ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

LAB 03

BÀI BÁO CÁO CÁC THUẬT TOÁN SẮP XẾP

NHÓM 16 - LỚP 21CTT5

❖ **Sinh Viên Thực Hiện:**

- Trần Minh Quang - MSSV: 21120539
- Lâm Hoàng Quốc - MSSV: 21120542
- Nguyễn Đặng Quốc - MSSV: 2112054
- Trần Quốc Thịnh - MSSV: 21120562

❖ **Giảng Viên Hướng Dẫn: Lê Đình Ngọc**

MỤC LỤC

I. MỘT SỐ LƯU Ý:

a. Kiểu sắp xếp:

- Trong đồ án này thì tất cả thuật toán đều sắp xếp mảng đầu vào thành mảng tăng dần.
- Cấu trúc Heap trong thuật toán Heap Sort là Max Heap.

b. Ký hiệu:

- a: Mảng đầu vào.
- n: Số lượng phần tử mảng đầu vào.
- a[i]: phần tử ở vị trí i của mảng a. Trong báo cáo này, vị trí bắt đầu của mảng là 0, vị trí cuối cùng trong mảng là n – 1.
- l/start: vị trí bắt đầu xét ở lần đầu tiên (thường là 0) ; r/end: vị trí kết thúc xét đầu tiên (thường là n – 1).
- root : gốc của cây nhị phân, là node trên cùng.
- Index (viết tắt là i): vị trí/ chỉ số trong mảng.
- minVal: giá trị phần tử nhỏ nhất trong mảng
- maxVal: giá trị phần tử lớn nhất trong mảng.

c. Giới hạn các biến:

- n: int và n <= 1,000,000.
- a[i]: int và 0 <= a[i].

d. Độ Phức Tạp Về Thời Gian:

Độ Phức Tạp Về Thời Gian kết luận cho một thuật toán là độ phức tạp của trường hợp tệ nhất có thể xảy ra khi chạy thuật toán.

e. Nhận Xét Thuật Toán:

Các nhận xét được rút ra từ những số liệu trong bảng thời gian chạy và số lần so sánh trong 4 bảng so sánh 11 thuật toán đo được.

f. Biểu đồ và bảng:

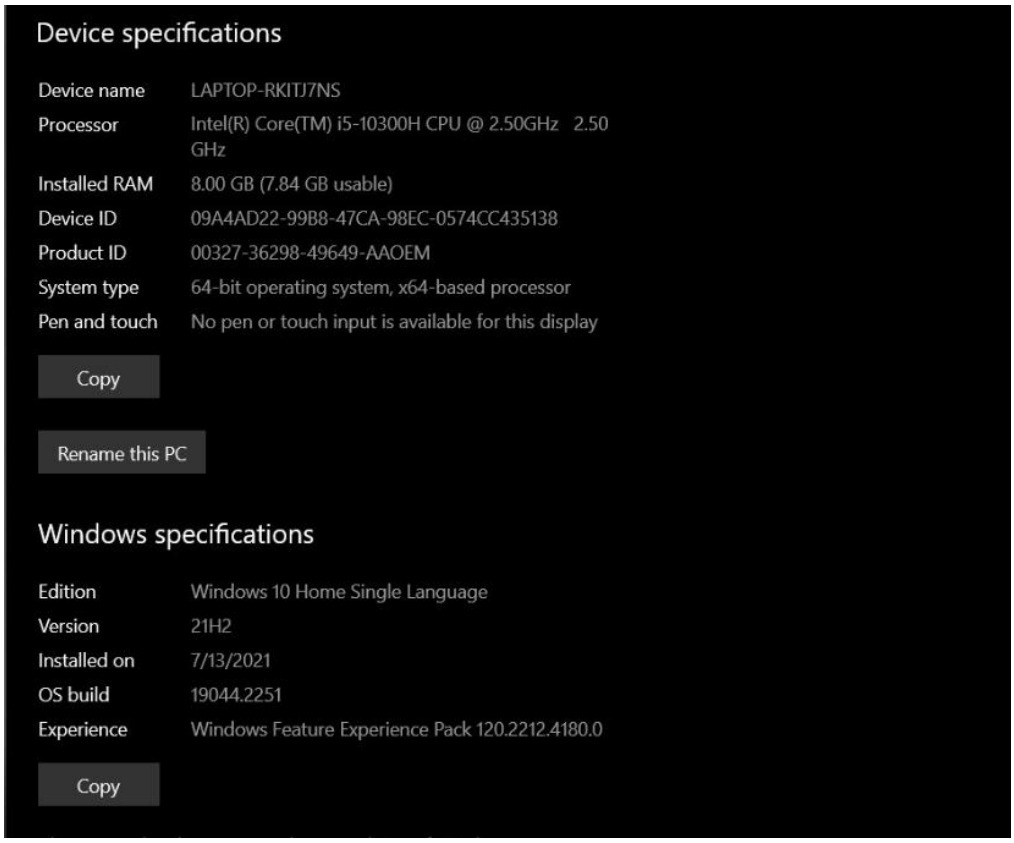
- Đơn vị đo thời gian là milisecond (ms).
- Đơn vị đo số lần so sánh là lần.

g. Cấu Hình Máy Tính Sử Dụng để đo thời gian:

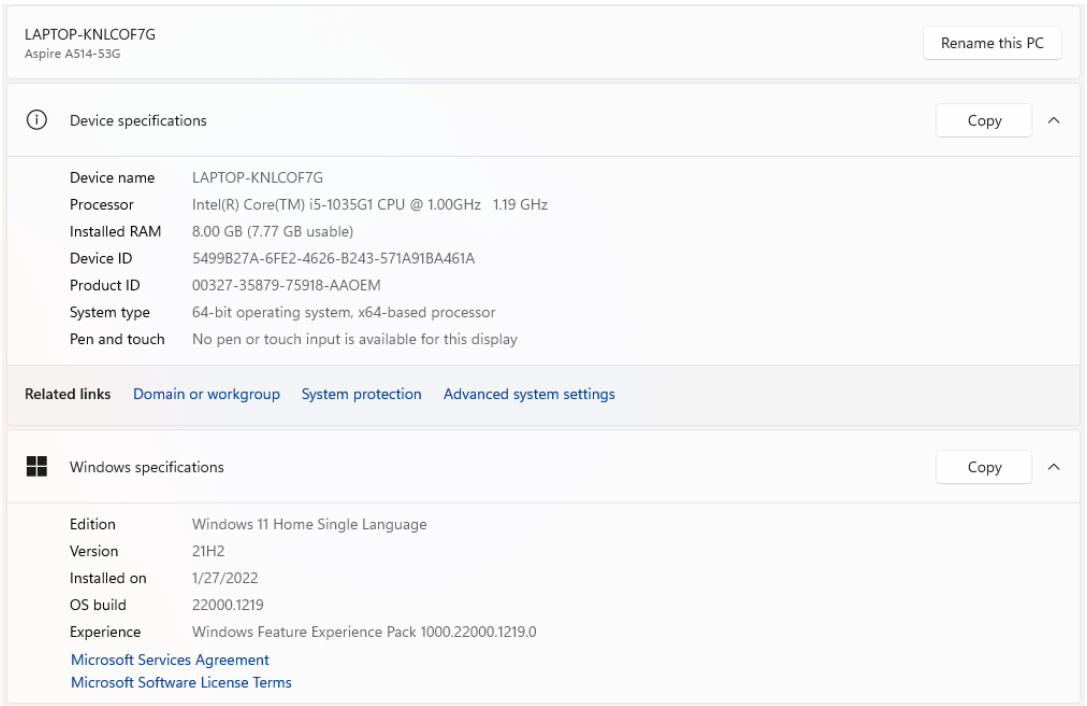
Để đo thời gian và ghi vào bảng thì 2 bạn phụ trách đo thời gian chạy thuật toán đã sử dụng 2 máy tính có cấu hình sau đây:

21120539 – 21120542 – 21120543 - 21120562

-21120539:

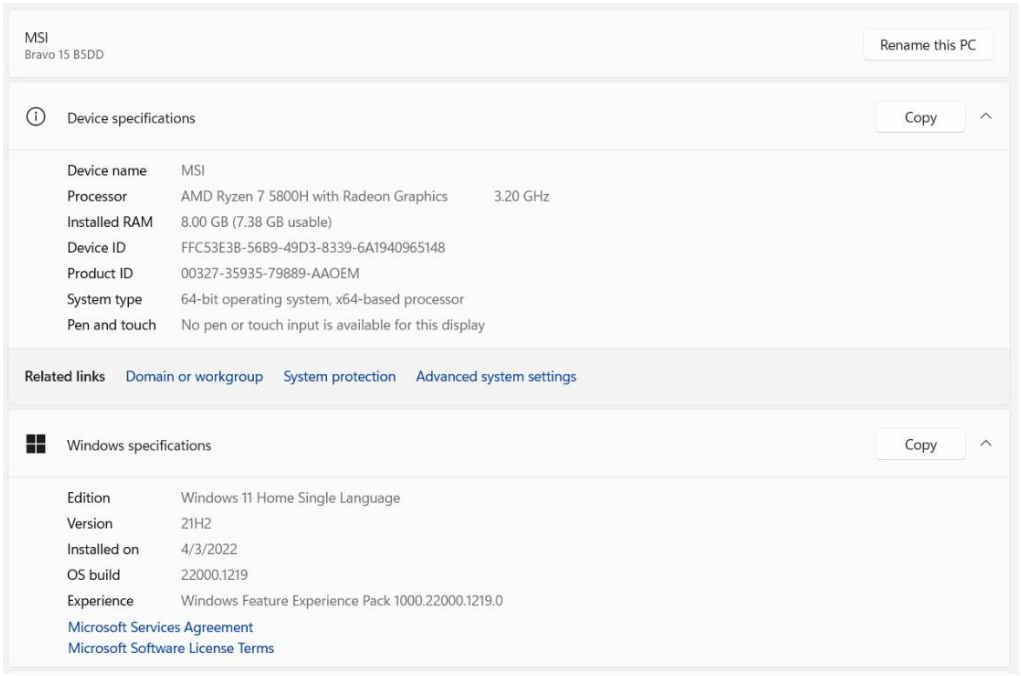


-21120542:

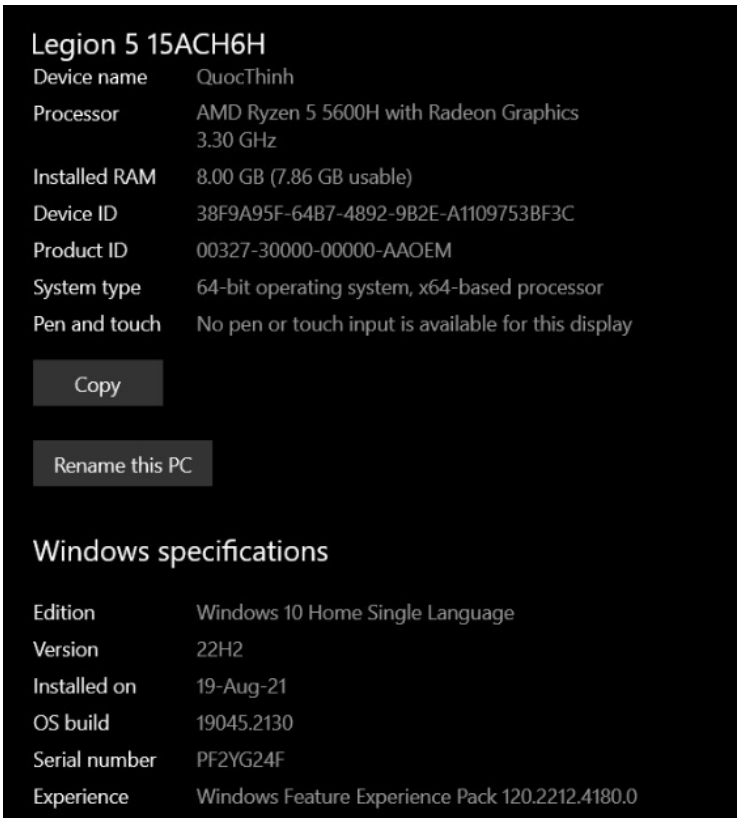


Ngoài ra, cũng có sử dụng máy tính của 2 bạn còn lại trong nhóm để chạy tính thời gian thuật toán và tham khảo số liệu tính được trên 2 máy đó. Đây là cấu hình máy tính của 2 bạn còn lại:

-21120543:



-21120562:



II. TRÌNH BÀY THUẬT TOÁN

1. Bubble Sort

a. Ý tưởng

- Giả sử cần sắp xếp tăng dần một danh sách có n phần tử a_0, a_1, a_2,…,a_{n-1}.
- Xuất phát từ cuối danh sách, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về đúng vị trí theo thứ tự tăng dần. Cuối cùng, phần tử đầu tiên trong danh sách sẽ là phần tử nhỏ nhất.
- Sau đó, sẽ không xét đến phần tử đầu tiên đó nữa ở bước tiếp theo. Do vậy, ở lần xử lí thứ i, phần tử đầu được xét sẽ có vị trí là i.
- Lặp lại các bước xử lý trên cho đến khi không còn cặp phần tử nào để xét.

b. Mã giả

- Bước1: i = 0; //lần xử lý đầu tiên
- Bước 2 :j = n-1; //duyệt từ cuối dãy ngược về vị trí i

Trong khi (j > i) xét trường hợp: Nếu a[j]<a[j-1] thì đổi chỗ a[j] và a[j-1].

j = j-1;

- Bước 3: i = i+1; //lần xử lý kế tiếp

Nếu i = n-1 -> hết dãy -> Dừng.

Ngược lại -> Lặp lại Bước 2

c. Nhận xét

- Thời gian: O(n), $O(n^2)$
- Bộ nhớ: O(1)
- Độ ổn định: Có

2. Shaker Sort

a. Ý tưởng

- Shaker Sort cũng dựa trên nguyên tắc đổi chỗ trực tiếp nhưng tìm các khắc phục nhược điểm của Bubble Sort. Trong mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau.
- Lượt đi: đẩy phần tử nhỏ về đầu mảng
- Lượt về: đẩy phần tử lớn về cuối mảng
- Ghi nhận lại những đoạn đã sắp xếp nhằm tiết kiệm các phép so sánh thừa

b. Mã giả

- Bước 1: l=0; r=n-1; //Đoạn l->r là đoạn cần được sắp xếp
  k=n; //ghi nhận vị trí k xảy ra hoán vị sau cùng để làm cơ sơ thu hẹp đoạn l->r
- Bước 2:

(*)

j=r; //đẩy phần tử nhỏ về đầu mảng
Trong khi j>l
nếu a[j]<a[j-1] thì {swap (a[j], a[j-1]): k=j;}
j--;
l=k; //loại phần tử đã có thứ tự ở đầu dãy

(**)

j=l
Trong khi j<r
nếu a[j]>a[j+1] thì {swap (a[j], a[j+1]); k=j;}
j++;
r=k; //loại phần tử đã có thứ tự ở cuối dãy

- Bước 3: Nếu l<r lặp lại bước 2
  Ngược lại: dừng

c. Nhận xét

- Thời gian: O(n), $O(n^2)$
- Bộ nhớ: O(1)
- Độ ổn định: Có

3. Radix Sort

a. Ý tưởng

- Trước tiên, ta giả sử mỗi phần tử a_i trong dãy $a_1, a_2, ..., a_n$ là một số nguyên có tối đa m chữ số.
- Phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm,…

b. Mã giả

- Bước 1: // k cho biết chữ số dùng để phân loại hiện hành.

  $k = 0$; // $k = 0$: hàng đơn vị; $k = 1$: hàng chục;…

- Bước 2: // Tạo các lô chứa các loại phần tử khác nhau.

  Khởi tạo 10 lô $B_0, B_1,…, B_9$ rỗng;

- Bước 3: For $i = 1..n$ do

  Đặt a_i vào lô B_t với t = chữ số thứ k của a_i;

- Bước 4: Nối $B_0, B_1,…, B_9$ lại theo đúng trình tự thành a.
- Bước 5: $k = k + 1$.

  Nếu $k < m$: lặp lại bước 2.

  Ngược lại: Dừng

c. Nhận xét

- Thời gian: O(n)
- Chi phí bộ nhớ: O(1)
- Độ ổn định: Có

4. Heap Sort:

a. Ý tưởng:

- *Cấu Trúc Binary Heap:*

- Cấu trúc này là một dạng đặc biệt của cây nhị phân hoàn chỉnh, trong đó node con được so sánh với node cha và được sắp xếp một cách phù hợp. (Cây nhị phân hoàn chỉnh là cây có đầy đủ con ngoại trừ các node cuối)
- Thuật toán Heap Sort trong đồ án này sử dụng cấu trúc Max Heap: khóa của node cha luôn lớn hơn 2 node con.

- *Ý tưởng thuật toán Heap Sort:*

- Ta coi dãy cần sắp xếp là một cây nhị phân hoàn chỉnh, sau đó hiệu chỉnh cây thành dạng cấu trúc heap (vun đống)
- Dựa vào tính chất của cấu trúc heap, ta có thể lấy được ra phần từ lớn nhất hoặc nhỏ nhất của dãy, phần tử này chính là gốc của heap. Giảm số lượng phần tử của cây nhị phân và tái cấu trúc heap cho đến khi mảng được sắp xếp hoàn toàn thì dừng lại.
- Đưa phần tử đỉnh heap về đúng vị trí của dãy ở cuối mảng, sau đó giảm số lượng phần tử của mảng (không xét tới phần tử cuối nữa)
- Tái cấu trúc heap và lặp lại việc lấy phần tử gốc của cấu trúc heap cho tới khi danh sách ban đầu chỉ còn 1 phần tử. Đưa phần tử này về đúng vị trí và kết thúc thuật toán. (Ta phải thực hiện tái cấu trúc heap, vun lại đống bởi vì sau khi lấy ra phần tử gốc heap, cấu trúc heap cũ không còn đúng nữa.)

b. Mã Giả:

Thuật toán Heap Sort trải qua 2 giai đoạn:

Giai đoạn 1: Hiệu chỉnh dãy số ban đầu thành max heap.

Giai đoạn 2: Sắp xếp dãy số dựa trên max heap:

- Bước 1: + Đưa phần tử lớn nhất về vị trí đúng ở cuỗi dãy:

+ r = n – 1 ;

+ Hoán vị (a_0, a_r);

- Bước 2: + Loại bỏ phần tử lớn nhất ra khỏi heap: r = r – 1;

+ Hiệu chỉnh phần còn lại của dãy tử a_0 , a_1, … , a_r thành một max heap

- Bước 3: + Nếu r >= 0 (heap còn phần tử): Lặp lại B2.

+ Ngược lại: dừng thuật toán.

c. **Nhận xét**:

- Thời gian: O(nlogn)
- Chi Phí Bộ Nhớ: O(1)
- Độ Ổn Định: Không

- Thời gian xử lý trung bình của Heap Sort khá nhanh vì nằm trong nhóm các thuật toán có độ phức tạp là O(nlogn)
- Trường hợp chạy chậm nhất của Heap Sort là bộ dữ liệu ngẫu nhiên (-rand). Vì sẽ mất nhiều thời gian hơn để tạo max heap - đưa phần tử lớn nhất lên đỉnh heap. Lúc này Heap Sort có độ phức tạp là O(nlogn)

- Trường hợp chạy nhanh nhất của Heap Sort là bộ dữ liệu được sắp xếp ngược (sắp xếp giảm dần, -rev). Vì không cần tốn thời gian tạo Max Heap, với bộ dữ liệu này thì phần tử lớn nhất trong mảng đã ở trên đỉnh Max Heap rồi, ta chỉ cần đưa phần tử đó về đúng chỗ của nó trong mảng. Lúc này Heap Sort có độ phức tạp là O(n).
- Heap Sort không cần tạo mảng phụ mà chỉ sử dụng mảng đầu vào (cho cả việc tạo Max Heap và sắp xếp dữ liệu) nên chi phí của nó là O(1).

5. **Counting Sort:**

a. Ý tưởng:

Ý tưởng của counting sort là đếm số lượng lần xuất hiện của 1 giá trị trong mảng đầu vào, sau đó lưu nó vào vị trí có giá trị đó của 1 mảng con. Sau đó điền lại giá trị của mảng đầu vào dựa trên các phần tử của mảng con.

b. Mã giả:

- Bước 1: Tìm phần tử lớn nhất (maxVal) từ mảng đầu vào
- Bước 2: Khởi tạo mảng con là count có độ dài là maxVal+1. Mảng con này được sử dụng để lưu trữ số lượng các phần tử trong mảng, (index có giá trị từ 0 đến max).
- Bước 3: Đếm số lần xuất hiện của 1 phần tử trong mảng gốc rồi lưu số lần xuất hiện đó vào mảng con count
- Bước 4: Sắp xếp mảng đầu vào bằng cách lấy số lần xuất hiện của từng giá trị trong mảng con count viết thành dãy rồi ghi đè vào mảng đầu vào

c. Nhận xét:

- Thời gian: O(maxVal) (Vì có vòng for đơn không lồng chạy từ i = 0 đến i = maxVal). Trong một số trường hợp thì có thể xấp xỉ bằng O(n).
- Chi Phí Bộ Nhớ: O(maxVal)
- Độ Ổn Định: Có

- Thời gian xử lý trung bình của Counting Sort là nhanh nhất trong 11 thuật toán tìm hiểu trong đồ án này, độ phức tạp thời gian là O(maxVal) với mọi trường hợp.
- Trường hợp chạy chậm nhất của Counting Sort là bộ dữ liệu ngẫu nhiên (-rand). Vì tốn thời gian để thuật toán biết xem là phần tử đang xét trong mảng đầu vào thuộc vị trí nào trong mảng con count và sau đó tăng giá trị tại vị trí đó lên 1 đơn vị.

- Trường hợp chạy nhanh nhất của Counting Sort là bộ dữ liệu đã được sắp xếp tăng dần (-sorted).
- Counting Sort sử dụng 1 hàm đếm phụ (count) có số lượng phần tử là maxValue + 1. Với maxValue là giá trị lớn nhất trong mảng đầu vào. Nên độ phức tạp về Chi Phí Bộ Nhớ của nó là O(maxValue+1) = O(maxValue).
- Counting Sort có một nhược điểm lớn là chỉ có thể sử dụng cho mảng có phần tử không âm.

6. **Flash Sort:**

- Là một biến thể của Bucket sort. Trong báo cáo này sẽ sử dụng "lớp" thay cho "xô " để chỉ những tập hợp phần tử được chia nhỏ ra từ một mảng đầu vào gốc.
- Khác với Bucket Sort, Flash Sort chỉ cần biết số lượng phần tử trong 1 lớp chứ không cần biết chính xác những phần tử nằm trong 1 lớp là phần tử nào.

a. Ý tưởng:

- Flash sort là một thuật toán áp dụng việc phân phối dữ liệu vào các phân lớp khác nhau sao cho với mỗi phần tử của phân lớp trước đều nhỏ hơn mỗi phần tử của phân lớp sau
- Sau khi chia xong thì sử dụng hoán vị toàn cục cho các phần tử trong các lớp.Hoán vị ở đây là đưa các phần tử nằm ở vị trí lớp chưa đúng về đúng lớp của phần tử đó, nếu thêm 1 phần tử vào lớp này thì phải đẩy 1 phần tử sang lớp kia. Sau khi hoán vị toàn cục thì ta sẽ thỏa được điều kiện "mỗi phần tử của phân lớp trước đều nhỏ hơn mỗi phần tử của phân lớp sau" ở trên.
- Cuối cùng sau khi hoàn thành bước hoán vị thì sử dụng Insertion Sort sắp xếp lại các phần tử trong các lớp sẽ ra được mảng đã được sắp xếp.

b. Mã giả:

(Tất cả công thức sử dụng đã được chứng minh)

- Bước 1: Tính số lớp chia ra: nCLass = n * 0.45.
- Bước 2: Tính phần tử c1 để xác định lớp mà phần tử a[i] được chia vào, c1 = (nClass – 1)/(maxVal – minVal)
- Bước 3: Thêm các phần tử vào các lớp:

+Vị trí lớp mà phần tử a[i] được thêm vào là k, k = (a[i] – minVal)*c1

+Đếm số phần tử trong từng lớp

- Bước 4: + Tính vị trí kết thúc của lớp thứ j theo công thức class[j] = class[j] + class[j-1]

(1<= j <= n-1)

- Bước 5: Hoán vị cục bộ: di chuyển các phần tử nằm ở lớp chưa đúng về đúng lớp của mình.
- Bước 6: Sắp xếp cục bộ: sắp xếp lại các phần tử trong phạm vi của từng lớp.

c. Nhận xét:

- Thời Gian: O(n^2)
- Bộ Nhớ O(nClass)
- Độ Ổn Định: Không

- Thời gian xử lý trung bình của Flash Sort nhanh chỉ thua Counting Sort trong 11 thuật toán tìm hiểu trong đồ án này.
- Trường hợp chạy chậm nhất của Flash Sort là bộ dữ liệu gần như sắp xếp (-nsorted). Vì tốn thời gian để chia các phần tử nằm rải rác không đều nhau vào các lớp và sau đó hoán vị những phần tử này. Lúc này Flash Sort có thể có độ phức tạp là O(n^2), (nhưng trường hợp này là rất hiếm xảy ra).
- Trường hợp chạy nhanh nhất của Counting Sort là bộ dữ liệu đã được sắp xếp tăng dần (-sorted). Lúc này Flash Sort có độ phức tạp là O(n)
- Flash Sort sử dụng mảng con class để lưu số lượng các phần tử nằm trong từng phân lớp. Class có chi phí bộ nhớ là O(nClass).
- Thuật toán Insertion Sort ở bước Sắp Xếp Cục Bộ có độ phức tạp là O(1) vì số lượng phần tử trong các lớp thường sẽ rất nhỏ và nhờ bước Hoán Vị Toàn Cục Flash Sort đã đưa mảng về trường hợp tốt nhất của Insertion Sort.

7. Selection Sort

a. Ý tưởng:

Thuật toán selection sort sắp xếp một mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất(giả sử với sắp xếp mảng tăng dần) trong đoạn chưa được sắp xếp và đổi chỗ phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng). Thuật toán sẽ chia mảng làm 2 mảng con

- Một mảng con đã được sắp xếp
- Một mảng con chưa được sắp xếp

Tại mỗi bước lặp của thuật toán, phần tử nhỏ nhất ở mảng con chưa được sắp xếp sẽ được di chuyển về đoạn đã sắp xếp.

b. Mã giả:

Cho một mảng chưa sắp xếp A gồm N phần tử:

- Bước 1: Cho i = 0 (i < N-1)
- Bước 2: Tìm phần tử A[min] nhỏ nhất trong dãy hiện hành từ A[i] đến A[N-1]
- Bước 3: Hoán vị A[min] và A[i]
- Bước 4: i = i + 1
  - Nếu i < N-1 thì Lặp lại Bước 2
  - Ngược lại: Dừng. (Mảng đã sắp xếp xong)

c. Nhận xét:

- Thời gian:
  + Tệ nhất: $O(n^2)$
  + Trung bình: $O(n^2)$
  + Tốt nhất: $O(n^2)$
- Không gian: O(1)
- Độ ổn định: Không

8. **Insertion Sort**

a. Ý tưởng:

Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

- Khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử(phần tử đầu tiên, phần tử có chỉ số 0)
- Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử.
- Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

b. Mã giả:

Cho một mảng chưa sắp xếp A gồm N phần tử:

- Bước 1: i = 1 (i ≤ N-1) (giả sử A[0] đã được sắp xếp)
- Bước 2: key = A[i]. Tìm vị trí pos thích hợp trong đoạn A[0] đến A[i]
- Bước 3:

Nếu pos = i thì đoạn từ A[0] đến A[i] đã được sắp xếp
Nếu pos < i thì dịch chuyển tất cả các phần tử lớn hơn i sang phải một vị trí

- Bước 4: A[pos] = key (đoạn A[0] đến A[i] đã được sắp xếp)
- Bước 5: i = i + 1

Nếu i ≤ N – 1: Lặp lại bước 2
Ngược lại: Dừng (Mảng đã sắp xếp xong)

c. Nhận xét:

- Thời gian:
  + Tệ nhất: $O(n^2)$
  + Trung bình: $O(n^2)$
  + Tốt nhất: O(n)
- Không gian: O(1)
- Độ ổn định: Có

9. **Merge sort:**

a. Ý tưởng:

Chia danh sách gồm n phần tử chưa được sắp xếp thành n danh sách con, mỗi danh sách chứa một phần tử (danh sách một phần tử được coi là đã sắp xếp).

Liên tục hợp nhất các danh sách con để tạo ra các danh sách con được sắp xếp mớ cho đến khi chỉ còn lại một danh sách. Đây sẽ là danh sách được sắp xếp.

b. Mã giả:

- Bước 1: Chia dãy cần sắp xếp thành 2 dãy con

Từ dãy con thu được lại tiếp tục chia thành 2 dãy con nhỏ hơn nữa

Quá trình phân chia tiếp tục cho đến khi thu được dãy con chỉ còn duy nhất 1 phần tử.

- Bước 2: Hòa nhập 2 dãy con nhỏ nhất thành dãy con lớn hơn sao cho đúng thứ tự

Từ hai dãy con lớn hơn lại hòa nhập thành 2 dãy con lớn hơn nữa….

Quá trình hòa nhập cứ tiếp tục như vậy cho đến khi thu được dãy số ban đầu đã được sắp xếp.

c. Nhận xét:

Thời gian: O(nlogn)
Không gian: O(n)
Độ ổn định: Có

10. **Shell sort**

a. Ý tưởng:

- Khởi tạo giá trị h.
- Chia list thành các sublist nhỏ hơn tương ứng với h.
- Sắp xếp các sublist này bởi sử dụng sắp xếp chèn (Insertion Sort)
- Lặp lại cho tới khi list đã được sắp xếp

b. Giải thuật:

- Bước 1: Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau h vị trí.
- Bước 2: Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối.
- Bước 3: Giảm khoảng cách h để tạo thành các dãy con mới. Dừng khi h=1.

Dãy h_k cần thỏa mãn 2 điều kiện:

+ $h_{st} > h_{st+1}$

+ $h_k = 1$

c. Nhận xét:

- Thời gian: O(nlogn), $O(n^2)$
- Không gian: O(1)
- Độ ổn định: Không

11. **Quick sort**

a. Ý tưởng:

- Chọn điểm đánh dấu cho mảng, ở đây mình sẽ chọn điểm đánh sấu là số cuối cùng của mảng.
- Tạo hai biến là trái và phải để trỏ tới bên trái và bên phải của danh sách.
- Thực hiện so sánh các phần tử với điểm đánh dấu. Nếu phần tử nhỏ hơn điểm đánh dấu thì dịch chuyển qua bên trái và ngược lại.
- Sau khi dịch chuyển thực hiện công việc sắp xếp các phần tử trong mảng con mới, trước khi tiếp tục phân đoạn tiếp theo.

b. Mã giả:

- Bước 1: Gán left là vị trí đầu danh sách bên trái, right là vị trí đầu danh sách bên phải.

- Bước 2**:** Nếu left >= right thì kết thúc (danh sách chỉ có 1 phần tử). Ngược lại thì:
- Bước 2.1: Chọn tùy ý một phần tử a[k] trong danh sách (left <= k <= right): x=a[k]; i=left; j=right;
- Bước 2.2: Tìm kiếm và đổi chổ cặp phần tử a[i], a[j] nằm sai chổ

Trong khi (a[i]<x) i++;

Trong khi (a[j]>x) j- -;

Nếu i<=j thì đổi chổ a[i], a[j].

i++; j- -;

- Bước 2.3: Nếu i<=j thì lặp lại bước 2.2.
- Bước 3:

Nếu left<j thì gọi đệ quy để phân hoạch danh sách con a[left],…,a[j].

Nếu i<right thì gọi đệ qui để phân hoạch danh sách con a[i],…,a[right].

c. Nhận xét:

- Thời gian: O(nlogn), $O(n^2)$
- Không gian: O(logn)
- Độ ổn định: Không

III. KẾT QUẢ THỰC THU VÀ NHẬN XÉT:

1. Bảng đo:

a) Với bộ dữ liệu ngẫu nhiên:

RANDOMIZE						
Data Size:	10.000		30.000		50.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	55,68	100.009.999	506,25	900.029.999	1.371,62	2.500.049.999
insertion	31,08	49.965.158	315,74	451.468.312	789,03	1.239.785.381
bubble	1045	100.009.999	8.873,00	900.029.999	25.099,00	2.500.049.999
shaker	926,00	66.172.897	8.664,00	599.733.680	24.223,00	1.664.751.498
shell	1,70	521.595	6,42	1.926.643	12,00	3.911.835
heap	7,44	569.240	22,45	1.919.400	48,27	3.368.474
merge	3,94	578.048	13,37	1.921.528	21,95	3.355.382
quick	1,65	250.787	6,28	805.170	11,46	1.448.192
counting	0,13	60.002	0,33	180.002	0,55	282.770
radix	0,76	140.056	2,95	510.070	4,96	850.070
flash	0,28	82.304	0,89	249.067	1,44	407.982
RANDOMIZE						
Data Size:	100.000		300.000		500.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	5.571,15	10.000.099.999	49.787,90	90.000.299.999	142.069,00	250.000.499.999
insertion	3.466,39	4.989.067.836	28.877,70	45.003.071.384	82.006,50	124.805.287.943
bubble	104.729	10.000.099.999	928.745,00	90.000.299.999	2.832.260,00	250.000.499.999
shaker	98.651,00	6.658.220.085	865.359,00	60.090.104.292	2.685.010,00	166.857.724.129
shell	24,70	8.824.879	99,66	30.230.301	203,50	55.976.493
heap	137,43	7.188.481	458,02	23.690.726	392,64	41.131.490
merge	49,86	7.111.032	129,77	23.220.724	227,67	40.124.584
quick	23,11	3.055.345	72,30	10.035.790	117,03	17.629.346
counting	1,22	532.770	3,53	1.532.770	6,72	2.532.770
radix	10,40	1.700.070	36,12	5.100.070	54,01	8.500.070
flash	4,53	756.145	13,32	2.366.755	34,35	3.925.985

b) Với bộ dữ liệu đã được sắp xếp (tăng dần):

SORTED						
Data Size:	10.000		30.000		50.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	54,19	100.009.999	506,83	900.029.999	1.401,37	2.500.049.999
insertion	0,02	29.998	0,06	89.998	0,10	149.998
bubble	66	100.009.999	554,00	900.029.999	1.532	2.500.049.999
shaker	12,00	20.002	35,70	60.002	69,57	100.002
shell	0,30	240.037	1,21	780.043	1,77	1.400.043
heap	4,35	606.457	14,40	2.020.674	25,02	3.544.753
merge	2,86	475.242	9,72	1.559.914	17,93	2.722.826
quick	0,38	161.659	1,24	531.788	2,04	923.557
counting	0,19	60.002	0,31	180.002	0,52	300.002
radix	0,92	140.056	3,05	510.070	5,35	850.070
flash	0,28	108.994	0,78	326.994	1,39	544.994
SORTED						
Data Size:	100.000		300.000		500.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	5.539,31	10.000.099.999	50.020,60	90.000.299.999	145.087,00	250.000.499.999
insertion	0,20	299.998	0,61	899.998	1,01	1.499.998
bubble	6.356,00	10.000.099.999	55.798	90.000.299.999	166.774,00	250.000.499.999
shaker	182.26	200.002	559,83	600.002	1.354,69	1.000.002
shell	3,89	3.000.045	14,11	10.200.053	23,33	17.000.051
heap	58,84	7.555.744	187,29	24.738.390	345,50	42.812.485
merge	39,27	5.745.658	110,41	18.645.946	183,93	32.017.850
quick	4,28	1.947.097	12,75	6.319.424	22,98	10.888.350
counting	1,71	600.002	4,05	1.800.002	6,95	3.000.002
radix	10,91	1.700.070	39,45	6.000.084	67,03	10.000.084
flash	2,90	1.089.994	7,86	3.269.994	12,92	5.449.994

c) Với bộ dữ liệu được sắp xếp ngược (giảm dần):

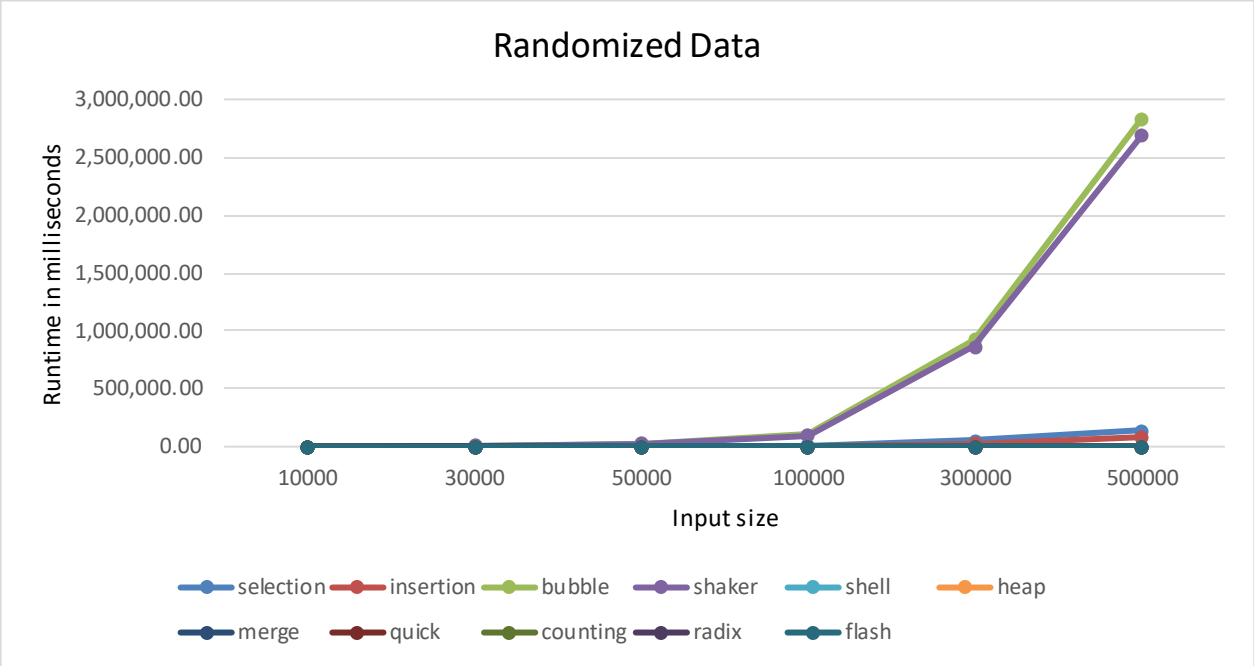
REVERSE SORTED						
Data Size:	10.000		30.000		50.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	58,30	100.009.999	539,14	900.029.999	1.475,53	2.500.049.999
insertion	62,51	100.009.999	612,15	900.029.999	1.627,55	2.500.049.999
bubble	1948	100.009.999	15.837,00	900.029.999	44.844,00	2.500.049.999
shaker	1.776,00	100.005.001	15.026,00	900.015.001	42.020,00	2.500.025.001
shell	0,46	365.157	1,62	1.194.027	3,21	2.194.603
heap	4,26	533.712	15,27	1.819.176	27,63	3.186.311
merge	2,81	466.442	9,73	1.543.466	17,09	2.683.946
quick	0,51	171.656	1,49	561.785	2,60	973.554
counting	0,16	60.002	0,45	180.002	0,71	300.002
radix	0,76	140.056	2,79	510.070	4,91	850.070
flash	0,24	93.753	0,72	281.253	1,14	468.753
REVERSE SORTED						
Data Size:	100.000		300.000		500.000	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	5.973,69	10.000.099.999	52.723,00	90.000.299.999	151.500,00	250.000.499.999
insertion	6.521,21	10.000.099.999	58.264,40	90.000.299.999	165.819,00	250.000.499.999
bubble	193.883,00	10.000.099.999	1.624.930,00	90.000.299.999	4.578.660,00	250.000.499.999
shaker	193.751,00	10.000.050.001	1.608.100,00	90.000.150.001	4.332.950,00	250.000.250.001
shell	5,95	4.689.165	21,48	15.201.813	42,55	25.857.555
heap	58,85	6.817.117	267,83	22.608.296	336,78	39.373.912
merge	37,85	5.667.898	108,38	18.408.314	183,99	31.836.410
quick	5,55	2.047.094	17,17	6.619.421	28,61	11.388.347
counting	1,85	600.002	3,92	1.800.002	6,31	3.000.002
radix	10,61	1.700.070	35,12	6.000.084	63,78	10.000.084
flash	2,39	937.503	7,41	2.812.503	12,48	4.687.503

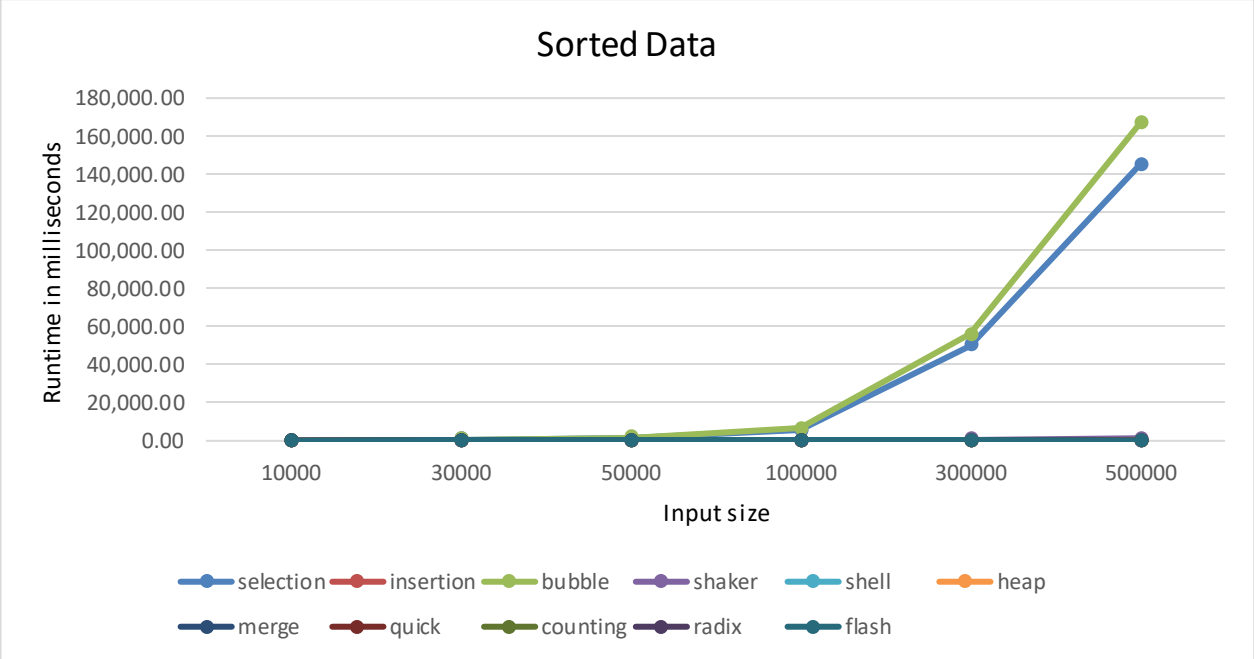
d)Với bộ dữ liệu gần như được sắp xếp:

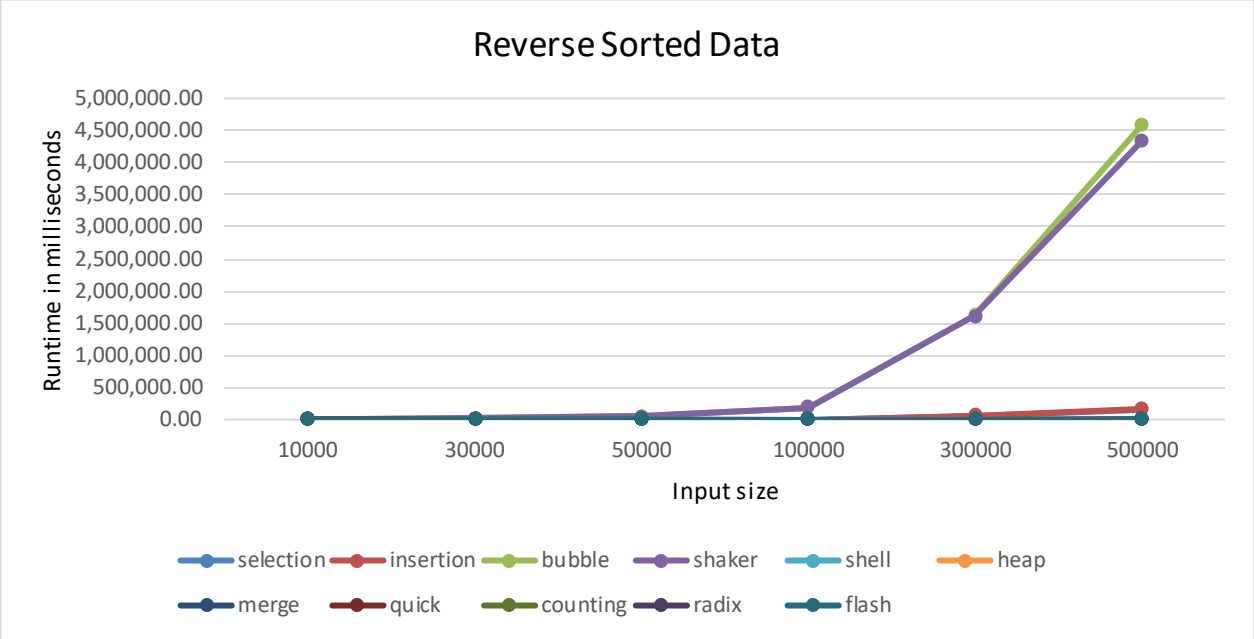
NEARLY SORTED						
Data Size:	10.000,00		30.000,00		50.000,00	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	55,19	100.009.999	512,07	900.029.999	1.373,82	2.500.049.999
insertion	0,18	222.266	0,56	654.350	0,50	641.822
bubble	63	100.009.999	562,00	900.029.999	1.541,00	2.500.049.999
shaker	2,00	155.474	7,00	606.708	9,00	571.964
shell	0,46	281.341	1,71	955.495	2,74	1.602.119
heap	4,94	605.860	15,16	2.020.443	28,39	3.544.277
merge	2,83	503.170	9,86	1.628.192	18,94	2.801.616
quick	0,42	161.691	1,19	531.816	1,97	923.601
counting	7,70	60.002	0,49	180.002	0,82	300.002
radix	0,91	140.056	3,36	510.070	6,05	850.070
flash	0,25	108.968	0,70	326.969	1,56	544.969
NEARLY SORTED						
Data Size:	100.000,00		300.000,00		500.000,00	
Resulting Statics	Running time	Comparisions	Running time	Comparisions	Running time	Comparisions
selection	5.710,45	10.000.099.999	50.762,40	90.000.299.999	143.313,00	250.000.499.999
insertion	0,67	871.698	1,05	1.471.010	1,32	1.894.826
bubble	6.356,00	10.000.099.999	55.536,00	90.000.299.999	173.756,00	250.000.499.999
shaker	13,00	707.441	25,00	1.216.214	56,00	1.586.567
shell	4,93	3.202.121	14,93	10.361.729	24,11	17.184.379
heap	52,59	7.555.720	193,97	24.738.215	342,89	42.812.540
merge	37,62	5.835.674	116,00	18.731.370	238,07	32.106.428
quick	4,13	1.947.141	13,35	6.319.452	22,41	10.888.378
counting	2,02	600.002	6,13	1.800.002	10,57	3.000.002
radix	11,14	1.700.070	38,75	6.000.084	63,15	10.000.084
flash	2,29	1.089.970	8,06	3.269.971	15,18	5.449.971

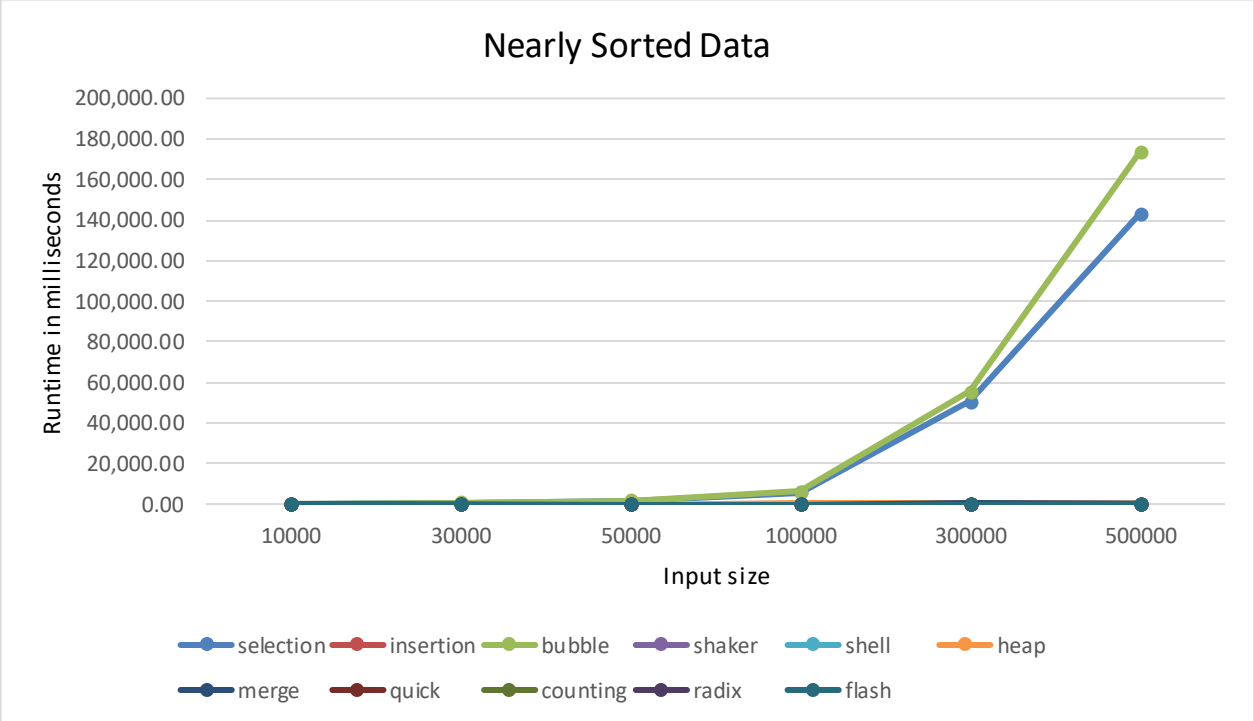
2. Biểu đồ:

a. Runtime









Nhận Xét Biểu Đồ:

- Với kích thước dữ liệu đầu vào nhỏ nhìn chung tốc độ chênh lệch của các thuật toán là không rõ để nhận thấy.

- Với mảng đã được sắp xếp, và gần như đã được sắp xếp thì Bubble Sort và Shaker Sort cho tốc độ nhanh hơn so với các trường hợp mảng khác do chi phí để biết được đây là mảng có thứ tự của 2 thuật toán trên là O(n).

- Với mảng gần như đã được sắp xếp thì Insertion Sort là sự lựa chọn tốt nhất do số phép hoán đổi phải thực hiện ít.

- Selection Sort cho tốc độ khá chậm trong đa số trường hợp do độ phức tạp luôn là $O(n^2)$, do đó Selection Sort chỉ nên dùng cho các trường hợp số lượng phần tử cần sắp xếp không quá nhiều.

- Với mảng gần như đã được sắp xếp thì Shaker Sort cho tốc độ nhanh hơn đáng kể so với Bubble Sort, do thu hẹp được khoảng phải duyệt tiếp theo sau khi duyệt.

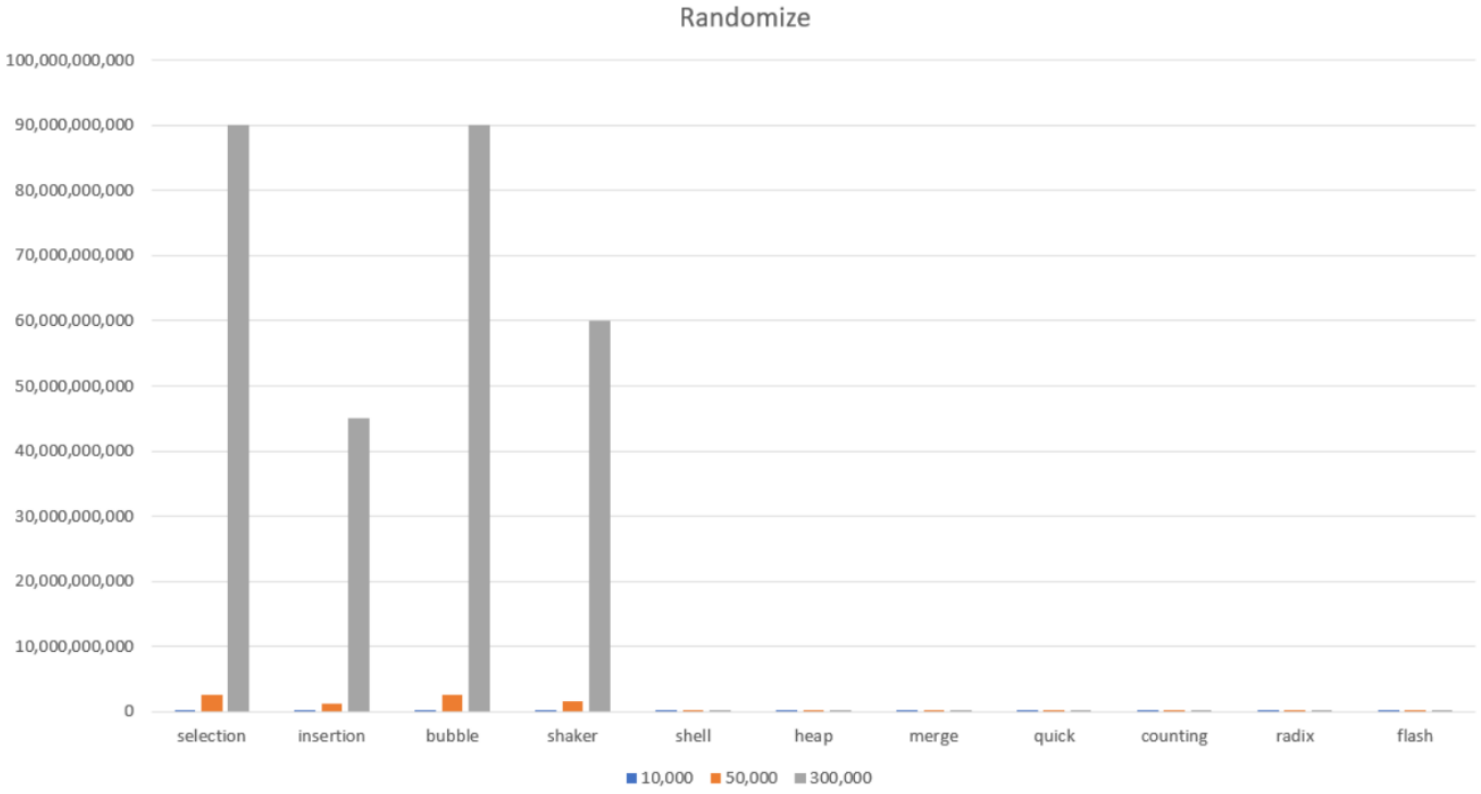
- Shell Sort, Heap Sort, Merge Sort và Quick Sort có tốc độ ổn định xuyên suốt cả 4 loại dữ liệu đầu vào.

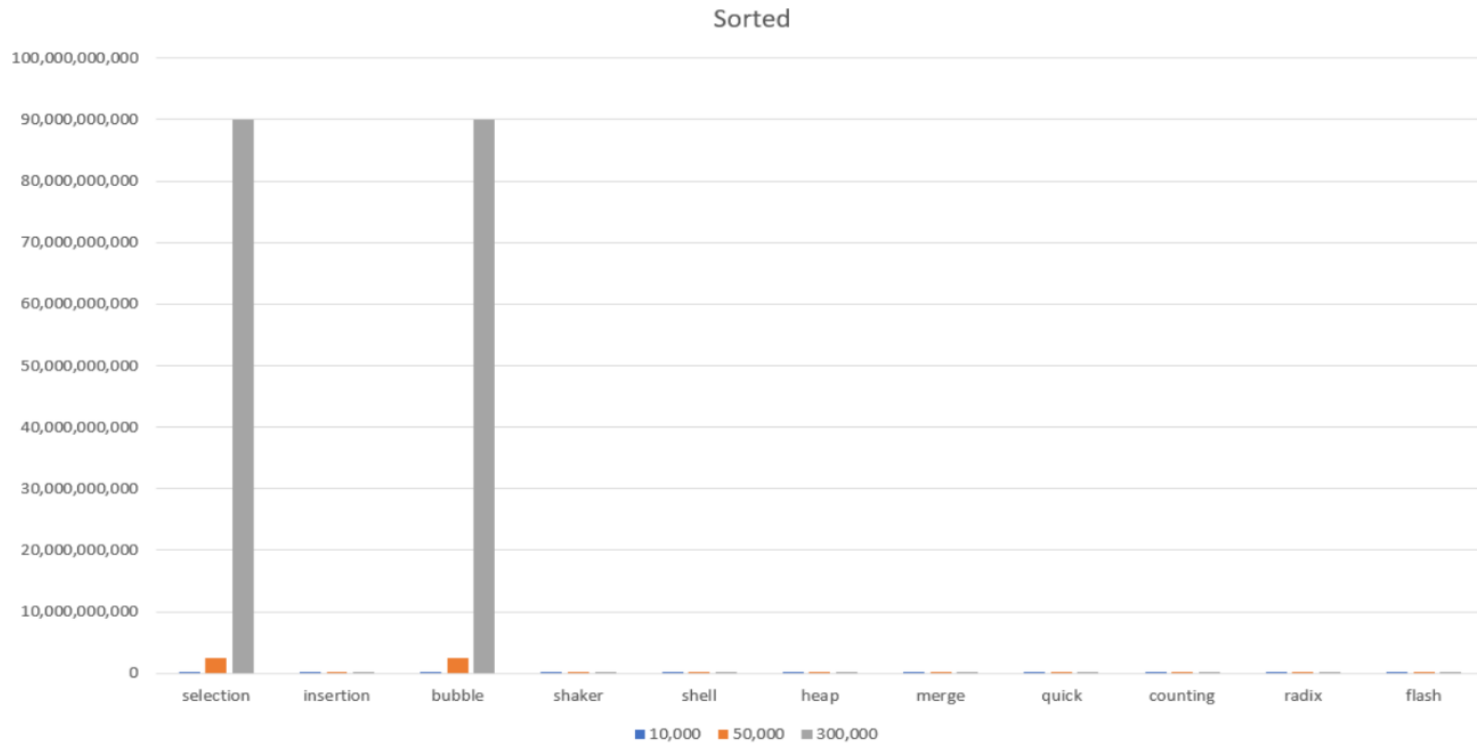
- Flash Sort là một thuật toán cho tốc độ nhanh và tiêu tốn rất ít bộ nhớ, tuy nhiên cách thức xây dựng thuật toán trên khá phức tạp.

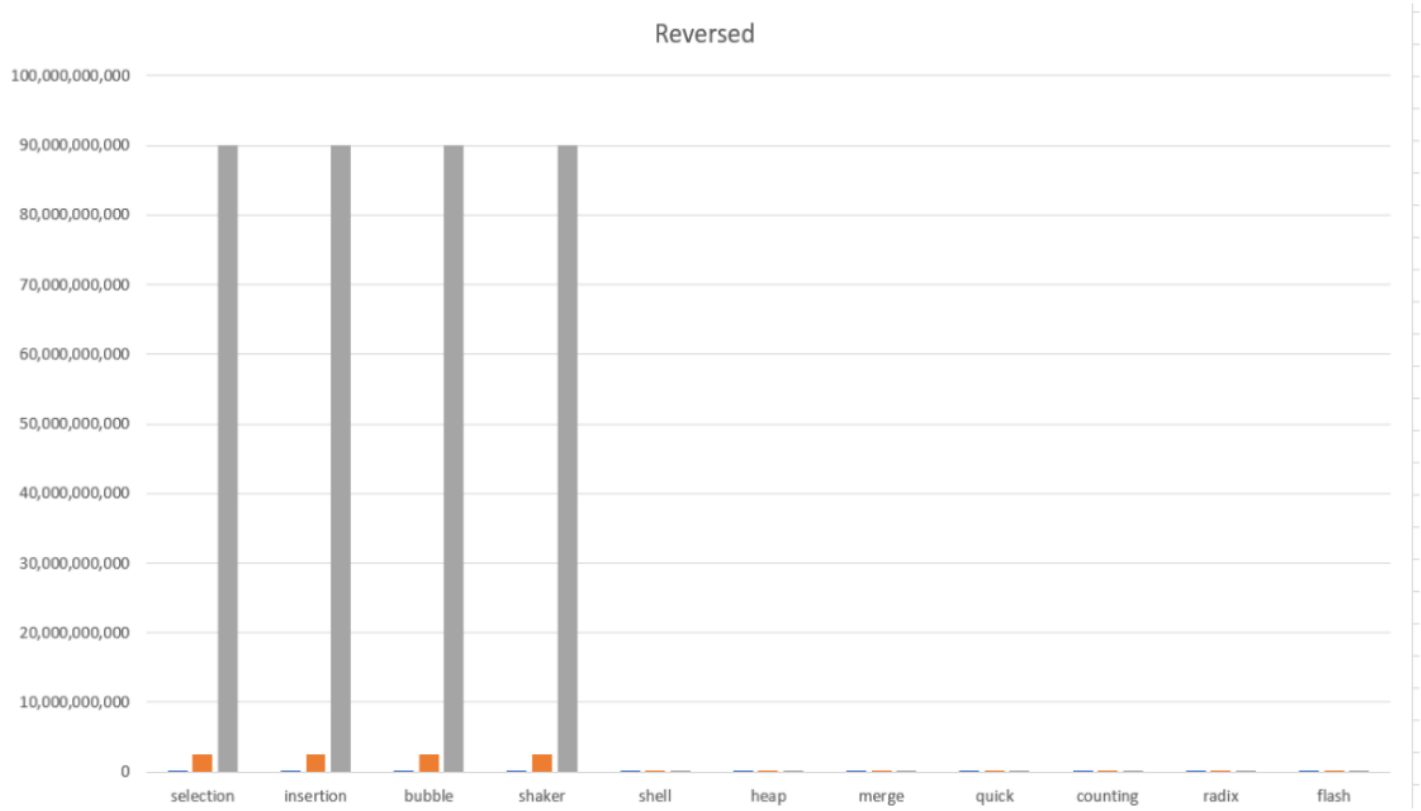
- Counting Sort và Radix Sort là những thuật toán cho tốc độ nhanh, tuy nhiên cần đánh đổi bằng cách sử dụng thêm bộ nhớ.
- Nhóm những thuật toán có Độ Ổn Định: Bubble Sort, Shaker Sort, Radix Sort, Counting Sort, Insertion Sort, Merge Sort.

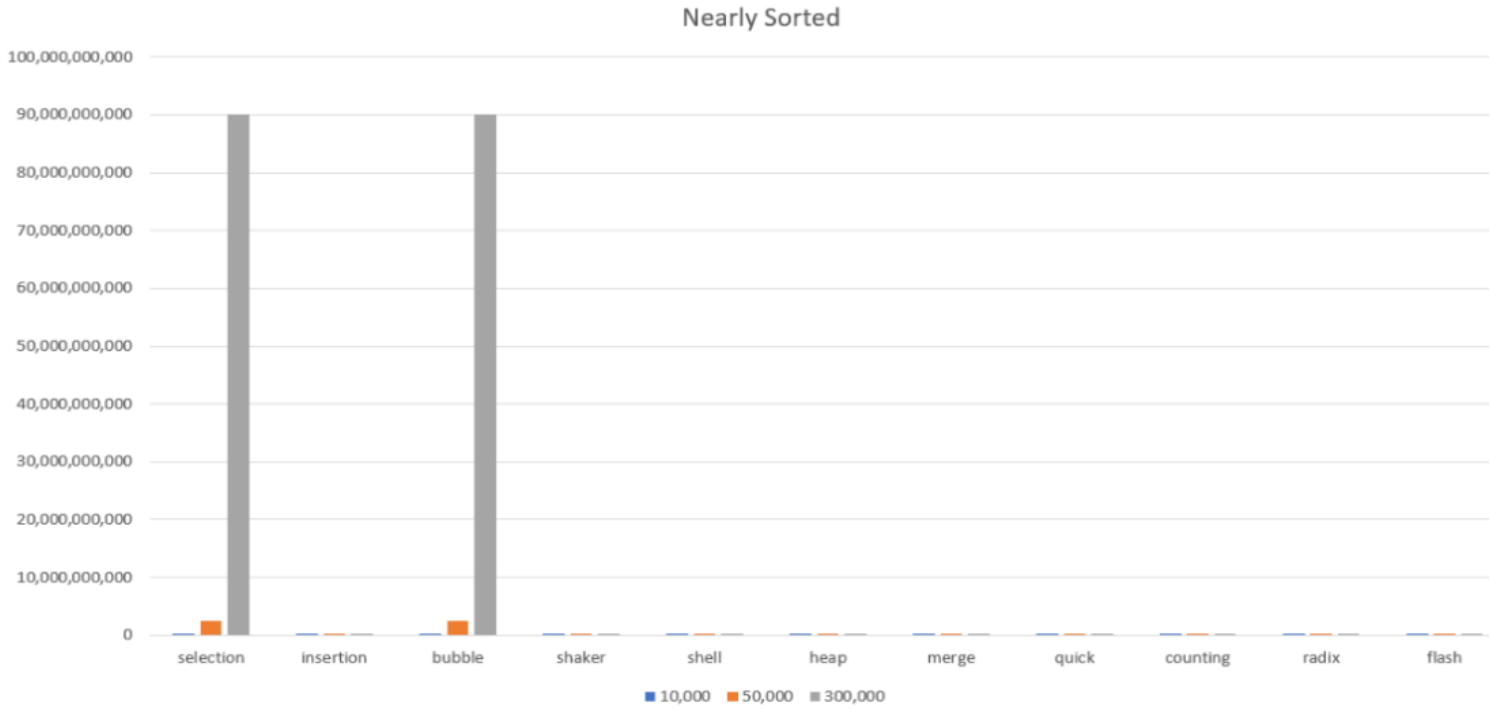
- Nhóm những thuật toán không có Độ Ổn Định: Heap Sort, Flash Sort, Selection Sort, Shell Sort, Quick Sort.

b. Comparisions






Reversed
100,000,000,000
90,000,000,000
80,000,000,000
70,000,000,000
60,000,000,000
50,000,000,000
40,000,000,000
30,000,000,000
20,000,000,000
10,000,000,000
0
selection
insertion
bubble
shaker
shell
heap
merge
quick
counting
radix
flash


Nearly Sorted
100,000,000,000
90,000,000,000
80,000,000,000
70,000,000,000
60,000,000,000
50,000,000,000
40,000,000,000
30,000,000,000
20,000,000,000
10,000,000,000
0
selection
insertion
bubble
shaker
shell
heap
merge
quick
counting
radix
flash
10,000
50,000
300,000

Nhận Xét Biểu Đồ:

- Với dữ liệu đầu vào nhỏ, số lượng phép so sánh của các thuật toán gần như tương đương.

- Bắt đầu từ 50.000, số lượng phép so sánh đã có sự khác biệt

.

- Mảng đã sắp xếp và mảng gần như sắp xếp thì bubble và selection có số lượng phép so sánh gần giống nhau, các thuật toán còn lại có số lượng phép so sánh ít như nhau

- Mảng sắp xếp ngược: selection, insertion, bubble và shaker có số lượng phép so sánh gần giống nhau, 7 thuật toán còn lại có số lượng phép so sánh khá ít và giống nhau

- Mảng có dữ liệu ngẫu nhiên, Selection và bubble có số lượng phép so sánh gần giống nhau, insertion và shaker gần giống nhau.

IV. TỔ CHỨC DỰ ÁN VÀ GHI CHÚ LẬP TRÌNH:

a. Tổ Chức Dự Án:

Dự án này được xây dựng với 2 mã nguồn (source code chính). Một là commands.cpp chứa code chạy những command line từ 1 đến 5. Hai là experiment.cpp chứa code chạy 4 cách sắp xếp dữ liệu với 6 bộ dữ liệu chạy cho 11 thuật toán, in kết quả thời gian chạy và đếm số lần so sánh ra màn hình hoặc file result.csv .

i) commands.cpp:

Chương trình này sử dụng 4 file header tự biên soạn và 1 file header được cung cấp sẵn đó là:

- SortingAlgorithm là header chứa các hàm của 11 thuật toán sắp xếp gốc (Hàm chính và các hàm phụ nếu có).
- Timing.h là header chứa các hàm đo thời gian chạy của 11 thuật toán sắp xếp trên.
- ComparingCounting.h là header chứa các hàm đếm số lần so sánh trong 11 thuật toán sắp xếp trên. (các hàm sẽ có thêm biến comparisions truyền vào và xử lý nên thời gian chạy có thể chậm hơn các hàm gốc trong header SortingAlgorithm.h)
- Commands.h là header chứa các hàm command line từ 1 đến 5.
- DataGenerator.h là header chứa các hàm giúp tạo bộ dữ liệu cho mảng dựa trên cách sắp xếp truyền vào (được cung cấp) .

ii) experiment.cpp:

Chương trình này sử dụng 3 file header tự biên soạn và 1 file header được cung cấp sẵn đó là:

- SortingAlgorithm là header chứa các hàm của 11 thuật toán sắp xếp gốc (Hàm chính và các hàm phụ nếu có).
- Timing.h là header chứa các hàm đo thời gian chạy của 11 thuật toán sắp xếp trên.
- ComparingCounting.h là header chứa các hàm đếm số lần so sánh trong 11 thuật toán sắp xếp trên. (các hàm sẽ có thêm biến comparisions truyền vào và xử lý nên thời gian chạy có thể chậm hơn các hàm gốc trong header SortingAlgorithm.h)
- DataGenerator.h là header chứa các hàm giúp tạo bộ dữ liệu cho mảng dựa trên cách sắp xếp truyền vào (được cung cấp).

b. Ghi Chú Lập Trình:

- File được ghi ra của commands.cpp có thể khác hoặc giống nhau tùy vào command line được chạy. File đọc vào của commands.cpp có thể được cung cấp sẵn hoặc đọc từ những File được ghi ra ở trên.
- File được ghi ra của experiment.cpp được dịnh dạng dựa vào bảng được cung cấp trong Lab03.pdf, file được ghi ra là result.csv nếu cần up lên các kho chứa đám mây (sử dụng cho việc làm nhóm) thì nên chuyển sang file result.xlsx để dễ chỉnh sửa.
- Các thư viện có sẵn đã sử dụng:
  - iostream: sử dụng cho việc đọc và xuất dữ liệu.
  - fstream: sử dụng cho việc đọc và xuất dữ liệu ra file.
  - iomanip: sử dụng cho việc căn chỉnh khoảng cách giữa ký tự hoặc từ (để dễ nhìn hơn).
  - chrono: sử dụng cho việc đo thời gian chạy của thuật toán với đơn vị là milliseconds.
  - string.h: để sử dụng hàm strcmp cho việc phân tích command line được nhập vào.

V. TÀI LIỆU THAM KHẢO

- Tài Liệu Giấy: Giáo Trình "Nhập Môn Cấu Trúc Dữ Liệu Và Giải Thuật - 2003" của trường.
- Tài Liệu Online (tham khảo từ web, video):

https://www.geeksforgeeks.org

https://www.youtube.com/@ProgramingUni

Github.com

Iostream.vn

https://codelearn.io/sharing/dau-moi-la-thuat-toan-sap-xep-tot-nhat