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

Tìm hiểu về các giải thuật sắp xếp

Thực hành cấu trúc dữ liệu và giải thuật

Sinh viên thực hiện

Bùi Quốc Trung
MSSV: 20120023
Lớp: 20CTT1TN

Giảng viên hướng dẫn

LT: Nguyễn Thanh Phương
TH: Bùi Huy Thông

Tháng 11 năm 2021

Mục Lục

1. Mở đầu

Sắp xếp là quá trình xử lí một danh sách các phần tử để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên dữ liệu mà các phần tử này lưu trữ. Vậy tại sao cần phải sắp xếp các dữ liệu này ?

Sắp xếp giúp chúng ta lưu trữ dữ liệu một cách có hệ thống, dễ dàng cho việc tìm kiếm, truy xuất, sử dụng sau này. Một số ví dụ như:

- Sắp xếp điểm thi các học sinh trong lớp.
- Sắp xếp giá các linh kiện điện thoại
- Danh sách các truy vấn được tìm kiếm nhiều nhất trên Google.
- Danh bạ điện thoại

Sắp xếp là công việc phải làm trong mọi lĩnh vực hiện nay. Để thuận tiện hơn cho việc sắp xếp dữ liệu, càng ngày các giải thuật sắp xếp càng được tạo ra và cải tiến nhiều hơn. Từ những giải thuật sắp xếp đầu tiên như: Selection Sort, Bubble Sort, Insertion Sort,... dần dần đã xuất hiện những giải thuật nhanh hơn, mạnh hơn như Quick Sort, Merge Sort, Radix Sort, Flash Sort để phục vụ nhu cầu xử lí số liệu ngày càng tăng của con người.

Trong bài báo cáo này, tôi xin được trình bày về 11 giải thuật sắp xếp nổi tiếng trong lập trình. Đó là: Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Flash Sort.

2. Giới thiệu về các giải thuật sắp xếp

2.1. Selection Sort

2.1.1. Ý tưởng

Đầu tiên, về ý tưởng thì Selection Sort xuất phát từ phương pháp sắp xếp trực quan nhất là đưa phần tử nhỏ nhất lên đầu mảng, sau đó không quan tâm đến nó nữa và tiếp tục đưa phần tử nhỏ nhất trong phần còn lại của mảng lên vị trí thứ 2,thứ 3... cứ thế cho đến phần tử cuối cùng.

Tóm lại, với một dãy gồm N phần tử, ý tưởng của giải thuật là thực hiện N-1 lần việc đưa phần tử nhỏ nhất lên đầu mảng được xét.

2.1.2. Các bước thực hiện

Bước 1: đặt biến đếm i=0.
Bước 2: tìm phần tử nhỏ nhất a[min] trong dãy từ a[i] đến a[N].
Bước 3: Hoán vị giá trị của a[min] và a[i].
Bước 4: Nếu i<N-1: tăng i lên 1 đơn vị và quay lại bước 2.
Nếu i=N-1: dừng (khi đó mảng đã được sắp xếp)

2.1.3. Đánh giá giải thuật

Độ phức tạp thuật toán

Ta thấy rằng ở lượt thứ i thì cần n-i phép so sánh để tìm ra phần tử nhỏ nhất trong mảng hiện hành. Ta có biểu thức sau:

$$\sum_{1}^{n}(n-i) = \frac{n(n-1)}{2}$$

Suy ra:

- Trường hợp tốt nhất: O(n^2)
- Trường hợp trung bình: O(n^2)
- Trường hợp xấu nhất: O(n^2)

Ưu điểm:

- Dễ hiểu
- Ít phải đổi chỗ các phần tử.
- Không phải mượn thêm mảng trong quá trình sắp xếp nên không gian lưu trữ là O(1)

Nhược điểm:

- Kém hiệu quả với các bộ dữ liệu lớn vì độ phức tạp O(n^2)
- Mảng đã được sắp xếp nhưng phải duyệt tất cả để kiểm tra

2.2. Insertion Sort

2.2.1. Ý tưởng

Duyệt tường phần tử của dãy và chèn phần tử đó vào dãy con ở trước nó (là dãy từ đầu đến trước nó 1 vị trí) sao cho đảm bảo dãy con đó là dãy con tăng. Cứ thế đến khi chèn phần tử cuối cùng của mảng là ta hoàn thành.

2.2.2. Các bước thực hiện

Bước 1: i = 1; a[0] là dãy con tăng đầu tiên
Bước 2: Gán temp = a[i] và tìm vị trí thích hợp để chèn a[i] trong đoạn từ a[0] -> a[i-1]
Bước 3: Dời các phần tử sau vị trí chèn tìm được sang bên phải 1 vị trí
Bước 4: Gán a[vị trí tìm được ở bước 2] = a[temp]
Bước 5: Tăng i lên 1 đơn vị
Bước 6: Nếu i < n thì quay lại bước 2; ngược lại kết thúc giải thuật

2.2.3. Đánh giá giải thuật

Đối với giải thuật này, các phép so sánh được thực hiện trong vòng lặp While để tìm ra vị trí thích hợp cần chèn phần tử tiếp theo vào đó. Số phép gán sẽ phụ thuộc vào sự phân bố ban đầu của mảng cần sắp xếp. Giải thuật thích hợp với các mảng đã được sắp xếp hoặc gần như là được sắp xếp

Độ phức tạp thuật toán (về thời gian):

- Trường hợp tốt nhất: Số phép so sánh cần thực hiện là n-1. Đó là khi mảng đã được sắp từ trước.=> O(n)
- Trường hợp trung bình: O(n^2)
- Trường hợp xấu nhất:cần n(n+1)/2 phép so sánh và cũng n(n+1)/2 phép gán để hoàn thành giải thuật. => O(n^2)

Không gian bộ nhớ: O(1)

Ưu điểm:

- Giải thuật cơ sở hiệu quả nhất
- Dữ liệu gần như đúng thứ tự thì giải thuật này chạy rất nhanh

Nhược điểm:

- Kém hiệu quả với các bộ dữ liệu lớn vì độ phức tạp O(n^2)
- Mảng đã được sắp xếp giảm thì chạy rất chậm

2.3. Bubble Sort

2.3.1. Ý tưởng

Giải thuật này có một cái tên khá hay là nỗi bọt. Giải thuật này được thực hiện giống như bột bóng nổi lên đó là bằng cách lần lượt đưa các phần tử nhẹ (các phần tử nhỏ) nổi lên trên cùng của dãy bằng cách đổi chỗ trực tiếp giá trị của hai phần tử đứng cạnh nhau trong mảng hiện tại. Lặp lại quá trình đến khi mảng được sắp

2.3.2. Các bước thực hiện

Đây là 1 giải thuật đã được cải tiến bằng cách lưu vị trí cuối cùng đã sắp xếp cũng như kiểm tra xem đoạn con đã xét có sắp xếp không để khỏi bị thực hiện dư thừa

Bước 1: i = n - 1; // duyen tat ca phan tu tu cuoi ve dau
Bước 2: gán k=n-1(k là vị trí cuối cùng cần đổi chổi); và một biến check để kiểm tra đoạn sau có cần đổi chổ không và j= 0 để duyệt đoạn từ 0 đến vị trí cuối cùng cần đổi. Nếu a[j] > a[j+1] thì mình đổi chổ 2 phần tử này và đánh dấu check là có đổi chổ và cập nhật vị trí đổi chổ cuối cùng là j + 1
Bước 3: nếu mà không có đổi chổ (check = false) thì tức là mảng đã đúng mình dừng việt sếp xếp. Ngược lại nếu i> 0 thì giảm i 1 đơn vị và quay lại bước 2

2.3.3. Đánh giá giải thuật

Đối với giải thuật Bubble Sort thì số phép so sánh không phụ thuộc vào sự phân bố của dữ liệu, chỉ phụ thuộc vào độ lớn của dữ liệu. Còn về số phép gán phải thực hiện lại phụ thuộc vào kết quả các phép so sánh. Độ phức tạp thuật toán:

$$\sum_{00}^{n-1}(n-i-1) = \frac{n(n-1)}{2}$$

- Trường hợp tốt nhất: Khi mảng được sắp sẳn.=> O(n)
- Trường hợp trung bình: O(n^2)
- Trường hợp xấu nhất:Khi mảng sắp xếp giảm. => O(n^2)

Không gian bộ nhớ: O(1)

Ưu điểm:

- Dễ hiểu, dễ dàng trong trong giảng dạy

Nhược điểm:

- Kém hiệu quả với các bộ dữ liệu lớn vì độ phức tạp O(n^2)
- Mảng đã được sắp xếp giảm thì chạy rất chậm

2.4. Shaker Sort

2.4.1. Ý tưởng

Cải tiếng từ bubble sort do trường hợp số bé nằm ở cuối dãy nên cần đổi chiều lược duyệt sau mỗi lần duyệt => shaker Sort

2.4.2. Các bước thực hiện

Bước 1: left=1, right=n-1 // left và right là chỉ số 2 biên của dãy cần xếp.
k = n -1// Flag là cờ hiệu để đánh dấu vị trí cuối cùng xảy ra hoán vị trong thao tác trước đó.
Bước 2: gồm có 2 vòng lặp.
Bước 2.11: j = right
Nếu a[j]<a[j-1]
Hoán vị(a[j], a[j-1])
k=j // đánh dấu vị trí hoán vị
j– // tiếp tục duyệt
left = k + 1;
Bước 2.22: j=left
Nếu a[j]>a[j+1] Hoán vị(a[j], a[j-1])
k=j //đánh dấu vị trí hoán vị
j++ //tiếp tục duyệt
right= k - 1
Bước 3: Nếu left < right : Lặp lại Bước 2. Ngược lại, giải thuật hoàn tất

2.4.3. Đánh giá giải thuật

Giải thuật là cải tiến của Bubble Sort với 2 lượt lặp đi, về trong mỗi lần lặp và thêm cờ hiệu vào giải thuật. Cờ hiệu giúp giảm thiểu phần nào các bước lãng phí.Về 2 lượt lặp thì tuy giảm số vòng lặp lớn đi một nữa nhưng không có quá nhiều giá trị vì vẫn phải lặp với độ phức tạp $O(n^2)$ mới có thể hoàn tất sắp xếp

- Trường hợp tốt nhất: Khi mảng được sắp sẵn.=> O(n)
- Trường hợp trung bình: $O(n^2)$
- Trường hợp xấu nhất :Khi mảng sắp xếp giảm. => $O(n^2)$

Không gian bộ nhớ: O(1)

Ưu điểm:

- Dễ hiểu
- giải được chi phí hơn bubble sort

Nhược điểm:

- Kém hiệu quả với các bộ dữ liệu lớn vì độ phức tạp $O(n^2)$
- Không quá tối ưu hơn Bubble sort có khi chậm hơn

2.5. Shell Sort

2.5.1. Ý tưởng

Đây là một cải tiến khác của Insertion Sort.Ý tưởng của giải thuật là chia dãy cần sắp xếp thành các dãy con cách nhau h vị trí. Sắp xếp các dãy con đưa các phần tử về đúng vị trí tương đối trong dãy con. Sau đó giảm dần khoảng cách h qua các bước thực hiện. Thuật toán dừng lại khi h=1, khi đó các phần tử đảm bảo rằng đã được so sánh với nhau và sắp xếp thành dãy hoàn chỉnh.

2.5.2. Các bước thực hiện

Bước 1: Khởi tạo khoảng cách k = $\log_2 n - 1$; và $h_k = 1$
Bước 2: Chia mảng thành các mảng con cách nhau h_i đơn vị
Bước 3: Sắp xếp các mảng con bằng thuật toán Insertion Sort
Bước 4: Lặp lại cho đến khi mảng được sắp xếp

2.5.3. Đánh giá giải thuật

Độ phức tạp của giải thuật này là O(n^2). Tuy nhiên, hiệu quả của giải thuật phụ thuộc vào việc chọn khoảng cách h. Ta có bảng độ phức tạp theo h:

General term ($k \geq 1$)	Concrete gaps	Worst-case time complexity
$\left\lfloor \frac{N}{2^k} \right\rfloor$	$\left\lfloor \frac{N}{2} \right\rfloor, \left\lfloor \frac{N}{4} \right\rfloor, \ldots, 1$	$\Theta\left(N^2\right)$ [e.g. when $N = 2^p$]
$2\left\lfloor \frac{N}{2^{k+1}} \right\rfloor + 1$	$2\left\lfloor \frac{N}{4} \right\rfloor + 1, \ldots, 3, 1$	$\Theta\left(N^{\frac{3}{2}}\right)$
$2^k - 1$	$1, 3, 7, 15, 31, 63, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$
$2^k + 1$, prefixed with 1	$1, 3, 5, 9, 17, 33, 65, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$
Successive numbers of the form $2^p 3^q$ (3-smooth numbers)	$1, 2, 3, 4, 6, 8, 9, 12, \ldots$	$\Theta\left(N \log^2 N\right)$
$\frac{3^k - 1}{2}$, not greater than $\left\lceil \frac{N}{3} \right\rceil$	$1, 4, 13, 40, 121, \ldots$	$\Theta\left(N^{\frac{3}{2}}\right)$
$\prod_I a_q$, where $a_0 = 3$ $a_q = \min\left\{n \in \mathbb{N} : n \geq \left(\frac{5}{2}\right)^{q+1}, \forall p : 0 \leq p < q \Rightarrow \gcd(a_p, n) = 1\right\}$ $I = \left\{0 \leq q < r \mid q \neq \frac{1}{2}\left(r^2 + r\right) - k\right\}$ $r = \left\lfloor \sqrt{2k + \sqrt{2k}} \right\rfloor$	$1, 3, 7, 21, 48, 112, \ldots$	$O\left(N^{1+\sqrt{\frac{\ln(5\ 2)}{\ln(N)}}}\right)$
$4^k + 3 \cdot 2^{k-1} + 1$, prefixed with 1	$1, 8, 23, 77, 281, \ldots$	$O\left(N^{\frac{4}{3}}\right)$
$\begin{cases} 9\left(2^k - 2^{\frac{k}{2}}\right) + 1 & k \text{ even}, \\ 8 \cdot 2^k - 6 \cdot 2^{(k+1)/2} + 1 & k \text{ odd} \end{cases}$	$1, 5, 19, 41, 109, \ldots$	$O\left(N^{\frac{4}{3}}\right)$

Không gian bộ nhớ: O(n)

Ưu điểm:

- Đạt được hiệu quả cao nếu chọn h hợp lí
- Độ phức tạp hiệu quả hơn nhiều so với Insertion Sort

Nhược điểm:

- Phải lưu ý trong việc chọn khoảng cách h.
- Dữ liệu lớn chạy hơi chậm

2.6. Heap Sort

2.6.1. Ý tưởng

Heap sort là kỹ thuật sắp xếp so sánh (*comparison-based sorting*) dựa trên cấu trúc dữ liệu Binary Heap - một dạng cây nhị phân hoàn chỉnh (Complete Binary Tree) mà trong đó các giá trị được lưu trữ theo một thứ tự đặc biệt sao cho giá trị trong các nút (node) cha luôn lớn hơn (hoặc luôn nhỏ hơn) so với giá trị trong các nút con của nó.

Ý tưởng thực hiện của heap sort tương tự như selection sort: chúng ta lần lượt tìm phần tử lớn nhất và đặt vào cuối dãy, nhưng thay vì mỗi thao tác tìm kiếm phải duyệt hết dãy tốn O(N) thì ta chỉ việc lấy ra từ Heap mất O(log(N)). Lặp lại quá trình cho tất cả các phần tử cho đến khi dãy được sắp xếp hoàn toàn.

2.6.2. Các bước thực hiện

Giải thuật Heap Sort trải qua 2 giai đoạn:
Giai đoạn 1 : Hiệu chỉnh dãy ban đầu thành heap.
Giả sử nửa sau của dãy ban đầu là heap hoàn chỉnh, ta dùng vòng lặp lần lượt thêm từ phần tử vào bên trái của heap. Vừa thêm vừa hiệu chỉnh vị trí của các phần tử để thỏa mãn điều kiện heap.
Giai đoạn 2 :
Bước 1: Đưa phần tử lớn nhất về cuối dãy: Hoán vị (a_0,a_r)
Bước 2: r=r-1 // Loại bỏ phần tử lớn nhất ra khỏi heap;
Hiệu chỉnh phần còn lại thành heap mới.
Bước 3: Nếu r > 0: Quay lại bước 2. Ngược lại: dừng

2.6.3. Đánh giá giải thuật

Giải thuật Heap Sort hiệu quả hơn các giải thuật cơ sở nhiều lần bởi vì tận dụng được kết quả so sánh từ các bước phía trước để sử dụng cho bước sau.Heap Sort rất phức tạp, nhưng đã chứng minh được trong trường hợp xấu nhất thì độ phức tạp là O($n\log_2 n$)

Không gian bộ nhớ: O(1)

Ưu điểm:

- Thuật toán chạy nhanh.
- Tích lũy được kết quả của các bước so sánh để tạo nên cấu trúc heap.

Nhược điểm:

- Trong mọi trường hợp hầu như HeapSort đều chạy tốt nhất O($n\log_2 n$) trong khi các thuật khác tường hợp tốt nhất là O(n)
- Chậm hơn so với các giải thuật O($n\log_2 n$) khác

2.7. Merge Sort

2.7.1. Ý tưởng

Mỗi phần tử riêng lẻ trong một dãy đề có thể được coi là một dãy không giảm. Merge Sort cũng dựa trên điều này. Tức là từ một dãy ban đầu phân hoạch dần thành các dãy con cho đến khi dãy con chỉ còn một phần tử. Sau đó trộn các dãy con lại thành một dãy lớn hơn bằng cách phân phối luân phiên các phần tử trong dãy con có tính thứ tự.

2.7.2. Các bước thực hiện

Bước 1: k=1 //k là chiều dài của chuỗi con trong bước hiện hành.
Bước 2: Tách dãy ban đầu a_0,a_1, ..., a_n nguyên tắc luân phiên thành 2 mảng b, c
Bước 3: Trộn từng cặp dãy con gồm k phần tử của b và c theo thứ tự tăng dần vào a.
Bước 4: k=k*2
Nếu k<n thì quay lại bước 2.
Nếu k>=n thì dừng

2.7.3. Đánh giá giải thuật

Ta thấy rằng số lần lặp ở bước 2 và bước 3 của Merge Sort là $\log_2 n$ do k tăng gấp 2 sau mỗi lần lặp. Dễ thấy chi phí thực hiện bước 2 và bước 3 tỉ lệ thuận với n. Do đó độ phức tạp của giải thuật này là O(n$\log_2 n$). Do không sử dụng thông tin nào về đặc tính của dãy sắp xếp nên trong mọi trường hợp thì chi phí Merge Sort là không đổi.

Không gian bộ nhớ: O(n)

Ưu điểm:

- Thuật toán chạy nhanh.
- Có tính ổn định cao

Nhược điểm:

- Tiêu tốn thêm bộ nhớ để lưu trữ các mảng con.

2.8. Quick Sort

2.8.1. Ý tưởng

Nếu như Merge Sort trộn các dãy con lại thành một dãy được sắp xếp hoàn chỉnh thì Quick Sort lại làm điều ngược lại. Quick Sort chọn một mốc pivot và chia dãy thành 2 dãy con:

- Một nửa bên trái nhỏ hơn pivot.
- Một nửa bên phải lớn hơn pivot.

Tiếp tục phân hoạch các dãy con theo nguyên tắc trên ta sẽ được một dãy đã sắp xếp.

2.8.2. Các bước thực hiện

Bước 1: Lấy phần tử chốt a[k] là ở đầu, giữa hoặc cuối mảngmảng.
Bước 2: Chia mảng theo phần tử chốt.
Bước 3: Sử dụng sắp xếp nhanh một cách đệ qui với mảng con bên trái.
Bước 4: Sử dụng sắp xếp nhanh một cách đệ qui với mảng con bên phải.

2.8.3. Đánh giá giải thuật

Hiệu quả của Quick Sort phụ thuộc nhiều vào việc chọn pivot.

Trường hợp tốt nhất là trong mỗi lần chọn pivot đều chọn được phần tử trung vị (phần tử chia mảng ra làm 2 phần có số phần tử bằng nhau.), khi đó thì cần $\log_2 n$ lần phân hoạch để dãy sắp xếp thành công.

Nếu chọn phải phần tử lớn nhất hoặc nhỏ nhất là pivot thì dãy sẽ bị phân hoạch thành 2 phần không bằng nhau, 1 phần chỉ bao gồm 1 phần tử và phần còn lại có n-1 phần tử. Do vậy cần phân hoạch n lần mới hoàn thành sắp xếp.

Do vậy trường hợp tốt nhất và trường hợp trung bình đều có độ phức tạp là O($n\log_2 n$) ,trường hợp xấu nhất độ phức tạp lên đến O(n^2)

Không gian bộ nhớ: O(n)

Ưu điểm:

- Thuật toán chạy nhanh nhất
- Được sử dụng rộng rãi

Nhược điểm:

- Tốc độ giải thuật phụ thuộc vào việc chọn pivot

2.9. Counting Sort

2.9.1. Ý tưởng

Counting sort là một thuật toán sắp xếp cực nhanh một mảng các phần tử mà mỗi phần tử là các số nguyên không âm; Ý tưởng của Counting Sort là không sử dụng các phép so sánh để xác định vị trí các phần tử nữa, thay vào đó sẽ dựa vào phân bố của các phần tử. Đếm số lần xuất hiện của mỗi phần tử và dựa vào đó tìm ra vị trí của chúng.

2.9.2. Các bước thực hiện

Bước 1: Tìm Max là phần tử lớn nhất của mảng, Khởi tạo mảng F với Max phần tử với giá trị 0
Bước 2: Tính tần xuất của các phần tử mảng và lưu vào F tương ứng
Bước 3: Đếm số phần tử nhỏ hơn a[i] xuất hiện trong mảng: F[a[i]] += F[a[i-1]]
Bước 4: Duyệt từ cuối mảng về đầu để xác định vị trí đúng vị trí của phần tử đó và lưu vào mảng phụ b Bước 5: Gán mang a theo mảng b. Hoàn thành

2.9.3. Đánh giá giải thuật

Counting Sort dựa vào sự phân bố nên chỉ cần duyệt và đếm, không cần so sánh hay hoán vị, tiết kiệm rất nhiều chi phí cho chương trình. Rõ ràng độ phức tạp thuật toán của Counting Sort là O(M+n) với M là giá trị lớn nhất của dãy.

Không gian bộ nhớ O(M)

Ưu điểm:

- Thuật toán chạy nhanh
- Không dựa vào so sánh sử dụng so sánh

Nhược điểm:

- Tốc độ giải thuật phụ thuộc vào giá trị lớn nhất của mảng
- Dữ liệu được sắp hoặc gần như sắp xếp thì thuật toán không tận dụng được điểm này
- Chỉ sắp xếp được số nguyên không âm

2.10. Radit Sort

2.10.1. Ý tưởng

Nhược điểm lớn nhất của Counting Sort là phải sử dụng mảng F với kích thước lớn, khó kiểm soát. Một cải tiến của Counting Sort đó là Radix Sort, với việc giảm kích thước mảng count xuống còn 10 và lặp lại số lần là số chử số của phần tử lớn nhất. Sắp xếp dựa trên cơ số là một kỹ thuật sắp xếp các phần tử bằng cách nhóm các chữ số riêng lẻ của một giá trị có cùng một vị trí. Sau đó, sắp xếp tăng các phần tử hàng đơn vị, hàng chục ...

2.10.2. Các bước thực hiện

Bước 1: k=0 // bắt đầu sắp xếp từ hàng đơn vị
tạo 10 lô rỗng b_0, b_1, b_2,.., b_9 (dùng để chứa các chữ sô khác nhau)
Bước 2: i: 0 -> n-1 (duyệt từ đầu đến cuối mảng)
Đặt a_i và lô b_j với j là chữ số thứ k của a[i]
Bước 3: Nối b_0, b_1, b_2,.., b_9 với nhau theo thứ tự thành dãy a
Bước 4: K=k+1;
Nếu k<m thì trở lại bước 2;
Ngược lại, dừng.

2.10.3. Đáng giá giải thuật

Với một dãy n nào đó có tối đa m chữ số, thuật toán thực hiện m lần các thao tác phân lô và ghép lô. Trong thao tác phân lô, mỗi phần tử chỉ được xét đúng 1 lần, khi ghép cũng vậy, chi phí thực hiện thuật toán hiển nhiên sẽ là O(2mn)= O(n)

Không gian lưu trữ: O(n + n_b); n_b là kích thước mảng b

Ưu điểm:

- Thuật toán chạy nhanh
- Khắc phục được Counting Sort

Nhược điểm:

- Chỉ sắp xếp được số nguyên không âm
- Tốn vùng nhớ

2.11. Flash Sort

2.11.1. Ý tưởng

Ý tưởng cơ bản đằng sau Flash Sort là trong một tập dữ liệu có phân bố đã biết, có thể dễ dàng ước tính ngay lập tức vị trí cần đặt phần tử sau khi sắp xếp khi phạm vi của tập hợp đã biết. Ví dụ: nếu cho một tập dữ liệu thống nhất trong đó giá trị nhỏ nhất là 1 và tối đa là 200 và 100 là một phần tử của tập hợp, thì sẽ hợp lý khi đoán rằng 100 sẽ ở gần giữa tập sau khi nó được sắp xếp. Vị trí gần đúng này được gọi là một lớp. Nếu các lớp được đánh số từ 1 đến m thì phần tửA_i có thể được xác định chỉ số lớp mà nó thuộc về thông qua công thức sau:

$$K(A_i) = 1 + (int)((m-1)\frac{A_i - A_{min}}{A_{max} - A_{min}})$$

Trong đó A là tập đầu vào và mọi lớp có số phần tử là bằng nhau, ngoại trừ lớp cuối cùng chỉ bao gồm (các) phần tử lớn nhất của dãy. Việc phân loại đảm bảo rằng mọi phần tử trong một lớp đều lớn hơn bất kỳ phần tử nào trong một lớp thấp hơn. Điều này giúp ích cho công việc sắp xếp dữ liệu và giảm số lần đảo ngược. Sắp xếp chèn Insertion Sort sau đó được áp dụng cho tập hợp đã phân loại. Miễn là dữ liệu được phân phối đồng đều, kích thước lớp sẽ nhất quán và sắp xếp chèn sẽ hiệu quả về mặt tính toán.

2.11.2. Các bước thực hiện

Bước 1: Chọn m.
Lời khuyên từ thầy Phương là nên chọn m = 0.43 n với n là độ lớn dữ liệu đầu vào. Thuật toán sẽ nhanh hơn
Tạo mảng L[0..m].
Bước 2: Phân hoạch dữ liệu thành các lớp khác nhau từ 1 đến m.
Đếm số lượng phần tử các lớp theo quy luật:
i:0 -> n-1

$$K(A_i) = 1 + (int)((m-1)\frac{A_i - A_{min}}{A_{max} - A_{min}})$$

Bước 3: Sử dung Insertion Sort để sắp xếp dữ liệu trong từng lớp
duyệt i: 1 -> m
InsertionSort(lớp m)

2.11.3. Đánh giá giải thuật

Độ phức tạp về thời gian:

- Trường hợp tốt nhất: O(n + m)
- Trường hợp trung bình: O(n + m)
- Trường hợp xấu nhất: O(n^2)

Không gian lưu trữ: O(n^2)

Ưu điểm:

- Chi phí thấp, ổn định cao
- Tận dụng tốt sự phân bố của dữ liệu
- Độ phức tạp tốt nhất gần như tuyến tínhtính

Nhược điểm:

- Dùng thêm mảng nên tốn bộ nhớ

3. Phân tích thông quaqua kết quả thực nghiệm

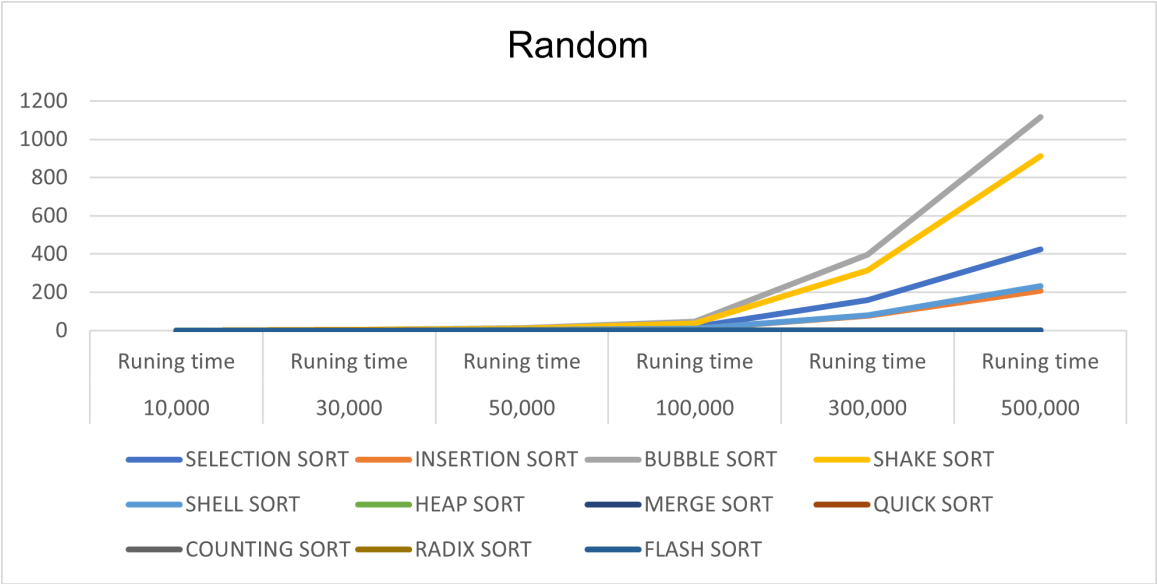
3.1. Bộ dữ liệu ngẫu nhiên

3.1.1. Bảng dữ liệu về thời gian chạy và số phép so sánh

Data order: Random												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions
SELECTION SORT	0.171875	100020001	1.5	900060001	4.14062	2500100001	18.0781	10000200001	158.203	90000600001	425.938	2.50001E+11
INSERTION SORT	0.0625	49550883	0.78125	451013275	2.07812	1250367805	8.57812	4982938183	77.0312	45123776116	208.891	1.24988E+11
BUBBLE SORT	0.421875	99966737	3.96875	899981451	11.75	2499903049	45.0156	10000037108	396.156	89997407851	1115.8	2.49996E+11
SHAKE SORT	0.359375	66143165	3.0625	601090929	10.25	1667291888	36.0469	6648635857	313.547	60144747298	913.094	1.66706E+11
SHELL SORT	0.09375	49744576	0.78125	451670593	2.35938	1251552356	9.04688	4985507199	78.5938	45132551939	232.594	1.25003E+11
HEAP SORT	0	379502	0.015625	1280139	0	2247905	0.015625	4794640	0.0625	15797625	0.109375	27423196
MERGE SORT	0	582859	0	1935824	0.015625	3377812	0.03125	7155370	0.109375	23358802	0.1875	40369354
QUICK SORT	0	272272	0.015625	930171	0	1599735	0.015625	3458778	0.046875	10994000	0.0625	19124517
COUNTING SORT	0	50002	0	150001	0	250002	0.015625	500003	0	1500003	0.03125	2500003
RADIX SORT	0.015625	140061	0.015625	510076	0.03125	850076	0.046875	1700076	0.1875	6000091	0.328125	10000091
FLASH SORT	0	339213	0	1114107	0	1931804	0.015625	4068954	0.03125	13165975	0.078125	22777783

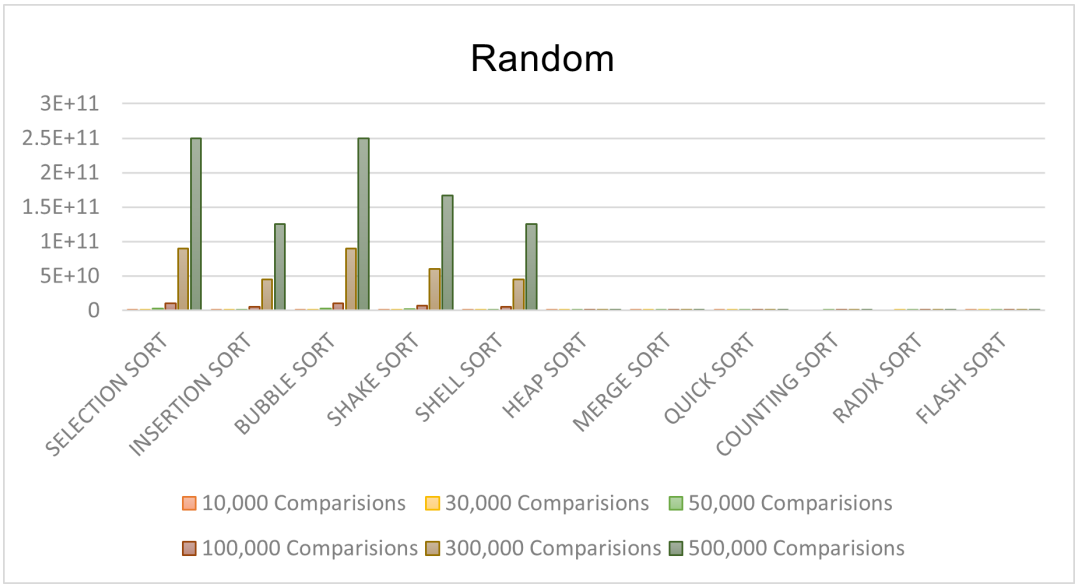
Bảng số liệu thời gian chạy (s) và số phép so sánh

3.1.2. Biểu đồ so sánh tốc độ chạy



Biểu đồ so sánh tốc độ chạy các giải thuật(Đơn vị: thời gian)

3.1.3. Biểu đồ so sánh số phép so sánh



Biểu đồ so sánh Phép so sánh

3.1.4. Nhận xét

Nhìn vào số liệu ta nhận thấy Counting Sort là thuật toán nhanh nhất với bộ dữ liệu 500000 phần tử. Sở dĩ có được điều này là do Counting Sort chạy với độ phức tạp thuật toán là O(n)à các giải thuật khác phải thực hiện nhiều thao tác hơn so với Counting Sort.

Các thuật toán có độ phức tạp O(n^2) đặc biệt là Bubble Sort và Shaker Sort chạy rất chậm lên đến 1115.8(s) và 914(s). Và các thuật toán này cũng sử dụng rất nhiều thao tác đặc biệt là Selection Sort.

Các thuật toán O(nlog_n) chạy tương đối ổn định không quá 1s. Và số thao tác cũng không quá nhiều.

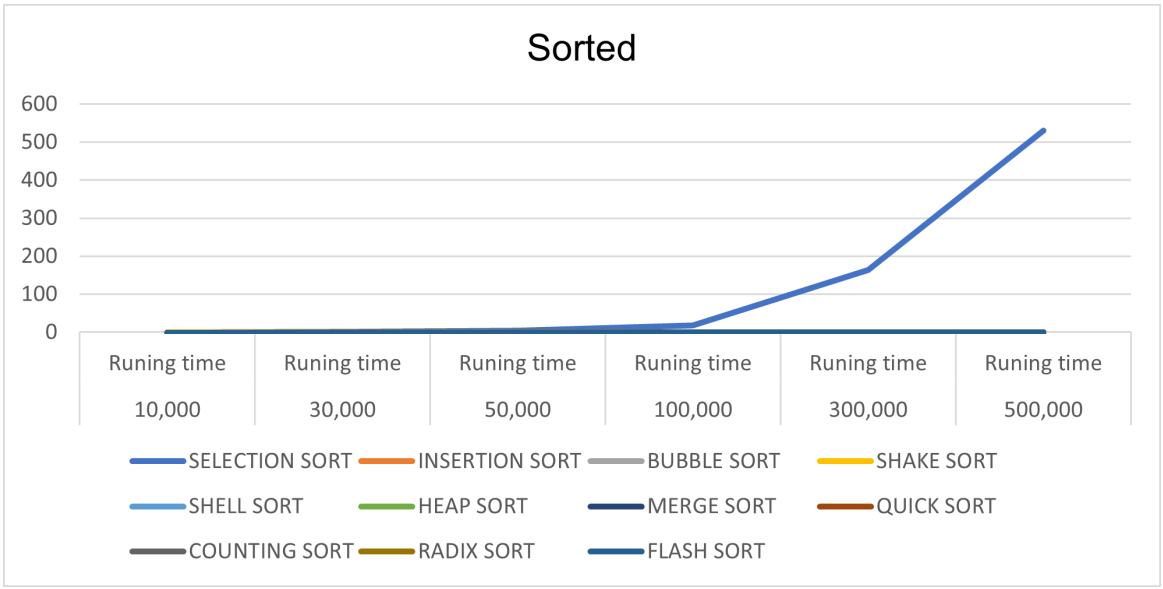
3.2. Bộ dữ liệu đã được sắp xếp tăngtăng

3.2.1. Bảng dữ liệu về thời gian chạy và số phép so sánh

Data order: Sorted												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions
SELECTION SORT	0.171875	100020001	1.64062	900060001	4.53125	2500100001	18.1406	10000200001	163.578	90000600001	529.719	2.50001E+11
INSERTION SORT	0	29998	0	89998	0	149998	0	299998	0	899998	0	1499998
BUBBLE SORT	0	20001	0	60001	0	100001	0	200001	0.015625	600001	0	1000001
SHAKE SORT	0	20001	0	60001	0	100001	0	200001	0	600001	0	1000001
SHELL SORT	0	223682	0	747303	0.015625	1334540	0.015625	2869009	0.015625	9675803	0.046875	16475803
HEAP SORT	0	396533	0	1326471	0	2328781	0.015625	4963737	0.046875	16292615	0.09375	28201279
MERGE SORT	0	466442	0	1543466	0.015625	2683946	0	5667898	0.078125	18408314	0.140625	31836410
QUICK SORT	0	149055	0	485546	0	881083	0.015625	1862156	0	5889300	0.03125	10048590
COUNTING SORT	0	50003	0.015625	150003	0	250003	0	500003	0.015625	1500003	0	2500003
RADIX SORT	0	140061	0.015625	510076	0.03125	850076	0.046875	1700076	0.1875	6000091	0.40625	10000091
FLASH SORT	0	367067	0	1194369	0	2067239	0.03125	4334497	0.03125	13949119	0.046875	23953017

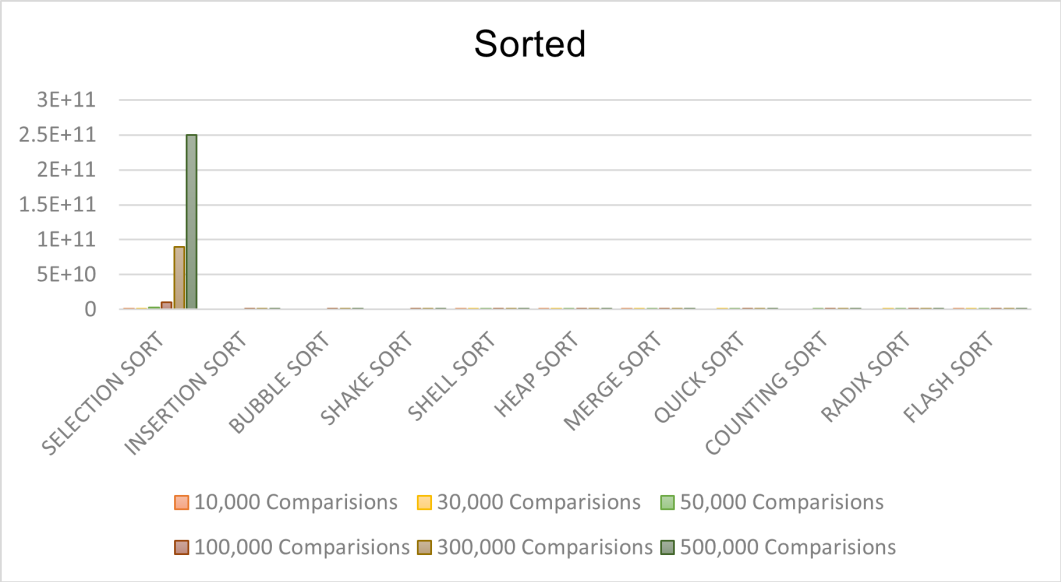
Bảng số liệu thời gian chạy (s) và số phép so sánh

3.2.2. Biểu đồ so sánh tốc độ chạy



Biểu đồ so sánh tốc độ chạy các giải thuật(Đơn vị: thời gian)

3.2.3. Biểu đồ so sánh số phép so sánh



Biểu đồ so sánh Phép so sánh

3.2.4. Nhận xét

Phần lớn các thuật toán đề có thời gian chạy cực nhanh và số phép thao tác ở bộ dữ liệu này, vì chúng có sẵn những đặc điểm trong thuật toán để tận dụng đặc điểm của bộ dữ liệu. Chỉ có Selection Sort là chạy chậm và có số thao tác lớn vì nó phải duyệt n - 1 vòng lặp để tìm ra giá trị nhỏ nhỏ nhất trong mỗi vòng lặp nên chi phí và thao tác vẫn O(n^2).

Ở kiểu dữ liệu này rất đặc biệt dó là Insertion và Shaker Sort có thời gian chạy thực thế cực nhanh. Bởi vì Insertion Sort đơn thuần tìm vị trí cần chèn vào dãy phía trước của một phần tử ở mỗi bước. Ở đây phần tử đó đã ở đúng vị trí của nó ngay từ đầu nên Insertion Sort chỉ việc duyệt tuần tự dãy với độ phức tạp O(n) là đã hoàn thành sắp xếp. Tương tự là Shaker Sort với kk đánh dấu vị trí cuối cùng cần hoán vị nên sau khi duyệt 1 lần thì thuật toán đã biết được dãy đã được sắp xếp.

Điều bất ngờ nhất là thuật toán Bubble Sort dạy rất rất nhanh trong khi với bộ dữ liệu trước chạy rất chậm. Vì Bubble Sort này là mình đã cải tiến cấp 2 có 1 biến check xem đoạn con vừa xét đã được sắp đúng chưa nếu đúng thì dừng ngay nên ở bộ dữ liệu này Bubble Sort chỉ cần chạy với DPT O(n).

Counting Sort vẫn chạy rất nhanh. Các giải thuật tốt vẫn chạy ổn định. Và với bộ dữ liệu này ta thấy thế mạnh của Quick Sort do pivot được chọn là phần tử trung tâm nên chạy rất nhanh.

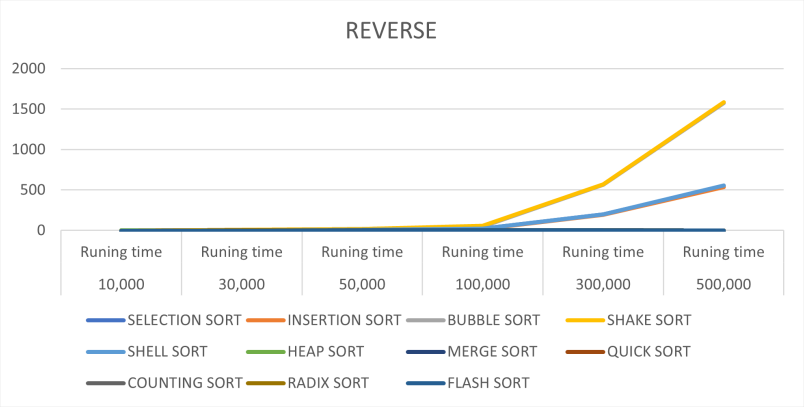
3.3. Bộ dữ liệu đã được sắp xếp giảm

3.3.1. Bảng dữ liệu về thời gian chạy và số phép so sánh

Data order: REVERSE												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions
SELECTION SORT	0.21875	100020001	1.95312	900060001	5.4375	2500100001	21.1719	10000200001	196.625	90000600001	544.859	2.50001E+11
INSERTION SORT	0.21875	100009999	1.9375	900029999	5.34375	2500049999	16.5469	10000099999	193.047	90000299999	535.734	2.5E+11
BUBBLE SORT	0.625	100039998	5.64062	900119998	15.7656	2500199998	48.0938	10000399998	565.234	90001199998	1576.53	2.50002E+11
SHAKE SORT	0.640625	100005000	5.6875	900015000	15.8281	2500025000	58.6875	10000050000	570	90000150000	1583.17	2.5E+11
SHELL SORT	0.203125	100213682	1.84375	900717303	5.28125	2501284540	22.125	10002769009	199.125	90009375803	553.656	2.50016E+11
HEAP SORT	0.015625	363417	0	1234995	0	2165045	0.015625	4624189	0.046875	15317245	0.109375	26647015
MERGE SORT	0	485241	0.015625	1589913	0	2772825	0.03125	5845657	0.078125	18945945	0.140625	32517849
QUICK SORT	0	159070	0	515556	0.015625	931094	0	1962168	0.015625	6189320	0.03125	10548604
COUNTING SORT	0	50003	0	150003	0	250003	0.015625	500003	0	1500003	0.015625	2500003
RADIX SORT	0	140061	0.015625	510076	0.03125	850076	0.0625	1700076	0.25	6000091	0.40625	10000091
FLASH SORT	0	352851	0	1151521	0.015625	1996587	0.015625	4193181	0.046875	13521275	0.0625	23237225

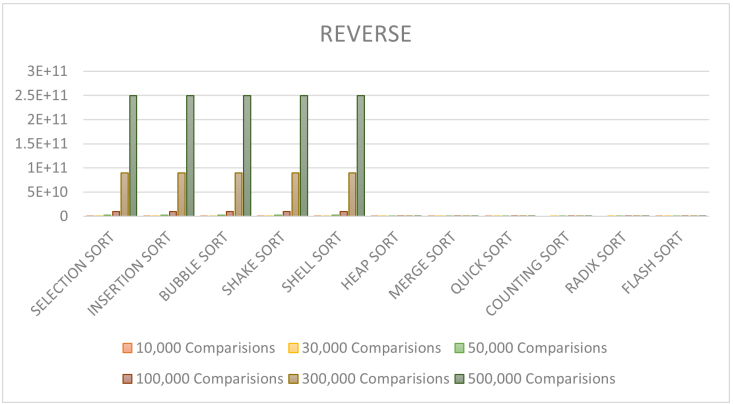
Bảng số liệu thời gian chạy (s) và số phép so sánh

3.3.2. Biểu đồ so sánh tốc độ chạy



Biểu đồ so sánh tốc độ chạy các giải thuật(Đơn vị: thời gian)

3.3.3. Biểu đồ so sánh số phép so sánh



Biểu đồ so sánh Phép so sánh

3.3.4. Nhận xét

Ở bộ dữ liệu này, Shaker Sort và Bubble Sort là những giải thuật làm việc kém hiệu quả nhất. Trong khi Bubble Sort phải hoán vị với số lần $\frac{n(n-1)}{2}$ để có thể đưa toàn bộ dãy về đúng vị trí của chúng. Những cải tiến của Shaker Sort, Bubble Sort hầu như vô dụng trong trường

hợp này. Flag luôn nằm ở biên của dãy nên không rút ngắn được quá nhiều thời gian chạy thuật toán so với Bubble Sort.

Các giải thuật tốt vẫn chạy ổn định, chạy chưa tới 1s. Counting sort vẫn chạy rất nhanh và đặc biệt Quick Sort tỏ ra vượt trội ở bộ dữ liệu này bởi vì với dãy bị đảo ngược thì phần tử ở giữa của mỗi dãy luôn là trung vị của dãy nên cách chọn pivot này giúp thuật toán lợi thế rất nhiều so với các thuật toán khác.

Các giải thuật cơ sở cần số lượng thao tác vô cùng lớn ở bộ dữ liệu này.

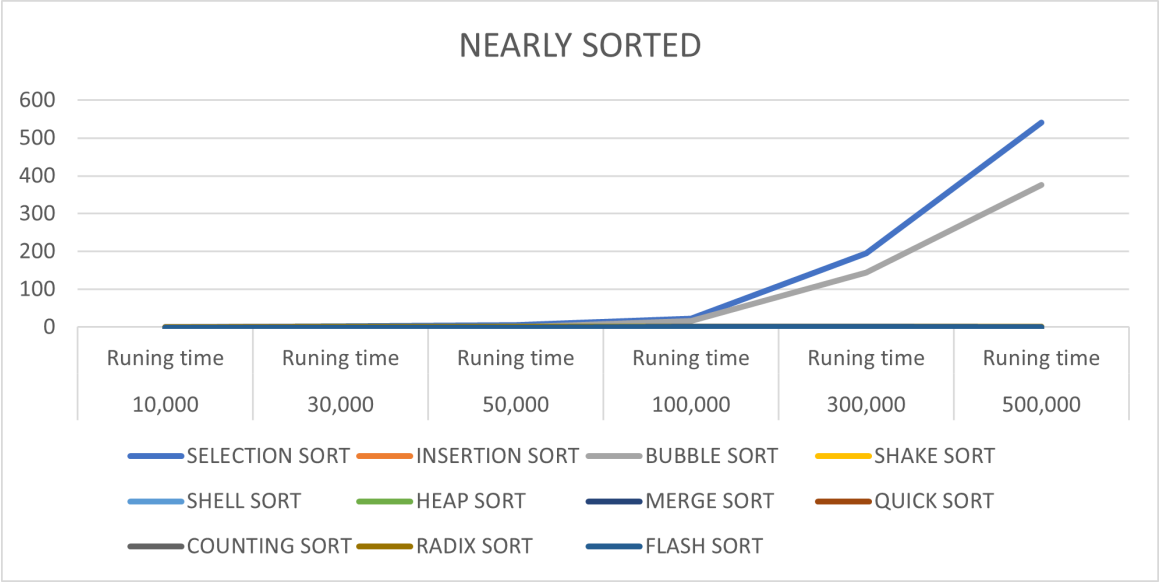
3.4. Bộ dữ liệu được sắp xếp gần như tăng

3.4.1. Bảng dữ liệu về thời gian chạy và số phép so sánh

Data order: NEARLY SORTED												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions	Runing time	Comparisions
SELECTION SORT	0.21875	100020001	1.9375	900060001	5.40625	2500100001	21.6406	10000200001	194.766	90000600001	541.016	2.50001E+11
INSERTION SORT	0	112186	0	434450	0.015625	1091786	0	1610290	0.015625	5064158	0	6220886
BUBBLE SORT	0.140625	71822053	1.45312	700887569	0.015625	2200961691	15.4531	7474868088	144.141	69655827607	376.406	1.81857E+11
SHAKE SORT	0	132848	0	471247	0.015625	1089964	0.015625	1736207	0.03125	5189047	0.03125	6564672
SHELL SORT	0	305870	0	1091755	0	2276328	0.015625	4179301	0.03125	13839963	0.0625	21196691
HEAP SORT	0	396419	0.015625	1326461	0.015625	2328735	0.015625	4963367	0.0625	16292041	0.09375	28196985
MERGE SORT	0	490961	0	1638901	0	2867910	0.015625	6001574	0.078125	19400810	0.140625	33264834
QUICK SORT	0	149087	0	485595	0	881143	0.015625	1862200	0.015625	5889344	0.03125	10048618
COUNTING SORT	0	50003	0	150003	0.015625	250003	0	500003	0.015625	1500003	0.015625	2500003
RADIX SORT	0.015625	140061	0.03125	510076	0.03125	850076	0.0625	1700076	0.234375	6000091	0.421875	10000091
FLASH SORT	0	367045	0	1194347	0	2067226	0.015625	4334479	0.046875	13949094	0.0625	23953003

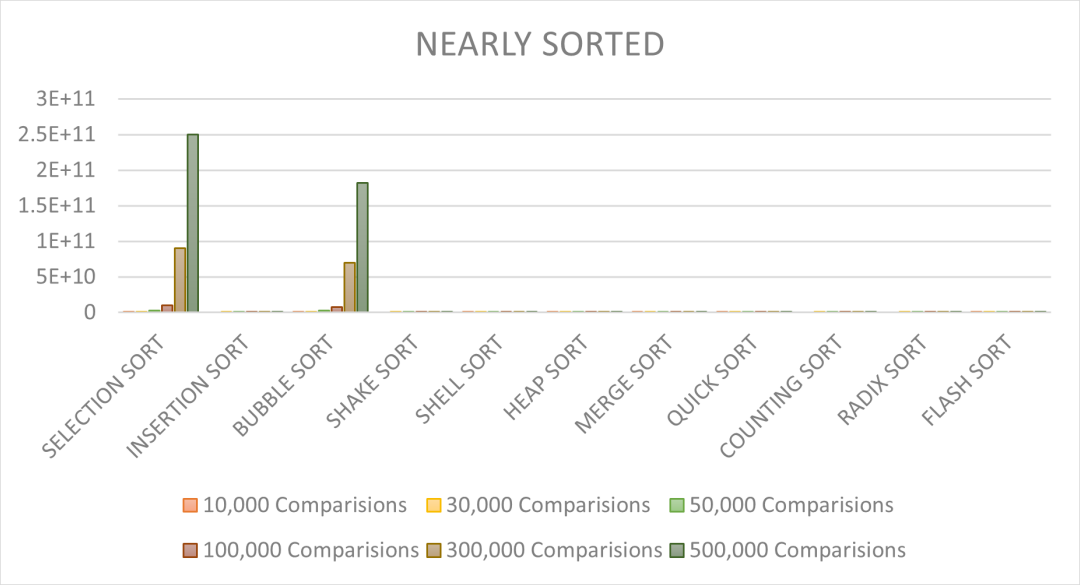
Bảng số liệu thời gian chạy (s) và số phép so sánh

3.4.2. Biểu đồ so sánh tốc độ chạy



Biểu đồ so sánh tốc độ chạy các giải thuật(Đơn vị: thời gian)

3.4.3. Biểu đồ so sánh số phép so sánh



Biểu đồ so sánh Phép so sánh

3.4.4. Nhận xét

Bộ dứ liệu này có khá nhiều điểm tương đồng so với bộ dữ liệu đã được sắp xếp. Tuy nhiên Bubble Sort chạy rất chậm là dãy gần như sắp xếp không phải đã được sắp xếp nên biến check(phần cải tiến) gần như không có tác dụng nên phải duyệt hết tất cả vị vậy DPT O(n^2)

Insertion Sort và Shaker Sort là những thuật toán sắp xếp cơ sở có thể tận dụng tốt đặc điểm bộ dữ liệu.

Quick Sort có cách chọn pivot ở giữa, luôn tìm được pivot gần với trung vị của dãy ở mỗi bước, rất mạnh ở những kiểu dữ liệu có sự phân bố đều như trường hợp thứ tư này

Counting sort vẫn lun giữ sự ổn định của mình luôn chạy rất nhanh vì Phần tử mảng không quá lớn. Các thuật toán tốt khác như Flash Sort, Merge Sort, Heap Sort thì luôn ổn định.

3.5. Đánh giá chung

Mỗi thuật toán sắp xếp đều có lợi thế riêng phụ thuộc mục đích sử dụng, bộ dữ liệu sử dụng và càng ngày càng được cải tiến để nhanh hơn mạnh mẽ hơn. Học và hiểu được các thuật toán sắp xếp này giúp ích rất nhiều cho việc học lập trình, nghiên cứu và cuộc sống.

4. Chú thích bài lập trình

4.1. Tập tin .cpp .h

1. Sort.cpp, Sort.h: Gồm code 11 hàm sắp xếp (có tính số thao tác)

2. Time.h Time.cpp: Dùng để tính thời gian thực thi của các hàm sắp xếp

3. DataGenerator.h DataGenerator.cpp: Dùng để sinh các bộ test

4. Command.h Command.cpp: Giúp nhập command line

5. main.cpp: Chương trình chính

4.2. Các tập tin dữ liệu

1. help.txt: thông tin để nhập command line

2. input.txt: chứa dữ liệu nhập

3. ouput.txt: Chứa kết quả sau khi sắp xếp

4.3. Các tập tin kết quả

1. DataResult.txt: Chứ kết quả thời gian chạy và số thao tác thực thi của từng hàm sắp xếp theo tất cả các kính thước và loại dữ liệu.

2. DataResult.xlsx: Chứa kết quả như DataResult.txt và có vẽ đồ thị minh họa.

5. Tài liệu tham khảo

- Shaker Sort: https://www.geeksforgeeks.org/cocktail-sort/
- Shell Sort: https://www.geeksforgeeks.org/shellsort/
- Merge Sort: https://www.geeksforgeeks.org/merge-sort/?ref=leftbar-rightbar
- Heap Sort: https://www.geeksforgeeks.org/heap-sort/
- Flash Sort: https://en.wikipedia.org/wiki/Flashsort