TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

LAB 3: CÁC THUẬT TOÁN SẮP XẾP

MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Giáo viên hướng dẫn:	Nguyễn Thanh Phương Bùi Huy Thông
Sinh viên thực hiện:	Dũ Quốc Huy MSSV: 20120101 Lớp: 20CTT1TN

Mục Lục

1 Mở đầu

Sắp xếp là quá trình bố trí lại các phần tử trong một tập hợp theo một trình tự nào đó nhằm mục đích giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn. Hiện nay có rất nhiều thuật toán sắp xếp khác nhau được các nhà khoa học máy tính đưa ra và ở chúng luôn có những ưu nhược điểm riêng của nó. Và phần tiếp theo đây, ta sẽ tìm hiểu về những thuật toán phổ biến nhất cũng như cách hoạt động, những lợi thế riêng của từng thuật toán để có thể vận dụng chúng một cách linh hoạt và hiệu quả nhất. Báo cáo này sẽ sử dụng các thuật toán để sắp xếp dữ liệu của mảng theo thứ tự tăng dần.

2 Các thuật toán sắp xếp phổ biến

2.1 Selection Sort

2.1.1 Ý tưởng thuật toán

Sắp xếp chọn là một trong những thuật toán sắp xếp đơn giản rất được phổ biến với người dùng. Nó sử dụng mô hình logic đơn giản liên quan đến việc tìm kiếm giá trị tối thiểu của phần dữ liệu chưa được sắp và di chuyển nó về đầu danh sách.

2.1.2 Các bước thực hiện

- Bước 1: Bắt đầu với vị trí đầu tiên của mảng, ta sẽ tìm phần tử nhỏ nhất của mảng từ vị trí đầu tiên đến hết mảng.
- Bước 2: Hoán đổi giá trị ở vị trí đầu tiên với giá trị nhỏ nhất vừa tìm được ở Bước 1 nếu 2 vị trí chứa 2 giá trị đó khác nhau.
- Bước 3: Sau khi thực hiện Bước 2: ta đã có được phần tử nhỏ nhất của mảng nằm đúng vị trí của nó là đầu mảng., phần dữ liệu chưa được sắp đã giảm đi 1 phần tử. Thực hiện lại Bước 1 và Bước 2 cho đến khi tất cả các phần tử của mảng đều nằm đúng vị trí.

2.1.3 Đánh giá thuật toán

Selection Sort là một thuật toán sắp xếp tại chỗ, đơn giản dễ thực hiện.
Với mảng có n phần tử, khi ở vị trí i, ta thực hiện (n-i) phép so sánh để có thể tìm ra giá trị nhỏ nhất, và việc này không phụ thuộc vào sự sắp xếp ban đầu của mảng, cho nên số lần so sánh là:

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

Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n^2)$
2. Trường hợp xấu nhất: $O(n^2)$

3. Trường hợp trung bình: $O(n^2)$

Độ phức tạp về không gian: $O(1)$,do một biến bổ sung được sử dụng.
Tính ổn định của thuật toán: không ổn định.

2.2 Insertion Sort

2.2.1 Ý tưởng thuật toán

Lấy ý tưởng giống như khi chúng ta sắp những quân bài trên tay, chia phần dữ liệu thành 2 phần, ví dụ phần đầu đã được sắp xếp, ta sẽ thực hiện chèn từng phân tử của phần chưa được sắp xếp vào phần đã được sắp sao cho luôn giữ được thứ tự.

2.2.2 Các bước thực hiện

- Bước 0: Thuật toán sẽ chia mảng thành 2 phần: phần bên phải sẽ là đã được sắp theo thứ tự, phần bên trái giống như mảng ban đầu.
- Bước 1: Bắt đầu với vị trí thứ 2 có giá trị là x, thực hiện chèn x vào phần mảng đã được sắp sao vẫn giữ được thứ tự tăng dần (tính đến vị trí thứ 2).
- Bước 2: Sau khi chèn 1 phần tử, phần mảng đã sắp xếp sẽ tăng lên 1, Ta thực hiện lại Bước 1 với phần tử tiếp theo cho đến khi toàn bộ mảng đều được sắp.

2.2.3 Đánh giá thuật toán

Insertion Sort là một thuật toán cơ bản, có ý tưởng đơn giản, thực hiện việc sắp xếp tại chỗ, với một số cả tiến bên dưới có thể sẽ giúp cho việc sử dụng Insertion Sort được tối ưu hơn.
Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n)$, khi mảng ban đầu đã được sắp đúng thứ tự
2. Trường hợp xấu nhất: $O(n^2)$,khi mảng được sắp xếp ngược, với mỗi vị trí ta phải thực hiện i phép so sánh

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

3. Trường hợp trung bình: $O(n^2)$, khi mảng ban đầu được sắp thứ tự lộn xộn (không tăng cũng không giảm)

Độ phức tạp về không gian: $O(1)$, do một biến bổ sung được sử dụng.
Tính ổn định của thuật toán: có sự ổn định.

2.2.4 Cải tiến thuật toán

Thay vì phải đi so sánh từng phần tử để tìm ra vị trí có thể chèn ở Bước 1, ta sử dụng tìm kiếm nhị phân để tiết kiệm thời gian tìm kiếm

```
l = 0; r = i - 1; v = a[i];
            while (l <= r)
            {
                    m = (r + l) / 2;
                    if (a[m] > v)
                            r = m - 1;
                    else
                            l = m + 1;
            }
```

***Note: Phần chương trình trong file thực thi đính kèm được sử dụng thuật toán sắp xếp chèn kết hợp cùng tìm kiếm nhị phân.**

2.3 Bubble Sort

2.3.1 Ý tưởng thuật toán

Sắp xếp nổi bọt là một thuật toán đơn giản với việc sắp xếp mảng bằng cách hoán đổi 2 vị trí về nhau nếu chúng không đứng đúng thứ tự. Phần tủ lớn nhất sẽ từ từ đi về cuối.

2.3.2 Các bước thực hiện

- Bước 0: Vòng lặp lớn để duyệt phần tử, vòng lặp nhỏ để so sánh và hoàn đổi phần tử
- Bước 1: Bắt đầu với phần tử đầu tiên, sử dụng vòng lặp từ $(i + 1)$ đến $(n - i+)$ để so sánh và hoàn đổi nếu 2 phần tử kề nhau không theo thứ tự tăng dần.
- Bước 2: Sau khi Bước 1 hoàn thành, phần tử lớn nhất sẽ được đưa về cuối mảng. Thực hiện lại Bước 1 bằng cách duyệt từ phần tử tiếp theo cho đến khi mảng hoàn toàn được sắp xếp.

2.3.3 Đánh giá thuật toán

Bằng việc hoán đổi 2 phần tử liên tiếp giống như việc một bột bong bóng nổi lên đẩy bột bong bóng khác xuống, thuật toán có thể áp dụng những cải tiến để tối ưu hóa xử lí dữ liệu.
Với mảng có n phần tử, khi ở vị trí i, ta thực hiện $(n - i)$ phép so sánh để quyết định xem có nên hoán đổi vị trí 2 phần tử hay không, và việc này không phụ thuộc vào sự sắp xếp ban đầu của mảng, cho nên số lần so sánh là:

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

Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n^2)$
2. Trường hợp xấu nhất: $O(n^2)$
3. Trường hợp trung bình: $O(n^2)$

Độ phức tạp về không gian: $O(1)$,do một biến bổ sung được sử dụng cho việc hoán đổi. Tính ổn định của thuật toán: có sự ổn đinh.

2.3.4 Cải tiến thuật toán

-Cải tiến 1: nếu phần sau của mảng đã được sắp đúng thứ tự (các phần tử liên tiếp từ vị trí i đến cuối mảng), nhưng vòng lặp nhỏ vẫn chạy để kiểm tra. Vì thể để tiết kiệm thời gian ta chỉ cần xác định vị trí hoán đổi cuối cùng của vòng lặp trước: sử dụng biến k để lưu lại vị trí cuối cùng hoán đổi của lần lặp trước, và biết rằng từ k trở đi mảng đã được sắp xếp.

-Cải tiến 2: mảng đã được sắp xếp sau x lần lặp ($x < n-1$), nhưng theo thuật toán thì vòng lặp lớn vẫn tiếp tục chạy cho đến khi tới phần tử $(n-1)$. Vì thế để chương trình có thể nhận biết được mảng đã được sắp và không cần chạy nữa ta sử dụng một biến kiểm tra: nếu lần duyệt tại vị trí thứ i không hề có sự hoán đổi nào (tức là các phần tử đã ở đúng vị trí), ta sẽ dừng vòng lặp và mảng khi đó đã được sắp xếp hoàn toàn.

Đánh giá thuật toán sau khi thực hiện cải tiến:

+Thời gian thực thi ở trường hợp tốt nhất là $O(n)$ nếu mảng ban đầu đã được sắp.

***Note: Phần chương trình trong file thực thi đính kèm được sử dụng thuật toán sắp xếp nổi bọt cải tiến.**

2.4 Shaker Sort

2.4.1 Ý tưởng thuật toán

Shaker Sort (hay còn gọi là Cocktail Sort) xem như một cái tiến của Bubble Sort bằng cách đảo ngược liên tục chiều duyệt, đưa phần tử lớn nhất về cuối, tiếp theo lại đưa phần tử nhỏ nhất về đầu. Do đó các phần tử về đúng vị trí cả 2 đầu của mảng nên sẽ cải thiện thời gian sắp xếp dãy số.

2.4.2 Các bước thực hiện

- Bước 1: Sử dụng các biến để lưu vị trí cần duyệt của phần mảng chưa được sắp xếp: $left$ (biến bên trái),$right$ (biên bên phải). Ban đầu $left = 0$, $right = n-1$

- Bước 2: Nếu ($left <= right$), ta thực hiện 2 lượt duyệt
  - Lượt đi: từ $left$ đến $right$, thực hiện hoán đổi vị trí 2 phần tử liên tiếp không đúng thứ tự tăng dần. Sau đó gán lại $right$ là vị trí hoán đổi lớn nhất. Phần tử lớn nhất sẽ được đưa về biên bên phải.
  - Lượt về: từ $right$ đến $left$, thực hiện hoán đổi vị trí 2 phần tử liên tiếp không đúng thứ tự giảm dần. Sau đó gán lại $left$ là vị trí hoàn đổi nhỏ nhất. Phần tử nhỏ nhất sẽ được đưa về biên bên trái.

  .
- Bước 3: Lặp lại Bước 2 cho đến khi 2 vị trí $left$ và $right$ trùng nhau.

2.4.3 Đánh giá thuật toán

Bằng cách sắp xếp từ cả 2 đầu của mảng, thuật toán giúp giảm số lần so sánh, đem lại hiệu hơn so với việc chỉ sắp xếp từ 1 đầu. Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n)$, mảng ban đầu đã được sắp theo thứ tự tăng dần
2. Trường hợp xấu nhất: $O(n^2)$, khi mảng ban đầu được sắp theo thứ tự giảm dần
3. Trường hợp trung bình: $O(n^2)$, mảng ngẫu nhiên không tăng, không giảm

Độ phức tạp về không gian: $O(1)$
Tính ổn định của thuật toán: có sự ổn định.

2.5 Shell Sort

2.5.1 Ý tưởng thuật toán

Shell Sort là một thuật toấn sắp xếp mang lại hiệu quả cao dựa trên Insertion Sort, giải thuật này nhằm tránh các trường hợp hoán đổi 2 vị trí cách xa nhau
Giải thuật này sử dụng giải thuật sắp xếp chọn trên các phần tử có khoảng cách xa nhau, sau đó sắp xếp các phần tử có khoảng cách hẹp hơn. Khoảng cách này còn được gọi là khoảng (interval).

2.5.2 Các bước thực hiện

- Bước 1: Sử dụng biến $interval$ để chia khoảng cho mảng ban đầu, $interval$ sẽ nhận các giá trị lần lượt là $\frac{n}{2}$, $\frac{n}{2}$, ... ,1
- Bước 2: Với mỗi giá trị của $interval$, ta sử dụng thuật toán $InsertionSort$ để đưa các phần tử cách nhau từng khoảng bằng $interval$ về đúng thứ tự tăng dần .
- Bước 3: Giảm giá trị $interval$ đi một nữa để thực hiện sắp xếp trong phạm vi hẹp hơn, thực hiện lại Bước 2 cho đến khi $interval = 1$

2.5.3 Đánh giá thuật toán

Bằng cách chia khoảng nhỏ ra để so sánh và sắp xếp, thuật toán đã tối ưu hóa việc so sánh các phần tử so với Insertion Sort.
Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n)$
2. Trường hợp xấu nhất: $O(n^2)$
3. Trường hợp trung bình: phụ thuộc vào số lượng khoảng cách

Độ phức tạp về không gian: $O(1)$
Tính ổn định của thuật toán: không ổn định.

2.6 Heap Sort

2.6.1 Ý tưởng thuật toán

Để hiểu rõ về thuật toán Heap Sort, ta cần biết được cấu trúc Heap (hay Binary Heap) là như thế nào? Cấu trúc Heap là một cây nhị phân hoàn chỉnh, nó có thể dễ dàng biểu diễn dưới dạng mảng, với mỗi nút cha I sẽ có 2 nút con là $2 * I + 1$ và $2 * I + 2$ (ví dụ mảng bắt đầu từ 0), nút cha có thể lớn hơn (hay nhỏ hơn) các nút con tùy vào thứ tự dữ liệu ta muốn sắp xếp.

- maxHeap: nút cha sẽ lớn hơn 2 nút con
- minHeap: nút cha sẽ nhỏ hơn 2 nút con

Thuật toán Heap Sort được xây dựng dựa trên việc tổ chức mảng có dạng cấu trúc Heap

2.6.2 Các bước thực hiện

Để sắp xếp mảng theo thứ tự tăng dần, ta sử dụng maxHeap

- Bước 1: Xây dựng một hàm tối ưu hóa dữ liệu đầu vào tạo nên một Heap (max-Heap).
- Bước 2: Gán $left = \frac{n-1}{2}$, $right = n - 1$
  Xây dựng cấu trúc Heap cho mảng ban đầu bằng cách duyệt các đỉnh trong từ vị trí $left$ đến $right$ và giảm $left$ cho đến khi $left = 0$
- Bước 3: Sau khi đưa mảng về cấu trúc Heap ở Bước 2, giá trị lớn nhất sẽ nằm đầu mảng. Thực hiện hoán đổi giá trị giữa phần tử đầu là phần tử ở vị . $right$ (giá trị lớn nhất sẽ nằm phía sau), Giảm $right$, xây dựng lại cấu trúc Heap từ vị trí 0 đến $right$. .
- Bước 4: Lặp lại Bước 3 cho đến khi $right = 1$. Mảng nhận được sẽ là mảng đã được sắp xếp theo thứ tự tăng dần.

2.6.3 Đánh giá thuật toán

Độ phức tạp về thời gian: $O(nlogn)$ cho mọi trường hợp. Giải thích: để xây dựng được cấu trúc maxHeap ta cần $O(logn)$ và để truyền phần tử phần tử hay hoán đổi cho việc sắp xếp cần $O(n)$.
Độ phức tạp về không gian: $O(1)$.
Tính ổn định của thuật toán: không ổn định.

2.7 Merge Sort

2.7.1 Ý tưởng thuật toán

Merge Sort là một thuật toán ứng dụng chia để trị. Bằng cách chia mảng thành 2 nữa, rồi lại tiếp tục chia thành 2 nữa cho đến khi mỗi mảng con chỉ có 1 phần tử, bắt đầu gộp từng 2 nữa đó lại theo đúng thứ tự tăng dần cho đến khi mảng ban đầu được sắp xếp.

2.7.2 Các bước thực hiện

- Bước 1: Sử dụng đệ quy để chia mảng ban đầu thành 2 mảng con, tiếp tục gọi để quy cho những mảng con được chia cho đến khi mỗi mảng con chỉ còn 1 phần tử.
- Bước 2: Sử dụng hàm gộp để gộp 2 mảng con thành một mảng lớn hơn theo thứ tự tăng dần giá trị các phần tử.
- Bước 3: Lặp lại Bước 1 và Bước 2 cho đến khi mảng ban đầu được sắp xếp.

2.7.3 Đánh giá thuật toán

Độ phức tạp về thời gian: $O(nlogn)$ trong cả 3 trường hợp vì Merge Sort luôn thực hiện chia mảng ban đầu thành 2 nữa và cần thời gian tuyến tính để có thể gộp các 2 nữa đó lại với nhau.
Độ phức tạp về không gian: $O(n)$, cần thêm bộ nhớ để có thể sao chép mảng con trong quá trình gộp mảng.
Tính ổn định của thuật toán: có sự ổn định.

2.8 Quick Sort

2.8.1 Ý tưởng thuật toán

Ứng dụng thuật toán chia để trị, bằng cách chọn phần tử chốt, ta đưa phần tử chốt về đúng vị trí khi mảng đúng thứ tư tăng dần, sao cho các phần tử nhỏ hơn nằm bên trái chốt, và ngược lại. Chia mảng thành 2 nữa (bên trái vị trí của chốt và bên phải vị trí chốt) và tiếp tục lặp lại công việc cho mỗi nữa.

2.8.2 Các bước thực hiện

- Bước 1: Chọn phần tử làm chốt.
- Bước 2: Dùng biến i để duyệt xuôi từ đầu mảng đến khi có phần tử lớn hơn phần tử chốt
- Bước 3: Dùng biến j để duyệt ngược từ cuối mảng đến khi có phần tử nhỏ hơn phần tử chốt.
- Bước 4: Nếu $i <= j$ thì ta hoán đối giá trị giữa chúng, Lặp lại Bước 2 và 3 cho đến khi $i > j$ thì j chính là phần tử chốt mới.
- Bước 5:
  - Nếu j chưa phải biên bên trái, ta thực hiện lại Bước 1 cho phần mảng con từ biên trái đến j
  - Nếu i chưa phải biên bên phải, ta thực hiện lại Bước 1 cho phần mảng con từ i đến biên trái

2.8.3 Đánh giá thuật toán

Độ phức tạp của thuật toán phụ thuộc vào việc chọn phần tử chốt. Có nhiều cách chọn phần tử chốt khác nhau: đầu mảng, cuối mảng, giữa mảng hay chọn ngẫu nhiên.
Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(nlogn)$
2. Trường hợp trung bình: $O(nlogn)$
3. Trường hợp xấu nhất: $O(n^2)$

Độ phức tạp về không gian: $O(logn)$
Tính ổn định của thuật toán: không ổn định.

2.9 Counting Sort

2.9.1 Ý tưởng thuật toán

Counting Sort là một thuật toán sắp xếp dữ liệu nguyên không âm mà không cần sử dụng đến so sánh. Thuật toán sẽ đếm số lượng xuất hiện của giá trị đó và thực hiện một số phép tính toán để có thể đưa giá trị đó vào đúng vị trí của mảng khi được sắp xếp

2.9.2 Các bước thực hiện

- Bước 1: Tìm số lần xuất hiện của từng phần tử trong mảng, lưu kết quả vào mảng **count**.
- Bước 2: Cộng dồn mảng **count** bằng cách **count[i] += count[i-1]** để giá trị của **count[i]** chính là chặn trên của phần tử i khi được sắp xếp.
- Bước 3: Đặt từng phần tử của mảng ban đầu vào mảng tạm **b** theo đúng chỉ số của mảng chứa giá trị đã sắp xếp dựa trên **count**.
- Bước 4: Mảng **b** là mảng đã được sắp xếp, có thể sao chép dữ liệu từ **b** sang mảng ban đầu để hủy vùng nhớ

2.9.3 Đánh giá thuật toán

Với k là giới hạn của dữ liệu đầu vào. Vì thuật toán thực hiện phép đếm số lượng và đưa vào mảng có chỉ số chính là giá trị phần tử nên độ phức tạp của thuật toán còn phụ thuộc vào giới hạn lớn nhất dữ liệu đầu vào.
Độ phức tạp về thời gian: $O(n+k)$ Độ phức tạp về không gian: $O(n+k)$

2.9.4 Cải thiện thuật toán

Thay vì phải khởi tạo mảng **count** đúng bằng giá trị lớn nhất của mảng. Ta có thể khởi tạo **count** có số lượng phần tử là $(max-min+1)$,vậy thì chỉ số của phần tử mảng đầu vào trong **count** chính là **a[i]-min**. Điều này sẽ làm giảm không gian lưu trữ, tối ưu hóa thuật toán.
Tính ổn định của thuật toán: có sự ổn định.

2.10 Radix Sort

2.10.1 Ý tưởng thuật toán

Radix Sort là một thuật toán sắp xếp theo cơ số không so sánh, mà dựa theo nguyên tắc phân loại thư. Các bức thư đến cùng một vùng nào đó sẽ được sắp chung với nhau để có thể gửi đi một cách thuận tiện nhất. Cứ tiếp tục như vậy thư sẽ đến tay người dùng sẽ theo một hệ thống phân loại nhất định. Thuật toán Radix Sort được chia thành 2 loại:

- LSD (Least significant digit): sắp xếp bắt đầu từ hàng nhỏ nhất (hàng đơn vị).
- MSD (Most significant digit): sắp xếp bắt đầu từ hàng lớn nhất.

***Note: Trong bài này và file thực thi đính kèm chỉ nhắc đến thuật toán Radix Sort theo LSD**

2.10.2 Các bước thực hiện

- Bước 1: Tìm phân tử lớn nhất của mảng, để xác định k là số lượng hàng nên duyệt (*0: hàng đơn vị, 1: hàng trăm,...*)
- Bước 2: Khởi tạo mảng **count** gồm 10 phần tử để lưu giá trị của hàng đó ($0- > 9$)
- Bước 3: Bắt đầu từ hàng đơn vị,đếm số lượng của từng giá trị xuất hiện ở hàng đơn vị của các phần tử mảng ban đầu.
- Bước 4: Cộng dồn mảng **count** sao cho **count[i]** chính là chặn trên của số lượng phần tử có tận cùng là i.
- Bước 5: Đặt từng phần tử mảng ban đầu vào mảng tạm **b** dựa chỉ số hàng đơn vị..
- Bước 6: Thực hiện gán lại từng phần tử của mảng **b** vào lại mảng ban đầu.
- Bước 7; Nếu sô hàng đã duyệt nhỏ hơn k, thực hiện lại Bước 2.

2.10.3 Đánh giá thuật toán

Gọi k là số lượng chữ số của phần tử có giá trị lớn nhất trong mảng ban đầu nên số lần chia nhóm và gộp lại là k. Với mỗi thao tác chia và gộp, mỗi phần tử chỉ được xét đúng một lần vì vậy độ phức tạp về thời gian là: $O(2kn) = O(n)$
Độ phức tạp về không gian: $O(k + n)$
Tính ổn định của thuật toán: có sự ổn định

2.11 Flash Sort

2.11.1 Ý tưởng thuật toán

Chia dữ liệu của mảng nhập vào thành m phân lớp khác nhau, mỗi phân lớp sẽ chưa một số lượng phần tử nhất định,sao cho giá trị nhỏ nhất của phân lớp đứng sau lớn hơn giá trị lớn nhất của phân lớp trước. Bây giờ chúng ta chỉ cần sắp xếp các phần tử trong từng phân lớp là có thể hoàn thành thuật toán. Thuật toán chia thành 3 giai đoạn:

- Giai đoạn 1: Phân lớp dữ liệu, xác định số phân lớp và số lượng phần tử mỗi phân lớp.
- Giai đoạn 2: Hoán vị toán cục, đưa các phần tử về đúng vị trí phân lớp.
- Giai đoạn 3: Sắp xếp toán cục, sử dụng thuật toán Insertion Sort để sắp xếp dữ liệu trong từng phân lớp.

2.11.2 Các bước thực hiện

Mảng nhập vào có n phần tử. Thuật toán Flash Sort được chia thành 3 giai đoạn:

- Giai đoạn 1: Phân lớp dữ liệu.
  - Bước 1: Xác định số phân lớp: **m=0.43n** và tạo mảng chưa số lượng từng phân lớp **L**
  - Bước 2: Xác định giá trị nhỏ nhất (minVal) và vị trí chứa phần tử lớn nhất của mảng (max). Nếu giá trị lớn nhất và nhỏ nhất trùng nhau, ta không cần thực hiện nữa.
  - Bước 3: Đếm số lượng phần tử của mỗi phân lớp theo công thức:

$$\frac{(a[i] - minVal) * (m-1)}{a[max] - minVal}$$

  - Bước 4: Tính vị trí kết thúc của từng phân lớp bằng cách cộng dồn mảng **L**
- Giai đoạn 2: Hóan vị toàn cục
  - Bước 1: Hoán đổi vị trí phần tử đầu tiên và phần tử lớn nhất, vì ta sẽ bắt đầu hoán vị từ phần tử đầu tiên.
  - Bước 2: Duyệt từng phần tử, nếu phần tử đó chưa nằm đúng phân lớp ($i < L[k] - 1$), ta sẽ tính phân lớp.
  - Bước 3: Cần đưa phần tử đang xét về đúng phân lớp, việc này sẽ kèo theo cả 1 chu trình hoán vị các phần tử khác cũng chưa nằm đúng phân lớp. Ta thực hiện vòng lặp để hoán vị. khi một phần tử đặt đúng phân lớp, số lương phần tử của phân lớp lưu trong mảng **L** sẽ giảm đi 1.
  - Bước 4: Lặp lại Bước 2 cho đến khi tất cả các phần đều nằm đúng phân lớp.
- Giai đoạn 3: Sắp xếp toán cục. Vì việc di chuyển các phẩn tử là không lớn nên ta sử dụng thuật toán Binary Insertion Sort để thực hiện việc sắp xếp.

2.11.3 Đánh giá thuật toán

Độ phức tạp về thời gian:

1. Trường hợp tốt nhất: $O(n)$
2. Trường hợp xấu nhất: $O(n^2)$

Độ phức tạp về không gian: $O(m)$
Tính ổn định của thuật toán: không ổn định.

3 Kết quả thử nghiệm các thuật toán

3.1 Thống kê thời gian và số phép so sánh

Các thuật toán sắp xếp: Selection Sort, Insertion Sort, Bubble Sort, Shaker Sort, Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, và Flash Sort.

Các kiểu dữ liệu: mảng ngẫu nhiên, mảng tăng dần, mảng giảm dần, mảng gần như tăng dần.

Các kích thước dữ liệu: 10.000, 30.000, 50.000, 100.000, 300.000, 500.000 phần tử.

Dưới đây là bốn bảng thông kế thời gian chạy (milliseconds) và số phép so sánh của 11 thuật toán sắp xếp, theo 4 kiểu dữ liệu và 6 kích thước dữ liệu khác nhau. Chương trình các thuật toán sẽ sắp xếp dữ liệu theo thứ tự **tăng dần**.

Data order: Randomized												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision
Selection Sort	150.628	100009999	1212.500	900029999	3352.710	2500049999	13462.700	10000099999	122261.000	90000299999	332438.000	250000499999
Insertion Sort	70.471	25375884	644.255	226506234	1777.420	626339103	7128.210	2503124764	63064.100	22535192165	175565.000	62475433512
Bubble Sort	402.119	99978554	3866.990	899513421	10875.300	2499861162	45408.700	9999197877	394796.000	89998659495	1093110.000	249992700166
Shaker Sort	303.559	66265080	2844.620	602447215	8046.900	1665537231	32590.600	6672386309	288081.000	60006840949	860469.000	166783655609
Shell Sort	2.146	660114	7.629	2214931	14.159	4595184	31.556	10148510	109.999	35106607	241.936	65181473
Heap Sort	1.762	497608	6.376	1680945	10.496	2952515	22.756	6304725	75.303	20798382	164.207	36118402
Merge Sort	2.071	583444	7.311	1937488	11.575	3383419	24.342	7166626	76.822	23382508	160.491	40382373
Quick Sort	1.327	284288	4.184	929728	7.500	1636356	15.450	3433899	50.729	10783196	92.152	18641275
Counting Sort	0.282	79998	1.017	240000	1.675	400002	3.483	800002	12.348	2399998	41.031	40382373
Radix Sort	1.095	140056	4.050	510070	6.629	850070	13.526	1700070	49.120	6000084	98.747	10000084
Flash Sort	1.026	346184	3.526	1115704	6.013	1933535	13.281	4056561	44.612	13150280	122.616	22592419

Bảng 3.1: Thời gian và số phép so sánh của các thuật toán theo dữ liệu ngẫu nhiên

Data order: Sorted												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision
Selection Sort	148.966	100009999	1410.180	900029999	4394.730	2500049999	16626.500	10000099999	148017.000	90000299999	339318.000	250000499999
Insertion Sort	0.699	277232	2.464	924464	4.560	1618928	10.337	3437856	31.431	11251424	58.329	19451424
Bubble Sort	0.035	20000	0.102	60000	4.560	100000	0.339	200000	0.951	600000	1.705	1000000
Shaker Sort	0.033	20001	0.101	60001	0.219	100001	0.325	200001	0.981	600001	2.049	1000001
Shell Sort	0.589	360042	1.972	1170050	5.599	2100049	7.774	4500051	24.814	15300061	49.117	25500058
Heap Sort	1.460	518705	4.889	1739633	10.153	3056481	18.819	6519813	51.446	21431637	96.077	37116275
Merge Sort	1.332	475242	5.691	1559914	9.587	2722826	17.035	5745658	57.228	18645946	85.937	32017850
Quick Sort	0.355	149055	1.147	485546	2.541	881083	5.203	1862156	13.330	5889300	23.696	10048590
Counting Sort	0.227	80002	0.691	240002	1.293	400002	2.304	800002	8.843	2400002	15.698	4000002
Radix Sort	1.111	140056	4.444	510070	9.373	850070	13.977	1700070	49.758	6000084	82.839	10000084
Flash Sort	1.093	368543	3.778	1198677	6.569	2074867	13.046	4349743	40.505	13992631	65.900	24023985

Bảng 3.2: Thời gian và số phép so sánh của các thuật toán theo dữ liệu tăng dần

Data order: Reverse sorted												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision
Selection Sort	135.003	100009999	1206.090	900029999	3356.220	2500049999	13395.300	10000099999	120926.000	90000299999	338388.000	250000499999
Insertion Sort	153.480	50252260	1268.390	450849494	3517.620	1251493960	14063.900	5003187890	126230.000	45010501462	355431.000	125018201462
Bubble Sort	427.530	100039997	3864.890	900119997	10797.100	2500199997	42837.000	10000399997	385787.000	90001199997	1083300.000	250001999997
Shaker Sort	418.787	100005000	3774.990	900015000	10541.600	2500025000	42121.400	10000050000	379503.000	90000150000	1065350.000	250000250000
Shell Sort	0.801	475175	3.496	1554051	4.914	2844628	10.530	6089190	34.100	20001852	60.367	33857581
Heap Sort	1.324	476739	4.610	1622791	7.527	2848016	15.806	6087452	51.611	20187386	95.086	35135730
Merge Sort	1.265	476441	4.587	1573465	6.879	2733945	15.026	5767897	46.221	18708313	82.325	32336409
Quick Sort	0.345	159070	1.189	515556	2.084	931094	4.348	1962168	13.896	6189320	24.277	10548604
Counting Sort	0.217	80002	0.692	240002	1.161	400002	2.380	800002	7.520	2400002	16.445	4000002
Radix Sort	1.023	140056	4.145	510070	6.433	850070	12.927	1700070	46.723	6000084	85.173	10000084
Flash Sort	0.960	353391	3.400	1153075	6.249	1999403	12.405	4198805	38.111	13537031	66.446	23262709

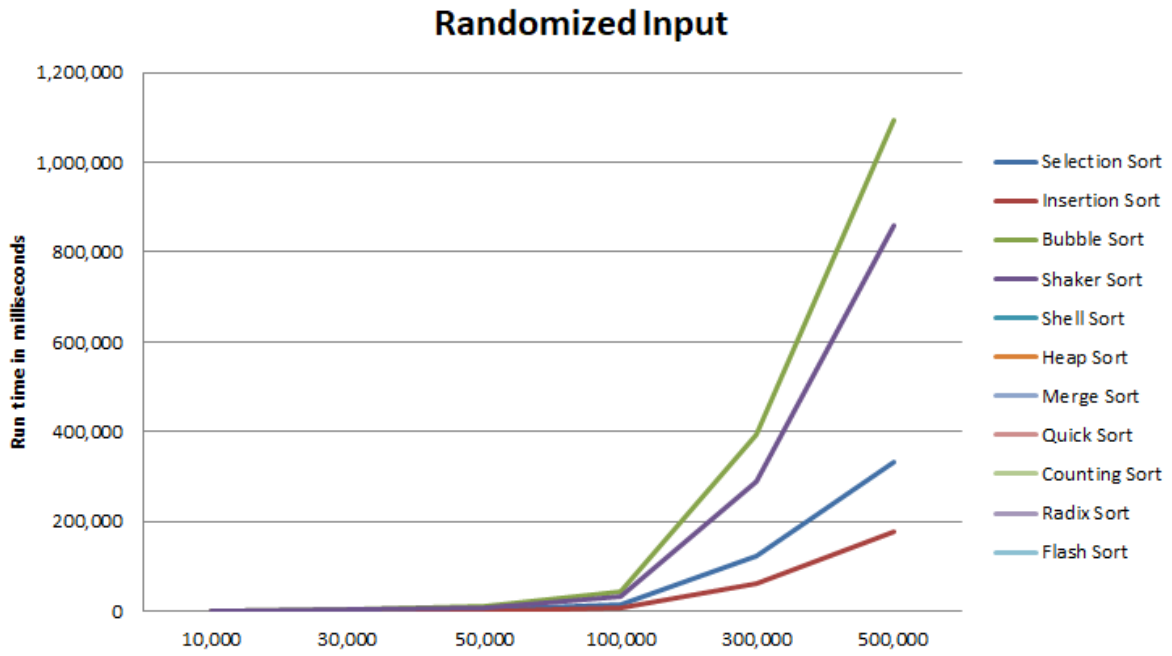
Bảng 3.3: Thời gian và số phép so sánh của các thuật toán theo dữ liệu giảm dần

Data order: Nearly sorted												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision	Running Time	Comparision
Selection Sort	129.041	100009999	1215.960	900029999	3375.100	2500049999	13542.600	10000099999	121407.000	90000299999	336734.000	250000499999
Insertion Sort	0.912	350512	2.846	1154808	5.058	1943216	10.908	4058260	37.140	12987350	61.095	21802102
Bubble Sort	122.516	85711491	1166.240	804081047	3031.240	1864317065	14219.900	9189777090	117986.000	74171734296	357735.000	191978858524
Shaker Sort	0.793	216043	2.328	594171	2.375	816867	5.980	2176214	14.103	4778894	29.234	10989758
Shell Sort	0.896	411894	2.969	1341958	4.489	2377334	10.835	5154308	30.438	16773712	50.211	28209788
Heap Sort	1.387	518442	4.973	1739553	7.738	3056606	16.884	6519627	54.778	21431159	91.185	37110448
Merge Sort	1.271	509808	4.054	1656263	7.030	2863392	14.780	6076154	48.053	19565354	81.539	33384140
Quick Sort	0.330	149091	1.142	485578	2.015	881123	4.264	1862212	13.474	5889793	22.829	10048634
Counting Sort	0.204	80002	0.670	240002	1.037	400002	2.100	800002	7.535	2400002	15.445	4000002
Radix Sort	1.095	140056	4.102	510070	7.575	850070	13.330	1700070	49.588	6000084	85.297	10000084
Flash Sort	2.051	368529	3.200	1198656	5.562	2074851	11.814	4349720	38.549	13992606	70.011	24023963

Bảng 3.4: Thời gian và số phép so sánh của các thuật toán theo dữ liệu gần tăng dần

3.2 Biểu đồ minh họa thời gian sắp xếp

3.2.1 Kiểu dữ liệu ngẫu nhiên

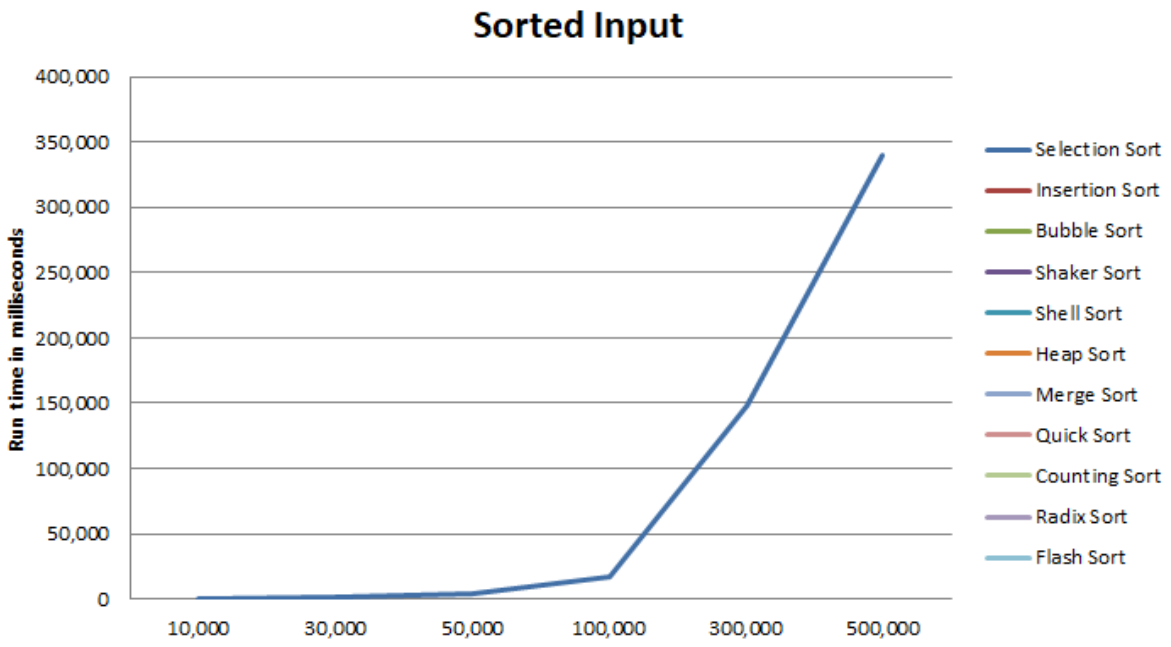


Hình 3.5: Thời gian sắp xếp của các thuật toán theo dữ liệu ngẫu nhiên

Qua biểu đồ so sánh thời gian chạy của từng thuật toán đối với dữ liệu được cho ngẫu nhiên:

- Thuật toán Counting Sort là nhanh nhất, với kích thước dữ liệu 500,000 chỉ cần 41ms để sắp xếp xong.
- Thuật toán Bubble Sort là chậm nhất,kích thước dữ liệu 100,000 đã cần 402ms, còn với 500,000 cần tới gần 1,100,000ms tương đương 18 phút để có thể sắp xếp xong.
- Nhìn vào biểu đồ, ta có thể nhận thấy ngay các thuật toán Shell Sort, Heap Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, Flash Sort có tốc độ vượt trội hơn hẳn so với các thuật toán còn lại.

3.2.2 Kiểu dữ liệu sắp xếp tăng dần

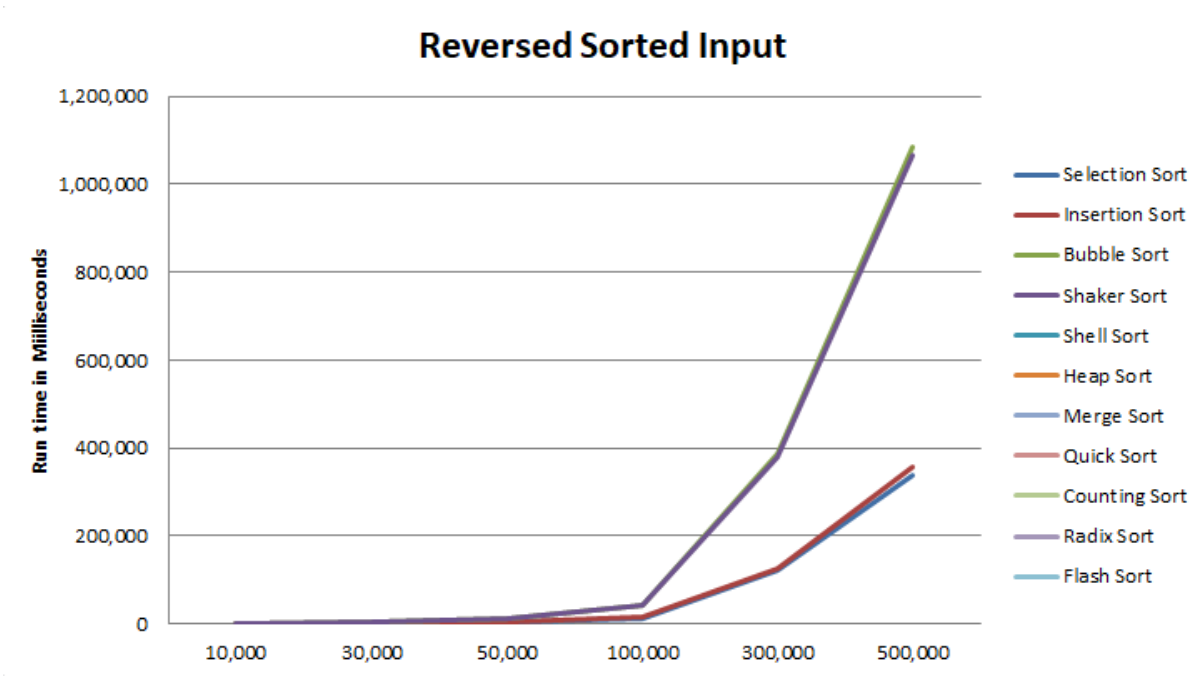


Hình 3.6: Thời gian sắp xếp của các thuật toán theo dữ liệu tăng dần

Với dữ liệu đầu vào được sắp xếp sẵn thứ tự tăng dần, Một vài thuật toán có thể nhận ra điều đó nhưng vẫn có những thuật toán vẫn cần cù xử lí cho tới phần tử cuối cùng:

- Thuật toán sắp xếp nhanh nhất là Bubble, ngay sau đó chính là Shaker Sort, kích thước dữ liệu 500,000 chỉ cần 2ms để hoàn thành, việc này chính là nhờ sự cải tiến nhận ra vị trí thay đổi cuối cùng từ đó rút ngắn thời gian thực hiện đi đáng kể.
- Thuật toán sắp xếp chậm nhất là Selection Sort, thời gian xử lí vẫn tương đương với kiểu dữ liệu ngẫu nhiên.
- Các thuật toán còn lại đều có tốc độ xử lí nhanh như nhau, chỉ vài chục ms với kích thước dữ liệ 500,000 phần tử.

3.2.3 Kiểu dữ liệu sắp xếp ngược

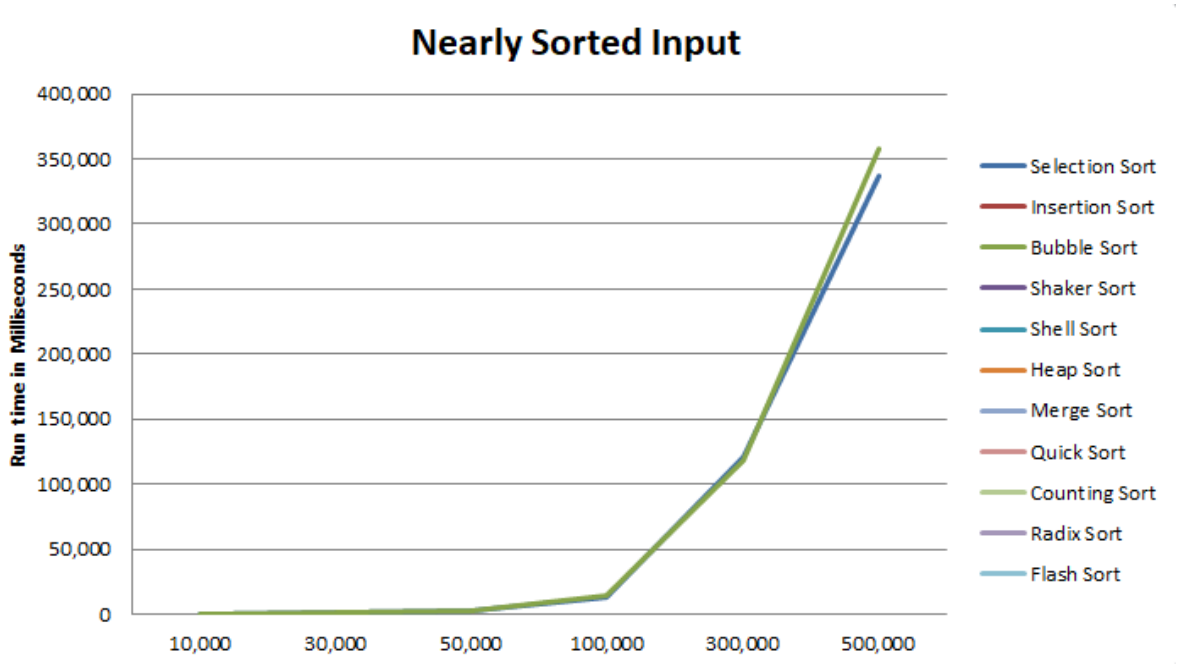


Hình 3.7: Thời gian sắp xếp của các thuật toán theo dữ liệu giảm dần

Với kiểu dữ liệu được sắp xếp giảm dần, ngược lại với thứ tự mong muốn, có thể rơi vào trường hợp xấu nhất của một vài thuật toán:

- Thuật toán sắp xếp nhanh nhất là Counting với 16ms với mảng 500,000 phần tử, so đó vẫn là Quick Sort với 24ms.
- Thuật toán sắp xếp chậm nhất Bubble Sort và Shaker Sort, khi bị đảo ngược chiều sắp xếp, việc thực hiện quá nhiều phép so sánh đã làm cho thuật toán bị chậm
- Thuật toán Insertion Sort nhờ vào thuật toán Binary Search đã giúp giảm lượng lớn số phép so sánh nhưng vẫn cần đến hơn 5 phút để sắp xếp mảng 500,000 phần tử.
- Các thuật toán Shell Sort, Heap Sort, Merge Sort, Radix Sort, Flash Sort vẫn có tốc độ xử lí rất nhanh, chỉ cần vài chục ms để xử lí xong kích thước dữ liệu lớn nhất

3.2.4 Kiểu dữ liệu gần như tăng dần



Hình 3.8: Thời gian sắp xếp của các thuật toán theo dữ liệu gần tăng dần

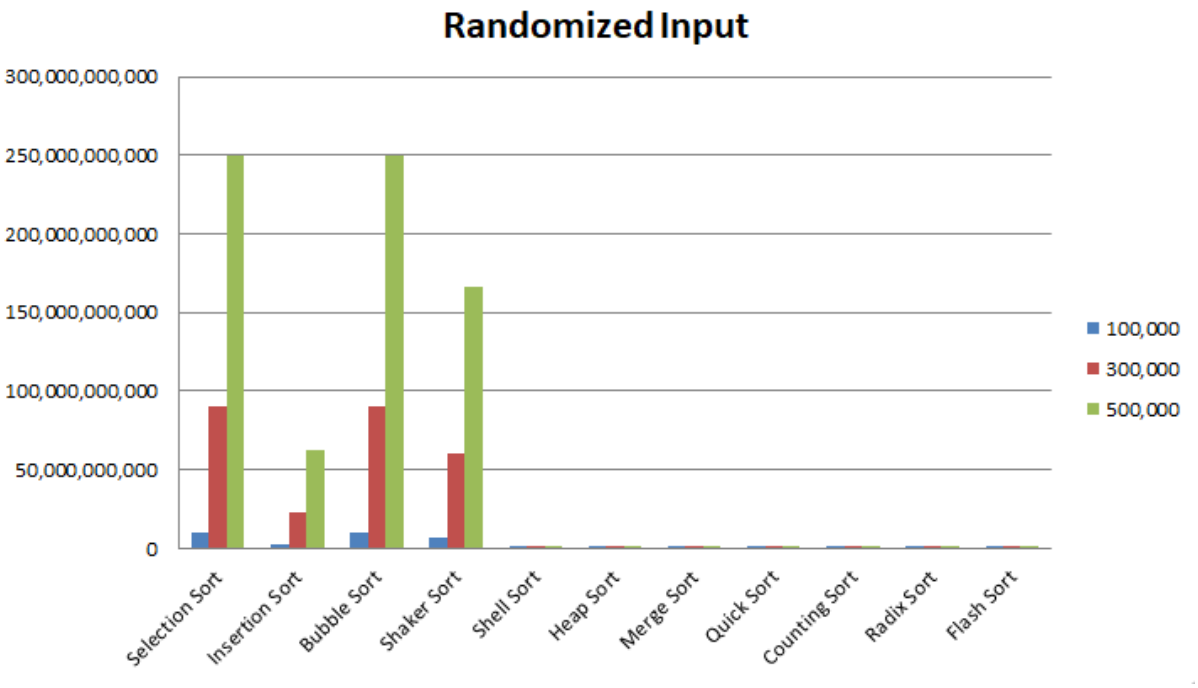
Đối với dữ liệu gần như sắp xếp, chỉ có 10 cặp số chưa ở đúng vị trí:

- Nhìn vào biểu đồ trên, gần như chỉ thấy 2 đường của Selection Sort và Bubble Sort. Việc sử dụng dụng vòng lặp để so sánh của Selection không thay đổi với bất kì kiểu dữ liệu nào. Bubble Sort phụ thuộc nhiều vào vị trí của những cặp số bị hoán đổi, bởi nếu 10 cặp số đó đều nằm ở những vị trí đầu tiên có thể làm thuật toán chạy rất nhanh.
- Thuật toán sắp xếp nhanh nhất là Counting Sort, sau đó vẫn là Quick Sort với lần lượt là 15ms và 22 ms với mảng 500,000 phần tử.
- Các thuật toán còn lại có thời xử lí gần như như nhau ở các kích thước dữ liệu, từ 30ms đến 90ms đối với kích thước dữ liệu lớn nhất

3.3 Biểu đồ minh họa phép so sánh

Để hạn chế việc quá nhiều dữ liệu, các biểu đồ dưới đây chỉ thể hiện số phép so sánh đối với 3 kích thước: 100000, 300000, 500000

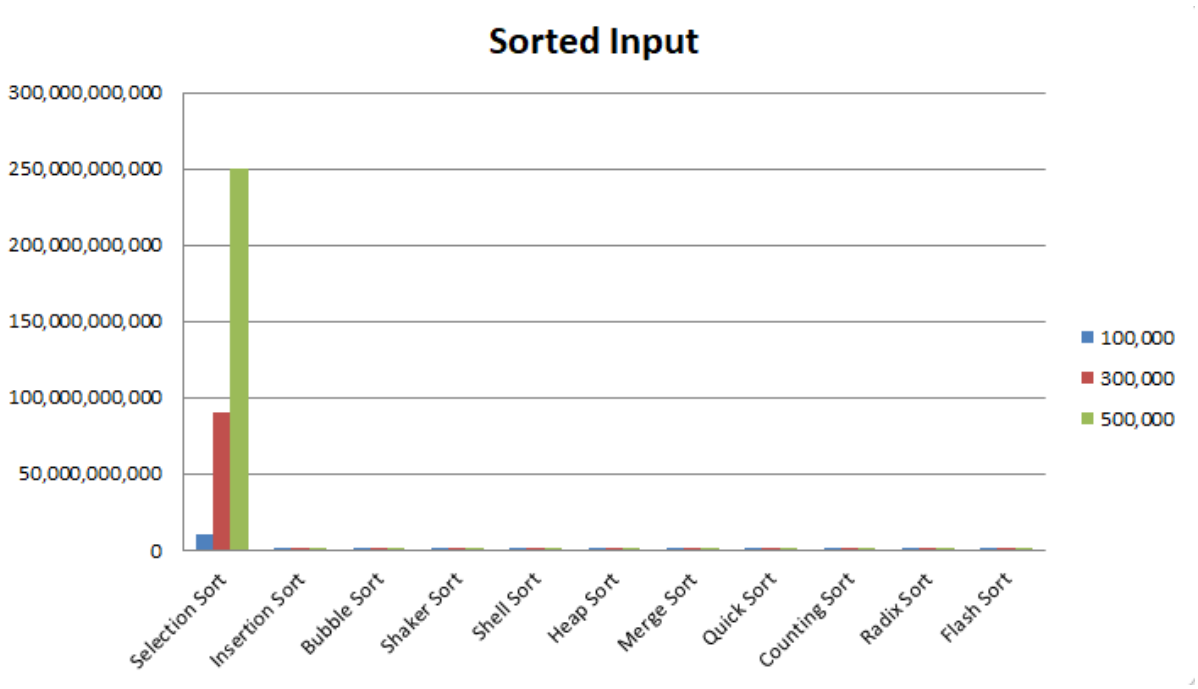
3.3.1 Kiểu dữ liệu ngẫu nhiên



Hình 3.9: Số phép so sánh của thuật toán theo dữ liệu ngẫu nhiên

- Nhìn vào biểu đồ gần như chỉ thấy cột so sánh của Selection Sort, Insertion Sort, Bubble Sort và Shaker Sort.
- Hai thuật toán Selection Sort, Bubble Sort có số phép so sánh nhiều nhất lên đến 250,000,000,000 phép so sánh, việc so sánh quá nhiều làm cho thuật toán bị chậm đi.
- Shaker Sort được giới thiệu như là một cái tiến của Bubble Sort, bằng việc sắp xếp cả 2 đầu mảng đã làm giảm đi đáng kể số phép so sánh.
- Bằng cách xử dụng thuật toán tìm kiếm nhị phân,Insertion Sort chỉ có số lượng bằng 1/3 so với 3 thuật toán trên.
- Các thuật toán còn lại đều có số phép so sánh nhỏ hơn 100,000,000, trong đó Quick Sort là có số phép so sánh ít nhất.

3.3.2 Kiểu dữ liệu sắp xếp tăng dần

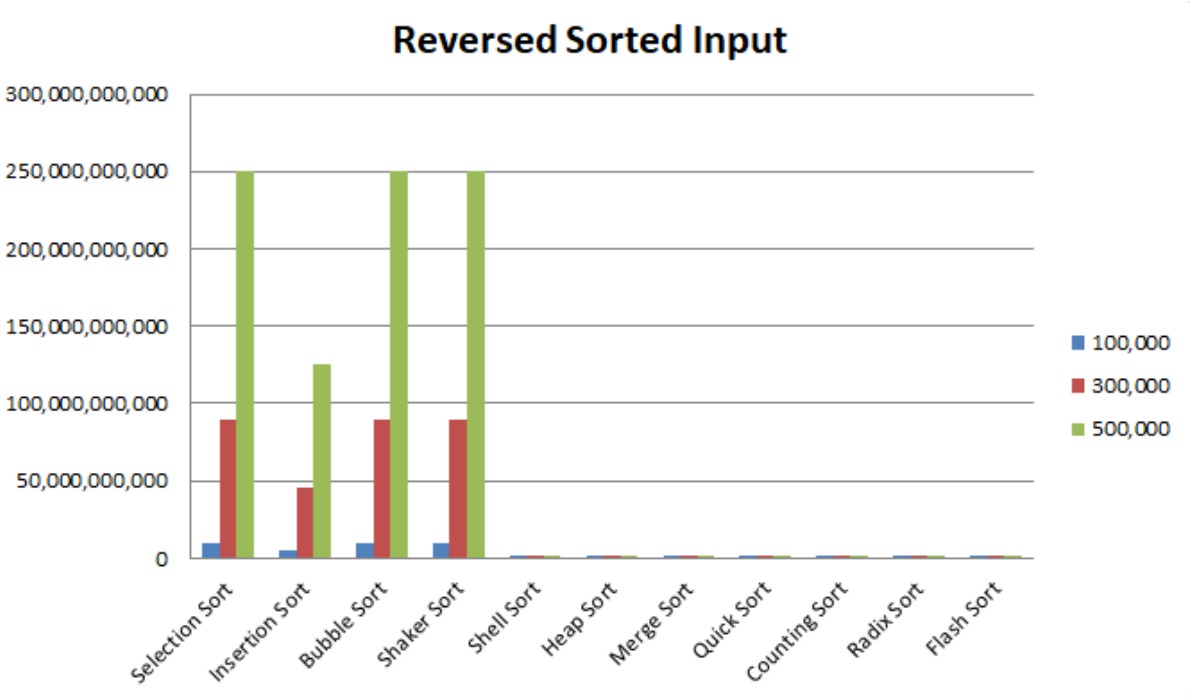


Hình 3.10: Số phép so sánh của thuật toán theo dữ liệu được sắp tăng dần

Đối với dữ liệu đã được sắp xếp sẵn:

- Thuật toán có nhiều phép so sánh nhất là Selection Sort, thuật toán luôn có độ phức tạp là $O(n^2)$ này luôn giữ vững số phép so sánh cho từng kiểu dữ liệu khác nhau.
- Bubble Sort và Shaker Sort khi có sử dụng biến để lưu giá trị hoán vị cuối có thể nhận ra rằng mảng đã được sắp nên chỉ cần 1,000,000 phép so sánh
- Việc dữ liệu được sắp sẵn làm cho Insertion Sort hạn chế được việc phải chèn phần tử.
- Ngoài 2 thuật toán có cách nhận biết mảng đã được sắp, Counting Sort có số phép so sánh nhỏ hơn hẳn các thuật toán còn lại chỉ 4,000,000 phép so sánh .

3.3.3 Kiểu dữ liệu sắp xếp ngược

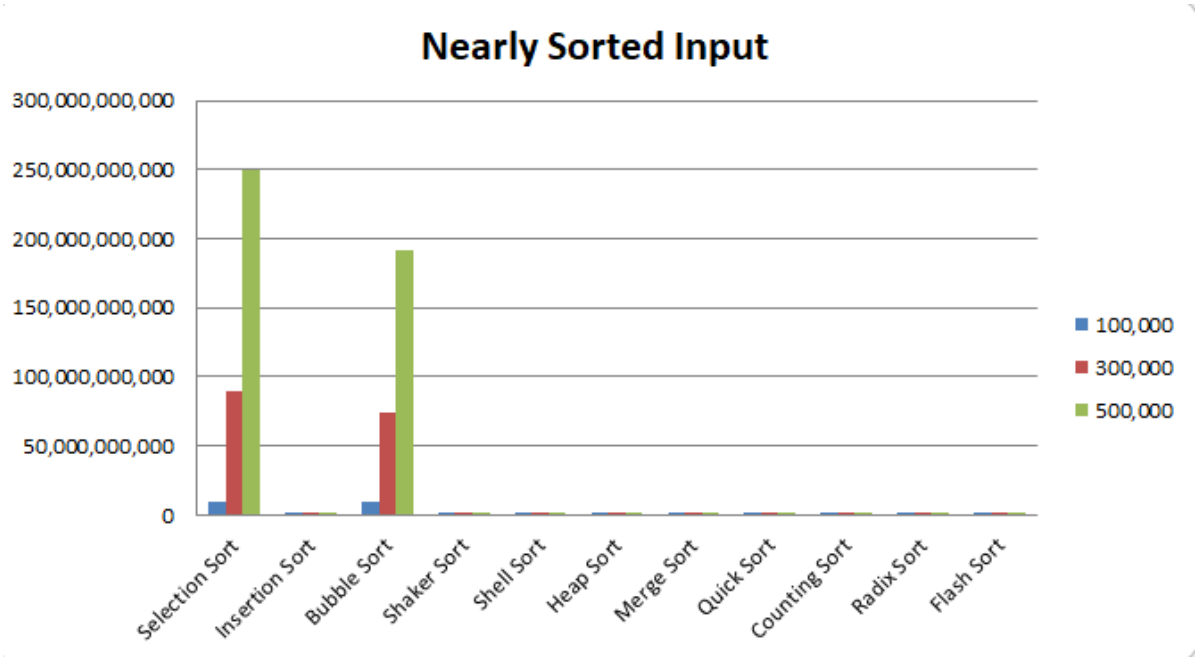


Hình 3.11: Số phép so sánh của thuật toán theo dữ liệu được sắp giảm dần

Đối với việc dữ liệu của mảng bị sắp xếp ngược với thứ tự mong muốn:

- Selection Sort vẫn giữ nguyên cho mình số phép so sánh như các kiểu dữ liệu khác
- Bubble Sort và Shaker Sort đều bị trường hợp xấu nhất, nên việc cải tiến dường như không đem lại hiệu quả ở trường hợp này, cả 2 đều có số phép so sánh nhiều nhất.
- Insertion Sort có số phép so sáng bằng 1/2 các thuật toán trên, vì phải tìm và chèn phẩn tử luôn rơi vào trường hợp đầu mảng.
- Counting Sort có số phép so sánh ít nhất, khi việc sắp xếp chỉ dựa trên việc đếm số lượng.

3.3.4 Kiểu dữ liệu gần như tăng dần



Hình 3.12: Số phép so sánh của thuật toán theo dữ liệu được sắp gần tăng dần

Chỉ cần hoán đổi 10 phần tử, số phép so sánh đã giảm đi rất nhiều:

- Selection Sort có số phép so sánh nhiều nhất, bởi vị thuật toán cố định việc sử dụng vòng lặp.
- Bubble Sort có số phép so sánh tùy vào vị trí các phần tử bị hoán đổi, Shaker Sort không cần đến quá nhiều phép so sánh để đưa 20 phần tử về đúng vị trí.
- Counting Sort có số phép so sánh nhỏ nhất, Radix Sort có số phép so sánh ít thứ 2. Bởi vì việc so sánh của 2 thuật toán chỉ đơn giản để đếm số lượng phần tử theo mục đích nhất định, còn quá trình sắp xếp thì chỉ dựa vào số lượng đã đếm được.

3.4 Đánh giá chung về các thuật toán

Thông qua 4 kiểu dữ liệu và 6 kích thước dữ liệu được nếu ra ở trên cùng những biểu đồ minh họa cho từng trường hợp khác nhau. Ta nhận thấy rắng:

- Thuật toán Selection Sort tuy đơn giản, chương trình ngắn gọn dễ hiểu nhưng độ phức tạp cao (luôn là $O(n^2)$), số lượng phép so sánh luôn rất lớn và giữ nguyên đối với mỗi kích thước dữ liệu dù là kiểu dữ liệu nào. Chỉ phù hợp với kích thước dữ liệu nhỏ để có thể đem lại hiệu quả. Đây còn là một thuật toán không có tính ổn định.

- Thuật toán Insertion Sort có ý tưởng đơn giản, dễ thực hiện thuật toán, với sự cái tiến kết hợp cùng Binary Search để có thể tối ưu thuật toán. Đối với dữ liệu bị sắp xếp ngược thứ tự mong muốn, thuật toán sẽ rơi vào trường hợp xấu nhất. Và ngược lại khi mảng đã được sắp sẵn, thuật toán chỉ đơn giản duyệt hết các phần tử mà không thực hiện thao tác chèn. Đây là một thuật toán có tính ổn định.

- Thuật toán Bubble Sort được cải tiến bằng cách lưu lại vị trí sắp xếp cuối đã đem lại hiệu quả rất lớn đặc biệt trong trường hợp mảng đã được sắp sẵn thứ tự mong muốn khi chỉ tốn $O(n)$ cho một lần duyệt.Nhưng đối với những trường hợp khác, việc so sánh quá nhiều làm thuật toán bị chậm đi, không thực sự đem lại hiệu quả. Đây là thuật toán có tính ổn định.

- Thuật toán Shaker Sort được biết đến như một cải tiến giúp tối ưu hóa việc "nổi bọt" và "chìm" của các phần tử. Giảm bớt số lần so sánh nhưng khi rới vào trường hợp mảng bị sắp ngược thứ tự, thuật toán không thực sự phát huy được hiểu quá. Nhưng đối với việc mảng khi có một thứ tự gần đúng, thuật toán đã cho ta thấy được những lợi ích mà nó đem lại. Đây là thuật toán ổn định.

Bốn thuật toán vừa nêu ở trên có phần ý tưởng đơn giản, dễ thực hiện, chương trình ngắn gọn, nhưng chỉ thực sự hiệu quả với nhưng trường hợp nhất định hoặc kích thước dữ liệu không quá lớn.

- Thuật toán Shell Sort là một thuật toán cải tiến từ Insertion Sort khi việc chèn phần tử đến vị trí đúng quá xa. Đem lại hiểu quả cao cho việc sắp xếp. Thuật toán có độ phức tạp về thời gian và số phép so sánh hầu như không đổi qua các kiểu dữ liệu, cũng không cần thêm không gian bộ nhớ. Nhưng đây là thuật toán không có tính ổn định.

- Thuật toán Heap Sort được thực hiện thông qua một cấu trúc được gọi là Heap. Đây là một sự sáng tạo khi ứng dụng cây nhị phân để tổ chức lại dữ liệu trong việc sắp xếp mà không cần thêm nhiều không gian bộ nhớ đêm lại sự hiểu quả cho việc sắp xếp. Có độ phức tạp về thời gian là $O(nlogn)$ trong các trường hợp nhưng nhận thấy rằng khi mảng được sắp xếp sẵn một thứ tự nào đó thuật toán sẽ nhanh hơn so với mảng ngẫu nhiên. Đây là một thuật toán không có tính ổn định.

- Thuật toán Merge Sort sử dụng phương pháp chia để trị và được sắp xếp nhờ việc gộp các mảng con theo thứ tự để tạo ra mảng hoàn chỉnh. Giống như Heap Sort, thuật toán cần nhiều thời gian hơn để có thể sắp xếp một mảng được cho ngẫu nhiên. Đồng thời thuật toán cần thêm nhiều không gian bộ nhớ $O(n)$ để lưu giá trị trong giai đoạn gộp. Đây là một thuật toán có tính ổn định.

- Thuật toán Quick Sort được xem là một thuật toán có tính phổ biến cao. Giống như Merge Sort thì Quick Sort cũng sử dụng phương pháp chia để trị nhưng ở đây thuật toán chọn phần tử chốt để cho thể phân chia ra các mảng con. Thuật toán rơi vào trường hợp xấu nhất khi mảng bị sắp xếp một cách ngẫu nhiên, và chạy nhanh khi mảng đã có sẵn thứ tự, đồng thời việc chọn phần tử chốt cũng đóng vai trò quan trọng trong độ phúc tạo của thuật toán. Đây là thuật toán không có tính ổn định.

- Thuật toán Counting Sort được biết đến là một thuật toán sử dụng phương pháp đếm để sắp xếp chứ không phải so sánh. Nhưng thuật toán không thể áp dụng khi mảng có số âm. Thuật toán chạy nhanh nhất khi mảng đã được sắp, các trường hợp còn lại vẫn cho ra kết quả cực kì nhanh và số phép so sánh hầu như không thay đổi.Nhưng trong thực thế thì độ phúc tạp phụ thuộc nhiều vào giá trị min, max của mảng ban đầu, việc phải sử dụng nhiều bộ nhớ và chỉ áp dụng cho số nguyên dương làm cho thuật toán chưa thật sự hiệu quả. Đây là thuật toán có tính ổn định

- Thuật toán Radix Sort giống như Counting Sort chỉ sắp xếp dựa vào đếm số lượng, nhưng lại là số lượng phần tử có giá trị nhất định tại một hàng nào đó. Đối với các kích thước dữ liệu nhỏ hơn hoặc bằng 300,000, thuật toán chạy nhanh nhất đối với kiểu dữ liệu bị sắp xếp ngược. Nhưng với dữ liệu 500,000 phần tử, mảng đã được sắp xếp lại cho ra kết quả nhanh nhất. Số phép so sánh của thuật toán được giữ cố định theo từng kích thước dữ liệu. Đây là một thuật toán có tính ổn định.

- Thuật toán Flash Sort là sự kết hợp nhưng điểm chung của các phương pháp sắp xếp. Bằng cách phân vùng một cách sáng tạo, thực hiện hoán vị theo chu trình và kết hợp cùng Insertion Sort để tạo ra một thuật toán có độ phức tạp về thời gian đạt đến $O(n)$. Trong 4 kiểu dữ liệu, Flash Sort chạy nhanh nhất khi mảng đã được sắp sẵn và chậm nhất khi mảng cho ngẫu nhiên, số lượng phép so sánh hầu như không thay đổi qua các kiểu dữ liệu. Đây là một thuật toán không có tính ổn định.

Trên đây là một vài nhận xét về các thuật toán sắp xếp nhưng ưu nhược điểm và trường hợp nên sử dụng.

4 Tài liệu tham khảo

- Sự hướng dẫn của thầy **Nguyễn Thanh Phương** và thầy **Bùi Huy Thông**
- Tham khảo chương trình tại
  - Thuật toán Shell Sort
  - Thuật toán Merge Sort
  - Thuật toán Radix Sort
  - Thuật toán Flash Sort
- Tham khảo thông tin các thuật toán:
  - Thông tin các thuật toán