10/24/2019

BÁO CÁO CTDLGT

Tìm hiểu 12 thuật toán sắp xếp

PHAN LONG HIẾU
18120377

SELECTION SORT

Ý tưởng :

Được xây dựng dựa trên việc sắp xếp tự nhiên của con người.

Thuật toán selection sort sắp xếp một mảng bằng cách sử dụng một màn ngăn vô hình ngăn cách mảng thành 2 phần, 1 phần đã được sắp xếp và một phần chưa được sắp xếp. Thuật toán sẽ tìm phần tử có giá trị nhỏ nhất (giả sử với sắp xếp mảng tăng dần) trong đoạn chưa được sắp xếp và đổi cho phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp (không phải đầu mảng).

Đánh giá thuật toán :

Độ phức tạp thời gian là o(n^2) trong bất cứ trường hợp nào

Không gian phụ trợ: O(1) – Đây là thuật toán in-place

Selection Sort hoạt động tốt nếu như dữ liệu có kích thước nhỏ, nếu dữ liệu có kích thước thì sẽ tốn nhiều thời gian để sắp xếp. Ngoài ra hiệu suất thực thi cũng không phụ thuộc vào việc dữ liệu random, sorted hay là nearly sorted như thế nào.

INSERTION SORT

Ý tưởng :

Cũng sử dụng một màn ngăn vô hình ngăn cách mảng thành 2 phần, 1 phần đã được sắp xếp và 1 phần chưa được sắp xếp giống như Selection Sort . Tuy nhiên Insertion Sort lại không tìm phần tử min trong mảng chưa được sắp xếp như Selection Sort. Thay vào đó Insertion Sort sẽ tìm cách chèn lần lượt các phần tử ở mảng chưa được sắp xếp vào đúng vị trí ở mảng đã được sắp xếp.

Đánh giá thuật toán :

Độ phức tạp thời gian :

- Trường hợp tốt nhất: O(n), tức là trường hợp mảng đã được sắp xếp tăng dần (nêu như ta muốn sắp xếp tăng dần).
- Trường hợp xấu nhất: O(n^2), trong trường hợp dữ liệu đã được sắp xếp nhưng ngược lại với chiều mà chúng ta muốn sắp xếp (kiểu dữ liệu Reversed)

Không gian phụ trợ: O(1) – Đây là thuật toán in-place

Insertion sort hoạt động tốt nếu như dữ liệu có kích thước nhỏ, nếu dữ liệu có kích thước thì sẽ tốn nhiều thời gian để sắp xếp. Ngoài ra thời gian thực thi cũng sẽ rất tốt nếu như dữ liệu gần như đã được sắp xếp (Nearly Sorted)

BINARY INSERTION SORT

Ý Tưởng :

Cũng giống như Insertion Sort nhưng mà có cải tiến thêm đó là Binary Insertion Sort sử dụng thuật toán tìm kiến nhị phân để tìm ra vị trí cần chèn phần tử. Điều này giúp thời gian tìm kiếm vị trí chèn giảm đi kha khá nhiều.

Đánh giá thuật toán :

Độ phức tạp thời gian trong trường hợp xấu nhất vẫn là O(n^2), tức là Binary Insertion Sort chỉ nhanh hơn trong việc tìm kiếm vị trí cần chèn thôi, nhưng sau khi tìm kiếm thì Binary Insertion Sort vẫn phải swap số lượng phần tử như là Insertion Sort nên độ phức tạp trong trường hợp này vẫn không có gì thay đổi.

Binary Insertion Sort hoạt động tốt nếu như dữ liệu có kích thước nhỏ, nếu dữ liệu có kích thước thì sẽ tốn nhiều thời gian để sắp xếp. Ngoài ra thời gian thực thi cũng sẽ rất tốt nếu như dữ liệu gần như đã được sắp xếp (Nearly Sorted)

SHELL SORT

Ý tưởng

Shell Sort hoạt động dựa trên Insertion Sort, đối với Insertion Sort nếu như vị trí cần chèn ở quá xa so với phần tử đang xét thì chúng ta sẽ tốn một khoảng chi phí khá là lớn, số lần di chuyển phần tử nhiều.

Phân hoạch dãy thành các dãy con sau đó sắp xếp các dãy con theo phương pháp chèn trực tiếp dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy

Đánh giá thuật toán:

Độ phức tạp về mặt thời gian:

Trong trường hợp tốt nhất: nlogn

Trường hợp trung bình và trường hợp xấu: Phụ thuộc vào khoảng cách chia ở mỗi lần, độ phức tạp ở trường hợp này rất tinh vi và vẫn còn nhiều tranh cãi, dự đoán rơi vào khoảng giữa O(n) và O(n^2).Theo Poonen Theorem, trường hợp xấu nhất là (NlogN)2/(log logN)2) hoặc (NlogN)2/log logN) hoặc (N(logN)2) .

Không gian phụ trợ: O(1) – Đây là thuật toán in-place

BUBBLE SORT

Ý tưởng:

Lấy ý tưởng giống như bong bóng nổi từ đáy hồ lên mặt nước, ta xét lần lượt các cặp phần tử, nếu cặp phần tử nào mà là giảm dần (trường hợp sắp xếp tăng dần) thì ta tiến hành hoán vị đổi chỗ hai phần tử này. Cứ như thế, ta sẽ đem được phần tử lớn nhất lên đầu dãy. Chúng ta sẽ thực hiện lại các thao tác trên cho đến khi dãy hiện hành chỉ còn 1 phần tử duy nhất.

Đánh giá thuật toán :

Độ phức tạp thời gian

- Trong trường hợp tốt nhất đọ phức tạp là O(n), tức là mảng đã được sắp xếp
- Độ phức tạp trong trường hợp trung bình là $O(n^2)$.
- Trong trường hợp xấu nhất đọ phức tạp là $O(n^2)$, khi mà mảng được sắp xếp reversed

Không gian phụ trợ: O(1) – Đây là thuật toán in-place

Bubble Sort mặc dù có độ phức tạp về mặt thời gian tương đương với Insertion Sort nhưng nó lại có sự khác biệt về số lần swap so với Insertion Sort rất nhiều (cụ thể là nhiều hơn). Bubble Sort kém hiệu quả hơn về mặt thời gian hơn so với Insertion Sort trong các trường hợp sắp xếp dữ liệu thuộc kiểu random

SHAKER SORT

Ý tưởng:

Là một cải tiến của Bubble Sort, không những dồn những phần tử lớn về phía bên phải (đối với sắp xếp tăng dần), nó còn dồn những phần tử nhỏ về phía bên trái nữa.

Đánh giá thuật toán:

Độ phức tạp thời gian

- Độ phức tạp cho trường hợp tốt nhất là O(n), tức là mảng đã được sắp xếp
- Độ phức tạp cho trường hợp xấu nhất $O(n^2)$.
- Độ phức tạp trong trường hợp trung bình là $O(n^2)$.

Không gian phụ trợ: O(1) – Đây là thuật toán in-place

Shaker Sort ưu thế hơn Bubble Sort trong trường hợp dữ liệu gần như được sắp xếp (Nearly Sorted), còn đối với các trường hợp như dữ liệu Random, Reversed thì hiệu suất gần như là tương đương với Bubble Sort.

HEAP SORT:

Ý tưởng:

Heap Sort sẽ sử dụng một trong những tính chất của cấu trúc Heap đó là phần tử lớn nhất luôn đứng ở vị trí đầu tiên, sau đó Heap Sort sẽ đẩy phần tử lớn nhất đó về cuối dãy và xây dựng lại dãy với cấu trúc Heap để tìm phần tử lớn nhất tiếp theo. Chính vì sử dụng cấu trúc Heap để tìm tiếm phần tử lớn nhất chứ không phải tìm kiếm tuyến tính như Selection Sort nên Heap Sort cải tiến hơn so với Selection Sort

Đánh giá thuật toán:

Độ phức tạp thời gian

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

Không gian phụ trợ: O(n)

Thuật toán sắp xếp Heap có đôi chút chậm hơn Quicksort .Tuy nhiên, cấu trúc dữ liệu Heap được sử dụng rất nhiều trong lập trình.

MERGE SORT

Ý tưởng:

Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp.

Đánh giá thuật toán sắp xếp Merge Sort

Độ phức tạp về mặt thời gian: O(nlog(n)) cho các trường hợp tốt, xấu, trung bình

Ưu điểm: Sắp sếp nhanh hơn so với các thuật toán cơ bản (Insertion Sort, Selection Sort, Interchage Sort), nhanh hơn Quick Sort trong một số trường hợp. Khi áp dụng Merge sort cho danh sách liên kết thì chúng ta không cần dùng thêm danh sách phụ trợ, chúng ta nối trực tiếp các node lại với nhau luôn. Đó là lí do Merge Sort thích hợp với sắp xếp danh sách liên kết.

Nhược điểm: thuật toán khó cài đặt, không nhận dạng được mảng đã được sắp, Merge sort sử dụng không gian phụ trợ O(n), nếu như dữ liệu lớn thì sẽ gây tốn kém bộ nhớ nhiều .

QUICK SORT

Ý tưởng:

Giống như Merge sort, thuật toán sắp xếp quick sort là một thuật toán chia để trị(Divide and Conquer algorithm). Nó chọn một phần tử trong mảng làm điểm đánh dấu(pivot). Thuật toán sẽ thực hiện chia mảng thành các mảng con dựa vào pivot đã chọn. Có nhiều cách chọn pivot, sau đây là một vài cách

- Luôn chọn phần tử đầu tiên của mảng.
- Luôn chọn phần tử cuối cùng của mảng
- Chọn một phần tử random.
- Chọn một phần tử có giá trị nằm giữa mảng(median element).

Đánh giá thuật toán sắp xếp Quick Sort

Độ phức tạp về mặt thời gian của Quick Sort

- Trường hợp tốt và trung bình: O(nlog(n))
- Trường hợp xấu: O(n^2), nếu như Quick Sort lấy trúng phần tử nhỏ nhất hoặc là phần tử lớn nhất làm phần tử pivot . Tuy nhiên chúng ta có thể tối ưu quick sort trong trường hợp này thành nlogn nếu như dùng giải thuật đệ qui đuôi.

Nhược điểm:

Độ phức tạp phụ thuộc vào việc chọn pivot. Nếu như pivot chọn được rơi vào

Quick Sort thích hợp với việc sắp xếp array hơn là Merge Sort, mặc dù cả 2 đều có cùng một độ phức tạp là nlogn. Lí do là vì Quick Sort nhìn chung là một dạng in-place sort, không đòi hỏi thêm vùng nhớ, trong khi đó Merge Sort O(n) thêm vùng nhớ.

Tuy nhiên về nhược điểm là Quick Sort lại không thích hợp với kiểu dữ liệu danh sách liên kết, lí do là vì danh sách liên kết muốn truy cập phần tử thứ n thì phải truy cập tuần tự qua n-1 phần tử trước, do đó chi phí sẽ tăng nhanh nếu như dùng Quick Sort cho danh sách liên kết, ví dụ như hàm swap.

COUNTING SORT

Ý tưởng

Counting sort hoạt động bằng cách duyệt qua mảng đầu vào, sau đó đếm số lượng xuất hiện của các phần tử trong đó và sử dụng số đếm đó để suy ra vị trí của các phần tử trong mảng được sắp xếp cuối cùng mà không cần dựa vào việc so sánh các phần tử

Đánh giá thuật toán:

Độ phức tạp về mặt thời gian: O(n) cho các trường hợp tốt, xấu lẫn trung bình

Không gian cần sử dụng:O(n)

Ưu điểm: Counting sort bởi vì có độ phức tạp tuyến tính nên nó nhanh hơn nhiều so với các thuật toán như Quick Sort hay là Merge Sort. Counting thích hợp với việc sắp xếp các mảng số nguyên không âm sẽ rất hiệu qua trong phần lớn các trường hợp.

Nhược điểm: Counting Sort chỉ hoạt động tốt nếu như dữ liệu đầu vào có phạm vị lớn hơn không quá nhiều so với số lượng các phần tử khác nhau trong mảng. Ngoài ra nếu như dữ liệu có kích thước lớn thì cũng sẽ gây tốn bộ nhớ bởi vì không gian bộ nhớ cần dùng của Counting Sort có độ phức tạp là O(n).

RADIX SORT

Ý tưởng

Radix Sort là một thuật toán tiếp cận theo một hướng hoàn toàn khác. Nếu như trong các thuật toán khác, cơ sở để sắp xếp luôn là việc so sánh giá trị của 2 phần tử thì Radix Sort lại dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó Radix Sort còn có tên là Postman’s sort.
Radix Sort không hề quan tâm đến việc so sánh giá trị của phần tử mà bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử

Đánh giá thuật toán:

Độ phức tạp về mặt thời gian: O(n) cho các trường hợp tốt, xấu lẫn trung bình

Thuật toán không có trường hợp xấu nhất và tốt nhất . Thuật toán có độ phức tạp tuyến tính nên hiệu quả khi sắp dãy có rất nhiều phần tử, nhất là khi khóa sắp xếp không quá dài so với số lượng phần tử

Vì thuật toán so sánh dựa trên các digit nên mình không thể áp dụng Radix Sort trên các kiểu dữ liệu phức tạp như là struct hay là so sánh các Object với nhau.

Các nguồn tham khảo:

https://en.wikipedia.org/wiki/Sorting_algorithm

https://www.geeksforgeeks.org/

https://www.stdio.vn/

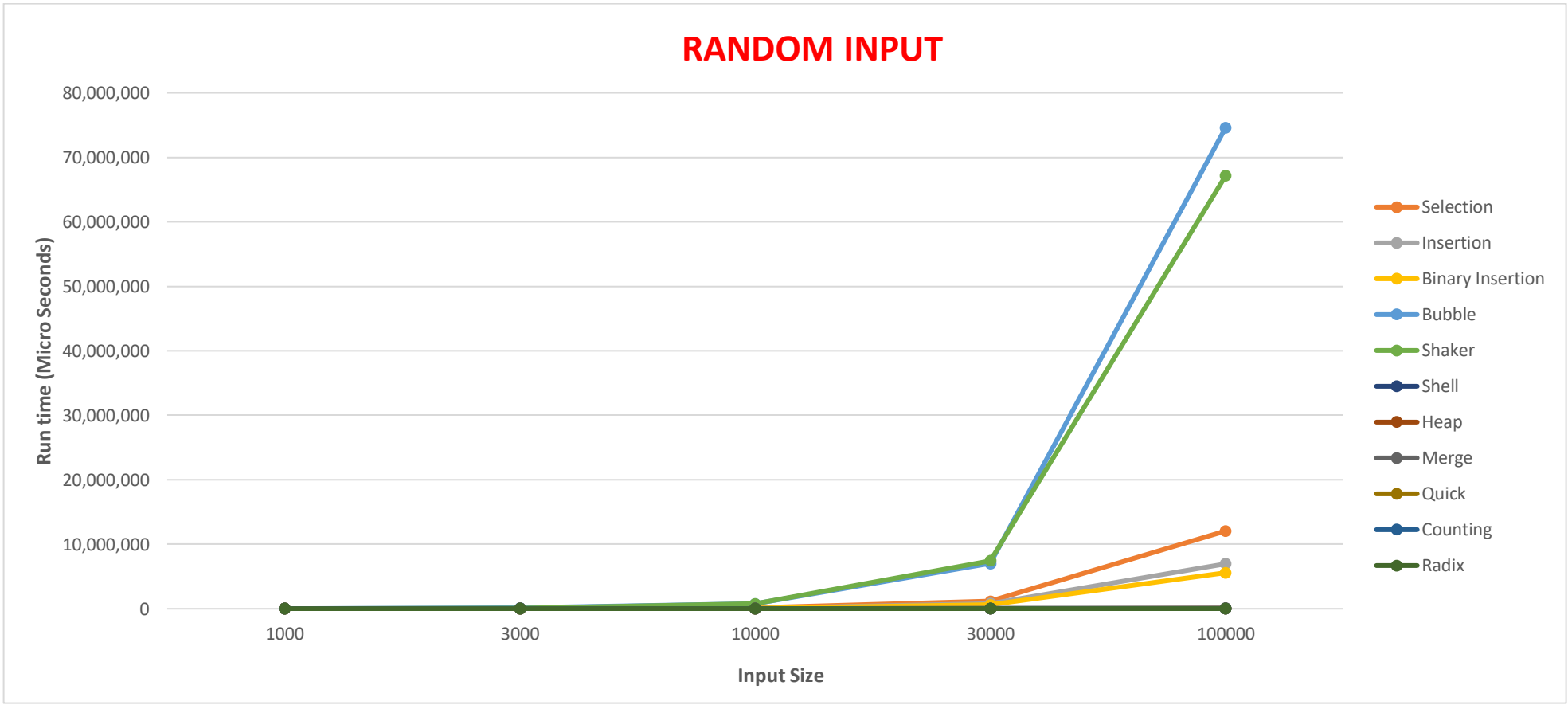
https://nguyenvanhieu.vn

https://www.hackerearth.com

https://stackoverflow.com

Input State	Input Size	Selection	Insertion	Binary Insertion	Bubble	Shaker	Shell	Heap	Merge	Quick	Counting	Radix	Flash
Random	1000	1,434	781	838	6,974	7,661	128	342	1,037	165	25	177	
	3000	10,581	5,999	6,760	130,585	62,738	552	1,097	2,744	598	98	352	
	10000	186,538	67,471	65,028	771,159	754,670	2,129	4,921	9,510	3,954	246	1,152	
	30000	1,173,245	797,162	629,427	6,995,869	7,431,318	8,224	14,668	63,055	7,004	833	4,189	
	100000	12,061,882	6,965,094	5,584,326	74,612,481	67,139,358	31,285	63,366	98,177	25,131	1,870	12,432	
Sorted	1000	1,410	10	221	8	15	35	313	798	68	25	84	
	3000	10,532	18	776	13	14	125	1,148	2,256	233	72	336	
	10000	174,844	43	3,679	28	29	702	4,117	7,626	775	227	990	
	30000	1,146,670	71	10,980	70	74	1,671	40,515	22,470	2,415	590	5,743	
	100000	12,326,262	368	41,354	227	239	16,581	63,869	75,029	10,906	2,249	13,718	
NearlySorted	1000	1,167	22	243	1,107	201	72	318	1,040	70	31	337	
	3000	10,744	65	1,152	9,800	617	199	1,208	2,210	228	68	350	
	10000	113,770	290	3,154	138,441	2,403	654	4,026	7,571	814	338	1,158	
	30000	1,214,897	719	18,632	1,071,156	5,069	2,072	13,735	23,493	2,493	602	4,093	
	100000	12,203,191	939	43,084	6,306,511	8,574	6,871	49,105	73,106	8,968	2,025	14,085	
Reversed	1000	1,491	2,435	3,062	10,763	11,273	53	296	869	74	29	83	
	3000	9,987	12,078	10,655	101,251	148,521	180	994	2,828	267	75	365	
	10000	157,031	273,178	116,458	1,130,302	1,297,272	688	4,059	7,541	1,100	251	1,059	
	30000	1,252,495	1,198,074	1,087,926	11,113,494	10,982,321	2,559	13,198	22,177	3,212	624	4,482	
	100000	12,207,592	13,837,586	11,319,462	108,017,592	107,870,186	10,662	60,149	74,876	11,430	2,224	12,624	

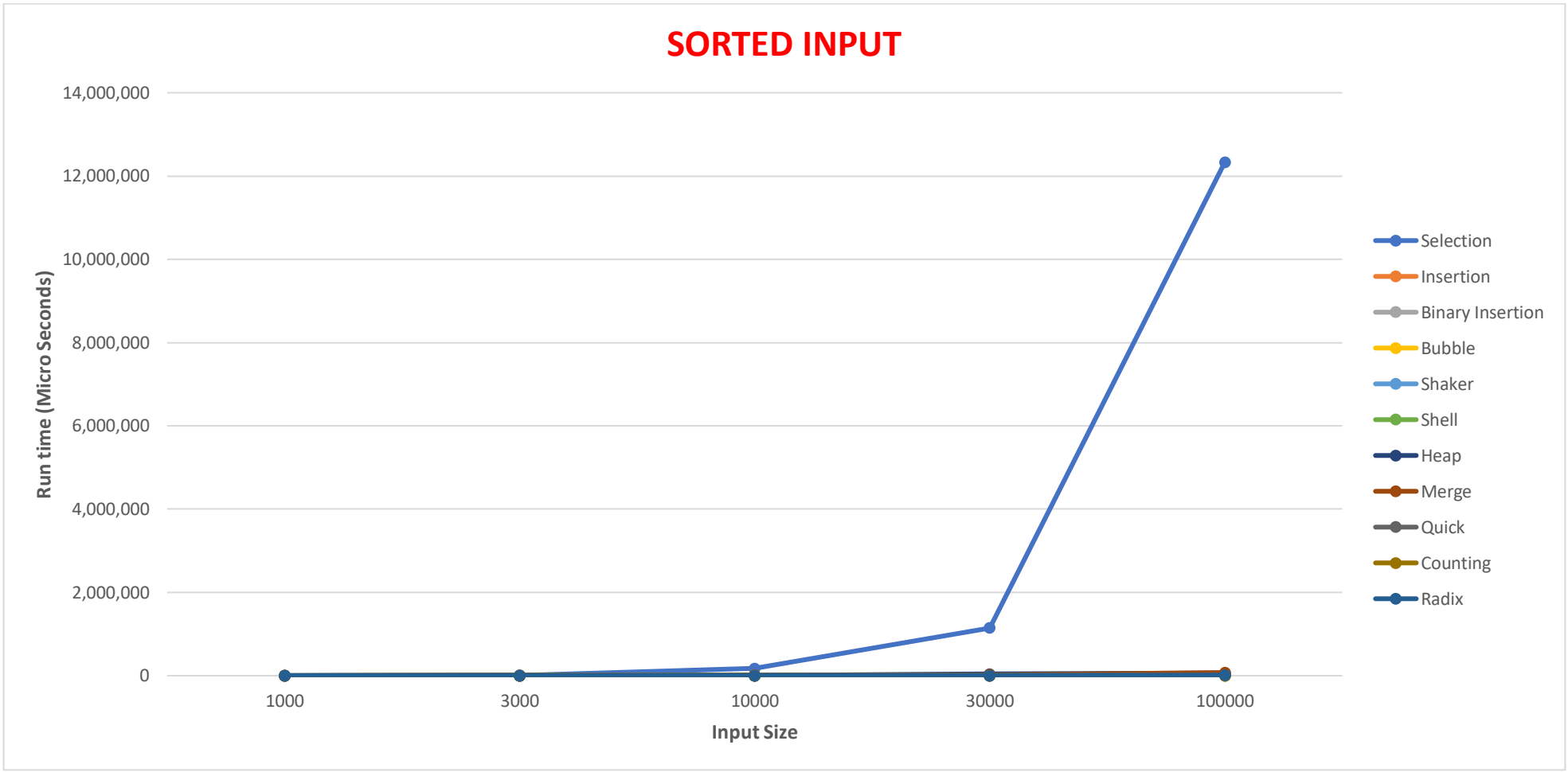
*** Đơn vị đo thời gian là Micro Second, dấu phẩy ở đây chỉ là ngăn cách phần nghìn, phần trăm,.. chứ không phải là dấu phẩy thập phân



Đối với kiểu dữ liệu random thì Bubble và Shaker cho thời gian thực thi lâu nhất mặc dù có cùng độ phức tạp với Selection Sort nhưng mà do phải swap rất nhiều lần nên thời gian lớn hơn nhiều

Theo sau thì đó chính là Insertion sort, Binary Insertion Sort

Và nhanh nhất thì chính là Counting, Merge, Quick, Radix, Heap Sort



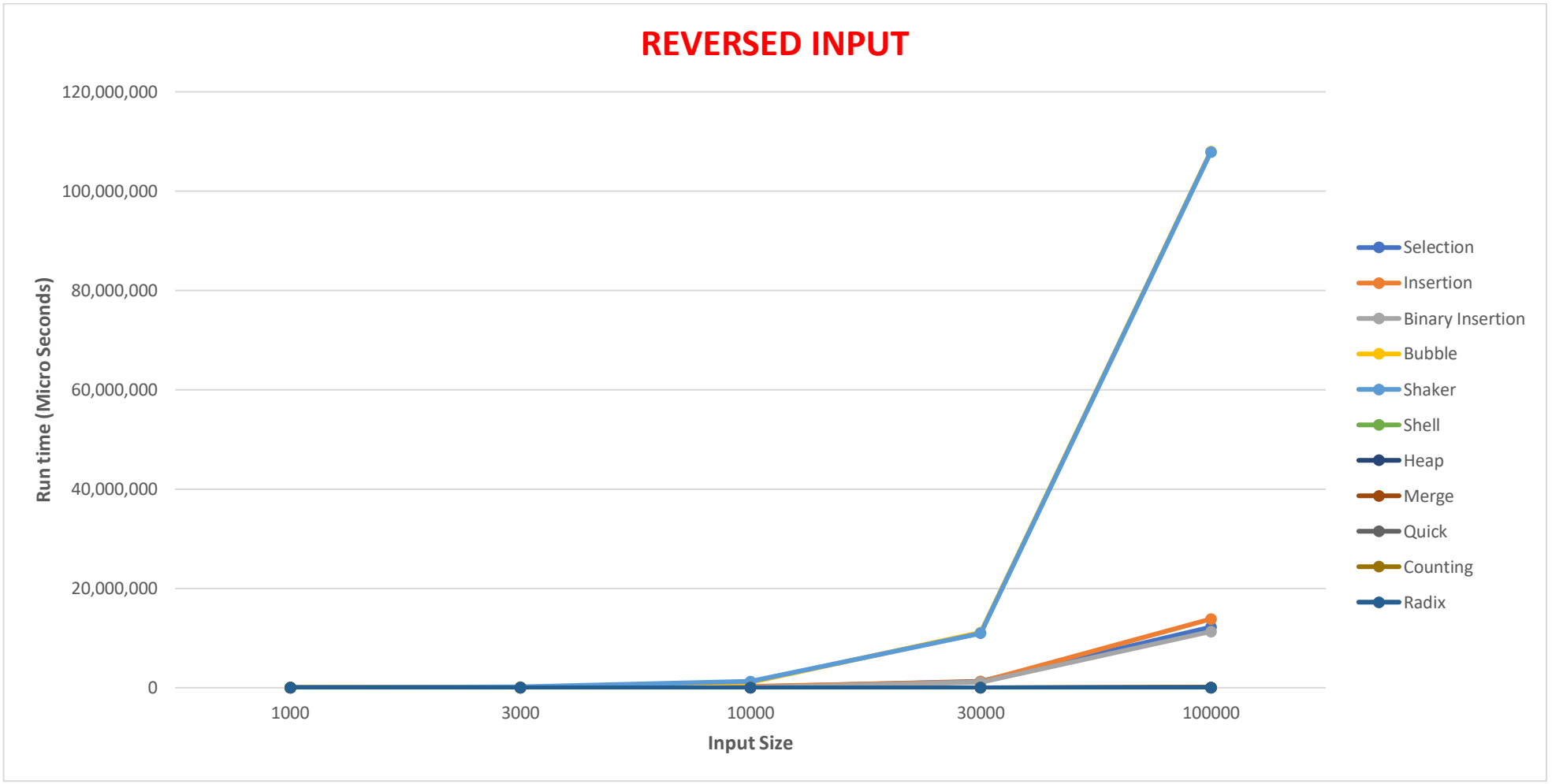
Đối với kiểu dữ liệu Sorted thì Slection Sort cho thời gian thực thi lâu nhất vì với Selection Sort nó không quan tâm dữ liệu như thế nào, độ phức tạp của nó luôn là O(n^2)

Theo sau thì đó chính là Insertion sort, Binary Insertion Sort

Và nhanh nhất thì chính là Bubble , Shaker, Insertion và Counting

Đối với kiểu dữ liệu Nearly Sorted thì tiếp tục Selection Sort cho thời gian thực thi lâu nhất vì với Selection Sort nó không quan tâm dữ liệu như thế nào, độ phức tạp của nó luôn là O(n^2)

Theo sau thì đó chính là Bubble Sort. Ở kích cỡ dữ liệu nhỏ thì Bubble không chênh lệch nhiều so với Merger Sort nhưng sau khi kích cỡ dữ liệu tăng dần thì Bubble sort ngốn rất nhiều thời gian để chạy.



Đối với kiểu dữ liệu Reversed thì Bubble , Shaker Sort cho thời gian thực thi lâu nhất vì đây chính là trường hợp xấu nhất của 2 thuật toán này

Theo sau thì đó chính là Insertion sort, Binary Insertion Sort và Selection Sort

Và nhanh nhất thì chính là Counting, Shell, Quick, Radix, Heap và Merge Sort