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 1 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 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, …, a_n.
   - Bước 3: Hoán vị hai phần tử a_{min} và a_n.
   - 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 một dãy 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 gồm n phần tử a_1, a_2, …, a_n.
   - Bước 1: Chọn i = 2.
   - Bước 2: Tìm vị trí k 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 chương trình.

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. Bubble Sort (Sắp xếp nổi bọt):

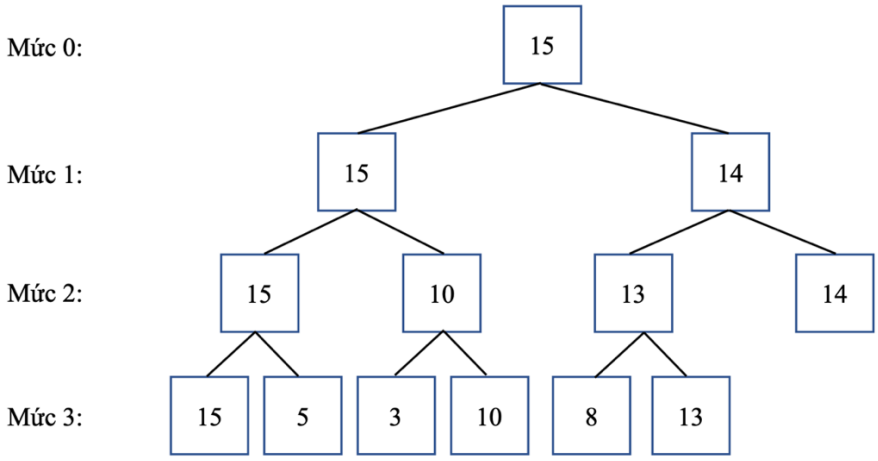
1. **Ý tưởng:** Bắt đầu duyệt từ cuối dãy và hoán vị 2 phần tử kề nhau để đưa được phần tử nhỏ hơn về đầu dãy. Và tiếp tục thực hiện như vậy sau khi đã bỏ ra phần tử đã được đưa về đầu dãy. Thực hiện cho đến khi dãy chỉ còn một phần tử, lúc này dãy đã được sắp xếp theo thứ tự.

2. **Thuật toán:** Cho mảng 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).

IV. 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 sắp xếp chọn (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:
     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 nó, 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 1 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}, …, 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}, …, a_R đã không còn là heap nên lặp lại bước 1 với dãy số a_j, a_{j+1}, …, a_R.
- Bước 3: Dừng thuật toán.

2. **Thuật toán:** Cho dãy số a_1, a_2, …, a_n.
- Bước 1: Xây dựng dãy số 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 dãy số).
- Bước 3: Hiệu chỉnh lại dãy số với n – 1 phần tử còn lại (bỏ phần tử cuối).
- Bước 4: Lặp lại bước 2 cho đến khi dãy 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).

V. 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 2 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 2 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, …, a_n và mảng b gồm m phần tử b_1, b_2, …, 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 gồm n phần tử a_1, a_2, ..., a_n.
   - Bước 1: Chọn k = 1 (chiều dài của mảng con ở bước hiện tại).
   - Bước 2: Tách mảng a_1, a_2, …, a_n thành 2 mảng b và c như sau:
     - Mảng b gồm: a_1, …, a_k, a_{2k+1}, …, a_{3k}, …
     - Mảng c gồm: a_{k+1}, …, a_{2k}, a_{3k+1}, …, a_{4k}, …
   - 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).

VI. Quick Sort (Sắp xếp nhanh):

1. **Ý tưởng:** Áp dụng thuật toán phân hoạch dãy a_1, a_2, …, a_n thành 2 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 3 phần:
   - Dãy 1: $a_k < x$ với k = 1…i.
   - Dãy 2: $a_k = x$ với k = (i + 1)…(j – 1).
   - Dãy 3: $a_k > x$ với k = j…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 dãy gồm n phần tử a_1, a_2, …, 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 2 phần tử a_i, a_j và tăng i thêm 1, giảm j xuống 1.
   - Bước 3:
     - Nếu $i \le j$ (chưa duyệt hết dãy số) thì lặp lại bước 2.
     - Ngược lại thì dừng thuật toán.

2. **Thuật toán:** Cho dãy a gồm n phần tử a_1, a_2, …, a_n.

- Bước 1: Chọn L = 1 và R = n.

- Bước 2: Áp dụng thuật toán phân hoạch dãy a_L, …, a_R thành 3 phần:
  - Dãy 1: $a_k < x$ với k = L…j.
  - Dãy 2: $a_k = x$ với k = (j + 1)…(i – 1).
  - Dãy 3: $a_k > x$ với k = i…R.
- Bước 3:
  - Nếu L < j (dãy 1 vẫn còn chứa nhiều hơn 1 phần tử) thì lặp lại bước 2 với R = j và giữ nguyên L.
  - Nếu i < R (dãy 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)).

VII. Radix Sort (Sắp xếp theo cơ số):

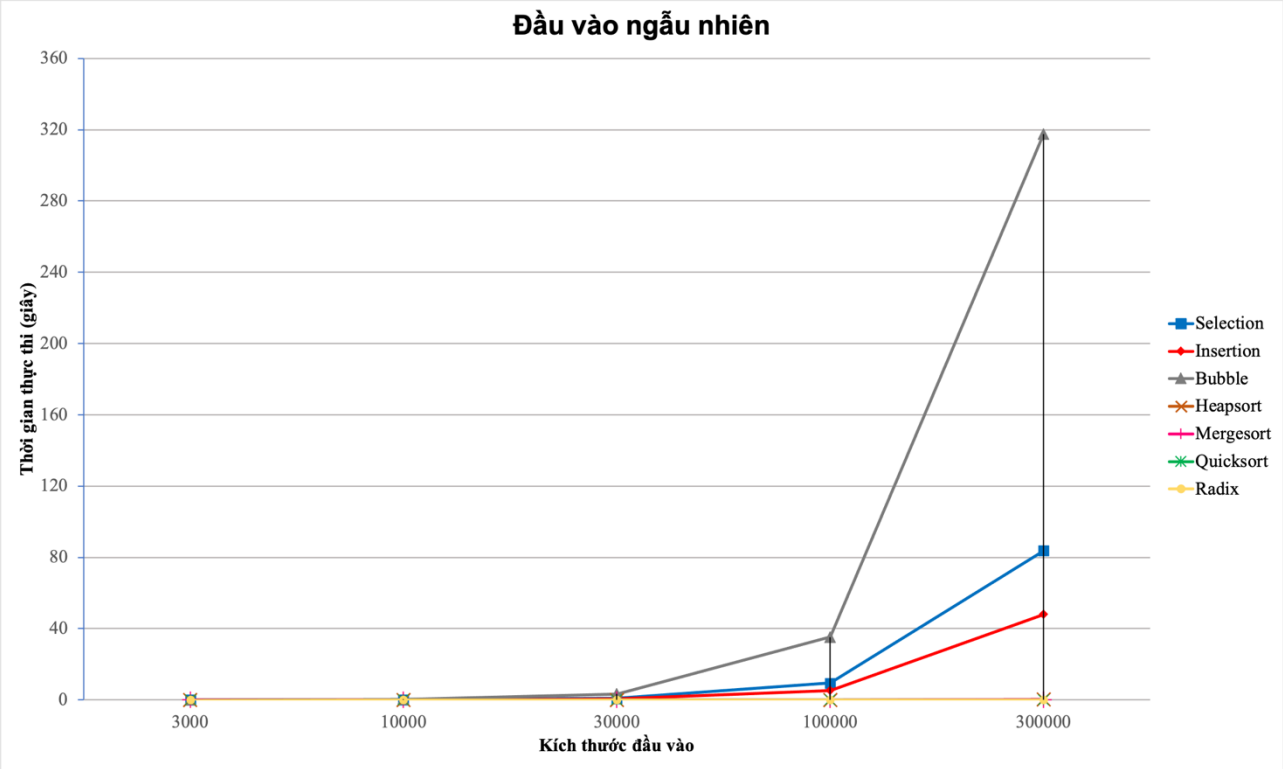
1. **Ý tưởng:** Phương pháp sắp xếp này không quá quan tâm đến việc so sánh giá trị của 2 phần tử mà dựa trên nguyên tắc phân loại thư của bưu điện. Cụ thể, để chuyển một khối lượng thư lớn đến tay người nhận. Bưu điện sẽ phân loại thư thuộc các tỉnh thành cụ thể và gửi đến tỉnh thành đó. Từ đó, các 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 dãy số 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 cột (có thể thay bằng một mảng chứa 10 hàng đợi (queue) để tiết kiệm bộ nhớ). 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:**
   - Độ phức tạp về thời gian: k số chữ số của phần tử lớn nhất.
     - 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).

B. THỰC NGHIỆM:

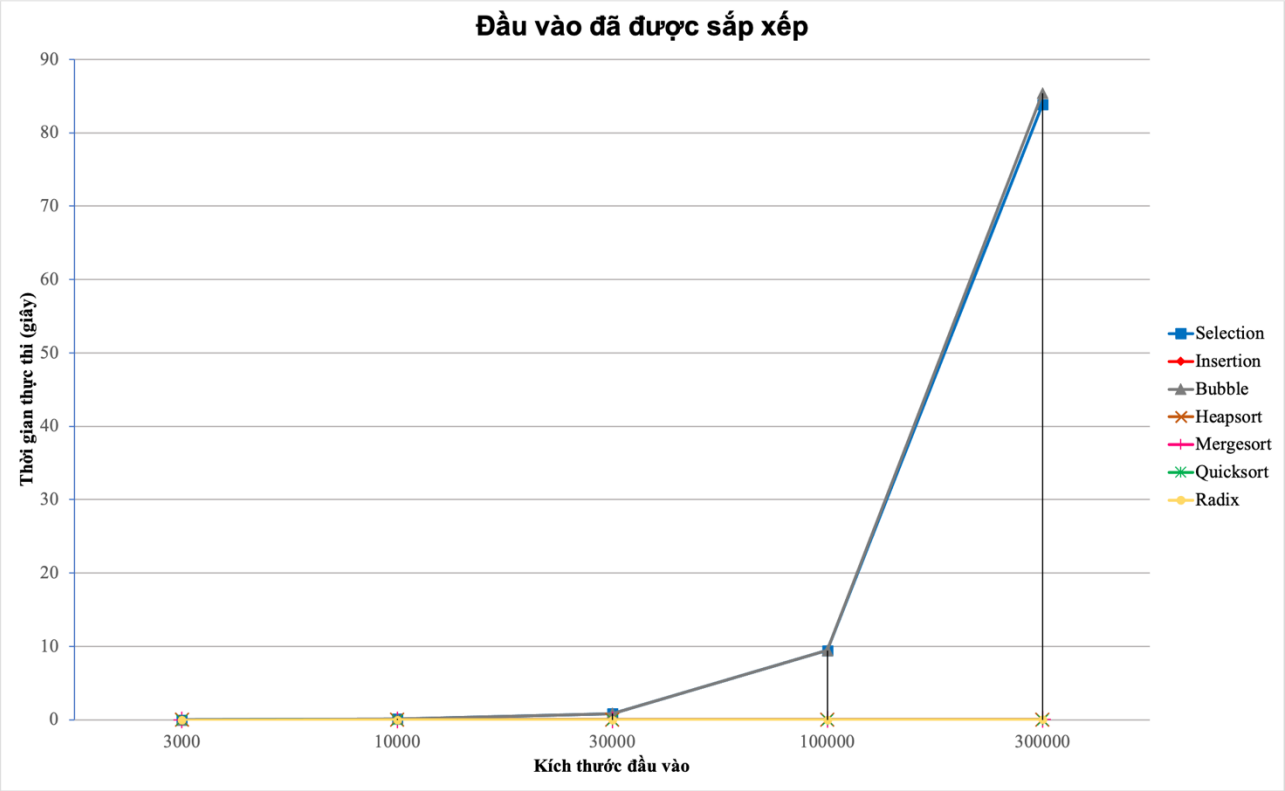
I. Đầu vào được ngẫu nhiên:



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: Radix Sort (sắp xếp theo cơ số). Vì độ phức tạp về thời gian của Radix Sort luôn luôn là O(kn) (k là số chữ số của phần tử lớn nhất) mà k < $\log_2$(kích thước đầu vào) trong tất cả kích thước đầu vào nên O(kn) < O(nlog(n)).
  - 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 Radix Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{0{,}019571 - 0{,}00017}}{0{,}019571 - 0{,}00017} \approx 789056711$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{317{,}550301 - 0{,}02831}}{317{,}550301 - 0{,}02831} \approx 3$ (phần tử / s^2).

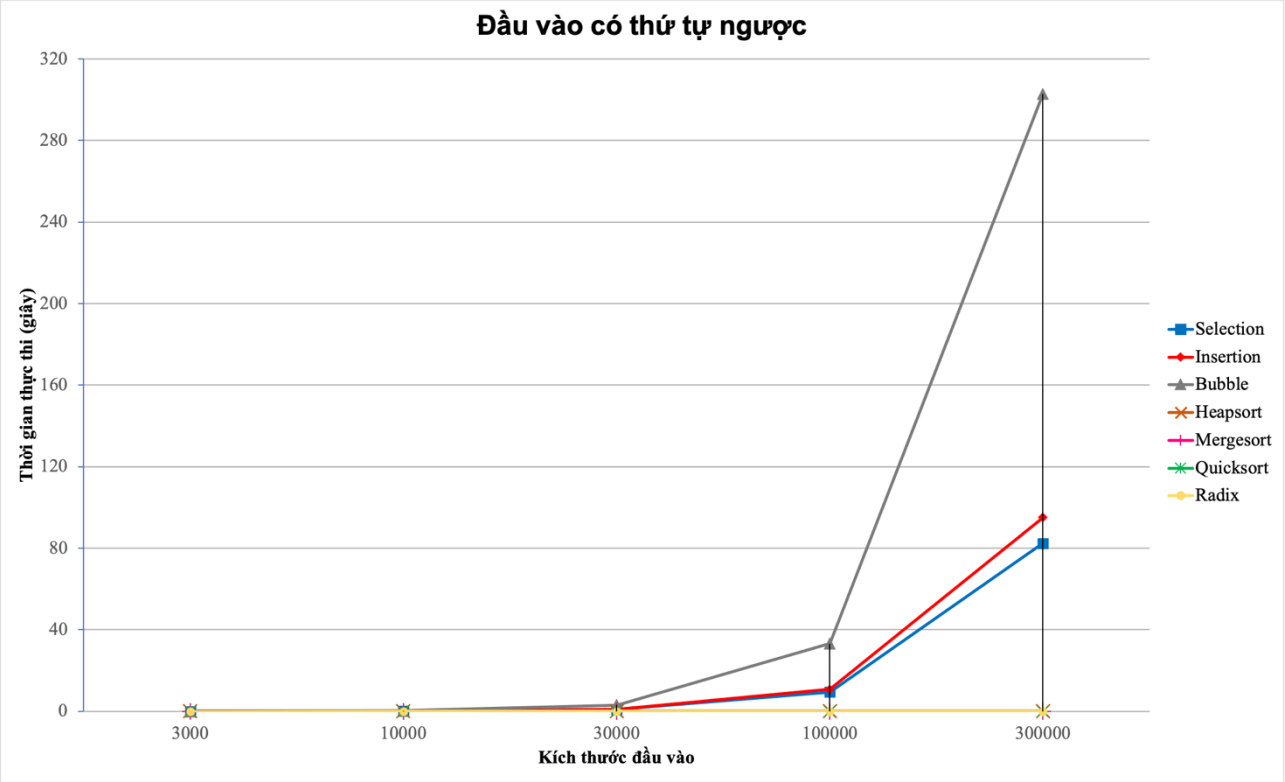
II. Đầu vào đã được sắp xếp:



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: Insertion Sort (sắp xếp chọn). 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 qua từng 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 Insertion Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{0{,}000889 - 0{,}000009}}{0{,}000889 - 0{,}000009} \approx 383522727273$ (phần tử / s^2).
  - Gia tốc của thuật toán Selection Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{83{,}845561 - 0{,}008066}}{83{,}845561 - 0{,}008066} \approx 42$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{85{,}36559 - 0{,}008443}}{85{,}36559 - 0{,}008443} \approx 41$ (phần tử / s^2).

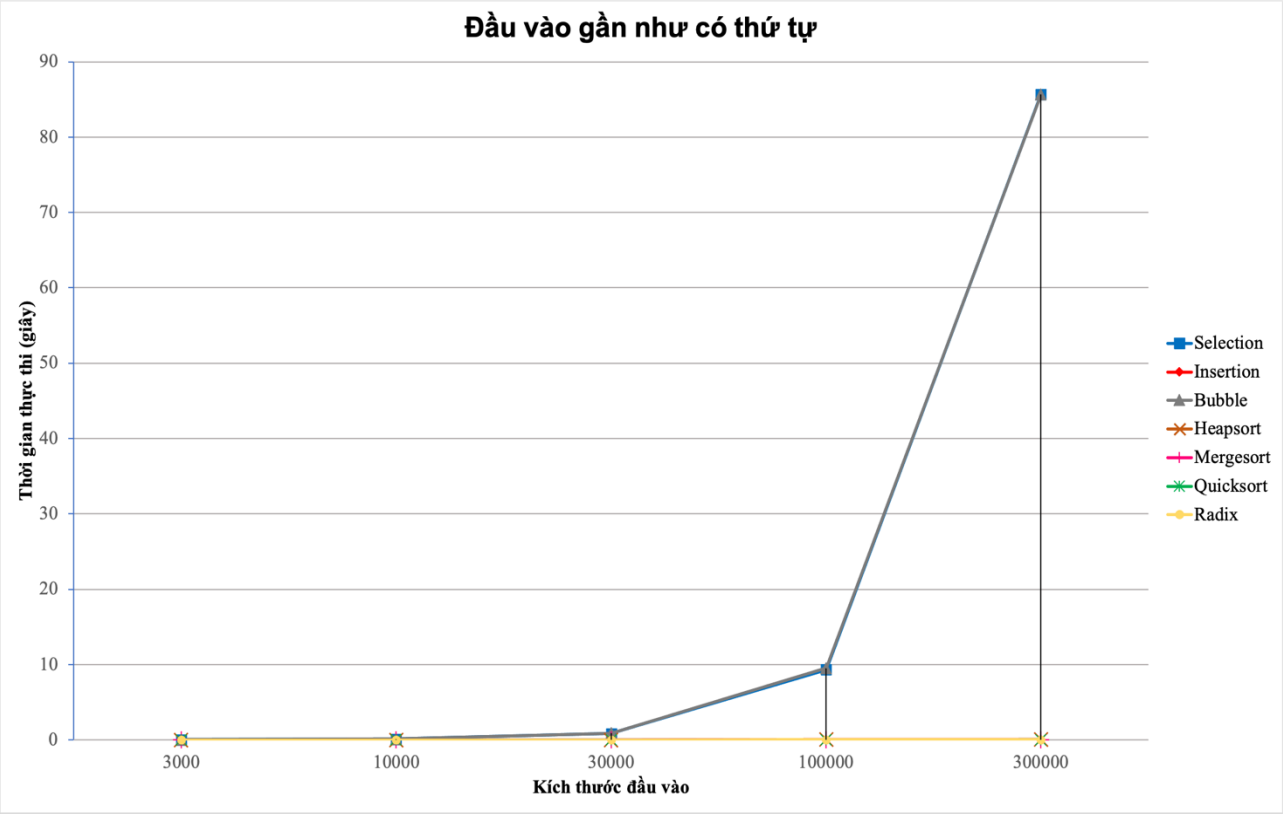
III. Đầu vào có thứ tự ngược:



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: Quick Sort (sắp xếp nhanh). Vì chỉ sau một lần phân hoạch thì mảng ban đầu đã được sắp xếp theo thứ tự.
  - Thuật toán chậm nhất trong tất cả kích thước đầu vào: Bubble Sort (sắp xếp nổi bọt). Đây là trường hợp xấu nhất của thuật toán này. Vì khi thực hiện một phép so sánh thì đồng thời nó sẽ thực hiện một phép hoán vị.
  - Gia tốc của thuật toán Quick Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{0{,}009448 - 0{,}00007}}{0{,}009448 - 0{,}00007} = 3377038350$ (phần tử / s^2).
  - Gia tốc của thuật toán Bubble Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{302{,}85627 - 0{,}029674}}{302{,}85627 - 0{,}029674} \approx 3$ (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:
    - Insertion Sort (sắp xếp chọn): Vì đây là trường hợp gần như tốt nhất của thuật toán với độ phức tạp về thời gian xấp xỉ O(n). Thuật toán chỉ cần duyệt qua từng phần tử và chèn lại một vài phần tử sai vị trí vào đúng vị trí với thời gian thực hiện việc dời phần tử qua trái là hằng số.
    - Quick Sort (sắp xếp nhanh): Vì chỉ cần sau vài lần phân hoạch thì các phần tử ở sai vị trí đã được đưa về lại vị trí đúng nhất.

    ➔ Sự chênh lệch của hai thuật toán này không nhiều và còn phụ thuộc vào vị trí và số lượng phần tử bị đặt sai vị trí.
  - 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 Insertion Sort:
    - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{0{,}005047 - 0{,}000052}}{0{,}005047 - 0{,}000052} = 11903795688$ (phần tử / s^2).

- Gia tốc của thuật toán Quick Sort:
  - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{0{,}009448 - 0{,}000070}}{0{,}009448 - 0{,}000070} \approx 3377038350$ (phần tử / s^2).
- Gia tốc của thuật toán Selection Sort:
  - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{85{,}688091 - 0{,}008135}}{85{,}688091 - 0{,}008135} \approx 40$ (phần tử / s^2).
- Gia tốc của thuật toán Bubble Sort:
  - $a = \frac{v}{t} = \frac{\frac{300000 - 3000}{85{,}635152 - 0{,}008733}}{85{,}635152 - 0{,}008733} \approx 41$ (phần tử / s^2).

❖ Kết luận:

- Thuật toán chạy nhanh nhất: Quick Sort (sắp xếp nhanh).
- Thuật toán chạy chậm nhất: Bubble Sort (sắp xếp nổi bọt).
- Nhóm thuật toán ổn định: Selection Sort, Insertion Sort, Bubble Sort, Merge Sort, Radix Sort.

```
Before          : 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a

After:
- Selection Sort: 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a
- Insertion Sort: 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a
- Bubble Sort   : 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a
- Merge Sort    : 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a
- Radix Sort    : 3a 3b 3c 4a 5a 5b 6a 6b 7a 8a
```

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

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

```
Before      : 3a 3b 3c 6a 4a 5a 5b 6b 7a 8a

After:
- Heap Sort : 3b 3a 3c 4a 5b 5a 6b 6a 7a 8a
- Quick Sort: 3c 3b 3a 4a 5b 5a 6b 6a 7a 8a
```

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