ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP

MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Tên sinh viên : Lê Hoàng Anh
Mã số sinh viên : 19127329
Lớp : 19CLC2

THÀNH PHỐ HỒ CHÍ MINH – Tháng 11, 2020.

A. THUẬT TOÁN:

I. Selection Sort (Sắp xếp chọn):

1. **Ý tưởng:** Ta thấy rằng khi một dãy a_i, …, a_n đã được sắp xếp thì phần tử a_i luôn là phần tử nhỏ nhất trong dãy a_i, …, a_n. Nên ta có được ý tưởng để sắp xếp một dãy theo phương pháp chọn: tìm vị trí của phần tử nhỏ nhất trong n phần tử ban đầu và đưa giá trị của phần tử đó về đầu dãy. Tiếp tục với n - 1 phần tử còn lại. Và đến khi còn lại một phần tử thì phần dừng và phần tử đó chắc chắn là số lớn nhất. Cuối cùng, dãy đã được sắp xếp theo thứ tự tăng dần.

2. **Thuật toán:** Cho mảng a gồm n phần tử a_1, a_2, …, a_n.
   - Bước 1: Chọn i = 1.
   - Bước 2: Tìm phần tử a_{min} nhỏ nhất trong mảng a_{i+1}, …, a_n.
   - Bước 3: Hoán vị hai phần tử a_{min} và a_i và tăng giá trị i lên thêm 1 (i = i + 1).
   - Bước 4:
     - Nếu $i \leq n - 1$ ➔ Lặp lại bước 2.
     - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: $O(n^2)$.
     - Trường hợp trung bình: $O(n^2)$.
     - Trường hợp tệ nhất: $O(n^2)$.
   - Độ phức tạp về không gian: O(1).

II. Insertion Sort (Sắp xếp chèn):

1. **Ý tưởng:** Đối với một dãy đã được sắp xếp, khi ta chèn một phần tử vào dãy đó ở một vị trí thích hợp thì dãy vẫn giữ nguyên được thứ tự ban đầu.
   - Cho dãy a gồm n phần tử a_1, a_2, …, a_n. Chia dãy ban đầu thành 2 dãy con a_1, a_2, …, a_i (gọi là dãy B) và a_{i+1}, a_{i+2}, …, a_n (gọi là dãy C). Giả sử dãy a_1, a_2, …, a_i đã có thứ tự (tăng dần). Chọn một phần tử từ dãy C, để tiện ta chọn a_{i+1} (tương đương C_0) và chèn vào dãy B sao cho nó vẫn giữ nguyên thứ tự (tăng dần).
   - Tìm vị trí k trong đoạn [1, i] thoả $a_{k-1} \leq a_{i+1} < a_k$, có 2 trường hợp xảy ra:
     - Nếu tồn tại k: dời tất cả phần tử trong đoạn [k, i] về sau (sang phải) một vị trí. Và sau đó cập nhật lại a_k bằng với a_{i+1}.
     - Ngược lại: các phần tử trong đoạn từ [1, i + 1] đã có thứ tự.
   - Tiếp tục thực hiện cho đến khi chèn hết phần tử từ dãy C sang dãy B. Kết quả là dãy B sẽ chứa tất cả các phần tử ban đầu theo thứ tự tăng dần.

2. **Thuật toán:** Cho mảng a gồm n phần tử a_1, a_2, …, a_n.
   - Bước 1: Chọn i = 2.

- Bước 2: Tìm vị trí k đầu tiên trong đoạn [1, i - 1] thoả $a_k > a_i$. Nếu k tồn tại thì dời các phần tử trong đoạn [k, i – 1] về sau một đơn vị (dịch sang phải một đơn vị) và chèn a_i vào vị trí k (gán $a_k = a_i$).
- Bước 3: Tăng i lên thêm 1 (i = i + 1).
    - Nếu $i \leq n$ ➔ Lặp lại bước 2.
    - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
    - Độ phức tạp về thời gian:
        - Trường hợp tốt nhất: O(n).
        - Trường hợp trung bình: $O(n^2)$.
        - Trường hợp xấu nhất: $O(n^2)$.
    - Độ phức tạp về không gian: O(1).

III. Binary Insertion Sort (Sắp xếp chèn nhị phân):

1. **Ý tưởng:** Đây là một phiên bản khác của thuật toán **Insertion Sort** nhằm giảm số lần so sánh khi tìm vị trí để chèn một phần tử vào một dãy con đã sắp xếp. Cụ thể, khi tìm vị trí k trong đoạn [1, i] thoả $a_{k-1} \leq a_{i+1} < a_k$ để chèn a_{i+1}, ta thấy dãy $a_1, a_2, \ldots, a_i$ đã được sắp xếp nên hoàn toàn có thể áp dụng thuật toán **Binary Search** (tìm kiếm nhị phân) để tìm được vị trí thích hợp để chèn phần tử a_{i+1} vào. Trong thuật toán Insertion Sort, ta cần O(n) phép so sánh trong trường hợp xấu nhất để thực hiện vị tìm vị trí. Nhưng khi dùng Binary Search thì chỉ cần O(log(n)) phép so sánh. Nhưng đối với trường hợp xấu nhất thuật toán vẫn chạy với độ phức tạp về thời gian là $O(n^2)$ vì vẫn cần một lượng phép hoán vị để dời các phần tử sang phải như thuật toán Insertion Sort.

    ❖ Thuật toán **Binary Insert** (Chèn nhị phân, vẫn là thuật toán tìm kiếm nhị phân nhưng thuật toán này là tìm và trả về vị trí để chèn một phần tử vào mảng ban đầu sao cho vẫn giữ được thứ tự tăng dần): Ta có mảnga gồm n phần tử $a_1, a_2, \ldots, a_n$.
    - Bước 1: Cho L = 1, R = n và x là giá trị cần chèn.
    - Bước 2: Trong khi L < R:
        - Chọn phần tử ở giữa làm mốc để so sánh với vị trí là mid = (L + R) / 2.
        - Nếu $x < a_{mid}$ ➔ Vị trí cần chèn chỉ nằm ở mảng bên trái ($a_1, a_2, \ldots, a_{mid-1}$) nên R = mid - 1.
        - Ngược lại ➔ Vị trí cần chèn chỉ nằm ở mảng bên phải ($a_{mid+1}, \ldots, a_R$) nên L = mid – 1.
    - Bước 3: Dời tất cả phần tử trong đoạn [L + 1, n] về sau một đơn vị (dịch sang phải một đơn vị) và chèn x vào vị trí L ($a_L = x$).
    - Bước 4: Dừng thuật toán.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
    - Bước 1: Chọn i = 2.
    - Bước 2: Tìm vị trí k thích hợp trong đoạn [1, i - 1] bằng cách áp dụng thuật toán **Binary Insert** (chèn nhị phân) đã được trình bày ở phần ***ý tưởng*** với các giá trị đầu vào cho thuật toán đó là mảng $a_1, a_2, \ldots, a_{i-1}$ và $x = a_i$.

- Bước 3: Tăng i lên thêm 1 (i = i + 1).
    - Nếu $i \leq n$ ➔ Lặp lại bước 2.
    - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
    - Độ phức tạp về thời gian:
        - Trường hợp tốt nhất: O(n).
        - Trường hợp trung bình: $O(n^2)$.
        - Trường hợp xấu nhất: $O(n^2)$.
    - Độ phức tạp về không gian: O(1).

IV. Bubble Sort (Sắp xếp nổi bọt):

1. **Ý tưởng:** Bắt đầu duyệt từ cuối dãy và hoán vị hai phần tử kề nhau để đưa được phần tử nhỏ hơn về đầu dãy. Và tiếp tục thực hiện như vậy sau khi đã bỏ ra phần tử đã được đưa về đầu dãy. Thực hiện cho đến khi dãy chỉ còn một phần tử, lúc này dãy đã được sắp xếp theo thứ tự.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
    - Bước 1: Cho i = 1 và j = n.
    - Bước 2: Trong khi j > i (đưa phần tử nhỏ nhất của mảng $a_i, a_{i+1}, \ldots, a_n$ về đầu mảng):
        - So sánh 2 phần tử kề nhau a_{j-1} và a_j và hoán vị giá trị nếu chúng là hai phần tử nghịch thế.
        - Giảm j xuống 1 (j = j – 1).
    - Bước 3: Tăng i lên thêm 1 (i = i + 1).
        - Nếu i < n (mảng còn nhiều hơn một phần tử) ➔ Lặp lại bước 2.
        - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
    - Độ phức tạp về thời gian:
        - Trường hợp tốt nhất: $O(n^2)$.
        - Trường hợp trung bình: $O(n^2)$.
        - Trường hợp xấu nhất: $O(n^2)$.
    - Độ phức tạp về không gian: O(1).

V. Shaker Sort:

1. **Ý tưởng:** Đây là một phiên bản nâng cấp của thuật toán **Bubble Sort** nhằm giảm số lần duyệt qua các dãy con đã được sắp xếp ở hai đầu của dãy số và tăng hiệu suất khi thực thi. Vì thuật toán Bubble Sort vẫn còn một vài hạn chế, có 2 điểm hạn chế cần nâng cấp như sau:
    - Thuật toán vẫn chạy và duyệt qua hết dãy số (hoặc một dãy con) trong khi dãy đó đã có thứ tự ➔ Giải quyết bằng cách ghi nhớ vị trí đổi chỗ cuối cùng trong mỗi lần duyệt qua dãy số.

- Khi phần tử lớn nhất ở đầu dãy nhưng vẫn phải chạy hết thuật toán ta mới đưa phần tử đó về được vị trí cuối dãy ➔ Giải quyết bằng cách duyệt một lần từ đầu đến cuối của dãy để đưa phần tử lớn nhất từ đầu về cuối dãy (ngược chiều so với cách duyệt để đưa phần tử nhỏ nhất về đầu dãy của thuật toán gốc).

2. **Thuật toán:** Cho mảng a gồm n phần tử a_1, a_2, …, a_n.
   - Bước 1: Cho L = 1, R = n (mảng a_L, a_{L+1}, …, a_R cần sắp xếp) và k = n (vị trí đổi chỗ cuối cùng).
   - Bước 2: Cho i = R. Trong khi i > L (đưa phần tử nhỏ nhất của mảng a_L, a_{L+1}, …, a_i về đầu mảng):
     - So sánh hai phần tử kề nhau là a_{i-1} và a_i ➔ Nếu hai phần tử này là nghịch thế ($a_{i-1} > a_i$) ➔ Hoán vị hai phần tử và lưu lại vị trí đã xảy ra hoán vị (k = i).
     - Giảm i xuống 1 (i = i – 1).
   - Bước 3: Thu gọn mảng cần sắp xếp lại (bỏ đi phần bên phải đã được sắp xếp): R = k.
   - Bước 4: Cho i = L. Trong khi i < R (đưa phần tử lớn nhất của mảng a_i, a_{i+1}, …, a_R về cuối mảng):
     - So sánh hai phần tử kề nhau là a_i và a_{i+1} ➔ Nếu hai phần tử này là nghịch thế ($a_i > a_{i+1}$) ➔ Hoán vị hai phần tử và lưu lại vị trí đã xảy ra hoán vị (k = i).
     - Giảm i xuống 1 (i = i – 1).
   - Bước 5: Thu gọn mảng cần sắp xếp lại (bỏ đi phần bên trái đã được sắp xếp): L = k.
   - Bước 6:
     - Nếu L < R (mảng còn nhiều hơn một phần tử) ➔ Lặp lại bước 2.
     - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(n).
     - Trường hợp trung bình: $O(n^2)$.
     - Trường hợp xấu nhất: $O(n^2)$.
   - Độ phức tạp về không gian: O(1).

VI. Shell Sort (Sắp xếp theo độ dài bước giảm dần):

1. **Ý tưởng:** Thuật toán này là một phiên bản cải tiến của thuật toán **Insertion Sort**. Với ý tưởng là sẽ chia dãy ban đầu thành nhiều đoạn (dãy con) gồm các phần tử ở cách nhau **h** vị trí:
   - Dãy ban đầu được xem như sự xen kẽ của các dãy sau:
     - Dãy con thứ nhất: a_1, a_{h+1}, a_{2h+1}, …
     - Dãy con thứ hai: a_2, a_{h+2}, a_{2h+2}, …
     - Dãy con thứ i: a_i, a_{2i}, a_{3i}, …
   - Ta áp dụng thuật toán Insertion Sort với từng đoạn (dãy con) của dãy ban đầu và nó sẽ đưa các phần tử về đúng vị trí trong đoạn đó. Sau đó ta sẽ giảm giá trị của h xuống để tạo ra các dãy con mới và có thể so sánh các phần tử không ở cùng dãy con trước đó. Thuật toán sẽ kết thúc khi h = 1, khi này đảm bảo tất cả các phần tử sẽ được so sánh với nhau và dãy số sẽ được sắp xếp lại theo đúng thứ tự.

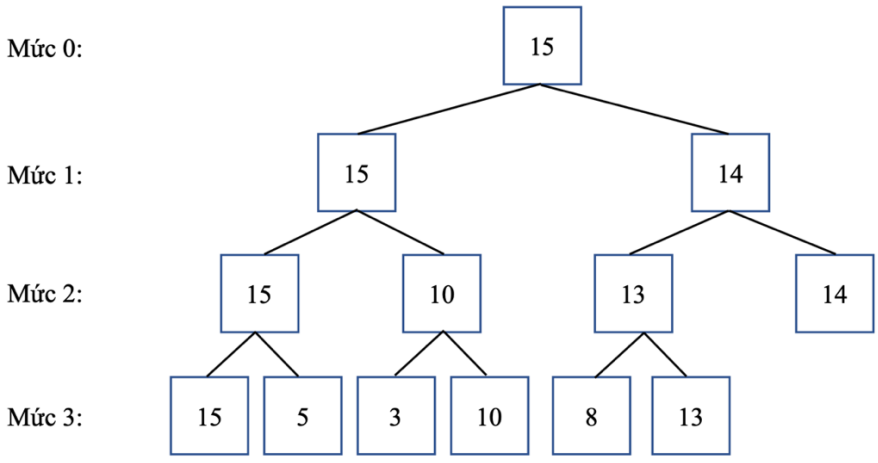
- Theo **Knuth** ta có 2 công thức để tính giá trị của h và k (số bước thực hiện hay số đoạn được chia):
    - Công thức thứ nhất: $h_i = (h_{i-1} - 1) / 3$ và $h_k = 1$, $k = \log_3 n - 1$.
    - Công thức thứ hai: $h_i = (h_{i-1} - 1) / 2$ và $h_k = 1$, $k = \log_2 n - 1$.
- Trong trường hợp này ta sẽ áp dụng công thức thứ 2.

2. **Thuận toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
- Bước 1:
    - Cho $k = \log_2 n - 1$ và $i = 1$.
    - Cho mảng h gồm k phần tử với $h_i = (h_{i-1} - 1) / 2$ và $h_k = 1$.
- Bước 2: Trong khi $i \leq k$:
    - Chia dãy ban đầu thành các mảng con mà mỗi phần tử cách nhau h_i vị trí.
    - Áp dụng thuật toán Insertion Sort để sắp xếp từng mảng con đó.
- Bước 3: Dừng thuật toán.

3. **Đánh giá thuật toán:**
- Độ phức tạp về thời gian:
    - Trường hợp tốt nhất: O(nlog(n)).
    - Trường hợp trung bình: $O(n^{4/3})$.
    - Trường hợp xấu nhất: $O(n^{3/2})$.
- Độ phức tạp về không gian: O(1).

VII. Heap Sort (Sắp xếp vun đống):

1. **Ý tưởng:**
- Khi thực hiện việc tìm vị trí phần tử nhỏ nhất ở bước i trong thuật toán **Selection Sort**, người dùng đã không tận dụng được kết quả của các phép so sánh ở bước i – 1. Để giải quyết vấn đề trên, mọi người đã tìm ra được một cấu trúc dữ liệu để lưu trữ các thông tin của phép so sánh ở các bước trước, có dạng cây như sau:
  Ví dụ: Ta có dãy số: 15 5 3 10 8 13 14.



Hình 1. Minh hoạ về cấu trúc của **Heap**.

Trong đó, phần tử ở mức i là phần tử lớn nhất ở mức i + 1. Do đó phần tử ở mức 0 (gốc của cây) sẽ là phần tử lớn nhất của dãy số. Vì vậy, khi loại bỏ phần tử ở gốc (tức là đưa phần tử lớn nhất về đầu dãy), ta chỉ cần quan tâm đến nhánh mà chứa phần tử đó và cập nhật lại nhánh đó, còn các nhánh khác đều được bảo toàn. Do đó, khi thực hiện bước kế tiếp ta đã tận dụng lại kết quả của phép so sánh ở bước trước đó.

Để cài đặt được thuật toán sắp xếp đó phải cần đến 2n – 1 ô nhớ để lưu trữ (mỗi ô là một nút trên cây) giá trị các phần tử trên cây. Nên để tối ưu hơn **J.Williams** đã đề xuất ra một khái niệm mới là **Heap** và thuật toán sắp xếp **Heap Sort** để giải quyết vấn đề trên mà chỉ cần n ô nhớ để lưu trữ.

- Định nghĩa **Heap**: Giả sử có một dãy số $a_1, a_2, \ldots, a_n$ và cần sắp xếp theo thứ tự tăng dần.
  - Các phần tử a_i với $i \in [1, n]$ luôn thoả: $a_i \geq a_{2i}$ và $a_i \geq a_{2i+1}$. Và (a_i, a_{2i}), (a_i, a_{2i+1}) được gọi là các cặp phần tử liên đới.
  - Có 3 tính chất sau:
    - Tính chất 1: Nếu $a_1, a_2, \ldots, a_n$ là một Heap thì khi loại bỏ một số phần tử ở hai đầu của heap, dãy còn lại vẫn là một Heap.
    - Tính chất 2: Nếu $a_1, a_2, \ldots, a_n$ là một heap thì phần tử ở đầu dãy luôn là phần tử lớn nhất.
    - Tính chất 3: Mọi dãy $a_L, a_{L+1}, \ldots, a_R$ với $2L > R$ là một Heap.

❖ Thuật toán xây dựng **Heap**: Cho một dãy số $a_1, a_2, \ldots, a_n$.

- Bước 1: Chọn $L = n / 2$. Vì dãy số $a_{n/2+1}, a_{n/2+2}, \ldots, a_n$ đã là một Heap theo tính chất 3.
- Bước 2: Trong khi $L \geq 0$ (thêm hết nửa dãy đầu vào dãy số đã là Heap phía sau để dãy số ban đầu trở thành một Heap):
  - Hiệu chỉnh dãy số $a_L, a_{L+1}, \ldots, a_n$ (dãy số $a_{L+1}, a_{L+2}, \ldots, a_n$ đã là một Heap nên khi thêm a_L vào đầu mảng cần hiểu chỉnh lại dãy số để dãy số sau khi thêm a_L vẫn là một Heap).
  - Giảm L xuống thêm 1.
- Bước 3: Dừng thuật toán.

❖ Thuật toán hiệu chỉnh dãy số: Có dãy số $a_L, a_{L+1}, \ldots, a_R$.

- Bước 1: Chọn $i = L$, $j = 2i$.
- Bước 2: Nếu $j \leq R$:
  - Tìm $\max(a_j, a_{j+1})$ (nếu có) để so sánh với phần tử cần hiệu chỉnh lại và gán lại j bằng với chỉ số phần tử lớn nhất.
  - Nếu $a_i > a_j$ ➔ Chuyển sang bước 3.
  - Ngược lại ➔ Hoán vị giá trị a_i và a_j. Vì việc hoán vị này đã thay đổi giá trị a_j nên có thể dãy số $a_j, a_{j+1}, \ldots, a_R$ đã không còn là heap nên lặp lại bước 1 với dãy số $a_j, a_{j+1}, \ldots, a_R$.
- Bước 3: Dừng thuật toán.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
   - Bước 1: Xây dựng mảng ban đầu thành Heap.
   - Bước 2: Hoán vị phần tử a_1 và a_n (để đưa phần tử lớn nhất a_1 về cuối mảng).

- Bước 3: Hiệu chỉnh lại mảng với n – 1 phần tử còn lại (bỏ phần tử cuối).
- Bước 4: Lặp lại bước 2 cho đến khi mảng chỉ còn 1 phần tử.

3. **Đánh giá thuật toán:**
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(nlog(n)).
     - Trường hợp trung bình: O(nlog(n)).
     - Trường hợp xấu nhất: O(nlog(n)).
   - Độ phức tạp về không gian: O(1).

VIII. Merge Sort (Sắp xếp trộn):

1. **Ý tưởng:** Dựa theo kỹ thuật chia để trị (divide and conquer). Lần lượt chia đôi dãy ban đầu thành hai dãy con (kích thước dãy ban đầu giảm một nửa). Tiếp tục việc chia đôi dãy con như vậy cho đến khi được n dãy con (mỗi dãy con chứa một phần tử). Trộn theo thứ tự hai dãy con được chia ra từ dãy con trước đó. Tiếp tục trộn như vậy cho đến khi nhận được một dãy có kích thước bằng với dãy ban đầu. Và dãy này cũng là dãy đã được sắp xếp theo thứ tự (tăng dần).

   ❖ Thuật toán trộn 2 mảng: mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$ và mảng b gồm m phần tử $b_1, b_2, \ldots, b_m$.
   - Bước 1: Khởi tạo mảng c với k = n + m phần tử và ic = ia = ib = 1.
   - Bước 2: Trong khi ia ≤ n và ib ≤ m:
     - Nếu $a_{ia} > b_{ib}$ (cặp phần tử nghịch thế) ➔ Thêm phần tử a_{ia} vào mảng c và tăng ia lên thêm 1.
     - Ngược lại thêm phần tử b_{ib} vào mảng c và tăng ib lên thêm 1.
     - Tăng ic lên thêm 1.
   - Bước 3:
     - Nếu ia ≤ n ➔ Thêm tất cả phần tử còn lại của a vào c.
     - Ngược lại nếu ib ≤ m ➔ Thêm tất cả phần tử còn lại của b vào c.
     - Ngược lại ➔ Dừng thuật toán.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
   - Bước 1: Chọn k = 1 (chiều dài của mảng con ở bước hiện tại).
   - Bước 2: Tách mảng $a_1, a_2, \ldots, a_n$ thành 2 mảng b và c như sau:
     - Mảng b gồm: $a_1, \ldots, a_k, a_{2k+1}, \ldots, a_{3k}, \ldots$
     - Mảng c gồm: $a_{k+1}, \ldots, a_{2k}, a_{3k+1}, \ldots, a_{4k}, \ldots$
   - Bước 3: Tiến hành trộn từng mảng con gồm k phần tử của 2 mảng b và c vào a.
   - Bước 4: Tăng k lên 2 lần (k = k * 2).
     - Nếu k < n ➔ Lặp lại bước 2.
     - Ngược lại ➔ Dừng thuật toán.

3. **Đánh giá thuật toán:**
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(nlog(n)).

- Trường hợp trung bình: O(nlog(n)).
- Trường hợp xấu nhất: O(nlog(n)).

- Độ phức tạp về không gian: O(n).

IX. Quick Sort (Sắp xếp nhanh):

1. **Ý tưởng:** Áp dụng thuật toán phân hoạch dãy $a_1, a_2, \ldots, a_n$ thành hai phần:
   - Dãy con 1: gồm các phần tử nhỏ hơn x.
   - Dãy con 2: gồm các phần tử lớn hơn x.

   Với x là một phần tử bất kỳ trong dãy ban đầu. Sau khi phân hoạch dãy ban đầu ta có được ba phần:
   - Dãy 1: $a_k < x$ với $k = 1 \ldots i$.
   - Dãy 2: $a_k = x$ với $k = (i + 1) \ldots (j - 1)$.
   - Dãy 3: $a_k > x$ với $k = j \ldots n$.

   Trong đó dãy 2 đã có thứ tự, nếu dãy 1 hoặc dãy 3 chỉ chứa 1 phần tử thì dãy đó cũng có thứ tự. Ngược lại, nếu dãy 1 hoặc dãy 3 chứa nhiều hơn 1 phần tử thì tiếp tục thực hiện việc phân hoạch với các dãy đó như đã trình bày.

   ❖ Thuật toán phân hoạch: có mảng gồm n phần tử $a_1, a_2, \ldots, a_n$.
   - Bước 1: Chọn $i = L = 1$, $j = R = n$ và $x = a_k$ với $L \le k \le R$.
   - Bước 2: Thực hiện việc chỉnh lại vị trí cho 2 phần tử a_i, a_j đang bị đặt sai vị trí:
     - Trong khi $a_i < x$ thì tăng i lên thêm 1.
     - Trong khi $a_j > x$ thì giảm j xuống 1.
     - Nếu $i \le j$ (có 2 phần tử đang bị đặt sai vị trí) thì hoán vị hai phần tử a_i, a_j và tăng i thêm 1, giảm j xuống 1.
   - Bước 3:
     - Nếu $i \le j$ (chưa duyệt hết mảng) ➔ Lặp lại bước 2.
     - Ngược lại ➔ Dừng thuật toán.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$.
   - Bước 1: Chọn L = 1 và R = n.
   - Bước 2: Áp dụng thuật toán phân hoạch mảng $a_L, \ldots, a_R$ thành ba phần:
     - Mảng 1: $a_k < x$ với $k = L \ldots j$.
     - Mảng 2: $a_k = x$ với $k = (j + 1) \ldots (i - 1)$.
     - Mảng 3: $a_k > x$ với $k = i \ldots R$.
   - Bước 3:
     - Nếu $L < j$ (mảng 1 vẫn còn chứa nhiều hơn 1 phần tử) thì lặp lại bước 2 với R = j và giữ nguyên L.
     - Nếu $i < R$ (mảng 3 vẫn còn chứa nhiều hơn 1 phần tử) thì lặp lại bước 2 với L = I và giữ nguyên R.
   - Bước 4: Dừng chương trình.

3. **Đánh giá thuật toán:**
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(nlog(n)).

- Trường hợp trung bình: O(nlog(n)).
- Trường hợp tệ nhất: $O(n^2)$.

- Độ phức tạp về không gian: O(log(n)).

X. Counting Sort (Sắp xếp đếm phân phối):

1. **Ý tưởng:** Xét dãy số chứa các phần tử riêng biệt (không trùng nhau). Đối với mỗi phần tử của một dãy số a có thứ tự thì số lượng phần tử mà bé hơn phần tử a_i chính là vị trí liền trước của phần tử a_i trong dãy số đó. Nên thuật toán này sẽ dựa trên việc đếm số lượng phần tử bé hơn phần tử a_i chứ không dựa trên các phép so sánh như các thuật toán khác.
   Ví dụ: Ta có dãy số đã được sắp xếp: 1 3 8 9.
   - Có 1 phần tử bé hơn 3 nên phần tử 3 sẽ ở vị trí 2.
   - Có 2 phần tử bé hơn 8 nên phần tử 8 sẽ ở vị trí 3.
   - Có 3 phần tử bé hơn 9 nên phần tử 9 sẽ ở vị trí 4.

2. **Thuật toán:** Cho mảng a gồm n phần tử $a_1, a_2, \ldots, a_n$. Qui ước cách viết phần tử thứ i là tên_mảng[i].
   - Bước 1: Tạo một mảng C với kích thước m = max(a[1], …, a[n]) với giá trị ban đầu của từng phần tử là 0.
   - Bước 2: Đếm số lần xuất hiện của từng phần tử trong mảng a và giá trị đó vào gán vào C[a[i]].
   - Bước 3: Duyệt qua từng phần tử (bỏ qua phần tử đầu) trong mảng C và cộng dồn giá trị với phần tử liền trước: C[i] = C[i] + C[i – 1].
   - Bước 4: Tạo một mảng tạm B với kích thước bằng n (để lưu các giá trị của mảng a mà đã được sắp xếp).
   - Bước 5: Duyệt từ cuối mảng a và thêm từng phần tử a[i] vào mảng B ở vị trí j = C[a[i]]:
     - Cho k = n. Trong khi k >= 1:
       - Gán B[C[a[k]] = a[k].
       - Giảm giá trị C[a[k]] xuống 1 (C[a[k]] = C[a[k]] - 1).
     - Lúc này mảng B chính là mảng a sau khi đã sắp thứ tự.
   - Bước 6: Sao chép toàn bộ giá trị từ mảng B sang mảng a.
   - Bước 7: Dừng thuật toán.

3. **Đánh giá thuật toán:** Ta có $k = \max(a_1, \ldots, a_n)$.
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(n).
     - Trường hợp trung bình: O(n + k).
     - Trường hợp xấu nhất: O(n + k).
   - Độ phức tạp về không gian: O(n + k).

❖ **<u>Lưu ý:</u>** Phần cài đặt thuật toán đã tối ưu bộ nhớ khi mảng chứa số lần xuất hiện của từng phần tử chỉ có kích thước là $m = \max(a_1, \ldots, a_n) - \min(a_1, \ldots, a_n) + 1$.

XI. Radix Sort (Sắp xếp theo cơ số):

1. **Ý tưởng:** Phương pháp sắp xếp này không quá quan tâm đến việc so sánh giá trị của 2 phần tử mà dựa trên nguyên tắc phân loại thư của bưu điện. Cụ thể, để chuyển một khối lượng thư lớn đến tay người nhận. Bưu điện sẽ phân loại thư thuộc các tỉnh thành cụ thể và gửi đến tỉnh thành đó. Từ đó, các tỉnh thành sẽ phân loại thư của từng quận, huyện và gửi về quận, huyện đó. Thực hiện tương tự với phường xã. Do đó, mô phỏng lại cách chuyển thư của bưu điện, ta sẽ sắp xếp dãy số dựa vào việc phân loại các phần tử theo chữ số hàng đơn vị, hàng chục, hàng trăm,….

2. **Thuật toán:** Cho mảng a gồm n phần tử a_1, a_2, …, a_n. Qui ước $a_i[k]$ là chữ số thứ k của phần tử a_i (từ trái sang phải).
   - Bước 1: Tạo một mảng hai chiều b với n dòng và 10. Xác định số lượng chữ số của phần tử lớn nhất (phần tử có nhiều chữ số nhất) và lưu vào k.
   - Bước 2: Trong khi k > 0:
     - Duyệt qua n phần tử và thêm phần tử thứ i vào cột $a_i[k] - 1$ của b.
     - Lấy các phần tử ra theo từng cột từ 1 đến 10 (theo đúng trình tự thêm vào của cột) và thêm vào a (theo thứ tự từ a_1, a_2, …, a_n).
     - Giảm k xuống 1 (k = k – 1).
   - Bước 3: Dừng thuật toán.

3. **Đánh giá thuật toán:** Ta có k số chữ số của phần tử lớn nhất.
   - Độ phức tạp về thời gian:
     - Trường hợp tốt nhất: O(kn).
     - Trường hợp trung bình: O(kn).
     - Trường hợp tệ nhất: O(kn).
   - Độ phức tạp về không gian: O(k + n).

XII. Flash Sort:

1. **Ý tưởng:** Chia mảng ban đầu thành các ngăn chứa các vài phần tử thích hợp. Sau đó, sắp xếp lại các phần tử trong từng ngăn theo thứ tự tăng dần và nối từng ngăn lại theo thứ tự đã chia ban đầu ta sẽ được một mảng các phần tử tăng dần. Công thức để xác định ngăn thứ k của phần tử a_i với m là số lượng ngăn (được tính theo công thức bên dưới):

$$k(a_i) = 1 + int\left((m-1)\frac{a_i - a_{min}}{a_{max} - a_{min}}\right)$$

   Trong đó: $m = \lfloor 0.42 \times n \rfloor$ (theo Karl-Dietrich Neubert).

2. **Thuật toán:** Cho mảng a gồm n phần tử a_1, a_2, …, a_n và m là số ngăn (được tính theo công thức bên trên). Qui ước cách viết phần tử thứ i là tên_mảng[i].
   - Bước 1:
     - Cho r = max(a[1], …, a[n]).
     - Tạo một mảng B gồm r phần tử với a[i] sẽ nằm ở ngăn thứ B[i] và tính giá trị B[i] theo công thức bên trên.

- Tạo một mảng FP gồm m + 1 (số ngăn) phần tử với FP[i] (FP[1] = 1, không sử dụng FP[m + 1])vị trí đầu tiên của phần tử thuộc ngăn thứ i trong mảng a sau khi đã sắp xếp.
- Cho i = 1. Trong khi i ≤ n:
  - Xác định ngăn của phần tử a_i:
$$B[a[i]] = 1 + int\left((m-1)\frac{a_i - a_{min}}{a_{max} - a_{min}}\right).$$
  - Ngăn thứ B[i] tăng thêm một phần tử:
$$FP[B[a[i]] + 1] = FP[B[a[i]] + 1] + 1.$$
- Hiện tại mảng FP đang chứa số lượng phần tử của từng ngăn. Ta tính vị trí đầu tiên của phần tử thuộc ngăn thứ i trong mảng a sau khi đã sắp xếp bằng cách duyệt từ phần tử thứ hai đến cuối mảng FP và cộng dồn với phần tử liền trước nó (FP[j] = FP[j] + FP[j - 1]).

- Bước 2:
  - Tạo một mảng A gồm n phần tử chứa các phần tử của từng ngăn (theo thứ tự từ ngăn 1 đến ngăn m).
  - Tạo một mảng P gồm m phần tử với P[i] là số lượng phần tử hiện tại của ngăn thứ i.
  - Cho i = 1. Trong khi i ≤ n:
    - Cho j = $FP\left[B[a[i]]\right]$ là vị trí của phần tử đầu tiên trong ngăn thứ B[a[i]] (ngăn của phần tử a[i]).
    - Cho h = $P\left[B[a[i]]\right]$ là số lượng phần tử hiện tại trong ngăn thứ B[a[i]] (ngăn của phần tử a[i]).
    - Chuyển phần tử a[i] vào đúng ngăn: A[j + h] = a[i].
    - Tăng $P\left[B[a[i]]\right]$ lên thêm 1: $P\left[B[a[i]]\right] = P\left[B[a[i]]\right] + 1$.
- Bước 3: Áp dụng thuật toán Insertion Sort để sắp xếp từng ngăn của mảng A.
- Bước 4: Sao chép toàn bộ phần tử từ mảng A (mảng a ban đầu sau khi đã được sắp xếp) qua mảng a.
- Bước 5: Dừng thuật toán.

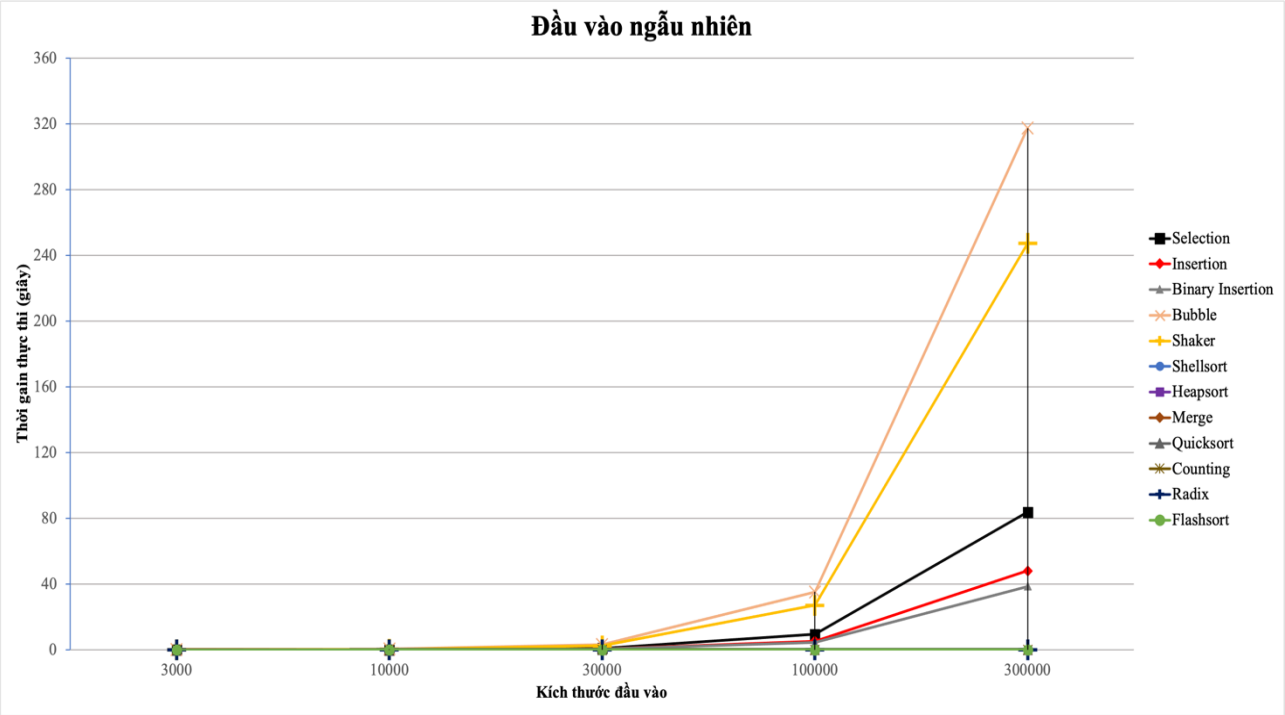
3. Đánh giá thuật toán:

- Độ phức tạp về thời gian:
  - Trường hợp tốt nhất: O(n).
  - Trường hợp trung bình: O(n + r) với $r = a_{max}$.
  - Trường hợp tệ nhất: $O(n^2)$.
- Độ phức tạp về không gian: O(n).

❖ **<u>Lưu ý:</u>** Phần cài đặt thuật toán đã tối ưu bộ nhớ khi mảng chứa số lần xuất hiện của từng phần tử chỉ có kích thước là $m = \max(a_1, \ldots, a_n) - \min(a_1, \ldots, a_n) + 1$.

B. KẾT QUẢ THỰC NGHIỆM:

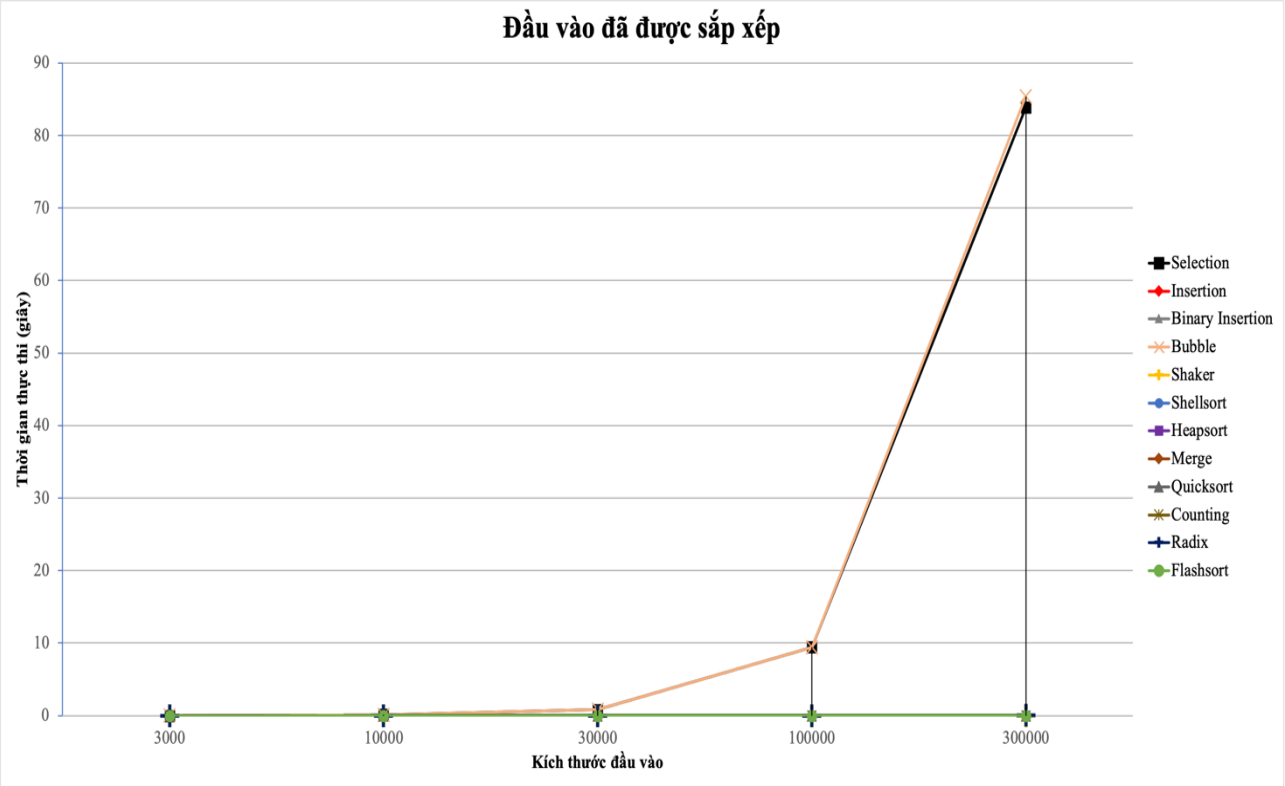
I. Đầu vào được ngẫu nhiên:



Hình 2. Đồ thị biểu diễn thời gian chạy của các thuật toán trong từng kích thước đầu vào (giá trị được sinh ngẫu nhiên).

- **Nhận xét:**
  - Thuật toán nhanh nhất trong tất cả kích thước đầu vào: Counting Sort. Vì độ phức tạp thuật toán là O(k + n) với k = max(a_1, …, a_n) – min(a_1, …, a_n) + 1 và k << n nên lúc này độ phức tạp sẽ là O(n). Do đó thuật toán này rất nhanh.
  - Thuật toán chậm nhất trong tất cả kích thước đầu vào: Bubble Sort (sắp xếp nổi bọt). Vì sau mỗi lần đưa phần tử nhỏ nhất về đầu mảng thì có thể một số phần tử đã được đặt đúng vị trí ở giữa mảng đã bị dời sang vị trí khác nên số lần thực hiện việc hoán đổi vị trí xảy ra quá nhiều dẫn đến việc thuật toán thực hiện quá lâu.
  - Gia tốc của thuật toán Counting Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(0{,}009378 - 0{,}000052)^2} = 3414802786$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(317{,}550301 - 0{,}02831)^2} \approx 3$ (phần tử / s^2).

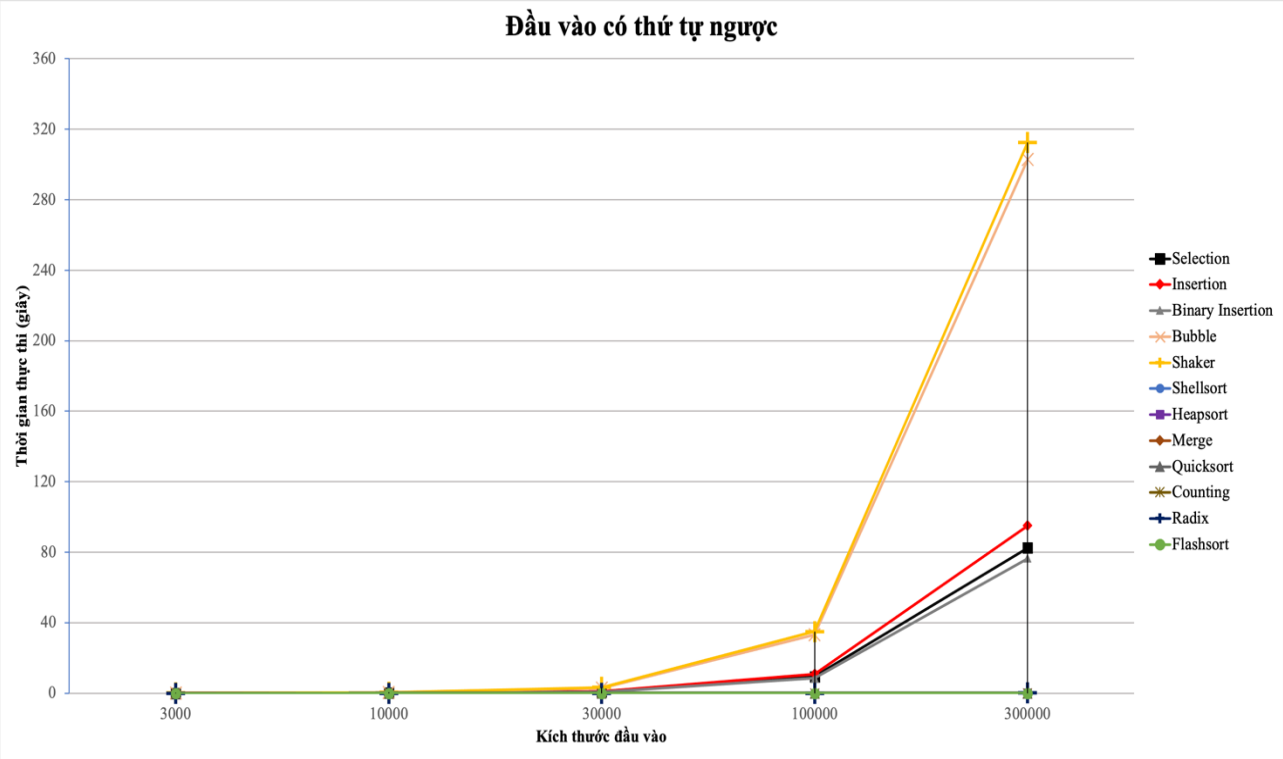
II. Đầu vào đã được sắp xếp:



Hình 3. Đồ thị biểu diễn thời gian chạy của các thuật toán trong từng kích thước đầu vào (giá trị đã được sắp xếp).

- **<u>Nhận xét:</u>**
  - Thuật toán nhanh nhất trong tất cả kích thước đầu vào: Shaker Sort. Vì đây là trường hợp tốt nhất của thuật toán với độ phức tạp về thời gian chỉ O(n). Thuật toán chỉ cần duyệt một lần qua n phần tử là xong.
  - Thuật toán chậm nhất trong tất cả kích thước đầu vào: Selection Sort (sắp xếp chọn) và Bubble Sort (sắp xếp nổi bọt). Cả hai thuật toán này với thời gian sắp xỉ nhau vì đều phải duyệt qua hết mảng trong mỗi lần duyệt đều thực hiện phép so sánh với độ phức tạp về thời gian của hai thuật toán là $O(n^2)$.
  - Gia tốc của thuật toán Shaker Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(0{,}000666 - 0{,}000011)^2} \approx 692267350387$ (phần tử / s^2).
  - Gia tốc của thuật toán Selection Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(83{,}845561 - 0{,}008066)^2} \approx 42$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(85{,}36559 - 0{,}008443)^2} \approx 41$ (phần tử / s^2).

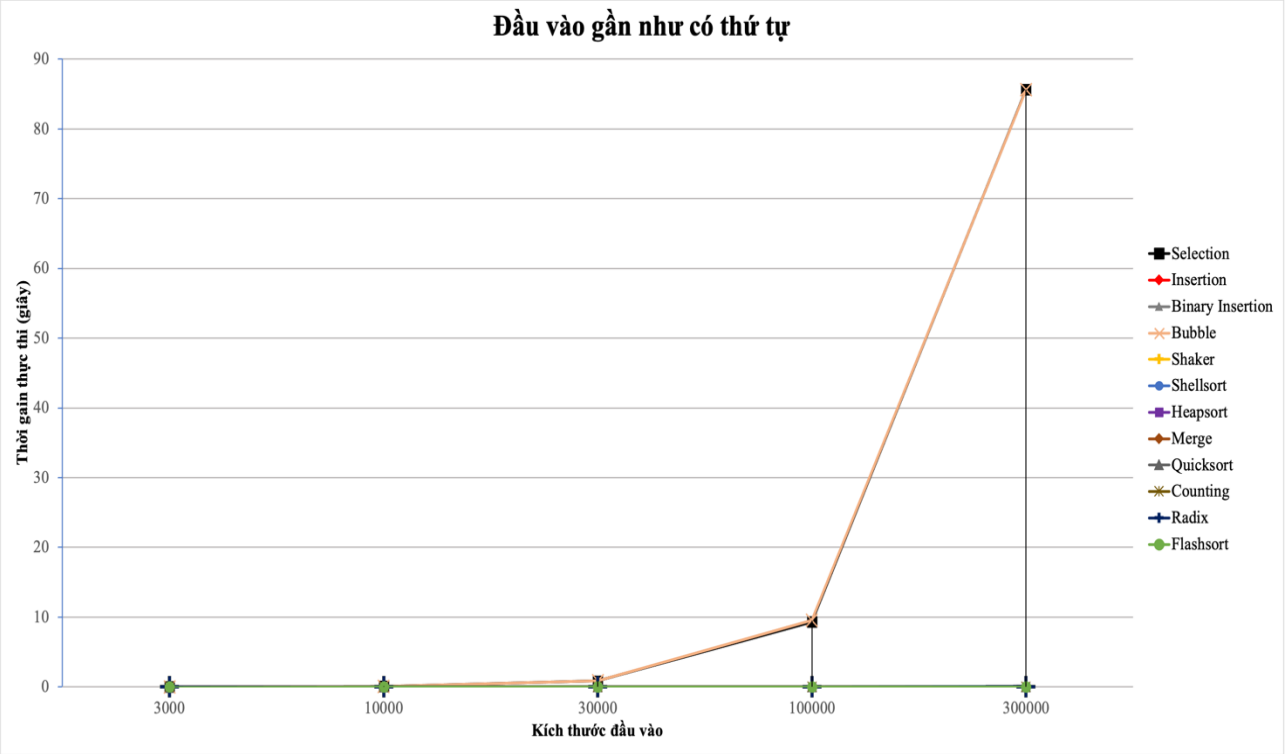
III. Đầu vào có thứ tự ngược:



Hình 4. Đồ thị biểu diễn thời gian chạy của các thuật toán trong từng kích thước đầu vào (giá trị có thứ tự ngược).

- **<u>Nhận xét:</u>**
  - Thuật toán nhanh nhất trong tất cả kích thước đầu vào: Couting Sort. Vì độ phức tạp thuật toán là O(k + n) với k = max(a_1, …, a_n) – min(a_1, …, a_n) + 1 và k << n nên lúc này độ phức tạp sẽ là O(n). Do đó, thuật toán này rất nhanh.
  - Thuật toán chậm nhất trong tất cả kích thước đầu vào: Shaker. Đây là trường hợp xấu nhất của thuật toán này. Thuật toán thực thi với độ phức tạp là $O(n^2)$.
  - Gia tốc của thuật toán Counting Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(0{,}003672 - 0{,}000042)^2} \approx 22539444027$ (phần tử / s^2).
  - Gia tốc của thuật toán Shaker Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(312{,}535868 - 0{,}032793)^2} \approx 3$ (phần tử / s^2).

IV. Đầu vào gần như có thứ tự:

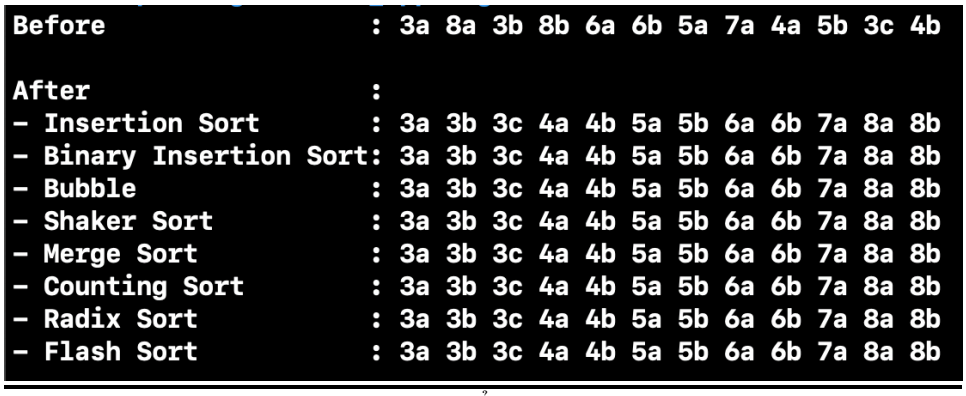


Hình 5. Đồ thị biểu diễn thời gian chạy của các thuật toán trong từng kích thước đầu vào (giá trị gần như có giá trị).

- **<u>Nhận xét:</u>**
  - Thuật toán nhanh nhất trong tất cả kích thước đầu vào: Counting Sort. Vì độ phức tạp thuật toán là O(k + n) với k = max(a_1, …, a_n) – min(a_1, …, a_n) + 1 và k << n nên lúc này độ phức tạp sẽ là O(n). Do đó thuật toán này rất nhanh.
  - Thuật toán chậm nhất trong tất cả kích thước đầu vào: Selection Sort (sắp xếp chọn) và Bubble Sort (sắp xếp nổi bọt). Cả hai thuật toán này với thời gian sắp xỉ nhau nhưng Bubble Sort có thể sẽ chậm hơn vì khi hoán vị có thể đã vô tình dời phần tử đang được đặt đúng vị trí sang một vị trí khác. Cả hai thuật toán này thực hiện chậm là vì đều phải duyệt qua hết mảng trong mỗi lần duyệt đều thực hiện phép so sánh với độ phức tạp về thời gian của hai thuật toán là O(n^2).
  - Gia tốc của thuật toán Counting Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(0{,}003515 - 0{,}000044)^2} \approx 24651719756$ (phần tử / s^2).
  - Gia tốc của thuật toán Selection Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(85{,}688091 - 0{,}008135)^2} \approx 40$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{300000 - 3000}{(85{,}635152 - 0{,}008733)^2} \approx 41$ (phần tử / s^2).

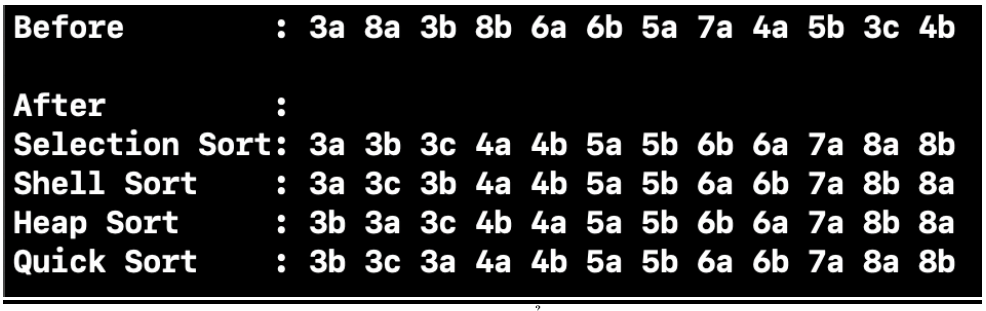
❖ Kết luận:

- Thuật toán chạy nhanh nhất: Counting Sort.
- Thuật toán chạy chậm nhất: Bubble Sort.
- Nhóm thuật toán ổn định: Insertion Sort, Binary Insertion Sort, Bubble Sort, Shaker Sort, Merge Sort, Counting Sort, Radix Sort, Flash Sort.



Hình 6. Minh hoạ sự ổn định của thuật toán.

- Nhóm thuật toán không ổn định: Selection Sort, Shell Sort, Heap Sort, Quick Sort.



Hình 7. Minh hoạ sự không ổn định của thuật toán.

- Ghi chú: mảng đầu vào cần sắp xếp trong hai hình trên là mảng số nguyên, các chữ cái a, b, c, … chỉ dùng để đánh dấu thứ tự xuất hiện của số đó trong mảng.

TÀI LIỆU THAM KHẢO

1. **Đỗ Xuân Lôi**: *"Cấu trúc dữ liệu và giải thuật"*. Nhà xuất bản Đại học Quốc gia Hà Nội – 2010.
2. **Trần Hạnh Nhi – Dương Anh Đức**: *"NHẬP MÔN CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT"*. Trường Đại học Khoa học Tự nhiên TP.HCM – 2003.
3. **Trần Đan Thư – Nguyễn Thanh Phương – Đinh Bá Tiến – Trần Minh Triết – Đặng Bình Phương**: *"KỸ THUẬT LẬP TRÌNH"*. Nhà xuất bản Khoa học và Kỹ thuật – 2019.
4. https://codelearn.io/learning/cau-truc-du-lieu-va-giai-thuat/853804
5. https://en.wikipedia.org/wiki/Insertion_sort
6. https://www.geeksforgeeks.org/binary-insertion-sort/
7. https://en.wikipedia.org/wiki/Flashsort
8. https://www.w3resource.com/javascript-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-12.php
9. https://en.wikipedia.org/wiki/Sorting_algorithm