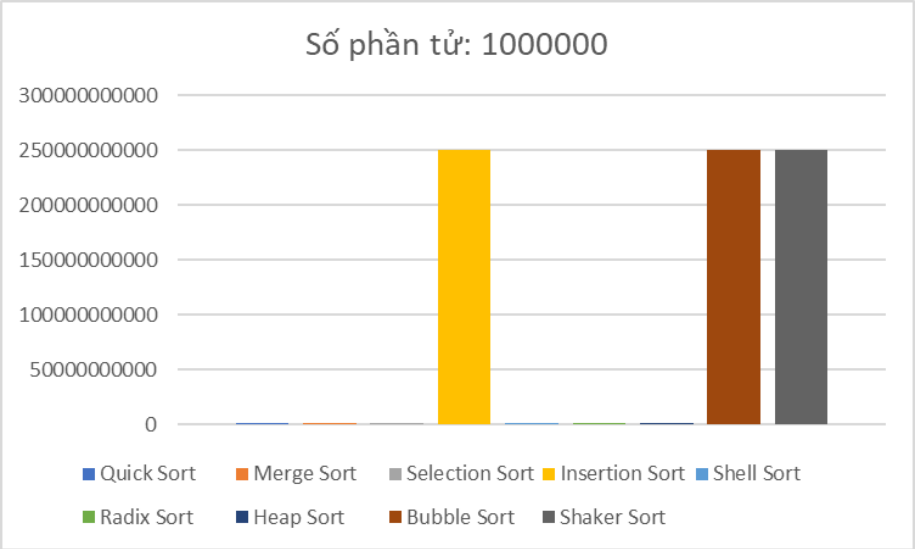
- Các biểu đồ với từng số phần tử:


Số phần tử: 100
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 1000
300000
250000
200000
150000
100000
50000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 10000
30000000
25000000
20000000
15000000
10000000
5000000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort





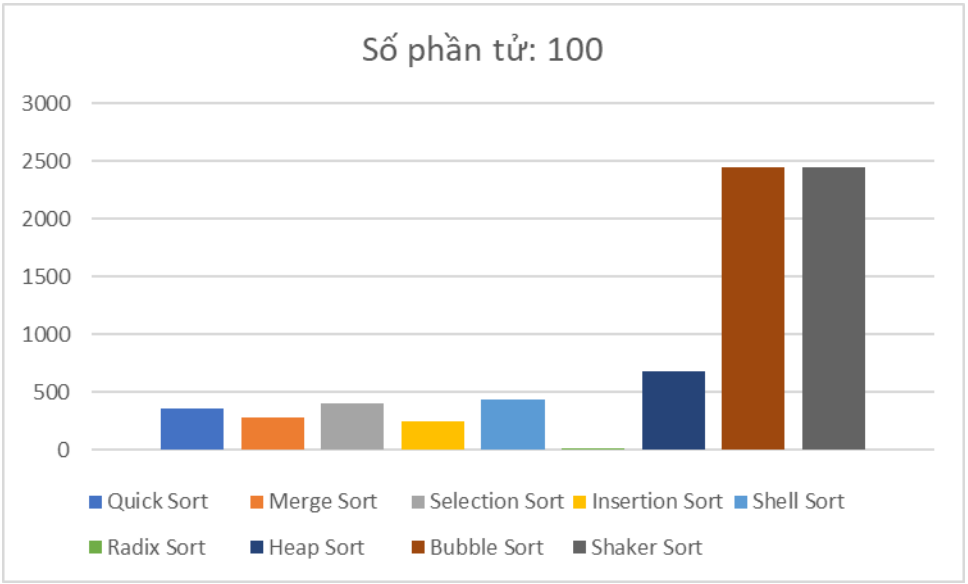
2. So sánh số lượng phép so sánh

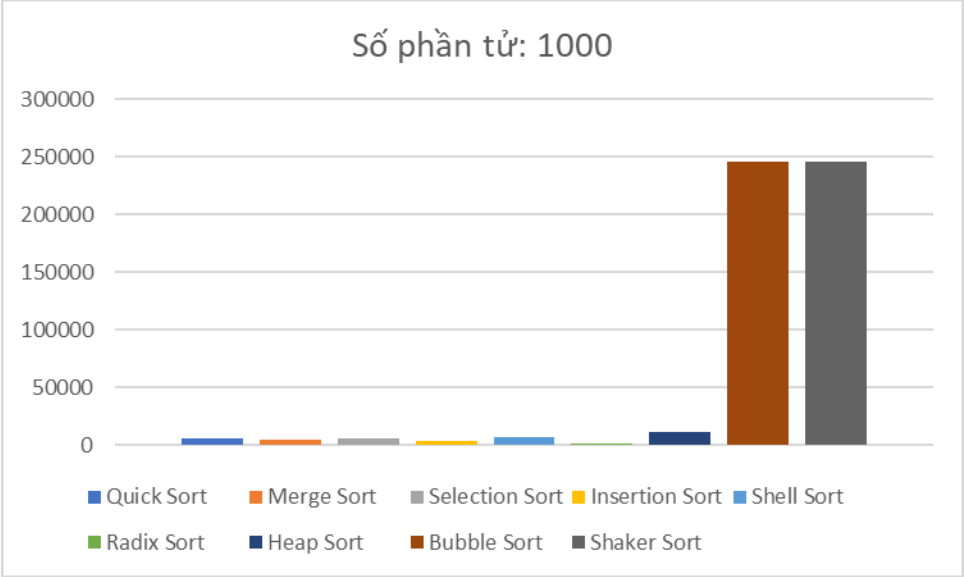
Kết quả số lượng phép so sánh của các thuật toán với số phần tử theo yêu cầu:

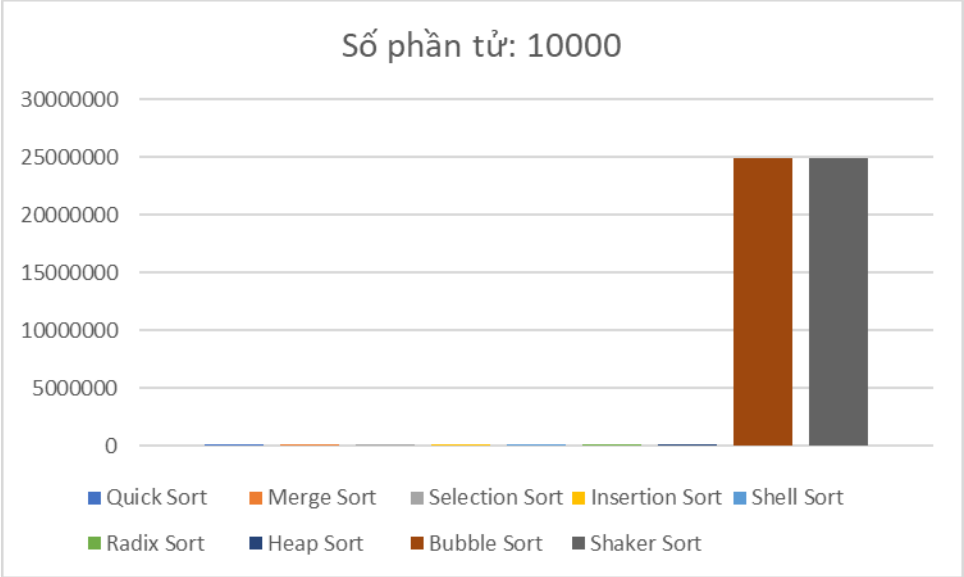
Tên thuật	Số phần tử				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	355	5673	74535	1111695	11992161
Merge Sort	285	4431	61271	776711	9392947
Selection Sort	405	6530	88480	1102783	13386202
Insertion Sort	245	4130	57708	743011	9102222
Shell Sort	442	7531	146908	2772731	49178918
Radix Sort	5	7	17	17	13
Heap Sort	680	11664	166535	2162886	26620497
Bubble Sort	2448	246391	24872456	2497208096	250173747016
Shaker Sort	2448	246391	24872456	2497208096	250173747016

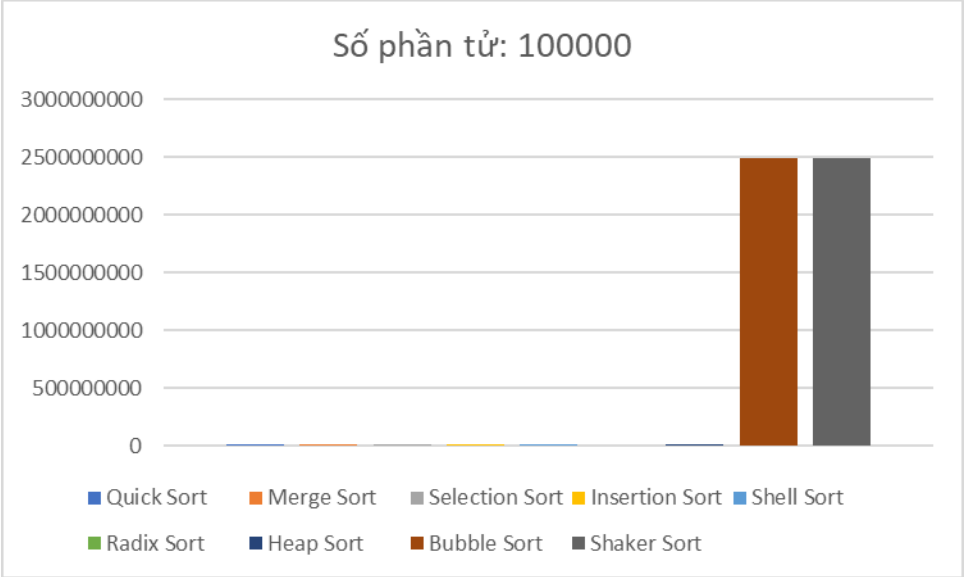
- Biểu đồ dựa vào bảng trên:

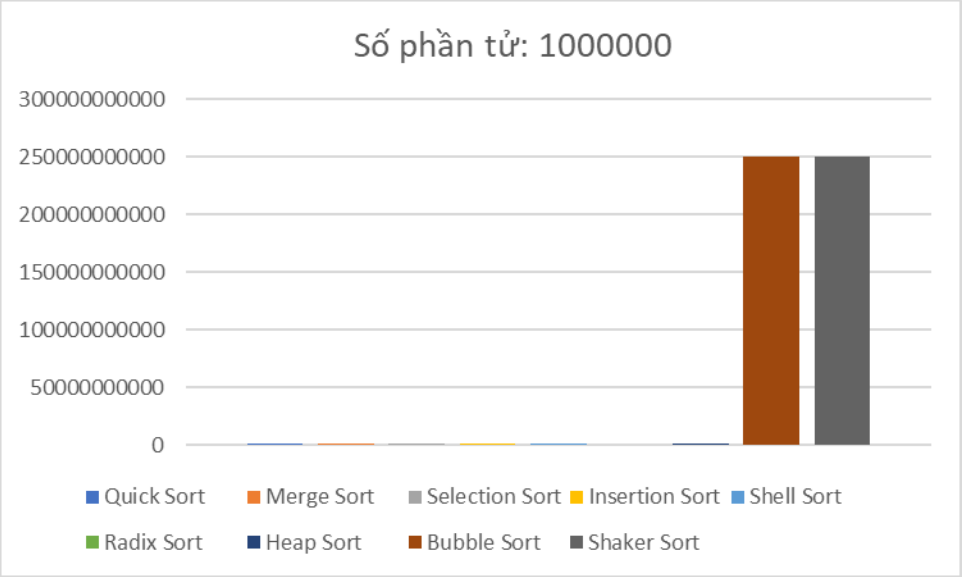
- Các biểu đồ với từng số phần tử:




Số phần tử: 1000
300000
250000
200000
150000
100000
50000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 10000
30000000
25000000
20000000
15000000
10000000
5000000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 100000
3000000000
2500000000
2000000000
1500000000
1000000000
500000000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort



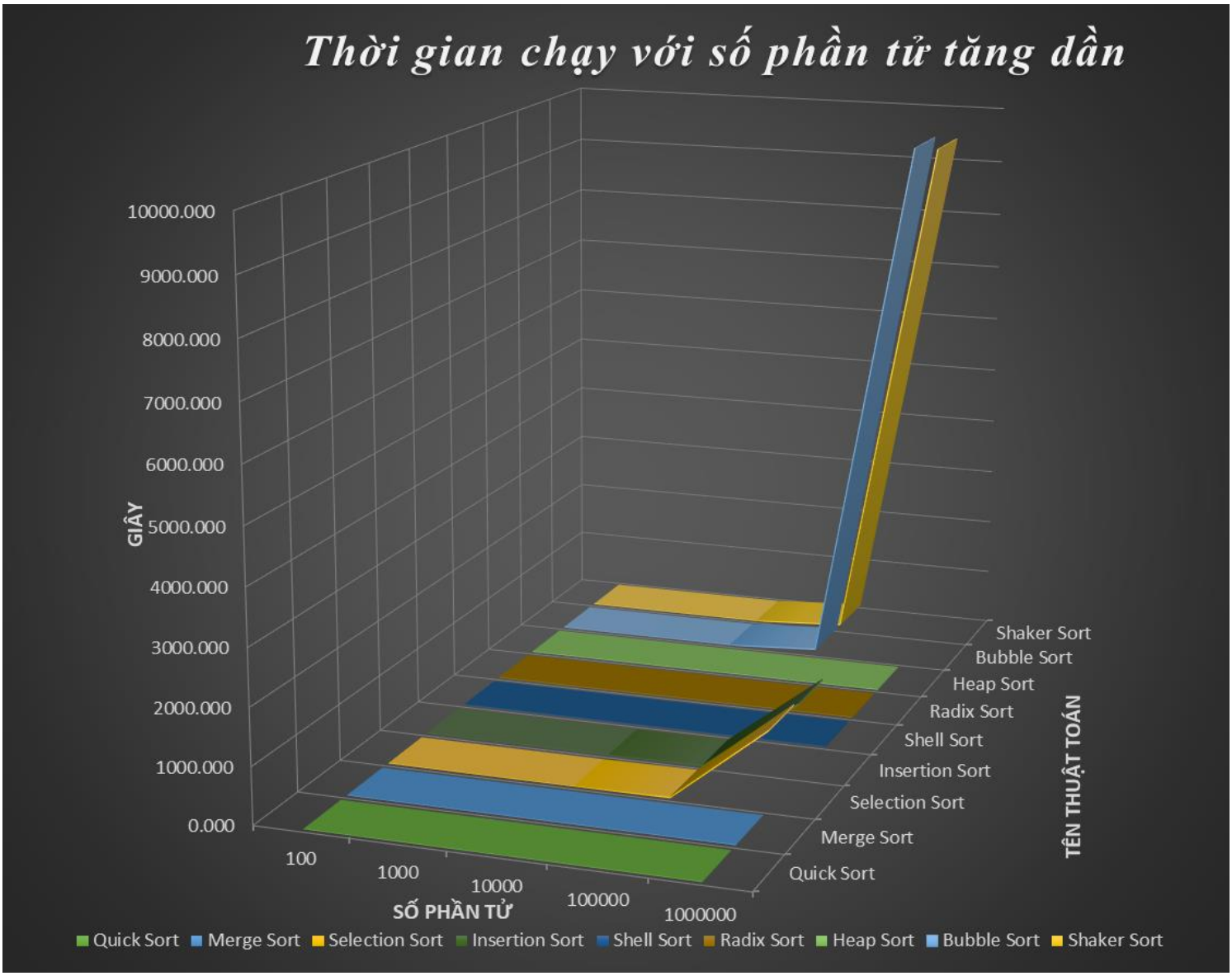
3. So sánh thời gian chạy:

a) Kết quả thời gian chạy của các thuật toán với số phần tử theo yêu cầu:

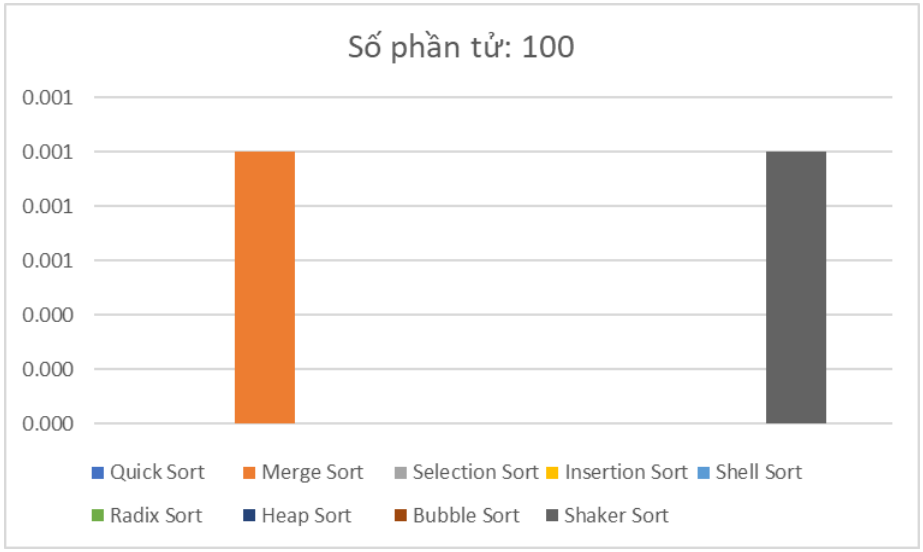
Tên thuật	***Số phần tử***				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	0.000	0.000	0.004	0.055	0.597
Merge Sort	0.001	0.001	0.010	0.111	1.002
Selection Sort	0.000	0.002	0.142	13.532	1375.656
Insertion Sort	0.000	0.001	0.090	7.611	1302.631
Shell Sort	0.000	0.000	0.003	0.034	0.630
Radix Sort	0.000	0.001	0.007	0.064	0.716
Heap Sort	0.000	0.001	0.012	0.109	1.513
Bubble Sort	0.000	0.009	1.091	96.550	9666.886
Shaker Sort	0.001	0.009	0.976	100.317	9412.003

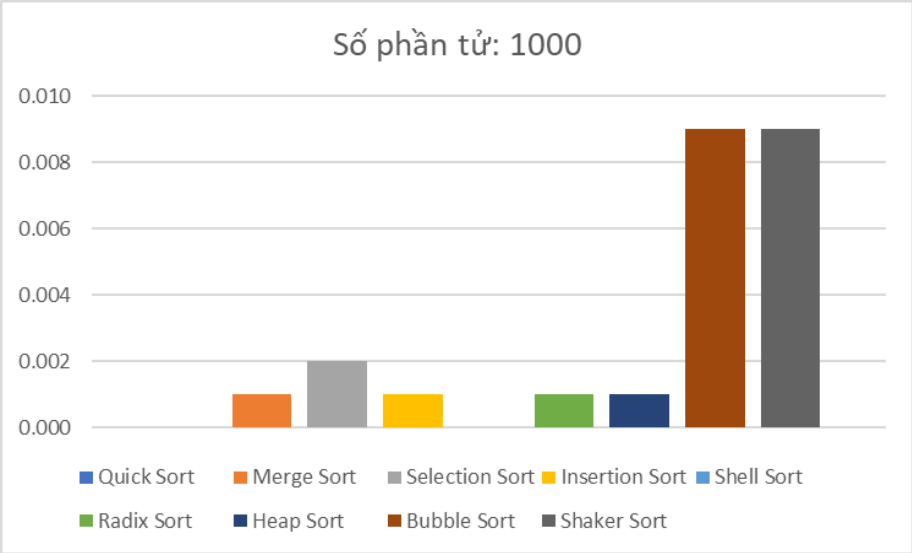
Đơn vị: ***s (giây)***

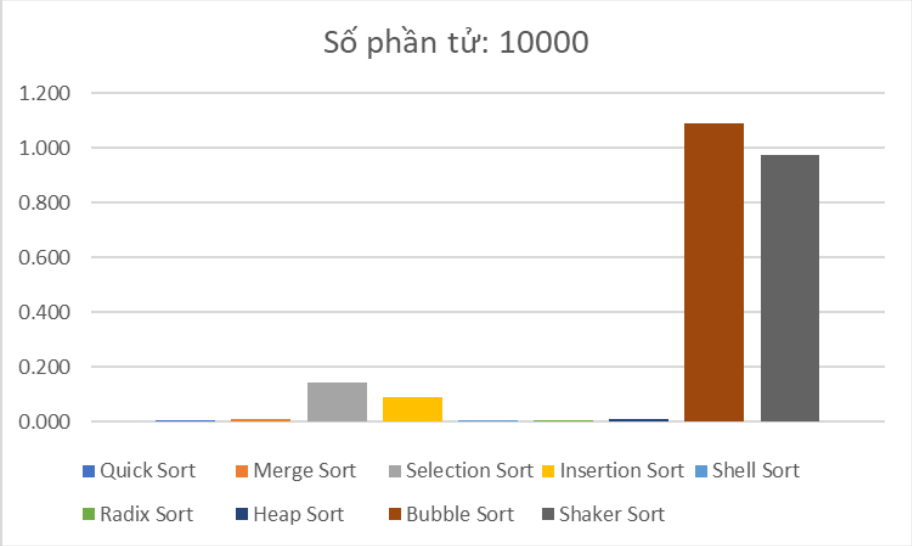
- Biểu đồ thời gian chạy dựa vào bảng trên:

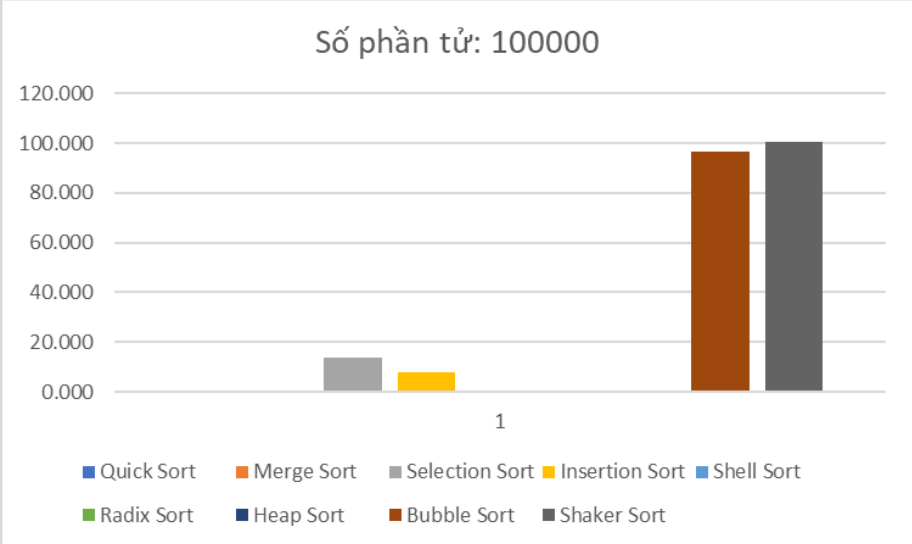


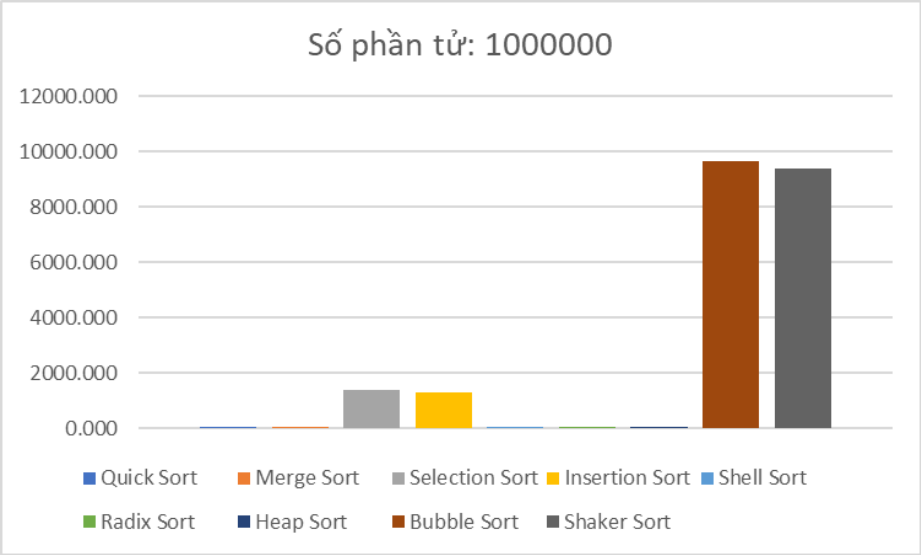
- Các biểu đồ với từng số phần tử trên:




Số phần tử: 1000
0.010
0.008
0.006
0.004
0.002
0.000
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 10000
1.200
1.000
0.800
0.600
0.400
0.200
0.000
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 100000
120.000
100.000
80.000
60.000
40.000
20.000
0.000
1
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


Số phần tử: 1000000
12000.000
10000.000
8000.000
6000.000
4000.000
2000.000
0.000
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort

III. Kết quả thống kê (Số lớn cực)

*Mỗi phần tử tạo ngẫu nhiên khoảng **35 chữ số***

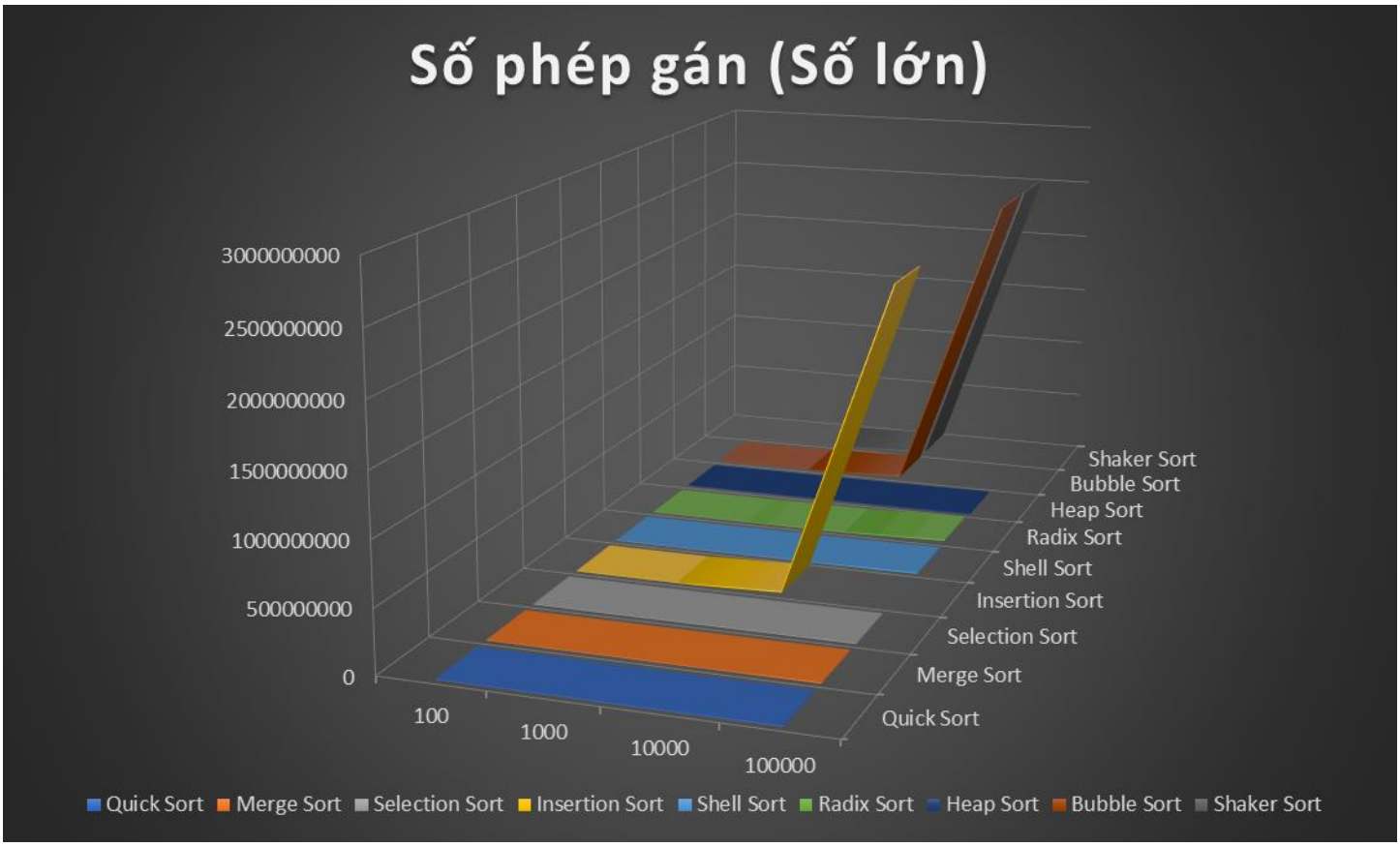
1. So sánh số lượng phép gán

Kết quả số lượng phép gán của các thuật toán với số phần tử theo yêu cầu:

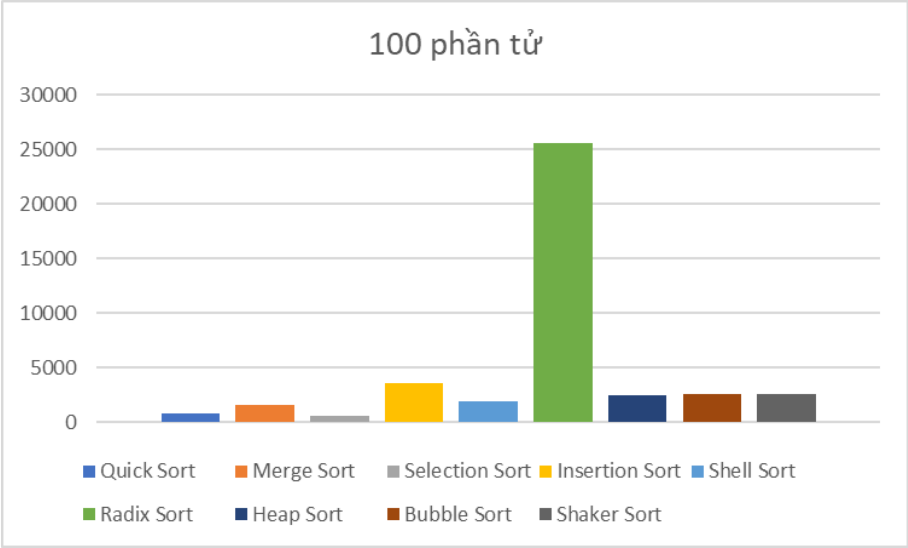
Tên thuật	***Số phần tử***				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	795	8586	119440	1355531	17272552
Merge Sort	1641	22949	297229	3637853	42902845
Selection Sort	621	8349	107159	1308537	X
Insertion Sort	3576	269804	25392224	2504825181	X
Shell Sort	1942	31731	516130	7337329	105504567
Radix Sort	25560	252362	2520363	25200369	252000365
Heap Sort	2470	37964	511504	6449504	77696052
Bubble Sort	2557	256223	25223227	2502802261	X
Shaker Sort	2557	256223	25223227	2502802261	X

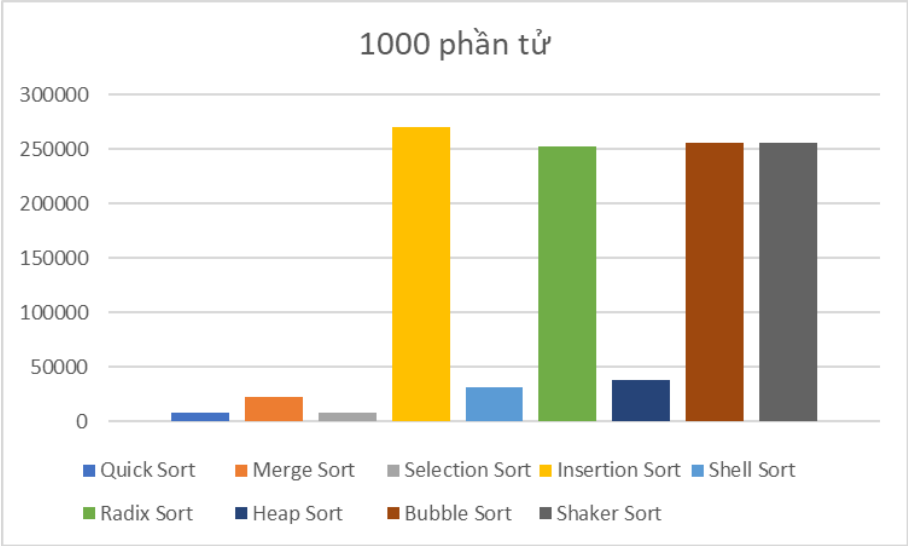
Chú thích: X ⇔ Thuật không hoàn thành trong 24 giờ

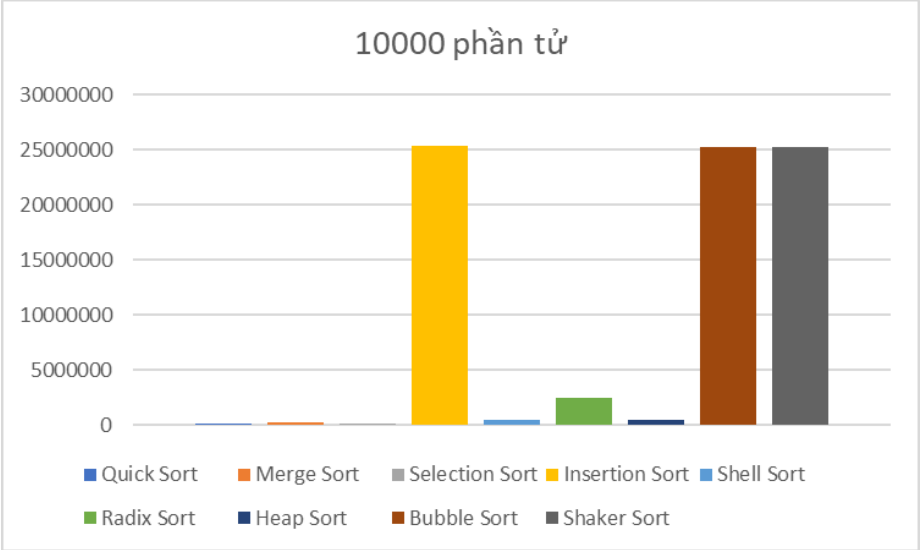
- Biểu đồ dựa vào bảng trên (trừ cột cuối)

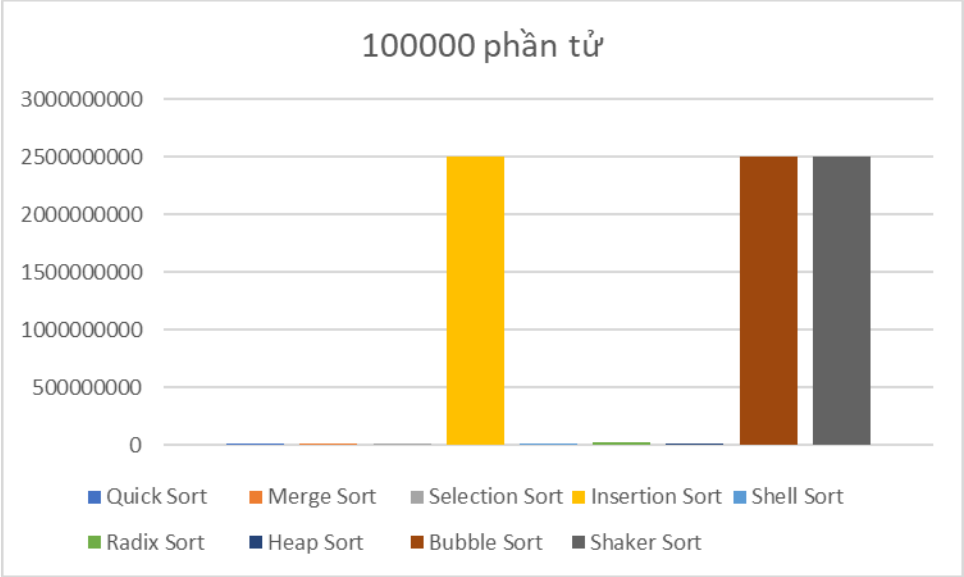


- Các biểu đồ với từng số phần tử:







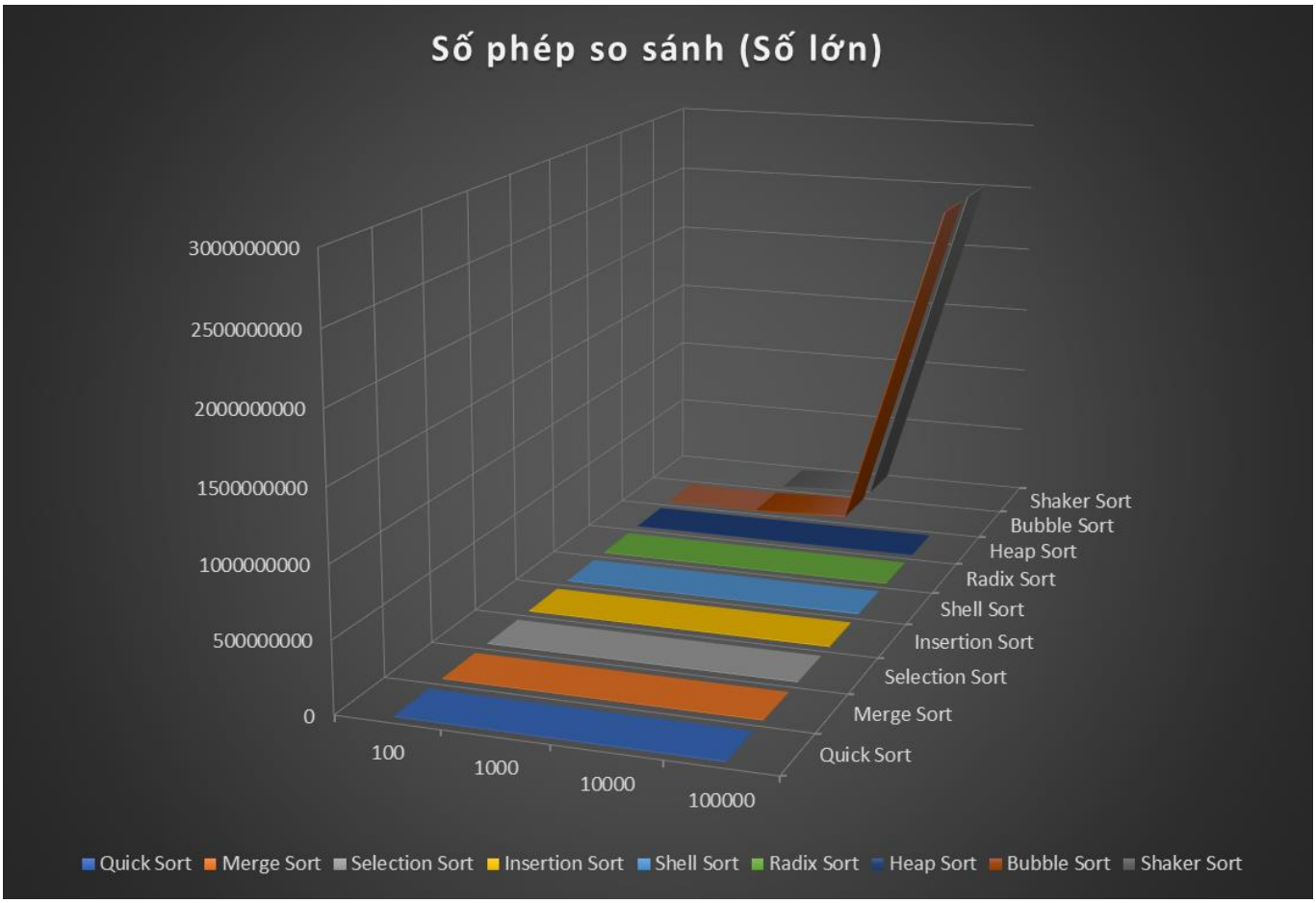


2. So sánh số lượng phép so sánh

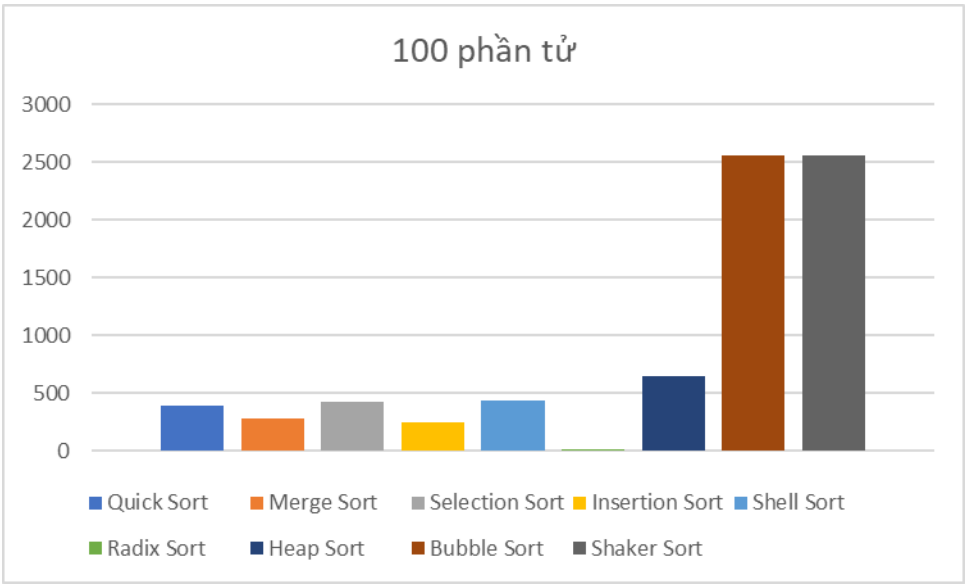
Kết quả số lượng phép gán của các thuật toán với số phần tử theo yêu cầu:

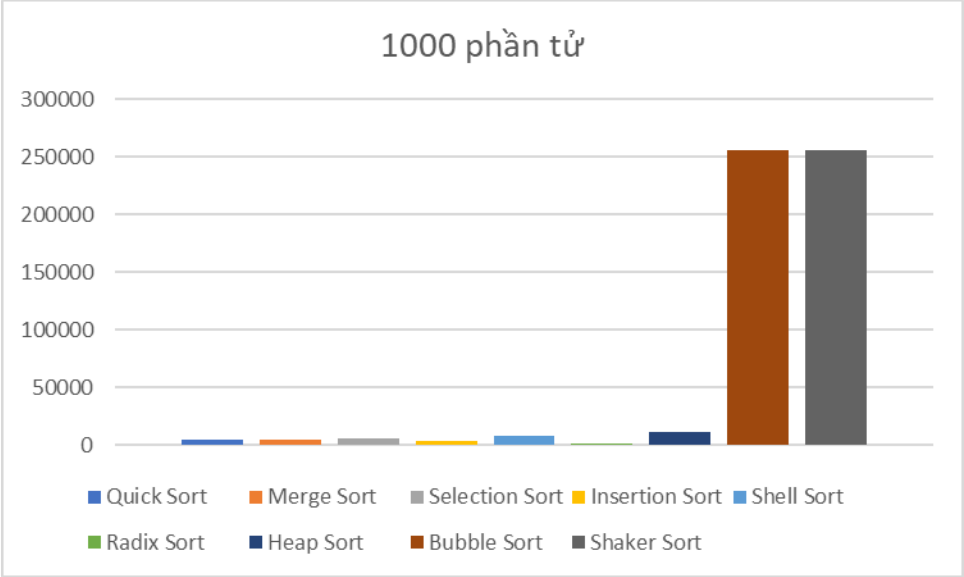
Tên thuật	***Số phần tử***				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	395	4586	79440	955531	13272552
Merge Sort	280	4394	61310	776605	9394820
Selection Sort	423	6351	87161	1108539	X
Insertion Sort	247	4134	57699	744347	X
Shell Sort	433	7713	156115	2837311	51504546
Radix Sort	4	6	7	13	9
Heap Sort	647	11682	166377	2163190	26623028
Bubble Sort	2557	256223	25223227	2502802261	X
Shaker Sort	2557	256223	25223227	2502802261	X

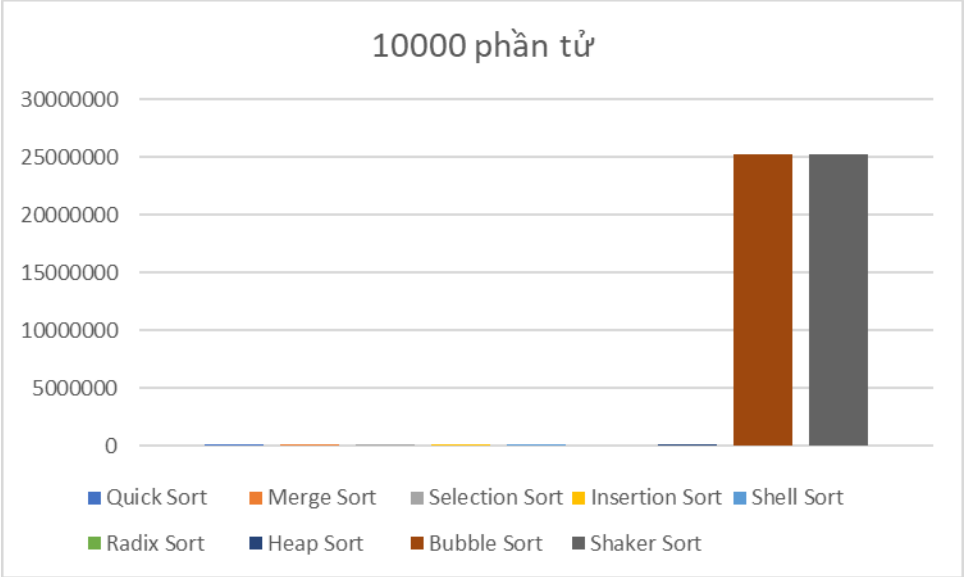
- Biểu đồ dựa vào bảng trên (trừ cột cuối)

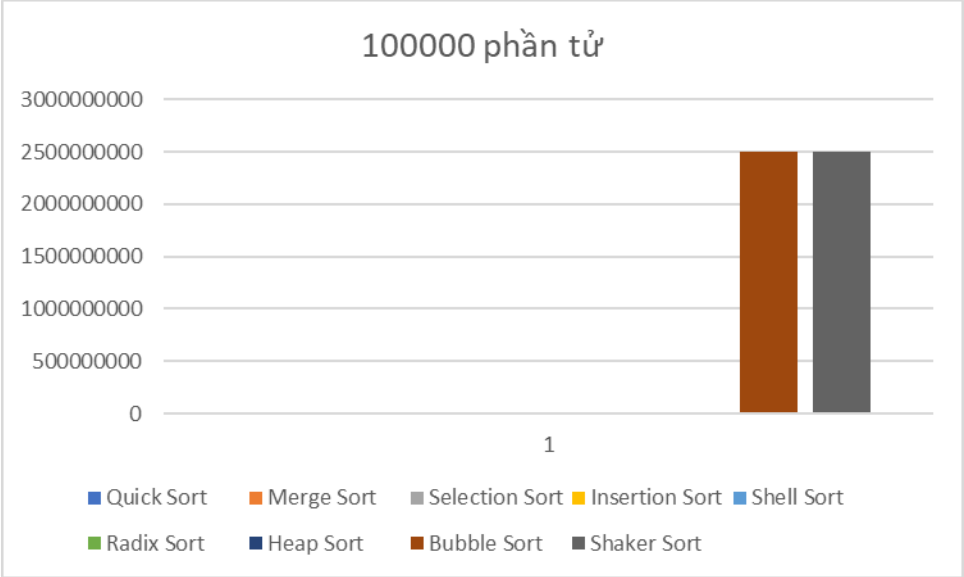


- Các biểu đồ với từng số phần tử:




1000 phần tử
300000
250000
200000
150000
100000
50000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


10000 phần tử
30000000
25000000
20000000
15000000
10000000
5000000
0
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort

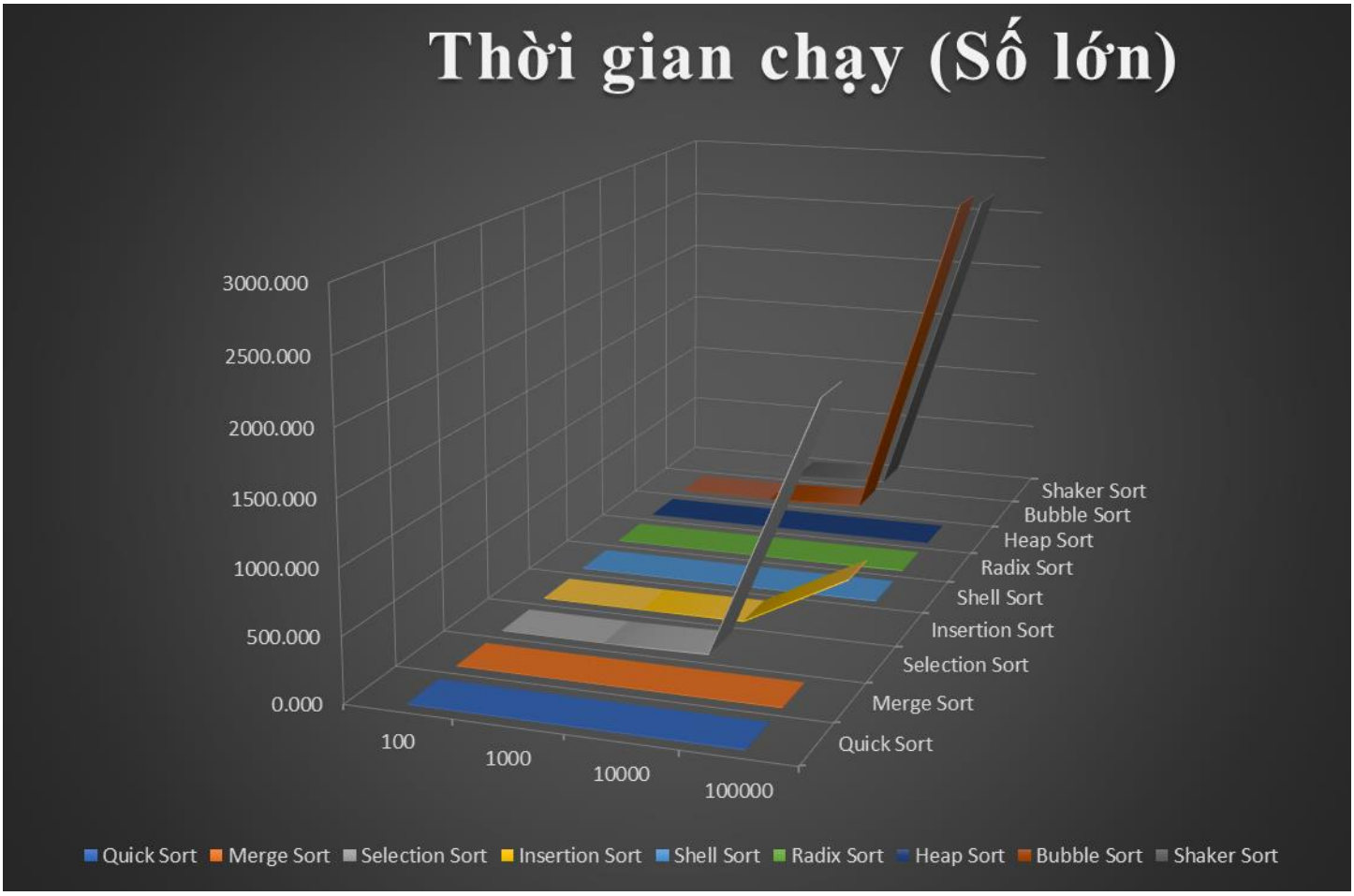

100000 phần tử
3000000000
2500000000
2000000000
1500000000
1000000000
500000000
0
1
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort

3. So sánh thời gian chạy

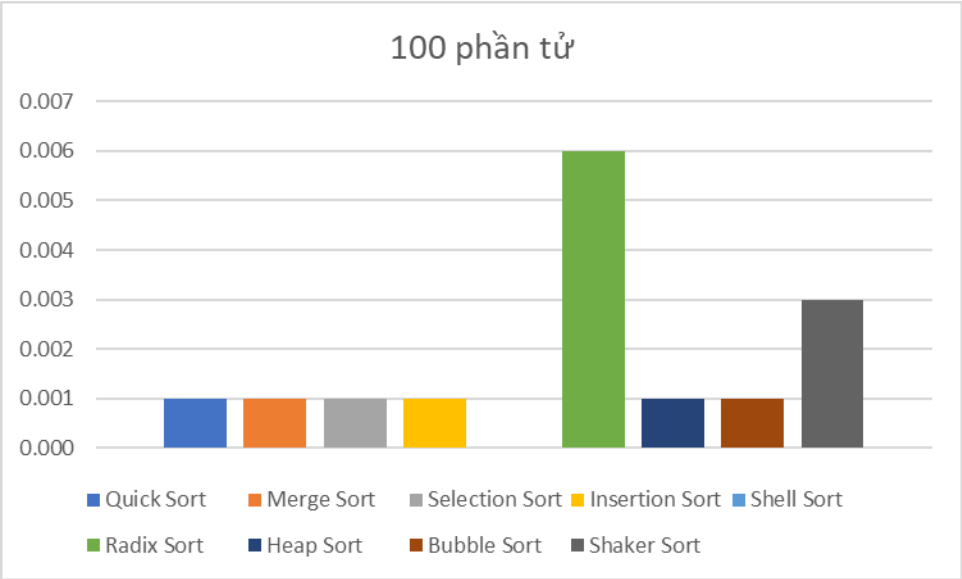
Kết quả thời gian chạy của các thuật toán với số phần tử theo yêu cầu:

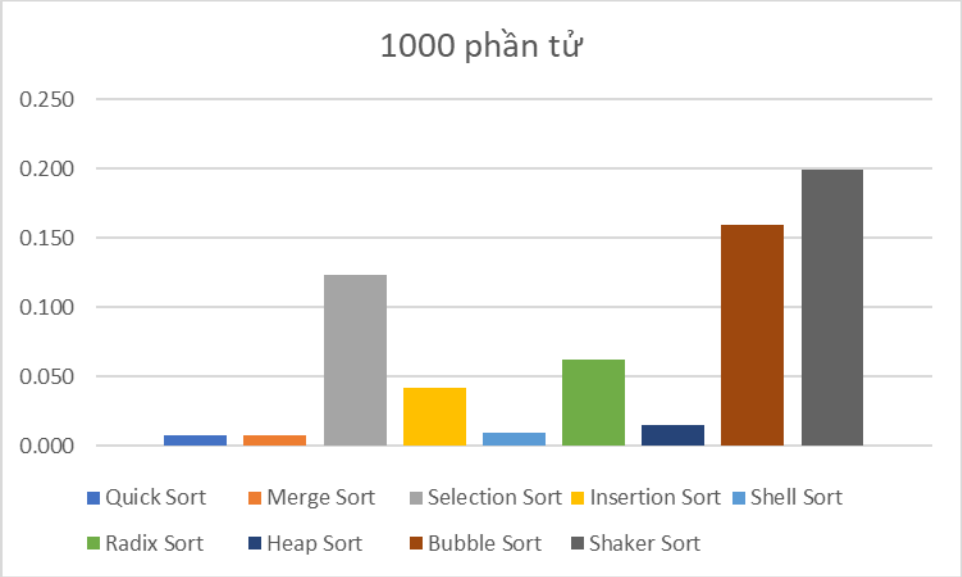
Tên thuật	*Số phần tử*				
	100	**1000**	**10000**	**100000**	**1000000**
Quick Sort	0.001	0.008	0.080	1.214	13.955
Merge Sort	0.001	0.008	0.102	1.418	14.633
Selection Sort	0.001	0.123	13.575	2037.998	X
Insertion Sort	0.001	0.042	3.717	431.503	X
Shell Sort	0.000	0.010	0.226	3.004	64.495
Radix Sort	0.006	0.062	0.733	6.075	73.059
Heap Sort	0.001	0.015	0.276	2.870	42.997
Bubble Sort	0.001	0.160	24.332	2776.659	X
Shaker Sort	0.003	0.199	25.649	2666.804	X

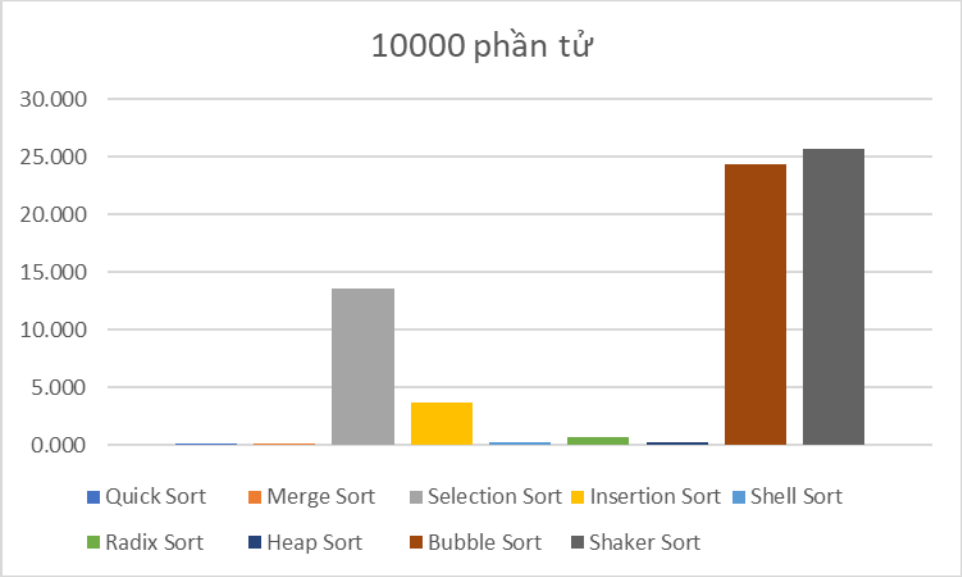
- Biểu đồ dựa vào bảng trên (trừ cột cuối):

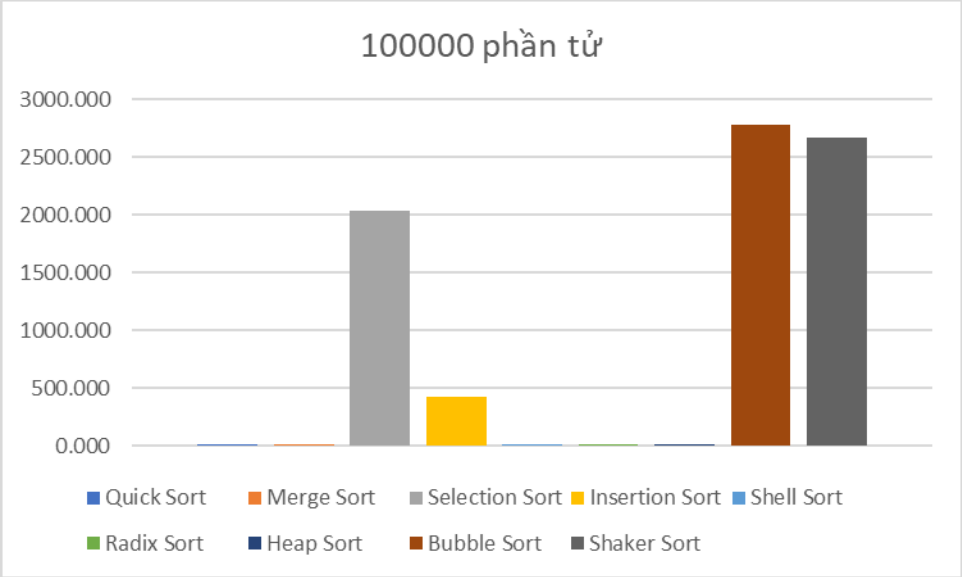


- Các biểu đồ với từng số phần tử:






10000 phần tử
30.000
25.000
20.000
15.000
10.000
5.000
0.000
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort


100000 phần tử
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
Quick Sort
Merge Sort
Selection Sort
Insertion Sort
Shell Sort
Radix Sort
Heap Sort
Bubble Sort
Shaker Sort

IV. Đánh giá, nhận xét:

Về số phép gán và so sánh:

- Các phép gán và so sánh có thể ảnh hưởng đến hiệu suất thuật toán khi đối tượng được sắp xếp là loại dữ liệu tổ hợp (struct,…) như kiểu dữ liệu số rất lớn trên (sử dụng danh sách liên kết đôi với nhiều node bên trong) làm chậm các thuật toán.
- Xét về mặt này dựa vào kết quả chạy thử trên ta đưa ra được kết luận:

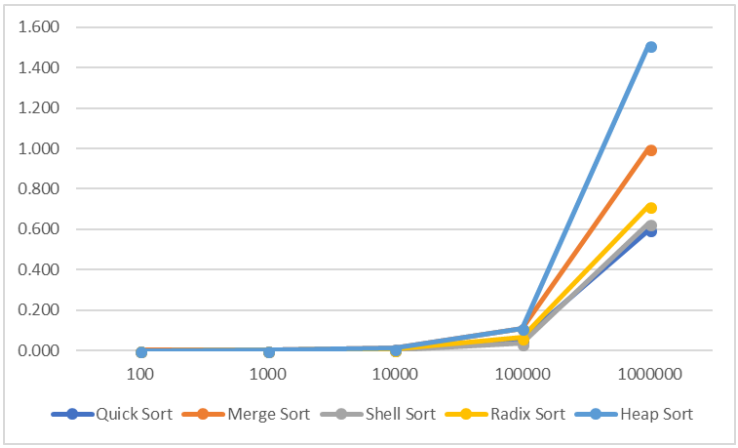
  Các thuật toán Bubble và Shaker sort có số lượng phép gán, so sánh và Insertion Sort có số lượng phép gán lớn so với các thuật toán còn lại.
- Vì sự ảnh hưởng của phép gán và so sánh làm chậm các thuật toán

  ➢ Để so sánh hiệu suất của các thuật toán trên cùng một kiểu dữ liệu ta chỉ cần xét hiệu quả của thuật toán khi kích thước đầu vào tăng dần, hay thường là dựa trên độ lớn các phần tử cần sắp xếp.

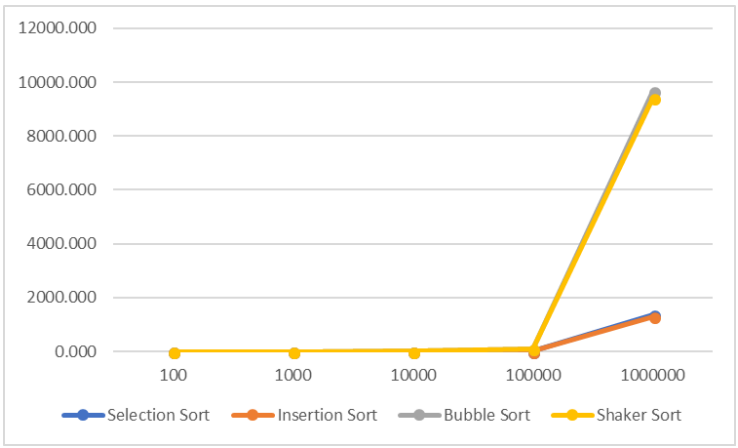
a) Kiểu dữ liệu int

- Dựa vào thời gian chạy, ta có thể sắp xếp hiệu suất các thuật toán như sau:

1. ***Quick Sort***
2. ***Shell Sort***
3. ***Radix Sort***
4. ***Merge Sort***
5. ***Heap Sort***



6. ***Insertion Sort***
7. ***Selection Sort***
8. ***Shaker Sort***
9. ***Bubble Sort***

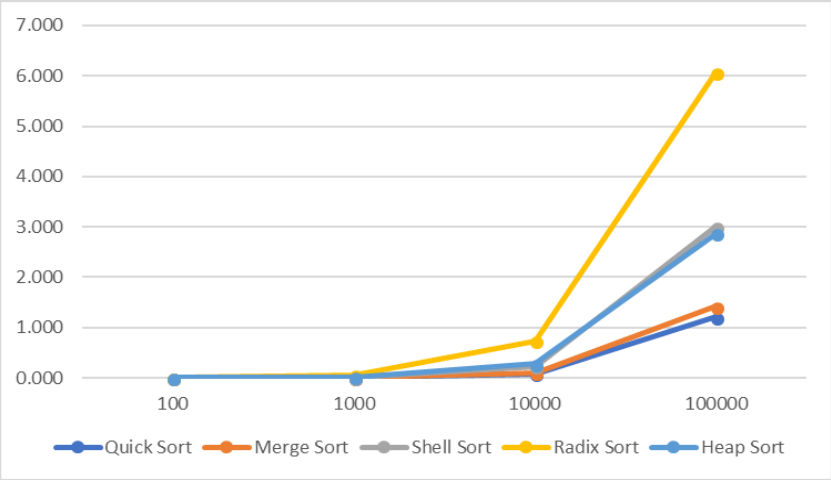


- Kết quả không ngạc nhiên vì:
  - Các thuật toán ***Quick, Radix, Merge, Shell, Heap sort*** đều được cho là các thuật toán có hiệu suất cao.
  - ***Insertion và Selection sort*** là hai thuật toán đơn giản phù hợp với dữ liệu kích thước nhỏ nhưng kém hiệu quả với dữ liệu kích thước lớn.
  - ***Bubble và Shaker sort*** là các thuật đơn giản nhưng hiệu quả kém, thường gặp trong các văn bản hướng dẫn nhưng ít được áp dụng thực tế.
- Tuy nhiên việc sắp xếp này hạn chế với việc ta chỉ chạy thử với các số được phát sinh ngẫu nhiên.
- Trong thực tế, ta có thể gặp các trường hợp mảng đã sắp xếp, mảng đã sắp xếp nhưng ngược, mảng gần như đã sắp xếp, mảng có ít phần tử riêng biệt (các phần tử lặp nhiều lần)… Khi xét hết các trường hợp có thể đưa ra kết quả khác so với hiện tại.

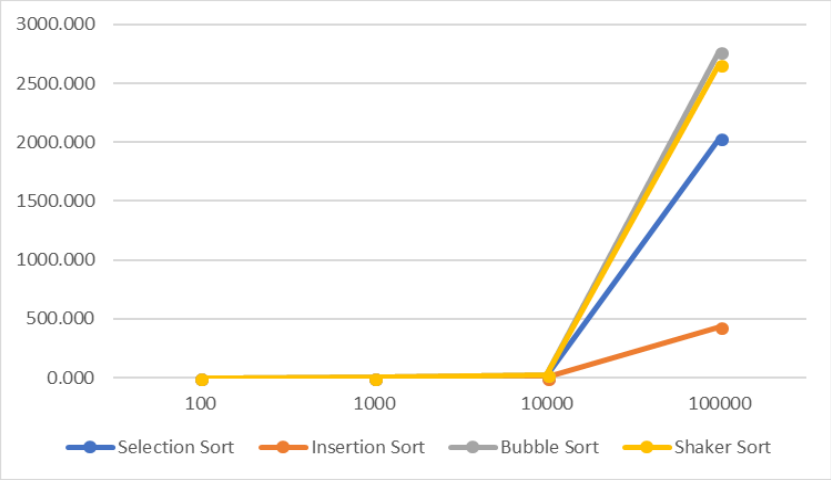
b) Kiểu dữ liệu số lớn

- Tương tự ta có thể sắp xếp hiệu suất các thuật toán như sau:

1. ***Quick sort***
2. ***Merge sort***
3. ***Heap sort***
4. ***Shell sort***
5. ***Radix sort***



6. ***Insertion sort***
7. ***Selection sort***
8. ***Shaker sort***
9. ***Bubble sort***



- Ở đây ta có 1 số khác biệt với kết quả ở kiểu dữ liệu int thường đa phần là do cách định nghĩa của kiểu số lớn, cách điều chỉnh để thuật toán chạy được với kiểu dữ liệu này và một số nguyên nhân như:
    - ***Radix sort*** nằm cuối nhóm các thuật chạy nhanh có thể vì mỗi phần tử có giá trị lớn (khoảng 35 chữ số).
    - ***Shell sort*** bị ảnh hưởng bởi số lượng phép gán và so sánh nhiều đáng kể so với các thuật còn lại trong nhóm.
    - ***Insertion sort*** nhanh hơn nhiều so với ***Selection, Shaker, Bubble sort*** vì 3 thuật đó có số phép gán, so sánh nhiều hơn so với ***Insertion sort***.

V. Các nguồn tham khảo

- https://en.wikipedia.org/wiki/Sorting_algorithm
- https://www.cprogramming.com/tutorial/computersciencetheory/sortcomp.html
- https://cs.stackexchange.com/questions/3/why-is-quicksort-better-than-other-sorting-algorithms-in-practice
- https://www.youtube.com/watch?v=ZZuD6iUe3Pc
- https://www.toptal.com/developers/sorting-algorithms