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

KHOA CÔNG NGHỆ THÔNG TIN



BÁO CÁO

GIẢI THUẬT SẮP XẾP

MÔN HỌC: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GIẢNG VIÊN HƯỚNG DẪN : PHAN THỊ PHƯƠNG UYÊN
SINH VIÊN THỰC HIỆN : NHÓM 5

THÁNG 11/2021

fit@hcmus

LỜI NÓI ĐẦU

Một trong những vấn đề cơ bản nhất của khoa học máy tính là sắp xếp một danh sách các phần tử. Hiện nay đã có rất nhiều lời giải cho vấn đề trên, được biết đến như là các thuật toán sắp xếp. Kể từ những ngày đầu tiên của thời đại máy tính, vấn đề sắp xếp đã thu hút rất nhiều sự nghiên cứu, có lẽ vì sự phức tạp của việc giải quyết vấn đề này một cách hiệu quả. Ngày nay người ta đã nghiên cứu ra rất nhiều thuật toán cơ bản và nâng cao khác nhau. Tuy nhiên không phải lúc nào chúng ta cũng có thể đánh giá thuật toán này tốt hay tệ hơn thuật toán khác được bởi vì hiệu năng của những thuật toán sắp xếp khác nhau phụ thuộc vào dữ liệu được sắp xếp là những dãy số ngẫu nhiên, dãy số được sắp xếp sẵn, dãy số đang bị đảo ngược, hay dãy số đã gần được sắp xếp và bị ảnh hưởng bởi cả số lượng phần tử cần sắp xếp.

Chính vì lý do này nên chúng em đã thực hiện đồ án nghiên cứu về thời gian chạy và số lần thực hiện phép so sánh của 11 thuật toán khác nhau bao gồm : Bubble Sort, Selection Sort, Insertion Sort, Count Sort, Radix Sort, Flash Sort, Heap Sort, Quick Sort, Merge Sort, Shaker Sort, Shell Sort. Thực hiện đồ án lần này đã giúp chúng em có những hiểu biết sâu sắc hơn về các thuật toán sắp xếp sau tiết học lý thuyết. Nhờ nó mà chúng em đã biết cách dùng thuật toán tối ưu nhất trong những tình huống khác nhau cũng như có cái nhìn tổng quan về thời gian chạy của những thuật toán và biết được để sắp xếp một dãy số thì một thuật toán cần thực hiện những số phép so sánh khác nhau như thế nào. Mặc dù chúng em đã rất cố gắng và nỗ lực để làm đồ án này do kinh nghiệm còn hạn chế và kiến thức em nắm chưa sâu nên chúng em biết sẽ không tránh khỏi những thiếu sót. Chúng em rất mong nhận được sự thông cảm và đóng góp của cô để lần sau làm đồ án được tốt hơn.

Hoàn thành đồ án về các thuật toán sắp xếp lần này là niềm vui của chúng em, chúng em rất là biết ơn cô Phan Thị Phương Uyên đã hướng dẫn chúng em tận tình trong suốt thời gian chúng em làm đồ án. Một lần nữa nhóm chúng em xin gửi lời cảm ơn chân thành nhất đến cô.

NHÓM 5

20120353 - Huỳnh Hữu Phước

20120366 - Phạm Phú Hoàng Sơn

20120397 - Bùi Quang Tùng

20120409 - Trần Thanh Tùng

MỤC LỤC

PHẦN 1 : CÁC THUẬT TOÁN SẮP XẾP	3
I. Selection Sort.....	3
II. Insertion Sort.....	6
III. Bubble Sort	8
IV. Shaker Sort.....	10
V. Shell Sort	12
VI. Heap Sort.....	14
VII. Merge Sort.....	20
VIII. Quick Sort.....	24
IX. Counting Sort.....	30
X. Radix Sort	32
XI. Flash Sort	35
PHẦN 2 : KẾT QUẢ THỰC NGHIỆM VÀ NHẬN XÉT	42
I. Cấu hình máy tính thực hiện	42
II. Kết quả thực nghiệm và nhận xét	42
1. Data order : Sorted.....	43
2. Data Order : Nearly Sorted.....	45
3. Data Order : Randomized.....	47
4. Data Order : Reverse Sorted.....	49
III. Kết luận chung về các kết quả thực nghiệm.....	52
PHẦN 3 : TỔ CHỨC MÃ NGUỒN.....	53
CÁC NGUỒN TÀI LIỆU THAM KHẢO.....	54

PHẦN 1 : CÁC THUẬT TOÁN SẮP XẾP

I. Selection Sort

1. Ý tưởng

- Thuật toán sẽ chia mảng ra thành 2 phần là mảng đã sắp xếp và chưa sắp xếp. Thực hiện sắp xếp mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất (vì đang xét tăng dần) trong đoạn chưa được sắp xếp và đổi chỗ nó với phần tử ở đầu đoạn chưa sắp xếp.
- Sau khi so sánh và đổi chỗ thì tăng số phần tử của phần đã sắp xếp lên 1 và giảm số phần tử của phần chưa sắp xếp xuống 1. Thực hiện đến khi phần chưa sắp xếp còn 1 phần tử.

2. Thuật toán

- Bước 1: Chọn phần tử đầu tiên của phần chưa sắp xếp ($i=0$).
- Bước 2: So sánh phần tử đã chọn với các phần tử phía sau nó (vị trí đã chọn + 1 đến n) để tìm phần tử nhỏ nhất.
- Bước 3: Đổi chỗ phần tử nhỏ nhất và phần tử đang chọn.
- Bước 4: Nếu $i < n$ tăng i lên 1 đơn vị và quay lại bước 1 2 và 3. Ngược lại thì dừng.

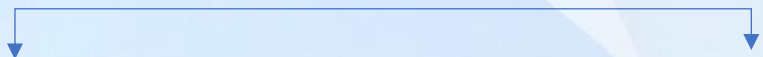
3. Ví dụ minh họa

Ta có bài toán sắp xếp một dãy Array với $\text{Array}[5] = \{12, 22, 20, 9, 15\}$ là một dãy không giảm. Gọi n là số phần tử của dãy ($n=5$), các phần tử còn lại có chỉ số từ 0 đến $n-1$ (như hình vẽ dưới):

12	22	20	9	15
0	1	2	3	4

Để minh họa , ta sẽ tô màu **xanh** vào các ô đã được sắp xếp đúng vị trí và màu **đỏ** là để chỉ phần tử nhỏ nhất trong dãy còn lại.

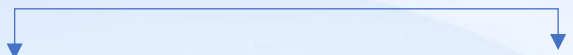
- Xét $\text{Array}[0]=12$ là phần tử đầu tiên của mảng chưa sắp xếp, sau đó tìm phần tử nhỏ nhất trong đoạn từ $\text{Array}[0]$ đến $\text{Array}[4]$, hoán đổi vị trí đó với $\text{Array}[0]$.



12	22	20	9	15
0	1	2	3	4

9	22	20	12	15
0	1	2	3	4

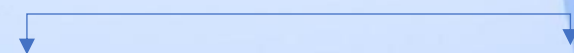
- Tương tự như trên ta chọn $\text{Array}[1]=22$, ta đi tìm phần tử nhỏ nhất trong đoạn $\text{Array}[1]$ đến $\text{Array}[4]$, sau đó hoán đổi vị trí đó với $\text{Array}[1]$.



9	22	20	12	15
0	1	2	3	4

9	12	20	22	15
0	1	2	3	4

- Tiếp tục chọn $\text{Array}[2]=20$, ta đi tìm phần tử nhỏ nhất trong đoạn $\text{Array}[2]$ đến $\text{Array}[4]$, sau đó hoán đổi vị trí đó với $\text{Array}[2]$.

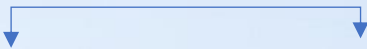


9	12	20	22	15
0	1	2	3	4

9	12	15	22	20
0	1	2	3	4

• Tiếp tục chọn Array [3]=12, ta đi tìm phần tử nhỏ nhất trong đoạn Array[3] đến Array[4], sau đó hoán đổi vị trí đó với Array[3].

9	12	15	22	20
0	1	2	3	4



• Còn lại 1 phần tử cuối cùng chắc chắn ở vị trí đúng, ta dừng thuật toán tại đây. Kết thúc ta có mảng Array[5]={9,12,15,20,22} đã được sắp xếp theo thứ tự tăng dần.

9	12	15	20	22
0	1	2	3	4

4. Độ phức tạp

- **Thời gian**
 - Tốt nhất : $O(n^2)$
 - Trung bình : $O(n^2)$
 - Xấu nhất : $O(n^2)$
- **Không gian:** bộ nhớ sử dụng $O(1)$ do chỉ dùng phép gán chứ không tạo thêm bất kì mảng phụ nào

5. Ưu điểm và nhược điểm

- **Ưu điểm**
 - Dễ cài đặt
 - Không tốn nhiều không gian bộ nhớ sử dụng
 - Không phụ thuộc vào cách tổ chức dữ liệu đầu vào
 - Tuy độ phức tạp là $O(n^2)$ nhưng chỉ yêu cầu $O(n)$ bước di chuyển

- Tốt cho kiểu dữ liệu so sánh khóa (địa chỉ vùng nhớ) ít tốn kém và di chuyển dữ liệu tốn kém
- **Nhược điểm**
 - Độ phức tạp luôn là $O(n^2)$ cho mọi trường hợp
 - Chạy rất lâu nếu số lượng phần tử lớn
 - Không phải là thuật toán chạy ổn định

II. Insertion Sort

1. Ý tưởng

Thực hiện duyệt từng phần tử trong mảng, sau đó chèn vào đúng vị trí trong mảng con (dãy số từ vị trí đầu đến trước nó) và giữ nguyên tính tăng dần của mảng con.

2. Thuật toán

- Bước 1: chọn phần tử thứ 2 trong mảng.
- Bước 2: duyệt và so sánh nó với các phần tử ở vị trí trước vị trí đang chọn ($i - 1$) đến vị trí đầu tiên. Dời vị trí các phần tử lớn hơn phần tử đang chọn lên 1 đơn vị.
- Bước 3: thực hiện chèn phần tử đang chọn vào vị trí thích hợp (giữ nguyên tính tăng dần của dãy con đó).
- Bước 4: nếu $i < n$ thì chọn vị trí tiếp theo, thực hiện lại 2 và 3. Ngược lại thì dừng.

3. Ví dụ minh họa

Cho dãy $\text{Array}[5] = \{1, 5, 6, 4, 2,\}$ là một dãy không giảm cần sắp xếp, gọi $n=5$ là số lượng phần tử của Array , các phần tử có chỉ số lần lượt từ 1 đến $n-1$ (ảnh minh họa bên dưới):

1	5	6	4	2
0	1	2	3	4

Ta quy ước các phần tử được tô màu **xanh** là các phần tử được sắp xếp vào mảng con . Các phần tử được tô màu **đỏ** là các phần tử đang được xét để so sánh với cách phần tử trong mảng con.

- Thực hiện duyệt từ đầu đến cuối mảng, Array[0] là phần tử đầu tiên của phần đã sắp xếp.
- Chọn vị trí $i = 1$, $key = \text{Array}[1]$ duyệt từ $i = 1$ trở về trước.

1	5	6	4	2
0	1	2	3	4

- Array[1] lớn hơn tất cả nên giữ nguyên vị trí, nó đã được sắp xếp và tăng i lên 1, chọn vị trí $i=2$, $key = A[2]$ duyệt từ $i = 2$ trở về trước.

1	5	6	4	2
0	1	2	3	4

- A[2] lớn hơn tất cả nên giữ nguyên vị trí, nó đã được sắp xếp và tăng i lên 1 , chọn vị trí $i=3$, $key = A[3]$ duyệt $i=3$ trở về trước.

1	5	6	4	2
0	1	2	3	4

- $A[0] < A[3] < A[2]$, tăng vị trí các phần tử từ A[1] đến A[2] lên 1 đơn vị, chèn key vào giữa A[0] và A[2], tăng i lên 1.
- Chọn vị trí $i = 4$, $key = A[4]$ duyệt từ $i = 4$ trở về trước.

1	4	5	6	2
0	1	2	3	4

- $A[0] < A[4] < A[2]$, tăng vị trí các phần tử từ A[1] đến A[3] lên 1 đơn vị, chèn key vào giữa A[0] và A[2], $i = 4$ dừng duyệt.

1	2	4	5	6
0	1	2	3	4

- Kết thúc thuật toán , ta có mảng $\text{Array}[5]=\{1,2,4,5,6\}$ đã được sắp xếp theo thứ tự tăng dần.

4. Độ phức tạp

- **Thời gian:**
 - Tốt nhất : $O(n)$
 - Trung bình : $O(n^2)$
 - Xấu nhất : $O(n^2)$
- **Không gian:** bộ nhớ sử dụng $O(1)$ vì chỉ dùng phép gán để lưu trữ tạm dữ liệu chứ không tạo mảng phụ.

5. Ưu điểm và nhược điểm

- **Ưu điểm:**
 - Dễ cài đặt.
 - Không tốn nhiều không gian bộ nhớ sử dụng.
 - Tốt cho dữ liệu đã được sắp xếp một phần hoặc dữ liệu nhỏ.
 - Thuật toán chạy ổn định.
- **Nhược điểm:**
 - Dữ liệu kích thước lớn chạy sẽ lâu.
 - Phụ thuộc vào việc tổ chức dữ liệu ban đầu.

III. Bubble Sort

1. Ý tưởng

Đưa phần tử bé nhất về đầu dãy. Xét lần lượt 2 phần tử liền kề nhau, bắt đầu từ phần tử cuối mảng :

- Nếu 1 phần tử lớn hơn phần tử đứng trước nó thì đổi chỗ hai phần tử đó.
- Lặp lại với các số tiếp theo cho đến khi xếp phần tử nhỏ nhất ở đầu mảng.

- Tiếp tục thực hiện lại từ cuối mảng vừa sắp xếp cho đến khi các phần tử được sắp xếp hoàn chỉnh.

2. Thuật toán

- Bước 1: sử dụng vòng lặp 1 với biến i , tổng cộng n phần tử (bắt đầu từ 0 đến n)-(lính canh để cài đặt vị trí đã sắp xếp)
- Bước 2: sử dụng vòng lặp 2 với biến j , tổng cộng n phần tử (bắt đầu từ phần tử cuối về phần tử i ở vòng lặp 1).
- Bước 3: so sánh phần tử thứ $[j]$ và $[j-1]$ nếu phần tử ở vị trí $[j] <$ phần tử vị trí $[j-1]$ thì đổi giá trị hai phần tử

3. Ví dụ minh họa

Ta có dãy cần sắp xếp là 3-1-5-6-2-4-7-8, thuật toán Bubble Sort sắp xếp dãy đã cho thành một dãy tăng dần như bảng dưới.

3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	2	6	4	7	8
3	1	5	2	6	4	7	8
3	1	2	5	6	4	7	8
3	1	2	5	6	4	7	8
3	1	2	5	6	4	7	8
1	3	2	5	6	4	7	8
1	3	2	5	6	4	7	8

Chú thích:

Đối tượng đang xét	Đối tượng hoán đổi	Đối tượng đang sắp xếp
--------------------	--------------------	------------------------

Lặp lại với dãy mới là: 3-2-5-6-4-7-8 cho đến khi dãy được sắp xếp.

4. Độ phức tạp

- **Theo thời gian:**
 - Tốt nhất là $O(n)$ khi mảng đã được sắp xếp.
 - Trung bình là $O(n^2)$.
 - Xấu nhất là $O(n^2)$ khi mảng được sắp xếp ngược.
- **Theo không gian:** $O(1)$. có thêm 1 biến để phục vụ cho việc hoán vị.

5. Ưu điểm và nhược điểm

- **Ưu điểm:** code ngắn gọn.
- **Nhược điểm:** hiệu suất thấp (số lần so sánh quá nhiều)
- **Cải tiến:** Thay thế bằng thuật toán Shaker Sort là cải tiến của Bubble Sort bằng cách giảm số lượng phần tử cần sắp xếp sau mỗi vòng lặp.

IV. Shaker Sort

1. Ý tưởng

Shaker Sort là cải tiến của Bubble Sort bằng cách giảm số lượng phần tử cần sắp xếp sau mỗi vòng lặp, sau khi đưa phần tử bé nhất về đầu dãy , tiếp tục sẽ đưa phần tử lớn nhất về cuối dãy. Giảm thời gian sắp xếp bằng cách giảm độ lớn mảng cần sắp xếp cho lần sắp xếp kế tiếp.

- Nếu 1 phần tử lớn hơn phần tử đứng trước nó thì đổi chỗ hai phần tử đó.
- Lặp lại với các phần tử tiếp theo cho đến khi xếp phần tử nhỏ nhất ở đầu mảng.
- Quay ngược lại và sắp xếp cho đến khi xếp phần tử lớn nhất cuối mảng
- Tiếp tục thực hiện lại từ cuối mảng vừa sắp xếp cho đến khi các phần tử được sắp xếp hoàn chỉnh.

2. Thuật toán

- Bước 1: Khai báo left= vị trí trái ; right = vị trí phải; biến k để đặt vị trí
- Bước 2: Nếu vị trí trái nhỏ hơn vị trí phải -> bước 3.
- Bước 3: Bắt đầu vòng lặp 1 với biến i ở vị trí trái (bắt đầu từ left đến right)

Xét nếu số tại vị trí $[i] >$ vị trí $[i-1]$ đổi chỗ hai vị trí và cài đặt lại biến k .

- Bước 4: đặt vị trí phải $right = k$ vì vị trí cuối dãy đã được sắp xếp bắt đầu vòng lặp 2 với biến l ở vị trí phải (bắt đầu từ vị trí $right$ về $left$) xét nếu số tại vị trí $[i] < [i-1]$ đổi chỗ hai vị trí và đặt lại biến k
- Bước 5: đặt $left = k \rightarrow$ quay lại bước 2.

3. Ví dụ minh họa

Ta có dãy cần sắp xếp là 3-1-5-6-2-4-7-8, thuật toán Shaker Sort sẽ thực hiện sắp xếp chúng thành một dãy tăng dần.

3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	6	2	4	7	8
3	1	5	2	6	4	7	8
3	1	5	2	6	4	7	8
3	1	2	5	6	4	7	8
3	1	2	5	6	4	7	8
3	1	2	5	6	4	7	8
3	1	2	5	6	4	7	8
1	3	2	5	6	4	7	8
1	3	2	5	6	4	7	8
1	2	3	5	6	4	7	8
1	2	3	5	6	4	7	8
1	2	3	5	6	4	7	8
1	2	3	5	4	6	7	8
1	2	3	5	4	6	7	8
1	2	3	5	4	6	7	8

Chú thích

Đối tượng đang xét	Đối tượng hoán đổi	Đối tượng đã sắp xếp
--------------------	--------------------	----------------------

Lắp lại cho với dây mới: 2-3-5-4-6-7 cho đến khi dây được sắp xếp.

4. Độ phức tạp

- **Theo thời gian:**
 - Tốt nhất là $O(n)$
 - Trung bình là $O(n^2)$
 - Xấu nhất là $O(n^2)$
- **Theo không gian:** độ phức tạp là $O(3)$ vì có thêm biến để thực hiện hoán vị và đặt left, right.

5. Ưu điểm và nhược điểm

- **Ưu điểm :** Hiệu suất cao hơn so với bubble sort, giảm số lượng phần tử sau mỗi vòng lặp.
- **Nhược điểm :** Hiệu suất còn khá thấp so với dãy ngược hoặc dãy random.

V. Shell Sort

1. Ý tưởng

- Là sự cải tiến của insertion sort nhằm giảm khoảng cách của các phần tử cần so sánh và sắp xếp.
- Giải thuật sử dụng insertion trên các phần tử có vị trí xa nhau một khoảng (interval) sau đó dần thu hẹp khoảng lại.
- Khoảng sẽ có giá trị: khoảng = số lượng phần tử / (2^k) (với k là số lần lặp cho đến khi khoảng = 1).

2. Thuật toán

- Bước 1: Khởi tạo biến interval (khoảng).
- Bước 2: Bắt đầu vòng lặp với biến interval có bước nhảy interval/2 cho đến khi interval=1.
- Bước 3: Bắt đầu vòng lặp 2 với biến i từ interval đến n-1.
- Bước 4: Đặt lính canh là phần tử tại vị trí i.
- Bước 5: Bắt đầu vòng lặp thứ 3 với biến j từ i.

Nếu $j \geq$ số khoảng (có tồn tại phần tử trước nó interval khoảng cách - (k)) và phần tử k lớn hơn phần tử lính cạnh thì tiến hành đổi chỗ hai phần tử đó.

- Bước 6: Lặp lại các bước đến khi mảng được sắp xếp hoàn toàn.

3. Minh họa thuật toán

Ta có dãy chưa sắp xếp là 3-1-5-6-2-4-7-8 (8 phần tử). Thuật toán Shell Sort sẽ thực hiện sắp xếp dãy trên thành dãy tăng dần. Xem ví dụ dưới.

- Interval của vòng lặp đầu $= 8/2 = 4$

3	1	5	6	2	4	7	8
2*	1	5	6	3*	4	7	8

- Tiếp tục chia interval cho 2 $= 4/2 = 2$

2	1	5	6	3	4	7	8
2	1	3*	4	5*	6	7	8

- Tiếp tục chia interval cho 2 $= 2/2 = 1$

2	1	3	4	5	6	7	8
1*	2*	3	4	5	6	7	8

Chú thích:

- Những đối tượng cùng màu là những đối tượng được so sánh với nhau.
- Những đối tượng có dấu * là những đối tượng đã được hoán đổi với ô cùng màu.

4. Độ phức tạp thuật toán

- Theo thời gian:
 - Tốt nhất: $O(n)$
 - Trung bình : $O(n)$
 - Xấu nhất: $O(n)$
- Theo không gian: $O(1)$ có thêm 1 biến để phục vụ cho việc hoán vị.

5. Ưu điểm và nhược điểm

- **Ưu điểm:** hiệu suất cao (ít phép so sánh), là sự cải tiến của Insertion sort.
- **Nhược điểm:** Không phù hợp khi sử dụng cho các tệp dữ liệu lớn.

VI. Heap Sort

1. Ý tưởng

Giải thuật Heap Sort còn được gọi là giải thuật vun đống, có thể được xem như bản cải tiến của Selection Sort khi chia các phần tử thành 2 mảng con

- Một mảng phần tử đã được sắp xếp.
- Một mảng các phần tử chưa được sắp xếp.

Giải thuật Heap Sort được chia thành 2 giai đoạn.

Giai đoạn 1:

Từ dãy giá trị input, chúng ta sắp xếp chúng thành một heap(dạng cấu trúc cây nhị phân). Heap này có thể là min heap (nút gốc có giá trị bé nhất), hoặc max heap (nút gốc có giá trị lớn nhất. Trong ví dụ minh họa bên dưới , ta sẽ sử dụng max heap với một số yêu cầu thỏa mãn sau:

- Nút cha sẽ lớn hơn tất cả các nút con, nút gốc của heap sẽ là phần tử lớn nhất.
- Heap được tạo thành phải là một cây nhị phân đầy đủ , tức là ngoại trừ các nút lá, các nút nhánh không được thiếu khi xét cùng một cấp độ của cây.

Giai đoạn 2 :

Giai đoạn này gồm các thao tác lặp đi lặp lại cho đến khi mảng dữ liệu được hoàn tất sắp xếp.

- Đưa phần tử nhỏ nhất của heap được tạo vào mảng kết quả, mảng này sẽ chứa các phần tử được sắp xếp.

- Sắp xếp lại heap sau khi loại bỏ nút gốc (có giá trị lớn nhất) để tìm phần tử có giá trị nhỏ nhất tiếp theo.
- Thực hiện lại thao tác 1 cho đến khi các phần tử của heap đều được đưa vào mảng kết quả.
- Như thế mảng kết quả sẽ chứa các phần tử được sắp xếp tăng dần.

Ta có thể biểu diễn heap bằng mảng:

- Thứ tự lưu trữ trên mảng được thực hiện từ trái qua phải.
- Nếu ta biết được chỉ số của 1 phần tử trên mảng, ta sẽ dễ dàng xác định được chỉ số của node cha và các node con của nó.
- Node gốc ở chỉ số 0..
- Node cha của node i có chỉ số $(i-1)/2$.
- Các node con của node i (nếu có) có chỉ số $[2*i + 1]$ và $[2*i + 2]$.
- Node cuối cùng có con trong 1 heap có n phần tử là $[n/2-1]$.

2. Thuật toán

Giả sử dãy ban đầu a cần sắp xếp có n phần tử $a_0, a_1, a_2, \dots, a_{n-1}$. Ta thực hiện công việc sắp xếp dãy a thành một dãy không giảm.

- Bước 1: Tạo max-heap từ dãy a .
- Bước 2: Hoán đổi node đầu và node cuối của heap, cập nhật dãy a tương ứng với heap.
- Bước 3: Xóa đi node cuối của heap.
- Bước 4: Nếu số node của heap > 1 , ta tạo lại max-heap từ heap đang có, lặp lại các bước 2,3. Ngược lại kết thúc thuật toán.

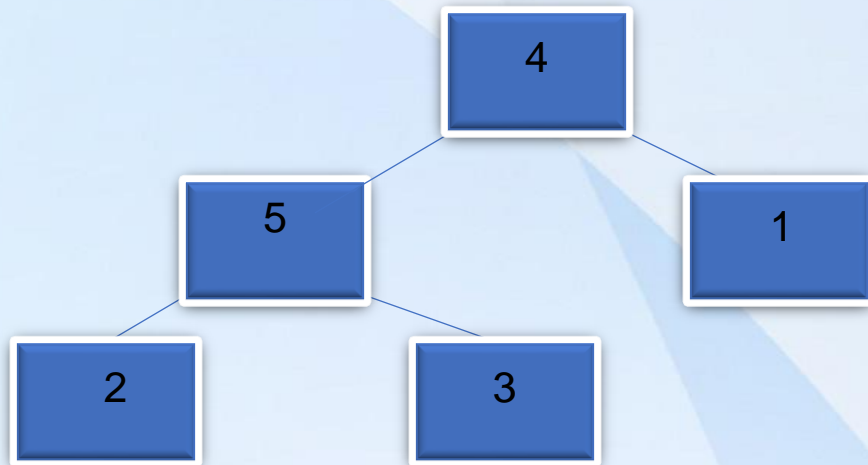
3. Ví dụ minh họa

Ta có bài toán sắp xếp dãy a thành một dãy không giảm với $a=[4,5,1,2,3]$. Gọi n là số phần tử của dãy a ($n=5$), các phần tử của dãy a có chỉ số từ 0 đến $n-1$ (như hình vẽ dưới), ta quy ước vùng màu **xanh** là các phần tử chưa sắp xếp, còn phần tử màu xanh **lá cây** là đã ở đúng vị trí được sắp xếp, màu **đỏ** là các vị trí đang xét.

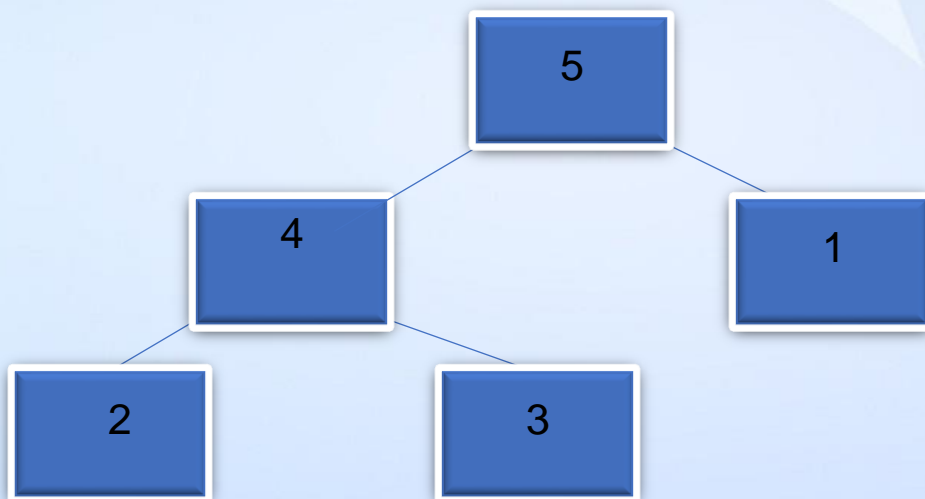
4	5	1	2	3
0	1	2	3	4

Giai đoạn 1 : Tạo max heap từ dãy a.

- Tạo heap từ dãy a:

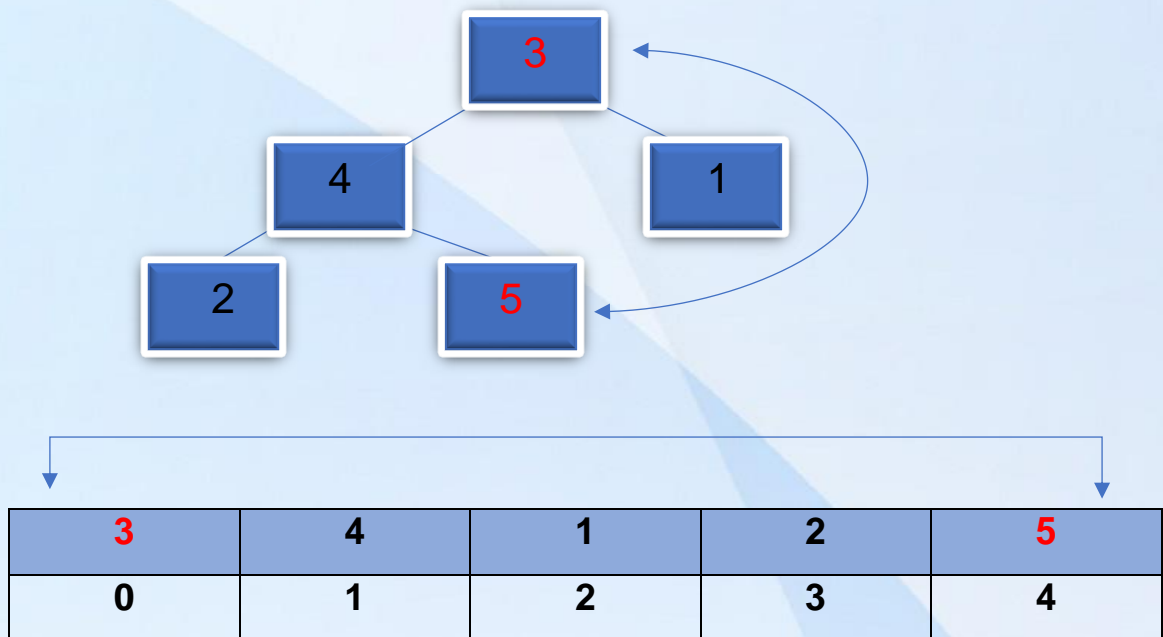


- Ta tiến hành tạo max-heap:

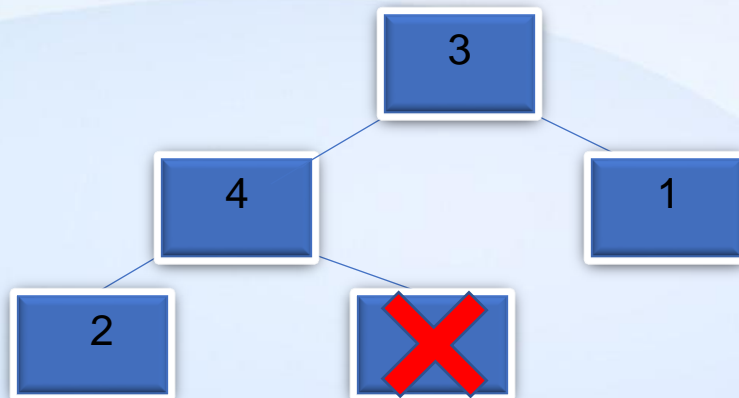


5	4	1	2	3
0	1	2	3	4

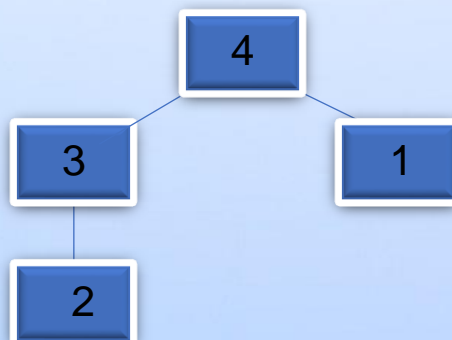
Giai đoạn 2 : Ta hoán đổi node đầu và node cuối của max-heap.



- Xóa đi node cuối:

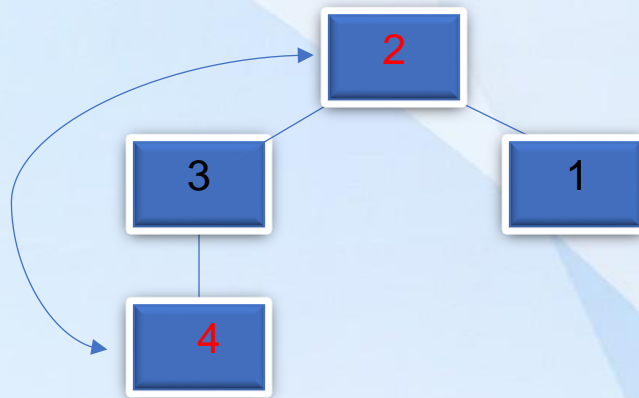


- Tạo max-heap:



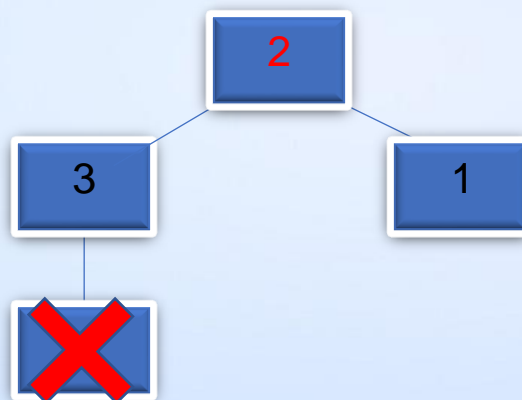
4	3	1	2	5
0	1	2	3	4

- Hoán đổi node đầu và node cuối của max-heap:



2	3	1	4	5
0	1	2	3	4

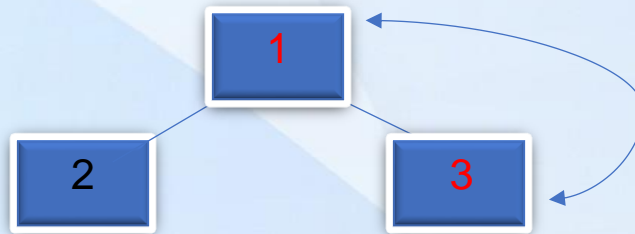
- Xóa đi node cuối :



- Tạo max-heap:

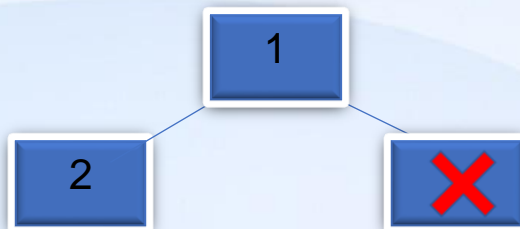


- Hoán đổi node đầu và node cuối của max-heap:



1	2	3	4	5
0	1	2	3	4

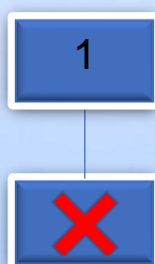
- Xóa đi node cuối :



- Ta thấy heap hiện tại đã là max-heap.

1	2	3	4	5
0	1	2	3	4

- Xóa node cuối của heap, ta thấy heap chỉ còn một node. Kết thúc thuật toán.



1	2	3	4	5
0	1	2	3	4

4. Độ phức tạp

- **Về thời gian**
 - Tốt nhất: $O(n \log_2 n)$
 - Trung bình: $O(n \log_2 n)$
 - Xấu nhất: $O(n \log_2 n)$
- **Về không gian: $O(1)$**

5. Ưu điểm và nhược điểm

- **Ưu điểm** : Cải thiện nhược điểm không tận dụng được thông tin ở bước $i-1$ để tìm phần tử nhỏ nhất ở bước i . Heap sort đã khắc phục nhược điểm nhờ sử dụng cấu trúc heap. Trong trường hợp xấu nhất và trường hợp mảng đã được sắp xếp một phần, Heap sort cho kết quả tốt hơn Quick sort. Heap sort cũng là một trong số các thuật toán có hiệu quả mà không sử dụng đệ quy.
- **Nhược điểm** : Chạy chậm hơn Quick sort trong trường hợp trung bình.

VII. Merge Sort

1. Ý tưởng:

Giống như Quick sort, Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(array, left, mid, right) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là array[left...mid] và array[mid+1...right] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

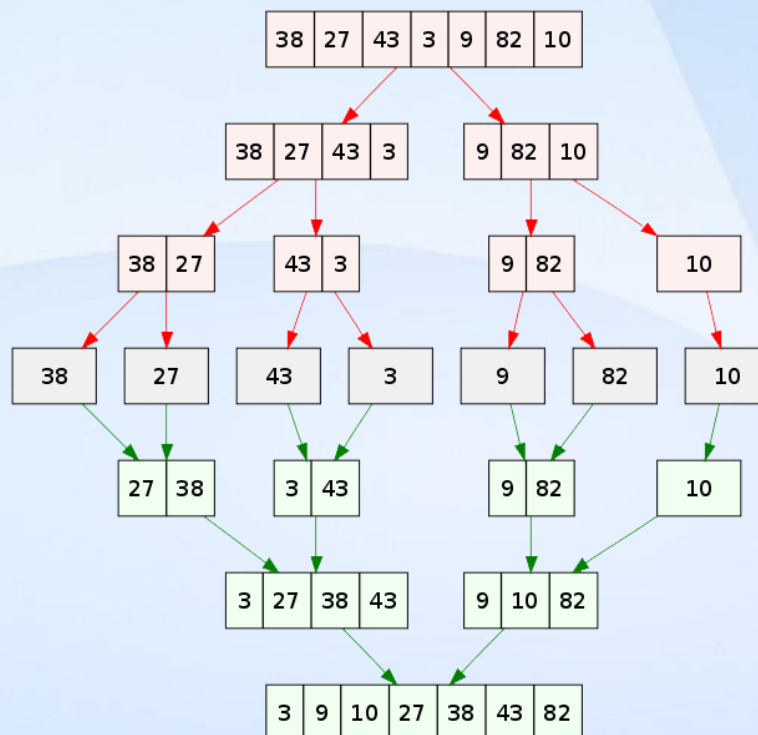
2. Thuật toán

Giả sử dãy ban đầu a cần sắp xếp có n phần tử a_0, a_1, \dots, a_{n-1} . Ta thực hiện công việc sắp xếp a thành một dãy không giảm.

- Bước 1: Đặt $left=0$, $right=n-1$
- Bước 2 : Trong khi $left < right$, ta thực hiện:
 - Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa : $mid = (left+right)/2$.
 - Gọi đệ quy mergeSort cho nửa đầu tiên: $mergeSort(arr, left, mid)$.
 - Gọi đệ quy mergeSort cho nửa thứ hai : $mergeSort(arr, mid+1, right)$.
 - Gộp 2 nửa mảng đã sắp xếp ở (2) và (3) $merge(arr, left, mid, right)$.

3. Ví dụ minh họa

Ta có bài toán sắp xếp dãy a thành một dãy không giảm với $a=[38,27,43,3,9,82,10]$.



38	27	43	3	9	82	10
3	9	10	27	38	43	82

Hình ảnh cho thấy toàn bộ sơ đồ tiến trình của thuật toán Merge Sort khi sắp xếp tăng dần cho dãy $a=[38,27,43,3,9,82,10]$. Nếu nhìn kỹ vào sơ đồ này,

chúng ta có thể thấy mảng ban đầu được lặp lại hành động chia cho tới khi kích thước các mảng sau chia là 1. Khi kích thước các mảng con là 1, tiến trình hợp (merge) sẽ bắt đầu thực hiện gộp lại các mảng này cho tới khi hoàn thành và chỉ còn một mảng đã sắp xếp. Với trường hợp khi 2 mảng con chỉ có 1 phần tử, ta chỉ việc xem phần tử nào nhỏ hơn và đẩy lên đầu, phần tử còn lại đặt phía sau. Do vậy, các mảng con cần gộp lại có tính chất luôn được sắp xếp tăng dần.

Ví dụ minh họa về thuật toán mergeSort, ta thực hiện gộp hai dãy $b=[3,27,38,43]$ và $c=[9,10,82]$:

- Tạo dãy rỗng d. Lúc đầu ta sẽ xét hai số là $b[0]$ và $c[0]$ thêm số nhỏ hơn vào dãy d. Ta thấy:
 $b[0] \leq c[0]$ nên ta lấy giá trị nhỏ hơn là $b[0]$ thêm vào dãy d.

DÃY B

3	27	38	43
---	----	----	----

DÃY C

9	10	82
---	----	----

DÃY D

3

- Xét $b[1]$ và $c[0]$, ta thấy $c[0] \leq b[1]$ nên ta lấy $c[0]$ thêm vào đầu dãy d.

3	27	38	43
---	----	----	----

9	10	82
---	----	----

3	9
---	---

- Tiếp tục thực hiện cho đến khi một trong hai dãy b và c hết phần tử để duyệt .

3	27	38	43
---	----	----	----

9	10	82
---	----	----

3	9	10	27	38	43
---	---	----	----	----	----

- Lúc này một trong hai dãy b và c đã hết phần tử để duyệt, ta thêm tất cả các phần tử còn lại vào dãy d.

3	9	10	27	38	43	82
---	---	----	----	----	----	----

4. Độ phức tạp

- **Về thời gian:**
 - Tốt nhất : $O(n \log_2 n)$
 - Trung bình : $O(n \log_2 n)$
 - Xấu nhất : $O(n \log_2 n)$
- **Về không gian:** $O(n)$

5. Ưu điểm và nhược điểm

- **Ưu điểm** : Sắp xếp nhanh hơn so với các thuật toán cơ bản như Insertion Sort, Selection Sort, Bubble Sort và nhanh hơn Quick Sort trong một số trường hợp.
- **Nhược điểm** : Thuật toán khó cài đặt, không nhận diện được mảng đã được sắp xếp. Khi cài đặt thuật toán đòi hỏi thêm không gian bộ nhớ để lưu trữ các dãy phụ. Hạn chế này khó chấp nhận trong thực tế vì các dãy cần sắp xếp thường có kích thước lớn. Vì vậy Merge Sort thường được dùng để sắp xếp các cấu trúc dữ liệu khác phù hợp hơn như danh sách liên kết hoặc file.

VIII. Quick Sort

1. Ý tưởng

Giống như Merge sort, thuật toán sắp xếp quick sort là một thuật toán chia để trị(Divide and Conquer algorithm). Nó chọn một phần tử trong mảng làm điểm đánh dấu(pivot). Thuật toán sẽ thực hiện chia mảng thành các mảng con dựa vào pivot đã chọn. Việc lựa chọn pivot ảnh hưởng rất nhiều tới tốc độ sắp xếp. Nhưng máy tính lại không thể biết khi nào thì nên chọn theo cách nào. Dưới đây là một số cách để chọn pivot thường được sử dụng:

- Luôn chọn phần tử đầu tiên của mảng.
- Luôn chọn phần tử cuối cùng của mảng.
- Chọn một phần tử random.
- Chọn một phần tử có giá trị nằm giữa mảng(median element).

Mấu chốt chính của thuật toán quick sort là việc phân đoạn dãy số . Mục tiêu của công việc này là: Cho một mảng và một phần tử x là pivot. Đặt x vào đúng vị trí của mảng đã sắp xếp. Di chuyển tất cả các phần tử của mảng mà nhỏ hơn x sang bên trái vị trí của x , và di chuyển tất cả các phần tử của mảng mà lớn hơn x sang bên phải vị trí của x .

Khi đó ta sẽ có 2 mảng con: mảng bên trái của x và mảng bên phải của x . Tiếp tục công việc với mỗi mảng con(chọn pivot, phân đoạn) cho tới khi mảng được sắp xếp.

2. Thuật toán

Giả sử dãy ban đầu a cần sắp xếp có n phần tử $a_0, a_1, a_2, \dots, a_{n-1}$. Ta thực hiện công việc sắp xếp a thành một dãy không giảm.

Các bước thực hiện *phân đoạn* trên dãy a trong đoạn từ $a_{\text{left}}, \dots, a_{\text{right}}$:

- Bước 1 : Chọn pivot bằng một kỹ thuật bất kỳ , hoán đổi nó với phần tử cuối cùng.
- Bước 2 : Đặt $i = \text{left} - 1, j = \text{right}$.

- Bước 3 : Khi $j < \text{right}$, ta thực hiện :
 - Nếu $a[j] \leq a[\text{right}]$:
 - Tăng i lên một đơn vị.
 - Hoán đổi $a[i]$ và $a[j]$.
 - Tăng j lên một đơn vị.
- Bước 4: Hoán đổi $a[i+1]$ và $a[\text{right}]$, ta có pivot_index chính là $i+1$. Trả về $i+1$. Kết thúc thuật toán.

Các bước thực hiện Quick Sort (phiên bản đệ quy):

Đặt $\text{left}=0$, $\text{right}=n-1$.

- Bước 1 : Nếu dãy a chỉ có một phần tử thì thuật toán kết thúc. Ngược lại thì thực hiện bước 2.
- Bước 2 : Chọn pivot trên dãy con đang xét trong đoạn từ $a_{\text{left}}, \dots, a_{\text{right}}$.
- Bước 3 : Thực hiện công việc phân đoạn (partition) cho dãy con đang xét.
- Bước 4 : Gọi đệ quy Quick Sort cho dãy con của a trong đoạn từ $a_{\text{left}}, \dots, a_{\text{pivot}-1}$.
- Bước 5 : Gọi đệ quy Quick Sort cho dãy con của a trong đoạn từ $a_{\text{pivot}+1}, \dots, a_{\text{right}}$.

Các bước thực hiện Quick Sort (phiên bản không đệ quy):

- Bước 1 : Khởi tạo stack rỗng, lần lượt push chỉ số của phần tử đầu và cuối của mảng đang xét vào stack.
- Bước 2 : Trong khi stack chưa rỗng, ta thực hiện các bước 3,4,5. Ngược lại, kết thúc thuật toán.
- Bước 3 : Lần lượt pop stack và cập nhật các giá trị pop từ stack lần lượt cho right và left .
- Bước 4 : Thực hiện công việc phân đoạn (partition) cho mảng đang xét trên dãy con của a có chỉ số từ left đến right .
- Bước 5 :
 - Nếu $\text{pivot_index} - 1 > \text{left}$, ta lần lượt push $\text{index}-1$ và left vào stack.

- Nếu $\text{pivot_index} + 1 < \text{right}$, ta lần lượt push $\text{pivot_index} + 1$ và right vào stack.

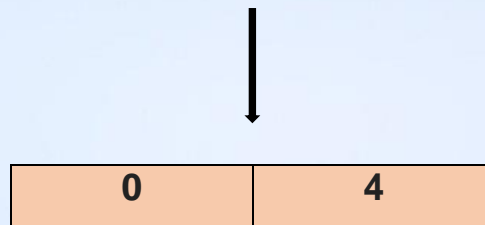
3. Ví dụ minh họa

Ta có bài toán sắp xếp dãy a thành một dãy không giảm với $a=[4,5,1,2,3]$. Gọi n là số phần tử của dãy $a(n=5)$, các phần tử của dãy a có chỉ số từ 0 đến $n-1$ (như hình vẽ bên dưới):

4	5	1	2	3
0	1	2	3	4

Trong ví dụ này ta sẽ sử dụng thuật toán Quick Sort (phiên bản không đệ quy) và chọn pivot bằng phương pháp *median of three* (phương pháp chọn pivot là phần tử trung vị trong ba phần tử đầu, giữa và cuối dãy).

Khởi tạo stack rỗng. Lần lượt push chỉ số của phần tử đầu và cuối của mảng đang xét vào stack.



- Ta thực hiện các công việc sau cho đến khi stack rỗng:
 - Lần lượt pop stack và cập nhật các giá trị pop từ stack lần lượt cho right và left : Ta có $\text{right}=4$, $\text{left}=0$.
 - Thực hiện công việc phân đoạn (partition) cho mảng đang xét trên dãy con a có chỉ số từ left đến right (*):
 - Chọn pivot bằng phương pháp *median of three* : ta thấy $1 \leq 3 \leq 4 \Rightarrow$ ta chọn 3 làm pivot ($\text{pivot_index}=2$). Ta thực hiện phân đoạn (partition) cho dãy đang xét:

1	2	3	5	4
0	1	2	3	4

- Hoán đổi pivot và $a[\text{right}]$:

1	2	4	5	3
0	1	2	3	4

- Đặt $i = \text{left} - 1$, $j = \text{left}$, khi $j < \text{right}$, ta thực hiện các bước sau:

➤ Nếu $a[j] \leq a[\text{right}]$, ta tăng i lên 1 đơn vị, rồi hoán đổi $a[i]$ và $a[j]$.

➤ Tăng j lên 1 đơn vị.

- Xét $j=0$; $i = -1$; ta có $a[0] = 1 \leq a[\text{right}] = 3$, ta tăng i lên một đơn vị ($i = -1 + 1 = 0$), sau đó hoán đổi $a[i]$, $a[j]$.

i ↓	j ↓			
1	2	4	5	3
0	1	2	3	4

- Xét $j=1$, $i=0$, ta có $a[1] = 2 \leq a[\text{right}] = 3$, ta tăng i lên 1 đơn vị ($i=1$), sau đó hoán đổi $a[i]$, $a[j]$.

i ↓	j ↓			
1	2	4	5	3
0	1	2	3	4

- Xét $j=2$, $i=1$, ta có $a[2] = 4 > a[\text{right}]$

i ↓	j ↓			
1	2	4	5	3
0	1	2	3	4

- Xét $j=3$, $i=1$, ta có $a[3]=5 > a[\text{right}] = 3$

	i			j	
	↓			↓	
1	2	4	5	3	
0	1	2	3	4	

- Xét $j=4 = \text{right}$, ta hoán đổi $a[\text{right}]$ và $a[i+1]$, ta có dãy a đã được phân vị với $\text{pivot} = 3$ và $\text{pivot_index} = i+1=2$

		$i+1$		j	
		↓		↓	
1	2	3	5	4	
0	1	2	3	4	

- Nếu $\text{pivot_index} - 1 > \text{left}$, ta lần lượt push index -1 và left vào stack (**).
- Nếu $\text{pivot_index} + 1 < \text{right}$, ta lần lượt push $\text{pivot_index}+1$ và right vào stack(***)
- Do $\text{pivot_index} - 1 = 2 > \text{left}$ ta thực hiện (**):

0	1
---	---

- Do $\text{pivot_index} + 1 = 3 < \text{right}$, ta thực hiện (***):

0	1	3	4
---	---	---	---

- Pop stack và cập nhật các giá trị từ stack lần lượt cho right và left:
Ta có $\text{right} = 4$, $\text{left} = 3$.
- Ta thực hiện (*), ta có dãy a sau khi được phân đoạn:

1	2	3	4	5
0	1	2	3	4

- Ta có $\text{pivot_index} = 3$. Do $\text{pivot_index} + 1 = 4 = \text{right}$ và $\text{pivot_index} - 1 = 2 < \text{left}$, ta không thực hiện (*) và (**). Pop stack và cập nhật các giá trị pop từ stack lần lượt cho right và left: Ta có $\text{right} = 1$, $\text{left} = 0$.
- Ta có $\text{pivot_index} = 0$. Do $\text{pivot_index} + 1 = 1 = \text{right}$ và $\text{pivot_index} - 1 = -1 < \text{left}$, ta không thực hiện (*) và (**).
- Lúc này stack rỗng . Dừng thuật toán.

1	2	3	4	5
0	1	2	3	4

4. Độ phức tạp

- **Về thời gian:**
 - Tốt nhất : $O(n \log_2 n)$
 - Trung bình : $O(n \log_2 n)$
 - Xấu nhất : $O(n^2)$
- **Về không gian :** Tùy vào cách thực hiện

5. Ưu điểm và nhược điểm

- **Ưu điểm:** là một trong các thuật toán sắp xếp chạy nhanh nhất đối với các tập dữ liệu có kích thước trung bình
- **Nhược điểm :** Quick Sort chậm hơn Merge Sort về thời gian chạy trong trường hợp xấu nhất khi mà độ phức tạp trong trường hợp xấu nhất là $O(n^2)$. Thời gian thực thi của Quick Sort ảnh hưởng rất nhiều bởi pivot. Việc chọn được một pivot tốt là rất khó thực hiện. Bên cạnh đó , khi thực hiện Quick Sort(phiên bản đệ quy) đối với tập dữ liệu lớn sẽ tiêu hao bộ nhớ, có thể dẫn đến tình trạng tràn bộ nhớ. Ta có thể khắc phục các vấn đề trên thông qua việc chọn pivot bằng các kỹ thuật chọn pivot như *median of three* và thực hiện Quick Sort không đệ quy.

IX. Counting Sort

1. Ý tưởng

Counting sort là một thuật toán sắp xếp cực nhanh một mảng các phần tử mà mỗi phần tử là các số nguyên không âm; Hoặc là một danh sách các ký tự được ánh xạ về dạng số để sort theo bảng chữ cái. Counting sort là một thuật toán sắp xếp các con số nguyên không âm, không dựa vào so sánh. Trong khi các thuật toán sắp xếp tối ưu sử dụng so sánh có độ phức tạp $O(n \log n)$ thì Counting sort chỉ cần $O(n)$ nếu độ dài của danh sách không quá nhỏ so với phần tử có giá trị lớn nhất.

2. Thuật toán

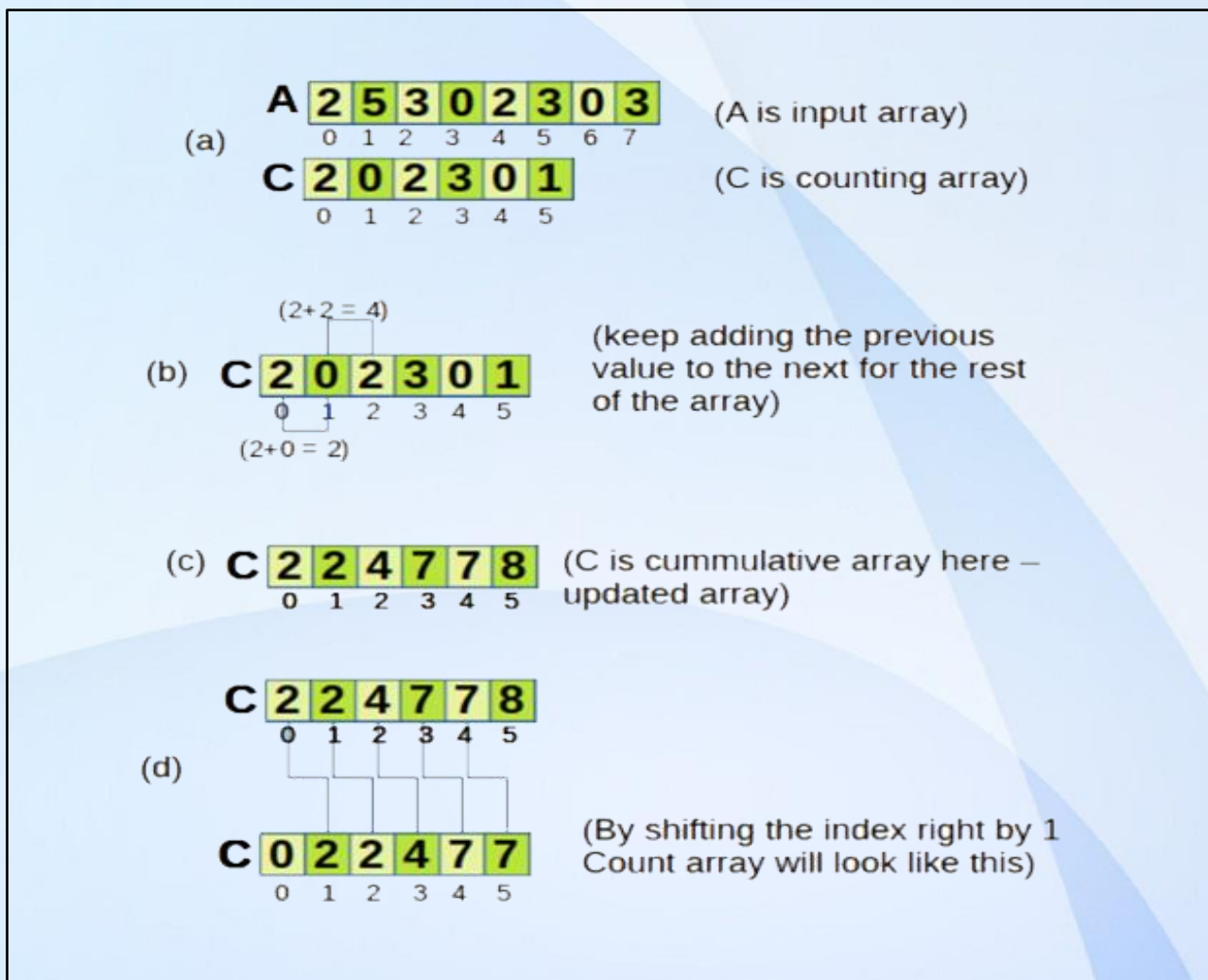
- Bước 1 : Tìm ra phần tử lớn nhất (giả sử là \max) từ mảng đã cho.
- Bước 2 : Khởi tạo một mảng count có độ dài là $\max+1$ với tất cả các phần tử 0. Mảng này được sử dụng để lưu trữ số lượng các phần tử trong mảng.
- Bước 3 : Lưu trữ số lượng của từng phần tử tại chỉ mục tương ứng của chúng trong mảng *đếm*.
Ví dụ: Nếu số phần tử của 3 là 2 thì 2 được lưu ở vị trí thứ 3 của mảng *đếm*. Nếu phần tử 5 không xuất hiện trong mảng, thì 0 được lưu ở vị trí thứ 5.
- Bước 4 : Lưu trữ tổng tích lũy của các phần tử của mảng *đếm*. Nó giúp đặt các phần tử vào chỉ số chính xác của mảng đã sắp xếp.
- Bước 5 : Tìm chỉ số của từng phần tử của mảng ban đầu trong mảng *đếm*. Điều này sẽ cho biết số đếm tích lũy.
- Bước 6 : Sau khi đặt mỗi phần tử vào đúng vị trí của nó, ta sẽ giảm số đếm của nó đi một đơn vị.

3. Ví dụ minh họa

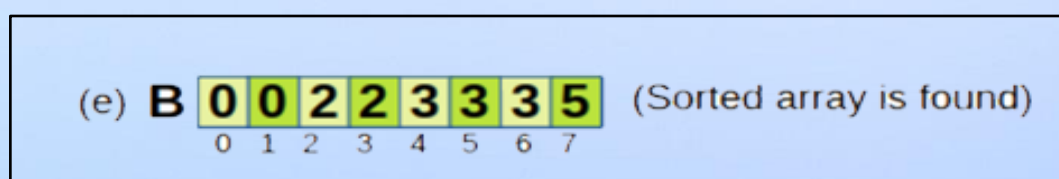
Phép toán ĐẾM - SẮP XẾP trên mảng đầu vào A []=[2,5,3,0,2,3,0,3], trong đó mỗi phần tử của A là một số nguyên không âm không lớn hơn k=5. Với A là mảng đầu vào cần được sắp xếp, C là mảng ta tạo riêng để thực hiện giải thuật đếm.

Thuật toán được tiến hành theo các bước a), b), c), d),e)

Xem ví dụ ở hình bên dưới.



Bây giờ lấy mảng B mới có cùng độ dài với mảng ban đầu. Đặt số ở giá trị tại chỉ số đó và sau đó tăng giá trị tại chỉ số của mảng đếm.



4. Độ phức tạp

- Về thời gian:
 - Tốt nhất : $O(n+k)$
 - Trung bình : $O(n+k)$
 - Xấu nhất : $O(n+k)$
- Về không gian : $O(k)$

5. Ưu điểm và nhược điểm

- Ưu điểm :
 - Độ phức tạp thời gian tuyến tính. Vì nó không phải là sắp xếp dựa trên so sánh, nó không bị giới hạn dưới độ phức tạp $O(n \log n)$.
 - Giảm độ phức tạp của không gian nếu phạm vi của các phần tử hẹp, tức là có nhiều tần suất xuất hiện các số nguyên gần nhau hơn.
- Nhược điểm :
 - Sự phức tạp về thời gian và không gian đều tăng vọt nếu phạm vi số lượng đầu vào lớn.
 - Nó chỉ hoạt động cho các giá trị rời rạc như số nguyên.
 - Trong trường hợp có liên quan đến các số nguyên âm, thì độ phức tạp tăng lên, cũng như các thay đổi nhất định trong thuật toán là bắt buộc.

X. Radix Sort

1. Ý tưởng:

Giả sử mỗi phần tử a_i trong dãy a_0, a_1, \dots, a_{n-1} là một số nguyên có tối đa m chữ số. Phân loại các phần tử này lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, ..., tương tự việc phân loại thư theo tỉnh thành, quận huyện, phường xã, ...

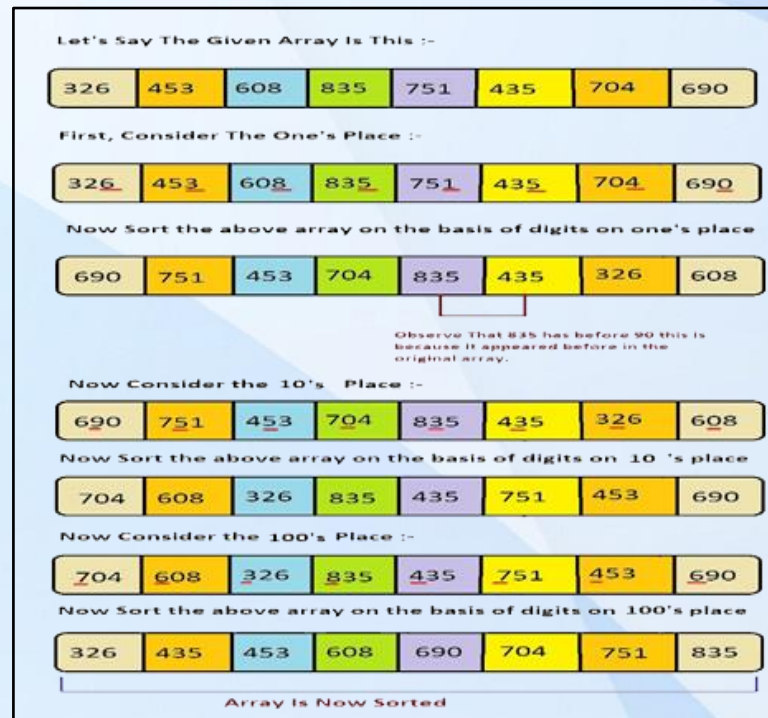
2. Thuật toán

- 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, \dots, B_9 rỗng;
- Bước 3: For $i = 1..n$ Đặt a_i vào lô B_t với $t =$ chữ số thứ k của a_i .
- Bước 3: Nối B_0, B_1, \dots, B_9 lại theo đúng trình tự thành a .
- Bước 4: $k = k + 1$.
 Nếu $k < m$: lặp lại bước 2. Ngược lại thì dừng.

3. Ví dụ minh họa

- Giả sử mảng đầu vào là:
- [326, 453, 608, 835, 751, 435, 704, 690]
- Dựa trên thuật toán, chúng ta sẽ sắp xếp mảng đầu vào theo chữ số của một người (chữ số có nghĩa nhỏ nhất).
- Mảng ban đầu được sắp xếp dựa trên [6, 3, 8, 5, 1, 5, 4, 0] bằng cách sử dụng sắp xếp đếm (Counting Sort)
- Vì vậy, mảng trở thành [690, 751, 453, 704, 835, 435, 326, 608]
- Bây giờ, chúng ta sẽ sắp xếp theo chữ số của mười:
- Mảng được sắp xếp một phần ở trên được sắp xếp dựa trên [9, 5, 5, 0, 3, 3, 2, 0] bằng cách sử dụng Sắp xếp đếm
- Bây giờ, mảng trở thành: [704, 608, 326, 835, 435, 751, 453, 690]
- Cuối cùng, chúng tôi sắp xếp theo chữ số hàng trăm (chữ số có nghĩa nhất):
- Mảng được sắp xếp một phần ở trên được sắp xếp dựa trên [7, 6, 3, 8, 4, 7, 4, 6] bằng cách sử dụng sắp xếp đếm(Counting Sort)
- Mảng trở thành: [326, 435, 453, 608, 690, 704, 751, 835] được sắp xếp.

Xem ảnh bên dưới để hiểu ví dụ



4. Độ phức tạp

- Về thời gian :
 - Tốt nhất : $O(n+k)$
 - Trung bình : $O(n+k)$
 - Xấu nhất : $O(n+k)$
- Về không gian : $O(n)$

5. Ưu điểm và nhược điểm

- Ưu điểm :
 - Nhanh khi các phím ngắn, tức là khi phạm vi của các phần tử mảng ít hơn.
 - Được sử dụng trong các thuật toán hàng số mảng hậu tố như thuật toán Manber và thuật toán DC3.
 - Radix Sort là sắp xếp ổn định vì thứ tự tương đối của các phần tử có giá trị bằng nhau được duy trì.
- Nhược điểm :

- Phụ thuộc vào chữ số hoặc chữ cái, Radix Sort kém linh hoạt hơn nhiều so với các loại khác. Do đó, đối với mọi loại dữ liệu khác nhau, nó cần phải được viết lại.
- Hằng số cho sắp xếp Radix lớn hơn so với các thuật toán sắp xếp khác.
- Nó chiếm nhiều không gian hơn so với Quicksort được sắp xếp tại chỗ.
- Sắp xếp theo căn cứ có thể chậm hơn các thuật toán sắp xếp khác như sắp xếp hợp nhất và sắp xếp nhanh, nếu các hoạt động không đủ hiệu quả. Các hoạt động này bao gồm chức năng chèn và xóa của danh sách con và quá trình cô lập các chữ số mà chúng ta muốn.
- Sắp xếp theo cơ số ít linh hoạt hơn các cách sắp xếp khác vì nó phụ thuộc vào các chữ số hoặc chữ cái. Sắp xếp theo cơ số cần được viết lại nếu kiểu dữ liệu bị thay đổi.

XI. Flash Sort

1. Ý tưởng :

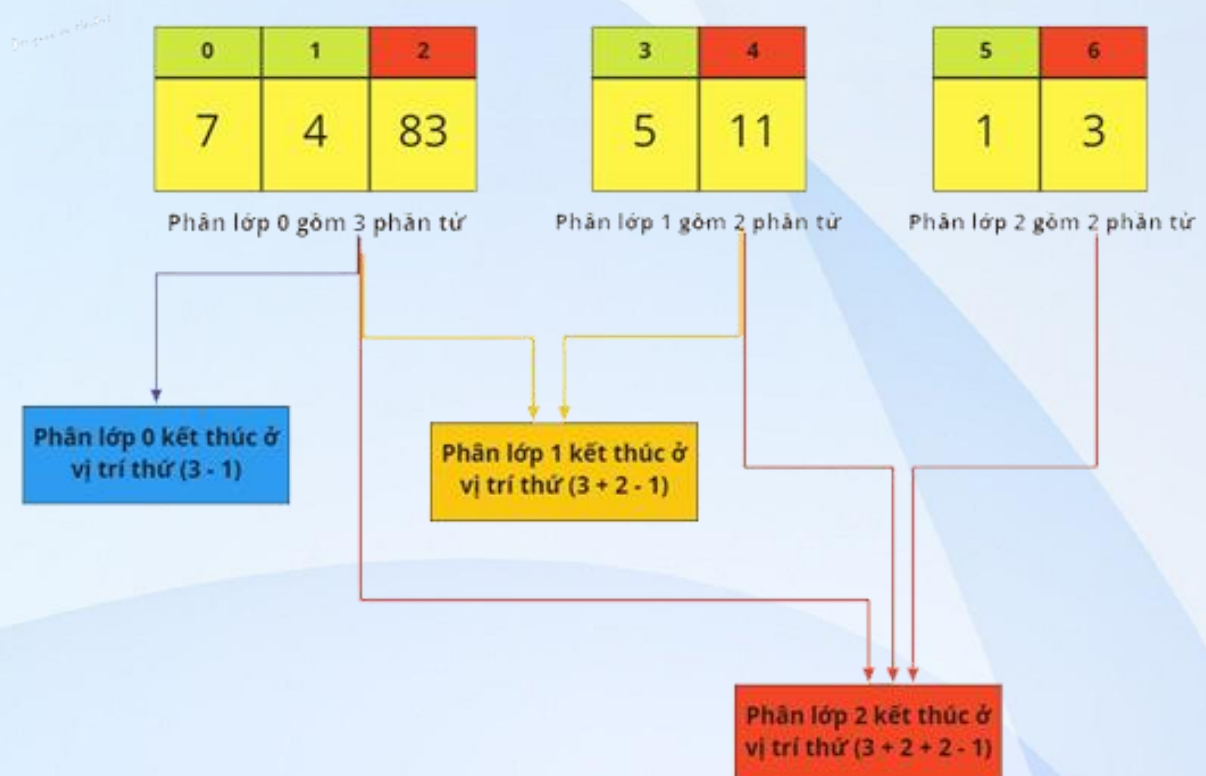
Flash sort là một thuật toán sắp xếp tại chỗ (in-situ, không dùng mảng phụ) có độ phức tạp $O(n)$, không đệ qui, gồm có 3 bước: Phân lớp dữ liệu, tức là dựa trên giả thiết dữ liệu tuân theo 1 phân bố nào đó, chẳng hạn phân bố đều, để tìm 1 công thức ước tính vị trí (lớp) của phần tử sau khi sắp xếp. Hoán vị toàn cục, tức là dời chuyển các phần tử trong mảng về lớp của mình. Sắp xếp cục bộ, tức là để sắp xếp lại các phần tử trong phạm vi của từng lớp.

2. Thuật toán

Giai đoạn 1 : Phân lớp dữ liệu:

- Bước 1: Tìm giá trị nhỏ nhất của các phần tử trong mảng(minVal) và vị trí phần tử lớn nhất của các phần tử trong mảng(max).
- Bước 2: Khởi tạo 1 vector L có m phần tử (ứng với m lớp, trong source code lần này chọn số lớp bằng $0.45n$).

- Bước 3: Đếm số lượng phần tử các lớp theo quy luật, phần tử $a[i]$ sẽ thuộc lớp $k = \text{int}((m - 1) * (a[i] - \text{minVal}) / (a[\text{max}] - \text{minVal}))$.
- Bước 4: Tính vị trí kết thúc của phân lớp thứ j theo công thức $L[j] = L[j] + L[j - 1]$ (j tăng từ 1 đến $m - 1$). Sở dĩ như vậy bởi ta có thể hình dung như sau:



Giai đoạn 2 : Hoán vị toàn cục:

Việc này sẽ hình thành các chu trình hoán vị: mỗi khi ta đem một phần tử ở đâu đó đến một vị trí nào đó thì ta phải nhấc phần tử hiện tại đang chiếm chỗ ra, và tiếp tục với phần tử bị nhấc ra và đưa đến chỗ khác cho đến khi quay lại vị trí ban đầu thì hoàn tất vòng lặp.

- Vấn đề 1: Vậy làm thế nào để khi cầm trong tay 1 phần tử nào đó, ta biết phải bỏ nó vào chỗ nào cho phù hợp? Câu trả lời thật đơn giản, ta tính xem nó phải nằm ở phân lớp nào, rồi bỏ nó vào cái vị trí hiện tại của phân

lớp đó, và vì ta bỏ vào phân lớp đó 1 phần tử nên phải lùi vị trí của phân lớp lại 1 đơn vị.

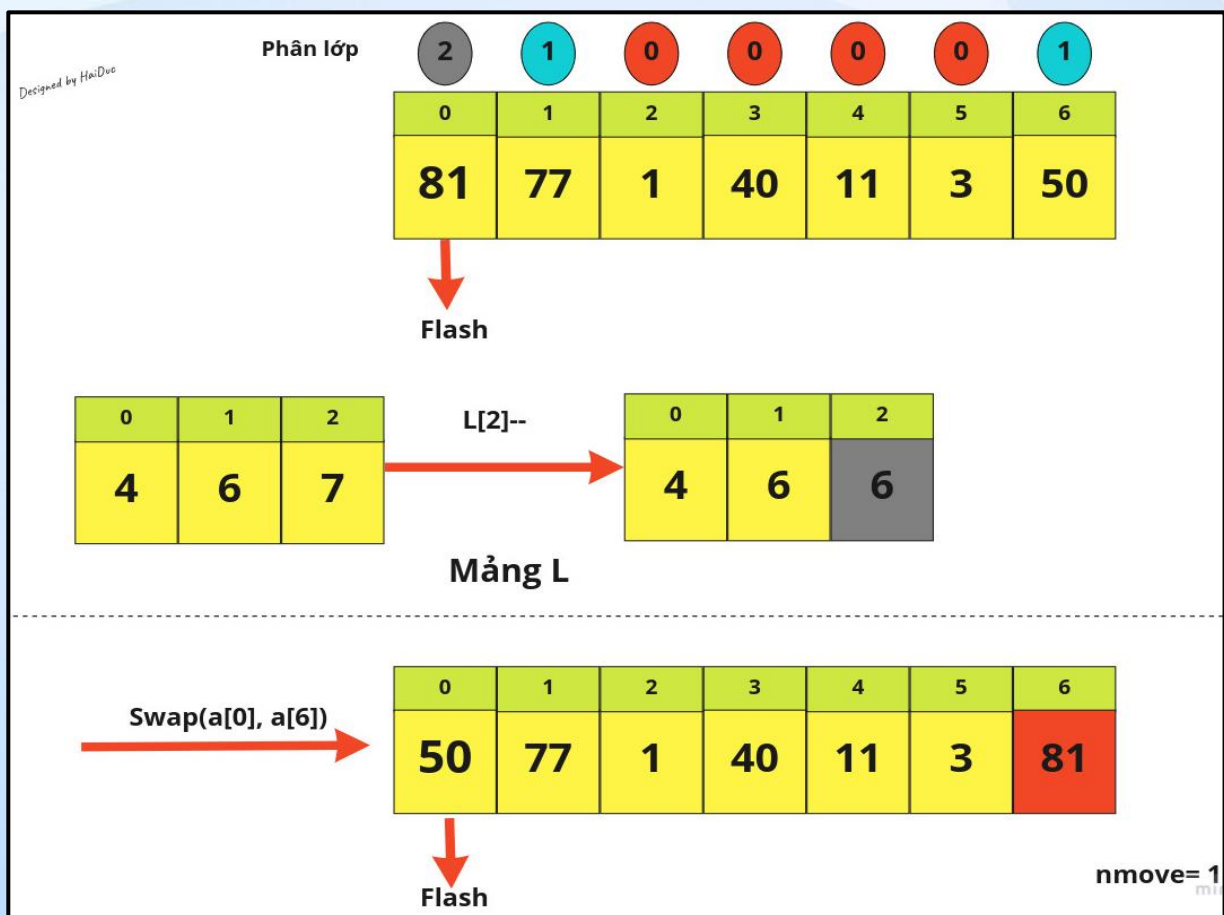
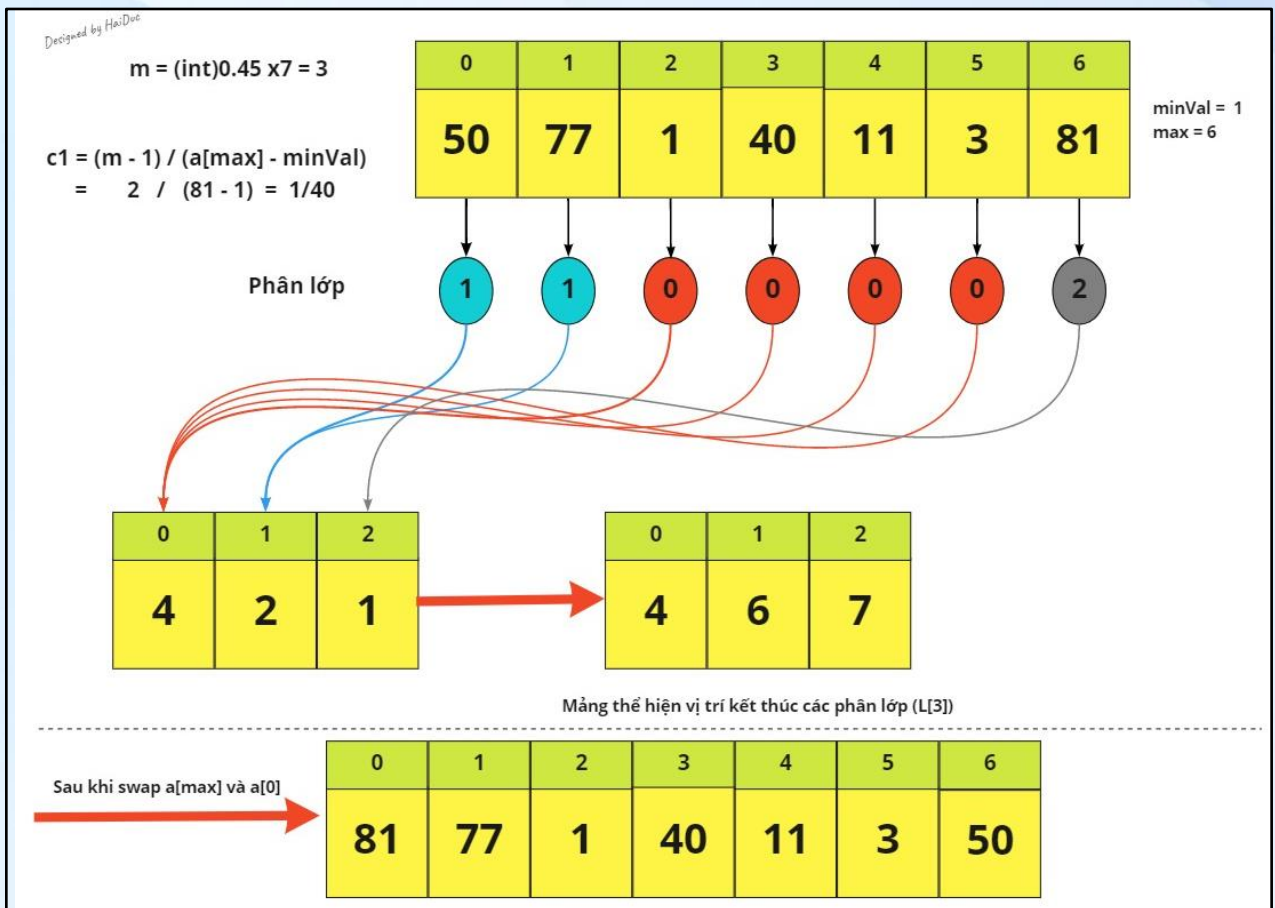
- Vấn đề 2: Mặc dù ta đã đi qua 1 chu trình nhưng vẫn còn những phần tử chưa ở đúng chỗ thì phải làm thế nào? Cũng đơn giản, tiến hành 1 chu trình khác cho đến khi không còn con thỏ nào trong chuồng bò và ngược lại nữa thì thôi.
- Vấn đề 3: Vậy là ta có một số các chu trình cần phải xử lý, vậy tại mỗi chu trình ta phải bắt đầu từ phần tử nào của mảng (Cycle leaders searching)? Đây cũng là một trong những điều đơn giản: ta chỉ việc duyệt từ đầu đến cuối mảng và tìm xem phần tử nào không nằm trong phân lớp ($I < L[K[I]]$), và bắt đầu bằng phần tử đó.

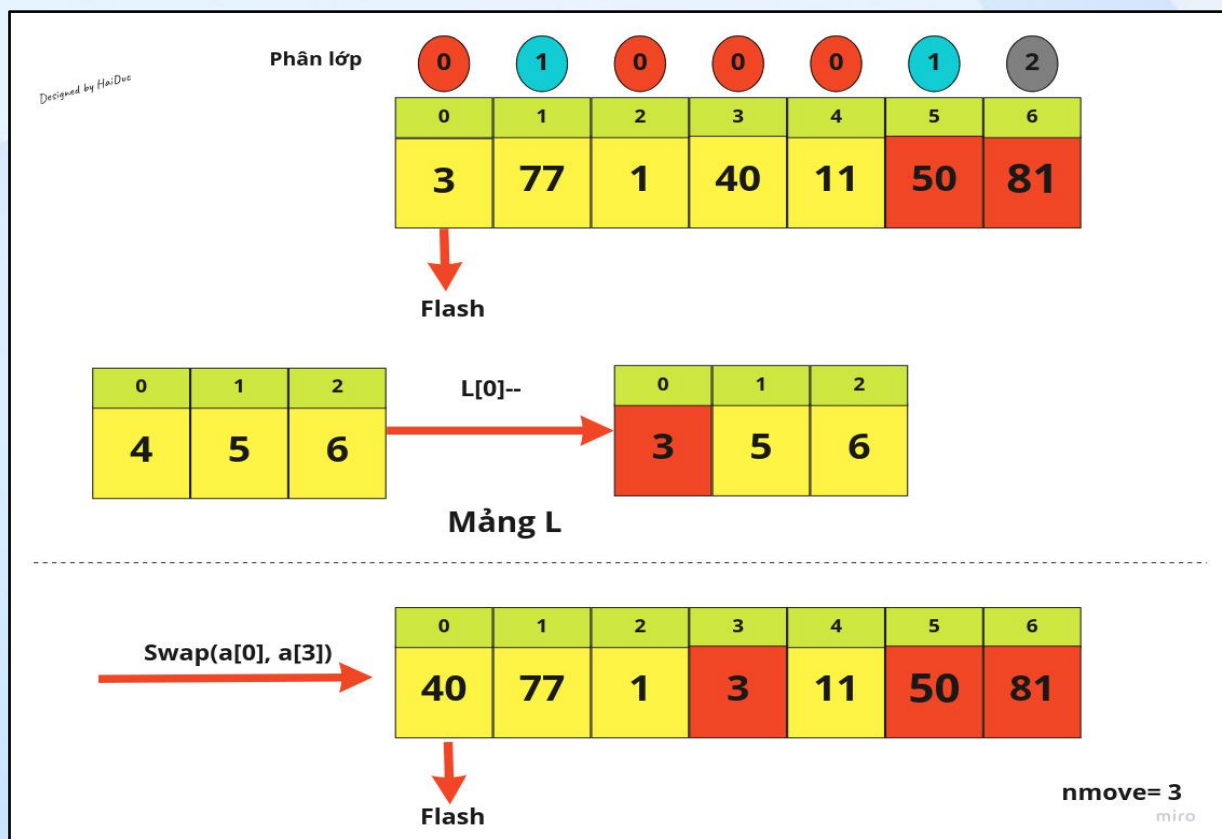
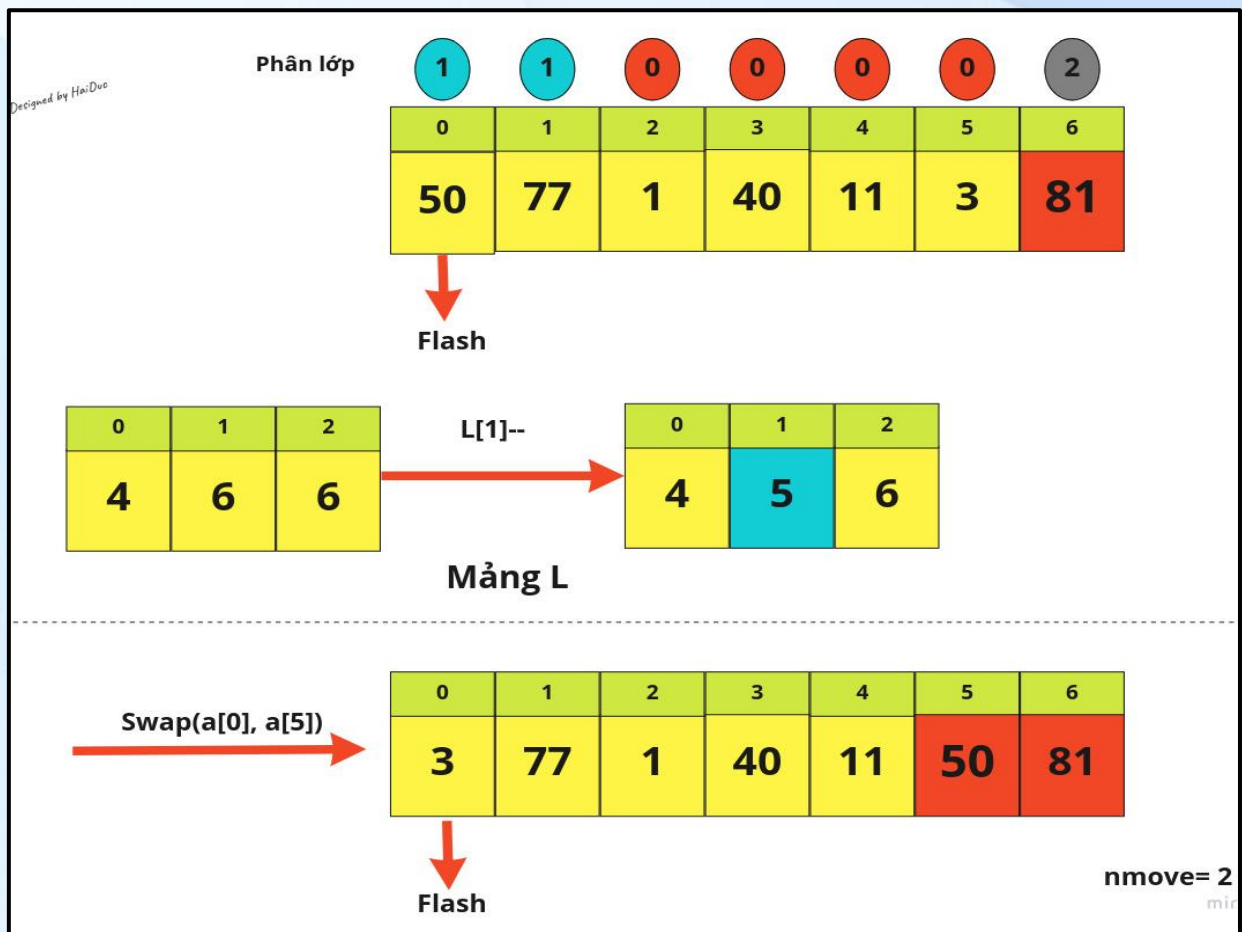
Giai đoạn 3 : Sắp xếp toàn cục:

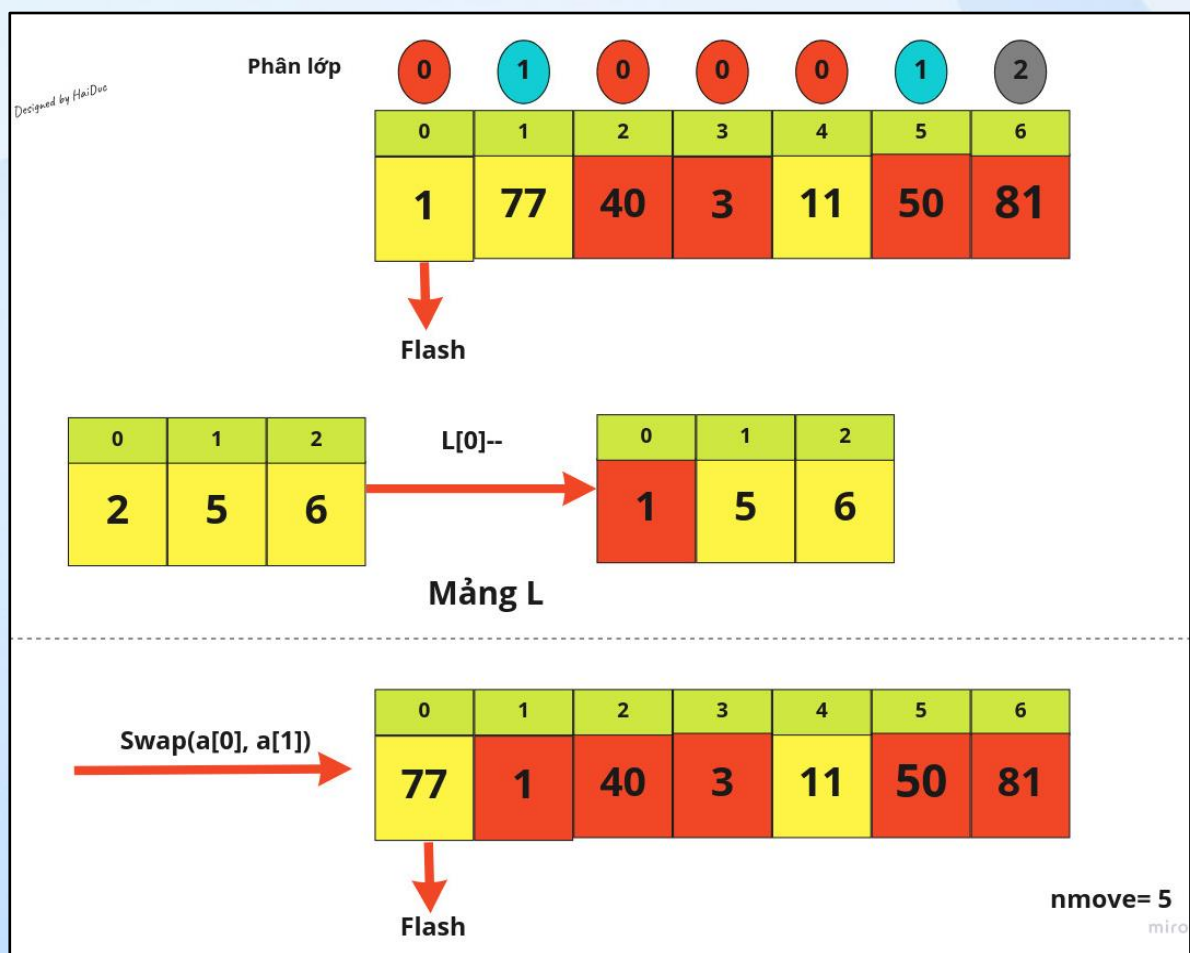
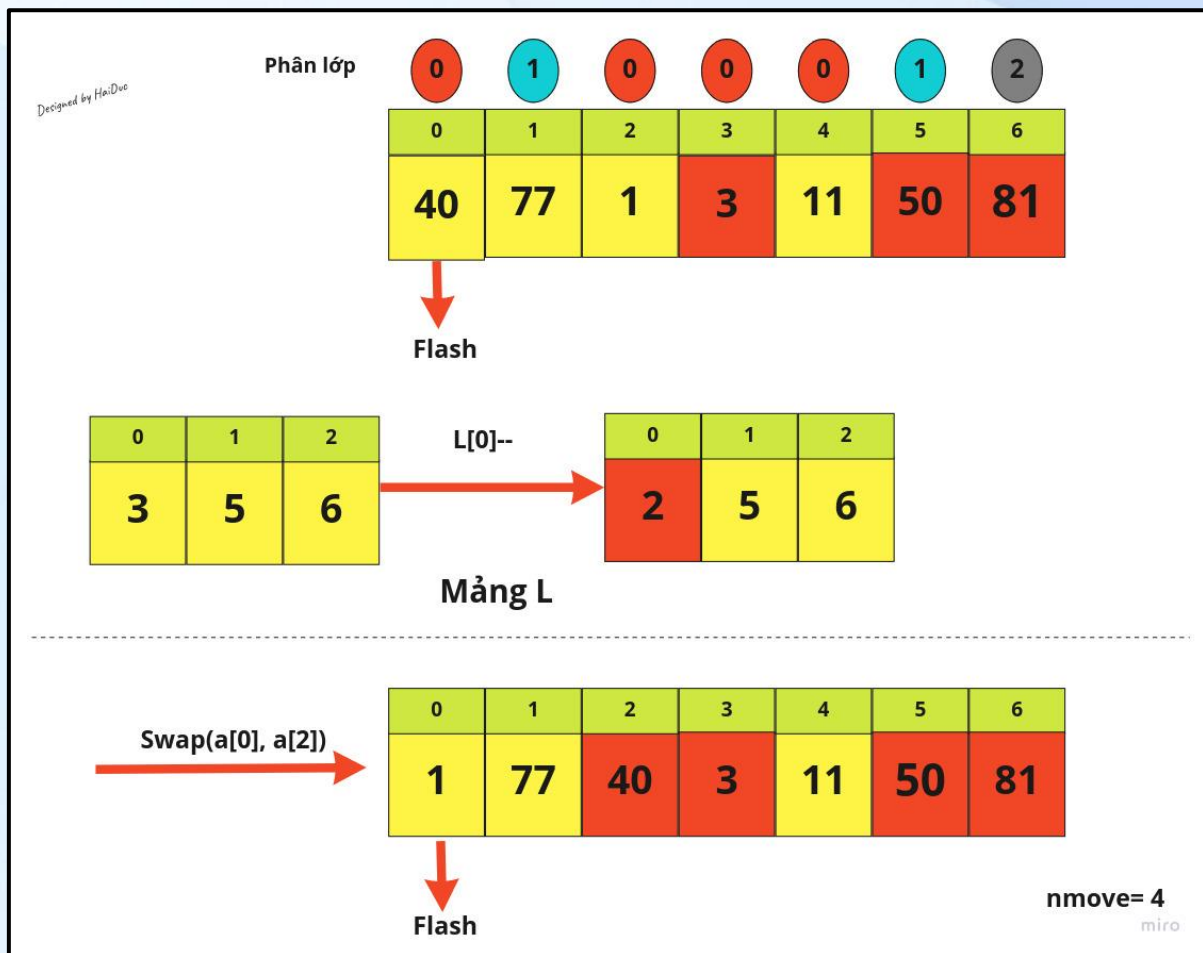
Sau bước *hoán vị toàn cục*, mảng của chúng ta hiện tại sẽ được chia thành các lớp (thứ tự các phần tử trong lớp vẫn chưa đúng) do đó để đạt được trạng thái đúng thứ tự thì khoảng cách phải di chuyển của các phần tử là không lớn vì vậy Insertion Sort sẽ là thuật toán thích hợp nhất để sắp xếp lại mảng có trạng thái như vậy.

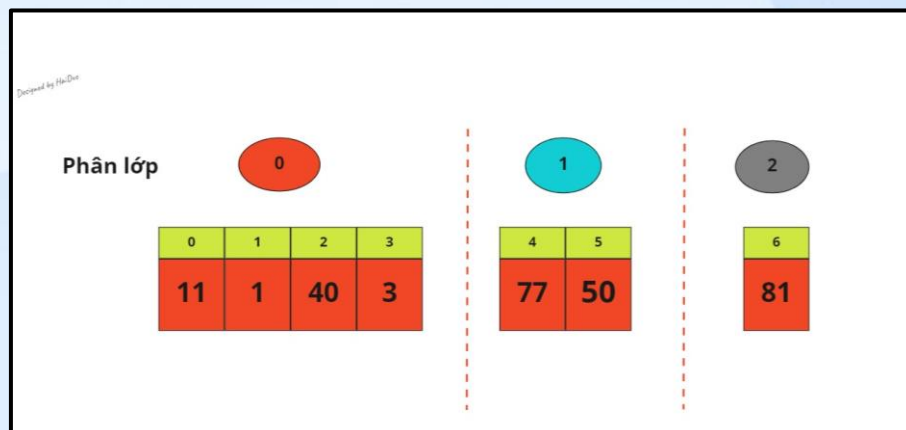
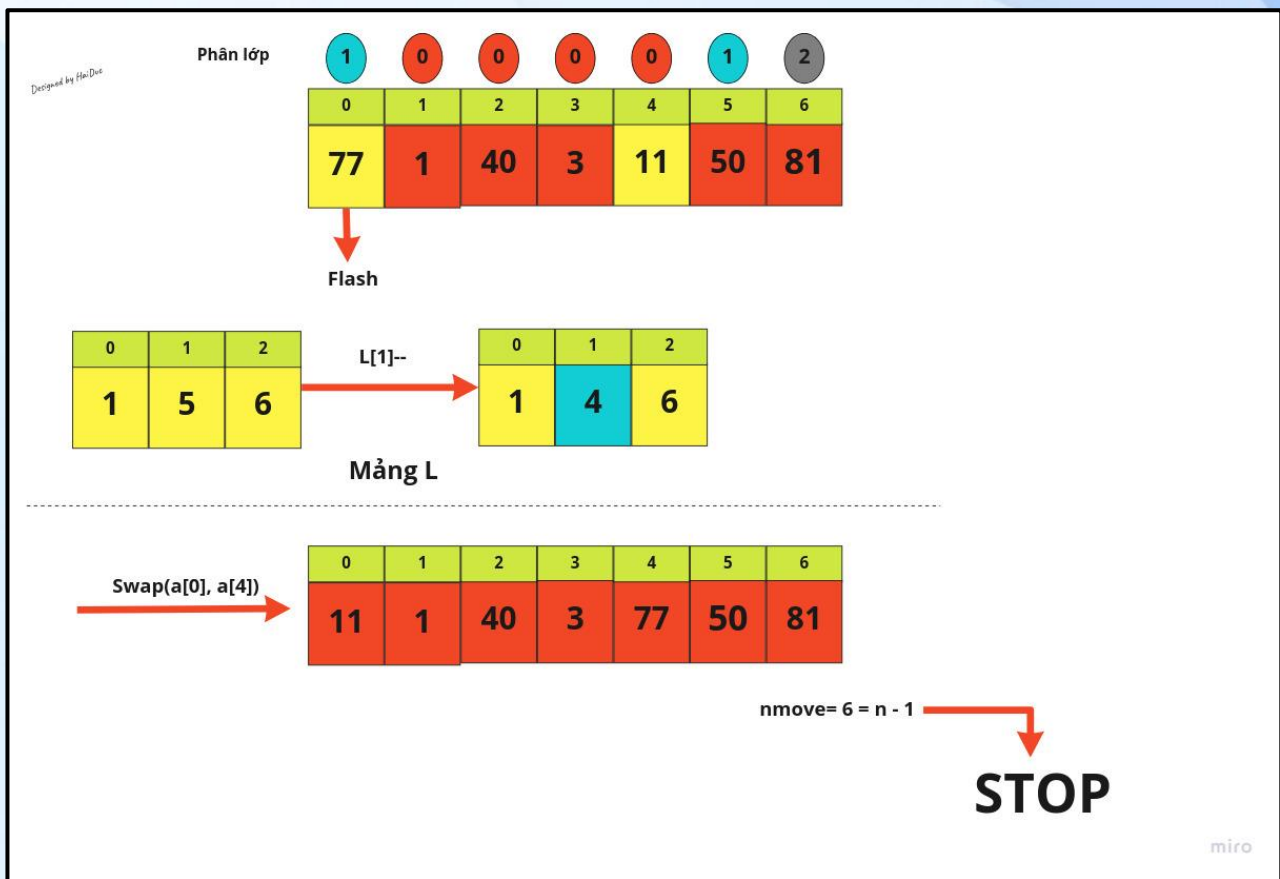
3. Ví dụ minh họa

Cho mảng a gồm 7 phần tử $a=\{50,77,1,40,11,3,81\}$, thực hiện giải thuật Flashsort như các hình minh họa bên dưới.









Đến đây, các phần tử của mảng sẽ về đúng với phân lớp của mình, do đó theo phân tích ở trên, ta dùng Insertion Sort để sắp xếp lại mảng này.

4. Độ phức tạp

- **Về thời gian :**
 - Tốt nhất : $O(1)$
 - Trung bình : $O(\log n)$
 - Xấu nhất : $O(n)$
- **Về không gian :** $O(n^2)$

5. Ưu điểm và nhược điểm

- **Ưu điểm :** Trong khi tiết kiệm bộ nhớ, Flashsort có nhược điểm là nó tính toán lại nhóm cho nhiều phần tử đã được phân loại. Điều này đã được thực hiện hai lần cho mỗi phần tử (một lần trong giai đoạn đếm nhóm và lần thứ hai khi di chuyển mỗi phần tử), nhưng việc tìm kiếm phần tử chưa được phân loại đầu tiên yêu cầu tính toán thứ ba cho hầu hết các phần tử. Điều này có thể tốn kém nếu các nhóm được chỉ định bằng cách sử dụng một công thức phức tạp hơn so với phép nội suy tuyến tính đơn giản.
- **Nhược điểm :** Nhược điểm là mảng được truy cập ngẫu nhiên, do đó không thể tận dụng bộ đệm dữ liệu nhỏ hơn toàn bộ mảng.

PHẦN 2 : KẾT QUẢ THỰC NGHIỆM VÀ NHẬN XÉT

I. Cấu hình máy tính thực hiện

- CPU: Intel Core I5 -9300HF(2.4 GHz – 4.1 GHz/8M cache/4 nhân, 8 luồng)
- RAM : 8GB DDR4 2400MHz
- VGA : NVIDIA GeForce GTX 1050 3GB GDDR5
- SSD : 512GB M.2 NVMe
- Hệ điều hành : Window 10 64bit Single Language
- Compiler : Microsoft Visual Studio 2019

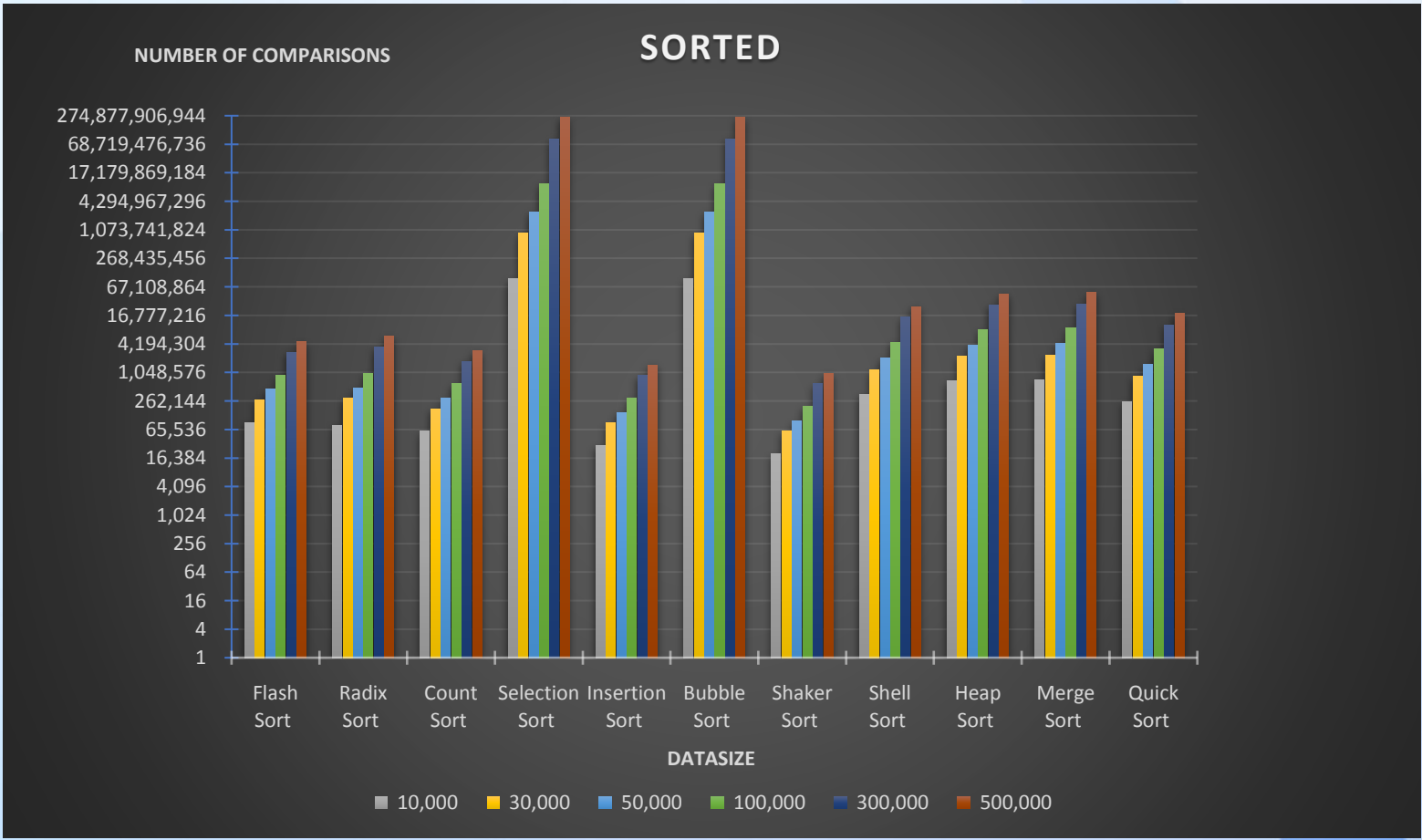
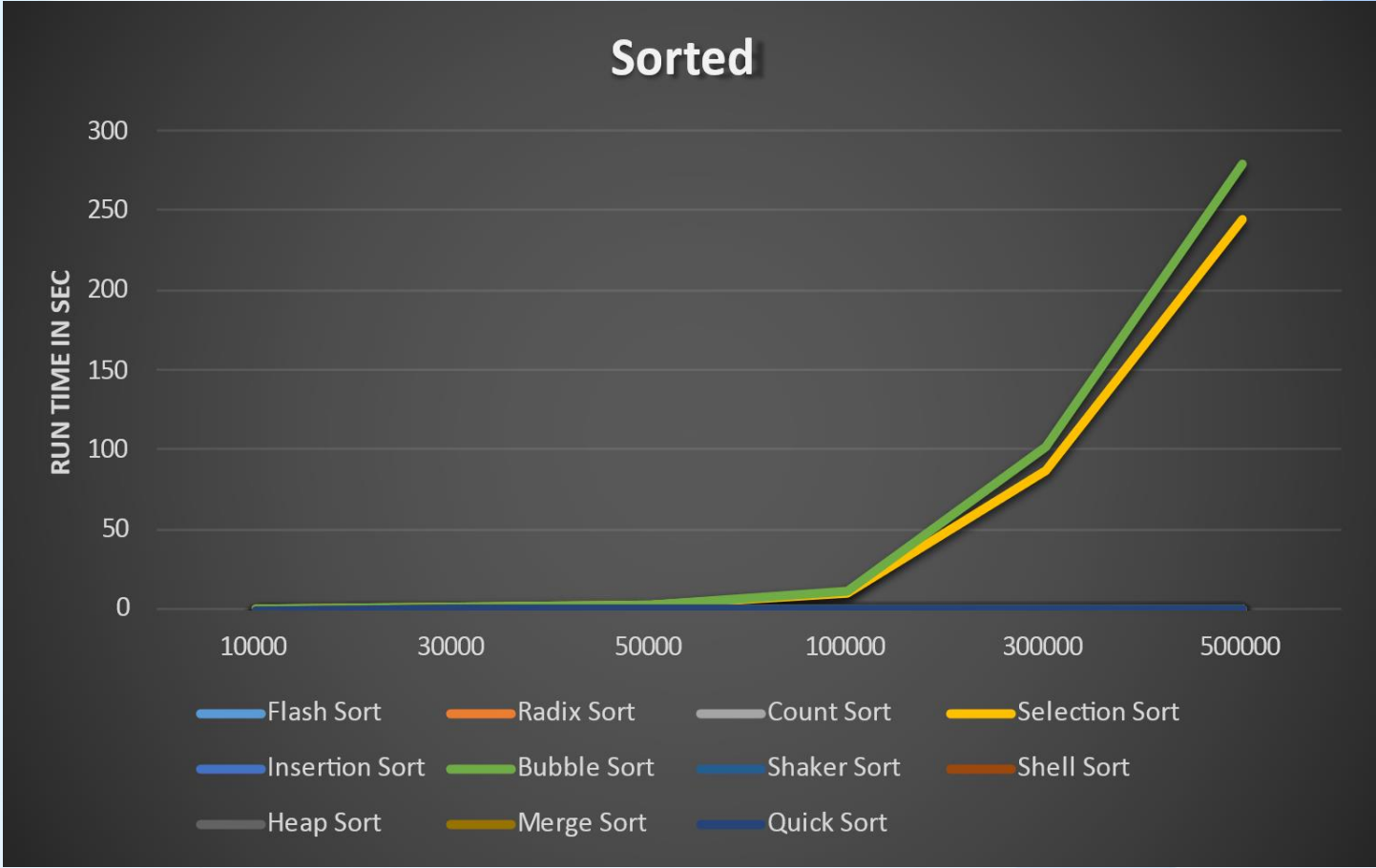
II. Kết quả thực nghiệm và nhận xét

Thực hiện lần lượt 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 với 4 Data Order Sorted Data,

Nearly Sorted, Reverse Sorted và Randomized với kích thước data đầu vào là 10000,30000,50000,100000,300000,500000. Input data là kiểu dữ liệu số nguyên không âm. Kết quả thực nghiệm theo thời gian và số lượng phép so sánh được tổng hợp thành các bảng và biểu đồ dưới đây.

1. Data order : Sorted

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
Flash Sort	1	92,904	2	278,704	3	464,504	7	929,004	22	2,787,004	37	4,645,004
Radix Sort	0	80,164	2	300,199	3	500,199	8	1,000,199	24	3,600,234	41	6,000,234
Count Sort	1	60,004	1	180,004	1	300,004	3	600,004	7	1,800,004	10	3,000,004
Selection Sort	184	100,009,999	1,642	900,029,999	4,559	2,500,049,999	18,602	10,000,099,999	165,548	90,000,299,999	467,575	250,000,499,999
Insertion Sort	0	29,998	0	89,998	1	149,998	1	299,998	2	899,998	4	1,499,998
Bubble Sort	180	100,009,999	1,626	900,029,999	4,918	2,500,049,999	19,005	10,000,099,999	165,810	90,000,299,999	460,260	250,000,499,999
Shaker Sort	0	20,002	0	60,002	0	100,002	1	200,002	1	600,002	2	1,000,002
Shell Sort	0	360,042	2	1,170,050	5	2,100,049	10	4,500,051	33	15,300,061	57	25,500,058
Heap Sort	11	684,780	36	2,275,500	66	3,991,520	138	8,504,270	450	27,805,800	787	48,028,500
Merge Sort	7	732,476	20	2,424,380	41	4,241,756	67	8,983,516	206	29,297,372	347	50,469,276
Quick Sort	1	257,324	4	864,466	7	1,518,930	14	3,237,858	37	10,651,426	63	18,451,426



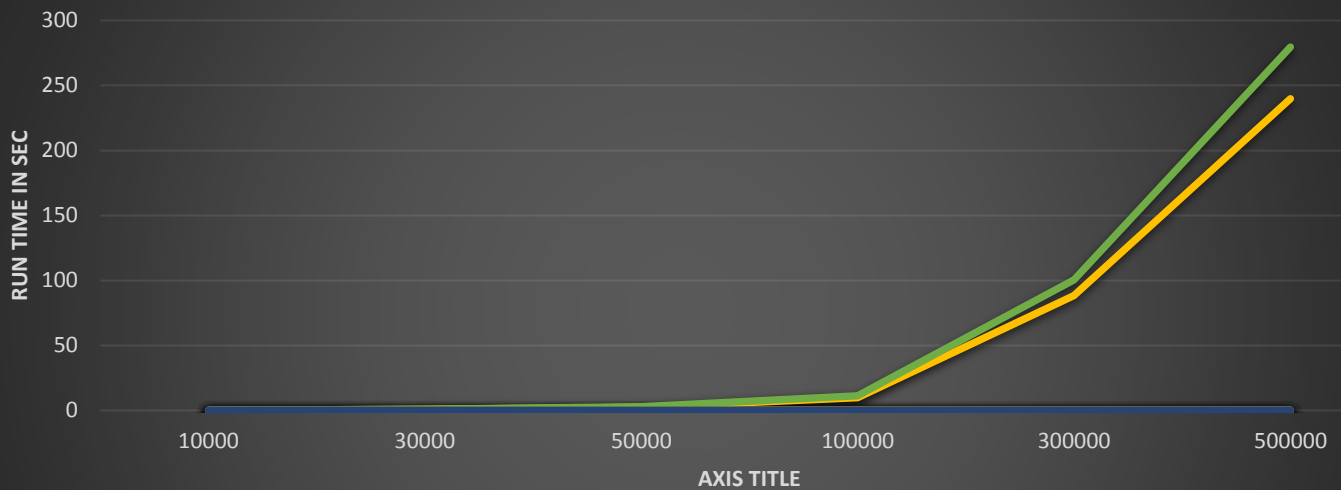
Nhân xét :

- Shaker sort: thuật toán có thời gian chạy nhanh nhất cùng với số lượng các phép so sánh ít nhất. Là một phiên bản cải tiến của bubble sort nhằm giảm số lượng phần tử cần sắp xếp dẫn đến việc giảm các phép so sánh và thời gian chạy. Sau đó là các thuật toán insertion sort, count sort, radix sort, flash và quick sort. đều là các thuật toán có hiệu suất cao (ít phép so sánh, thời gian chạy nhanh).
- Bubble sort: có thời gian chạy lâu nhất. mặc dù có số lượng phép so sánh bằng với Selection sort nhưng thời gian vẫn là lâu hơn với cùng số lượng phần tử. Vì có số lượng phép so sánh lớn nên việc gia tăng các phép so sánh với mỗi trường hợp số lượng đầu vào cũng có sự gia tăng nhanh nhất.
- Các thuật toán tầm trung bao gồm heap sort, merge sort.
- Về thời gian: do Bubble sort và Selection sort chỉ có độ hiệu quả cao đối với số lượng phần tử không quá lớn nên dẫn đến việc thời gian sắp xếp rất lớn ở các phép thử có số lượng đầu vào lớn.
- Lý do về số lượng so sánh selection sort và bubble sort không thể nhận diện các mảng đã sắp xếp nên việc lặp lại toàn bộ mảng dẫn đến việc số phép so sánh lớn.

2. Data Order : Nearly Sorted

Data Order: NearlySorted												
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
Flash Sort	1	92,880	3	278,681	4	464,477	7	928,980	22	2,786,980	38	4,644,982
Radix Sort	1	80,164	2	300,199	4	500,199	8	1,000,199	24	3,600,234	43	6,000,234
Count Sort	0	60,004	1	180,004	1	300,004	2	600,004	7	1,800,004	11	3,000,004
Selection Sort	184	100,009,999	1,666	900,029,999	4,602	2,500,049,999	18,460	10,000,099,999	165,912	90,000,299,999	463,339	250,000,499,999
Insertion Sort	0	142,586	0	417,886	0	550,554	1	637,794	2	1,275,554	3	1,998,678
Bubble Sort	183	100,009,999	1,643	900,029,999	4,636	2,500,049,999	18,297	10,000,099,999	165,794	90,000,299,999	457,415	250,000,499,999
Shaker Sort	0	145,551	0	502,071	1	588,110	0	628,009	1	935,793	2	1,517,729
Shell Sort	1	401,328	3	1,290,163	5	2,273,531	10	4,662,222	34	15,415,741	57	25,673,414
Heap Sort	11	684,230	37	2,275,390	65	3,990,500	138	8,504,180	455	27,805,830	786	48,028,530
Merge Sort	6	778,374	20	2,554,118	34	4,350,305	69	9,110,255	209	29,433,221	349	50,615,660
Quick Sort	1	268,360	3	899,014	6	1,541,020	12	3,286,766	38	10,706,210	63	18,488,546

Nearly Sorted



NUMBER OF COMPARISONS

NEARLY SORTED



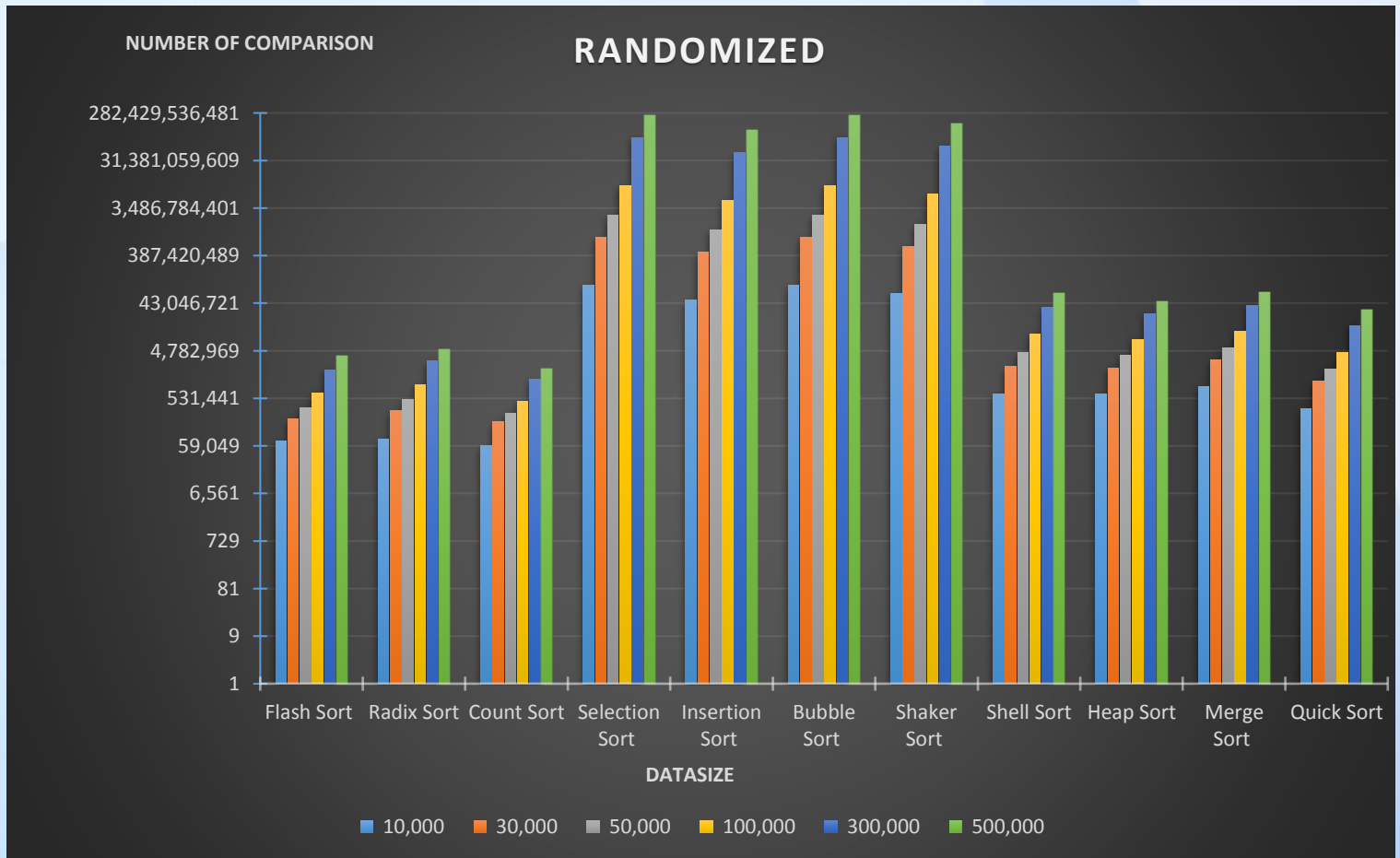
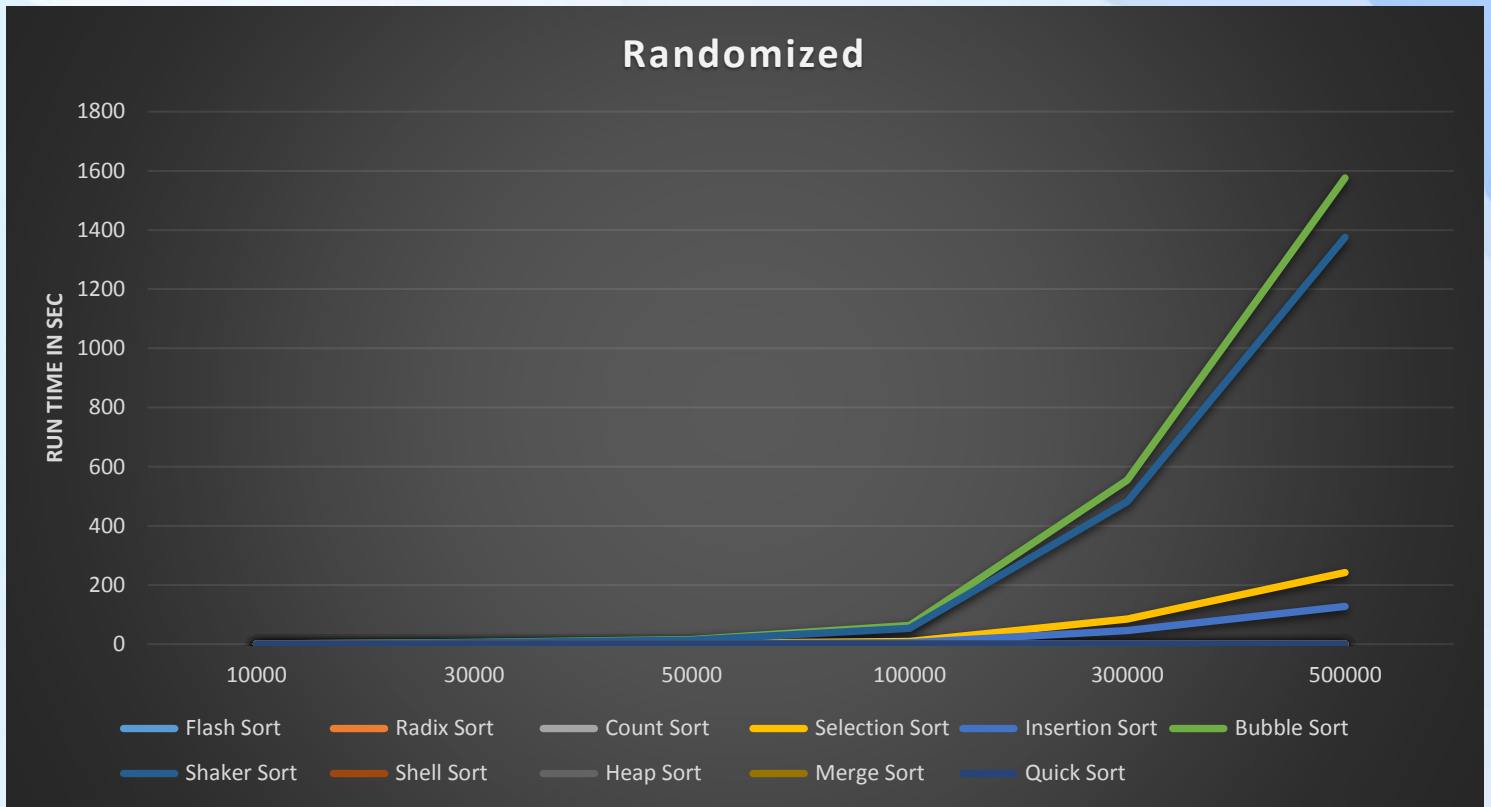
Nhận xét:

- Tương tự với Sorted, đầu vào đã gần như được sắp xếp có số liệu về thời gian cũng như phép so sánh thay đổi không quá nhiều với đầu vào đã được sắp xếp.
- Shaker sort và Bubble sort vẫn là những thuật toán hiệu suất thấp nhất (thời gian chạy lâu, số phép so sánh lớn).
- Thứ tự về hiệu suất của thuật toán gần như không thay đổi so với mảng đầu vào đã được sắp xếp (sorted).
- Các thuật toán tầm trung vẫn gồm heap sort và merge sort.
- Insertion sort và shaker sort vẫn là những thuật toán có hiệu suất cao nhất, sau đó là các thuật toán insertion sort, count sort, radix sort, flash và quick sort.
- Tương tự với mảng đầu vào đã được sắp xếp : selection sort và bubble sort không thể nhận diện các mảng đã sắp xếp nên việc lặp lại toàn bộ mảng mỗi vòng lặp dẫn đến việc các phép so sánh là rất lớn và thời gian chạy cũng rất lâu.

3. Data Order : Randomized

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
Flash Sort	1	73,349	2	202,458	4	349,381	8	685,955	26	1,990,792	53	3,699,932
Radix Sort	0	80,164	2	300,199	4	500,199	8	1,000,199	21	3,000,199	38	5,000,199
Count Sort	0	60,004	1	180,004	2	265,540	2	465,540	5	1,265,540	9	2,065,540
Selection Sort	186	100,009,999	1,725	900,029,999	4,643	2,500,049,999	18,512	10,000,099,999	173,160	90,000,299,999	472,041	250,000,499,999
Insertion Sort	0	49,884,391	0	450,219,602	1	1,253,127,977	1	5,014,100,284	2	44,977,970,692	4	124,754,172,417
Bubble Sort	185	100,009,999	1,715	900,029,999	4,595	2,500,049,999	18,403	10,000,099,999	172,080	90,000,299,999	466,100	250,000,499,999
Shaker Sort	0	67,664,487	0	597,682,559	1	1,668,964,282	0	6,674,349,307	1	60,057,461,867	2	170,089,628,423
Shell Sort	0	658,752	3	2,315,422	5	4,478,477	10	10,455,449	34	35,940,205	57	66,224,897
Heap Sort	13	645,820	37	2,173,775	65	3,812,330	140	8,126,430	496	26,728,095	779	46,375,020
Merge Sort	8	913,008	21	3,112,632	34	5,425,841	69	11,531,030	219	38,884,203	346	71,591,760
Quick Sort	1	329,338	5	1,175,282	9	2,042,732	20	4,365,006	64	15,501,914	108	31,209,398

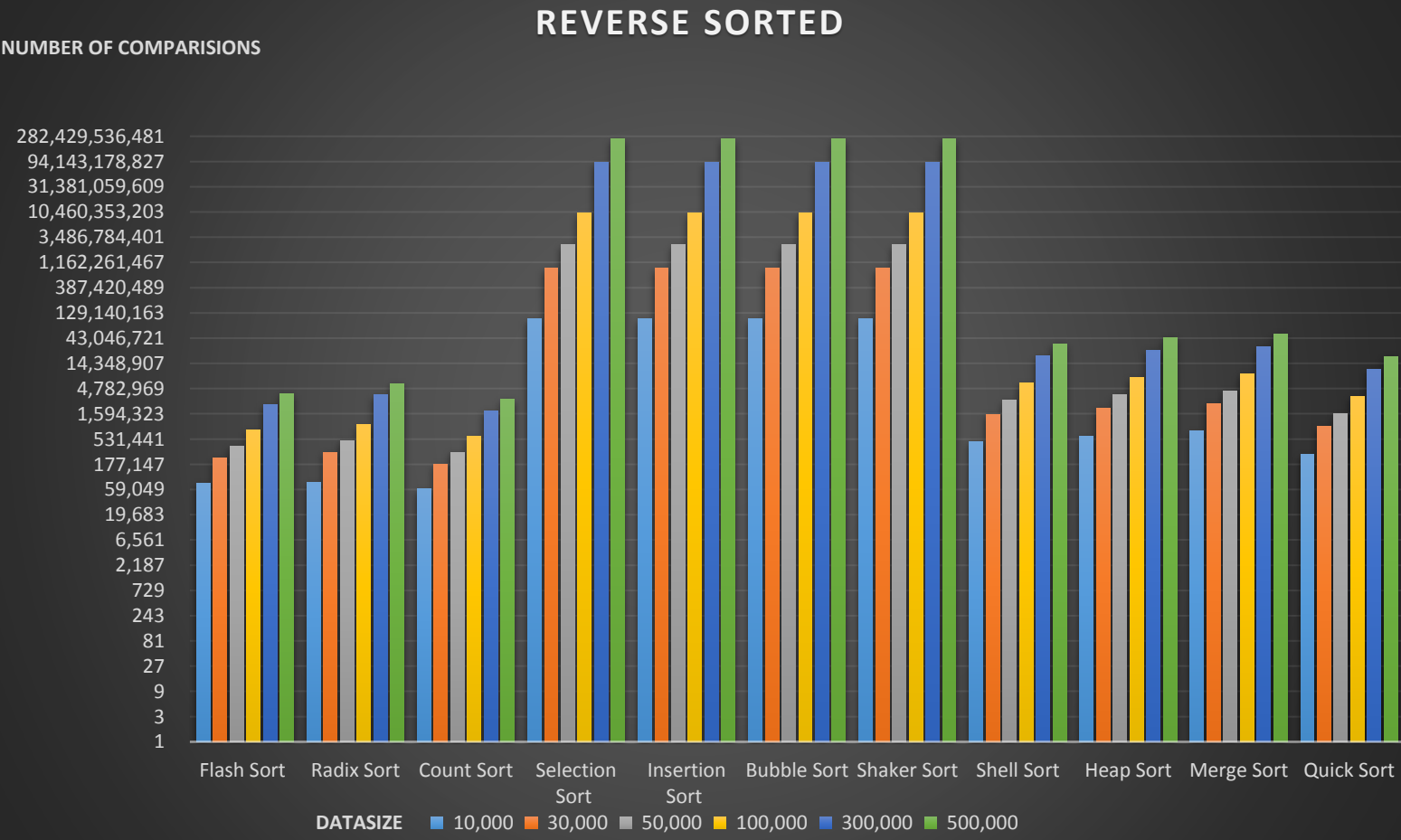
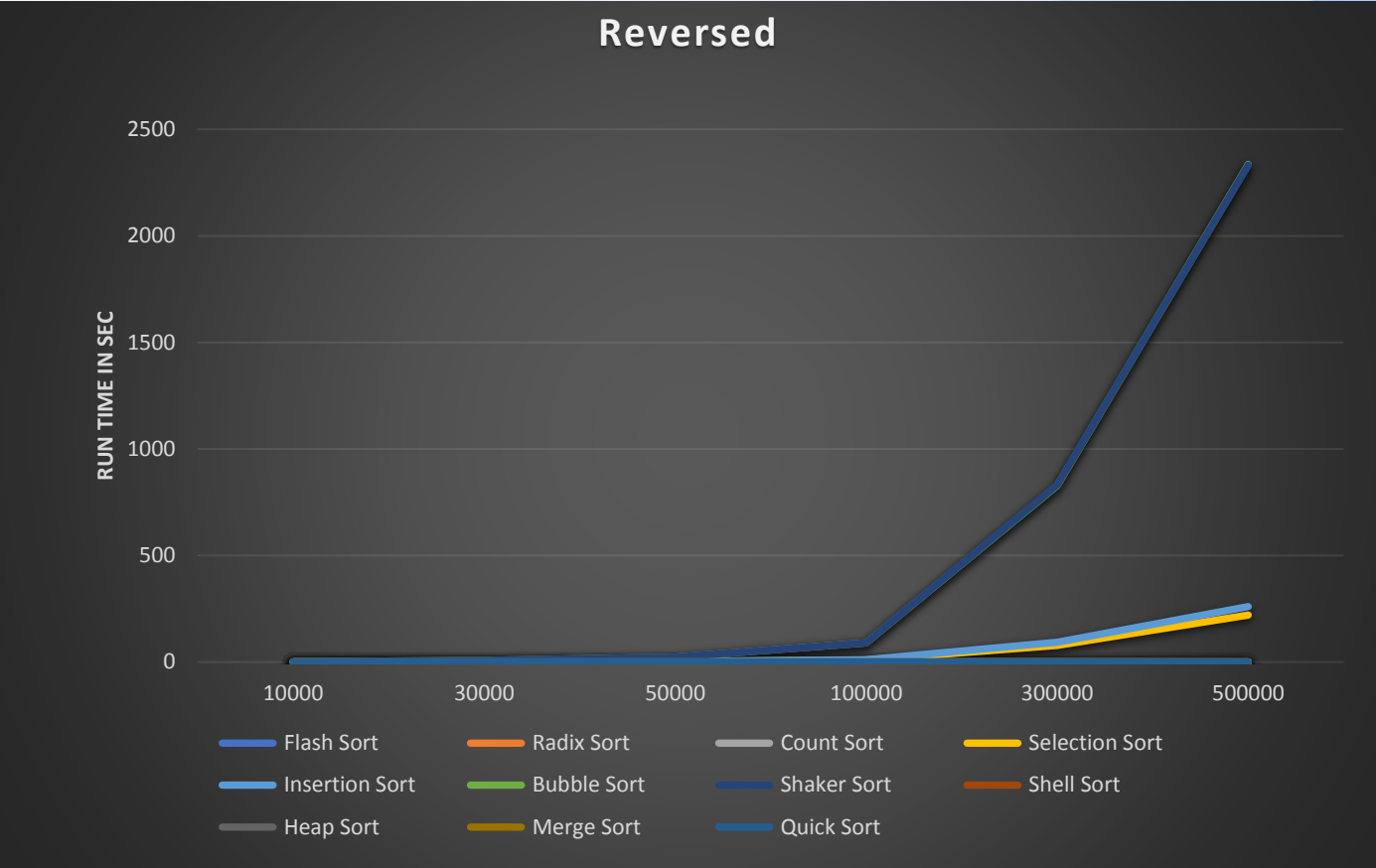


Nhân xét :

- Thuật toán chạy nhanh nhất là Counting Sort. Với độ phức tạp $O(n+k)$, data input là các số nguyên không âm và không sử dụng các phép so sánh để sắp xếp. Counting Sort đã thể hiện ưu thế vượt trội của mình so với các thuật toán sắp xếp có tốc độ nhanh và có sử dụng phép so sánh như Merge Sort, Quick Sort có độ phức tạp $O(n \log n)$.
- Thuật toán chạy chậm nhất là Bubble Sort. Kết quả không có gì bất ngờ. Việc sử dụng quá nhiều phép so sánh các phần tử liên tiếp nhau khiến Bubble Sort là thuật toán chậm nhất trong nhóm các thuật toán có độ phức tạp $O(n^2)$.
- Đối với các thuật toán còn lại, ta thấy rằng thời gian chạy của Quick Sort có ưu thế hơn so với Merge Sort và Heap Sort do đối với randomized data, việc chọn pivot có giá trị gần với trung vị của dãy là không khó khi ta sử dụng phương pháp *Median of Three*, nên Quick Sort rơi vào trường hợp hoạt động gần như tốt nhất.
- Về số cách phép so sánh, nhiều nhất là bao gồm Bubble Sort, Insertion Sort, Selection Sort và Shaker Sort, không có gì quá bất ngờ khi mà các thuật toán này có đặc điểm sử dụng nhiều phép so sánh và tận dụng các vòng lặp.

4. Data Order : Reverse Sorted

Data Order: Reversed												
Data size	10,000		30,000		50,000		100,000		300,000		500,000	
Resulting statics	Running time	Comparison	Running time	Comparison	Running time	Comparison	Running time	Comparison	Running time	Comparison	Running time	Comparison
Flash Sort	0	77,857	2	233,557	3	389,257	7	778,507	19	2,335,507	33	3,892,507
Radix Sort	0	80,164	2	300,199	4	500,199	7	1,000,199	23	3,600,234	41	6,000,234
Count Sort	1	60,004	1	180,004	1	300,004	2	600,004	6	1,800,004	10	3,000,004
Selection Sort	183	100,009,999	1,645	900,029,999	4,586	2,500,049,999	18,335	10,000,099,999	166,204	90,000,299,999	463,879	250,000,499,999
Insertion Sort	0	100,009,999	0	900,029,999	0	2,500,049,999	1	10,000,099,999	2	90,000,299,999	3	250,000,499,999
Bubble Sort	182	100,009,999	1,642	900,029,999	4,557	2,500,049,999	18,258	10,000,099,999	165,246	90,000,299,999	460,401	250,000,499,999
Shaker Sort	0	100,005,001	0	900,015,001	0	2,500,025,001	0	10,000,050,001	1	90,000,150,001	2	250,000,250,001
Shell Sort	1	475,175	2	1,554,051	4	2,844,628	10	6,089,190	34	20,001,852	57	33,857,581
Heap Sort	11	608,480	36	2,071,060	64	3,619,460	137	7,737,170	453	25,631,230	792	44,592,250
Merge Sort	7	746,159	20	2,481,621	34	4,326,137	68	9,153,957	207	29,764,765	351	51,546,155
Quick Sort	1	269,718	4	908,156	6	1,592,192	13	3,386,060	39	11,056,452	70	19,209,746



Nhân xét :

- Về thời gian, thuật toán nhanh nhất là thuật toán Counting Sort, thuật toán chậm nhất là thuật toán BubbleSort, kế đó là ShakerSort.
- Về số lượng so sánh, thuật toán có ít comparision nhất là CountSort, thuật toán thực hiện nhiều comparision nhất là SelectionSort, InsertionSort và BubbleSort, kế đó là ShakerSort.
- Về sự tăng tốc thời gian, các thuật toán BubbleSort và ShakerSort có sự tăng tốc thời gian lớn khi dữ liệu đầu vào lớn hơn 100 000 phần tử, các thuật toán SelectionSort, InsertionSort tăng vừa phải khi dữ liệu lớn hơn 300 000, các thuật toán còn lại tăng ít.
- Về sự tăng tốc số lượng so sánh, các thuật toán SelectionSort, InsertionSort, BubbleSort, ShakerSort có sự tăng tốc lớn, các thuật toán còn lại tăng thoải hơn.
- Lý do về thời gian, Thuật toán BubbleSort có thời gian chậm vì khi dữ liệu đầu vào ở dạng đảo ngược thì thuật toán sẽ trong tình trạng “worst case” lên đến $O(n^2)$ (worst case cũng diễn ra đối với InsertionSort, ShellSort trong trường hợp dữ liệu đảo ngược). BubbleSort thực hiện nhiều so sánh và hoán đổi. Nó chỉ có thể di chuyển một phần tử đúng 1 vị trí. CountSort nhanh vì nó là thuật toán dựa vào “keys” trong 1 phạm vi nhất định, đếm số lượng phần tử có “key” riêng biệt, sau đó tính toán vị trí của mỗi phần tử đầu ra.
- Lý do về số lượng so sánh, CountSort (cũng như RadixSort và FlashSort) là những thuật toán sắp xếp không dựa vào việc so sánh. Trong khi đó các thuật toán SelectionSort, InsertionSort, BubbleSort, ShakerSort phải thực hiện nhiều phép hoán vị phần tử, do đó khi dữ liệu đầu vào bị đảo ngược sẽ sinh ra số lần so sánh rất lớn.

III. Kết luận chung về các kết quả thực nghiệm

- Với kích thước dữ liệu đầu vào nhỏ nhìn chung độ chênh lệch của các thuật toán là không rõ để nhận thấy. Tuy nhiên khi số lượng phần tử lớn thì ta bắt đầu thấy sự chênh lệch rõ ràng của các thuật toán về số phép so sánh và thời gian thực thi trong mỗi kiểu dữ liệu
- Selection Sort cho tốc độ khá chậm trong tất cả kiểu dữ liệu 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 kiểu dữ liệu 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 và Insertion Sort có thể cài đặt khá dễ dàng
- Với dữ liệu đã được sắp xếp, Bubble Sort và Shaker Sort cho tốc độ nhanh nhất do chi phí là $O(n)$.
- Với kiểu dữ liệu gần như đã được sắp xếp thì Shaker Sort cho tốc độ nhanh hơn nhiều so với Bubble Sort, do thu hẹp phạm vi phải duyệt tiếp theo sau khi duyệt
- Với kiểu dữ liệu đảo ngược nó là thảm họa với Bubble Sort, Shaker Sort khi chúng tốn cực nhiều chi phí để có thể sắp xếp lại
- Counting Sort và Radix Sort là những thuật toán cho tốc độ nhanh, tuy nhiên cần đánh đổi bằng việc sử dụng thêm bộ nhớ. Tất nhiên đây là việc mà bạn phải chấp nhận để đổi lại tốc độ cao của thuật toán
- Shell Sort, Heap Sort, Merge Sort và Quick Sort ổn định với cả 4 kiểu dữ liệu đầu vào. Đây được xem như là thế mạnh của chúng bởi lẽ việc những thuật toán ổn định sẽ ít gặp vấn đề về lãng phí tài nguyên cũng như tránh xảy ra lỗi
- Flash Sort đúng như tên gọi của nó là một thuật toán cho tốc độ cực nhanh và tiêu tốn rất ít bộ nhớ, tuy nhiên đây là thuật toán mới và cách thức xây dựng thuật toán khá phức tạp.

- Trong từng trường hợp mỗi thuật toán sẽ có ưu điểm riêng nên việc nắm rõ điểm mạnh và yếu của mỗi thuật toán sẽ giúp những lập trình viên nâng cao hiệu quả trong làm việc.

PHẦN 3 : TỔ CHỨC MÃ NGUỒN

- Header.h định danh các hàm sử dụng trong chương trình
- datarandom.cpp chứa các hàm để tạo dữ liệu được cung cấp
- Source.cpp chứa các hàm sắp xếp và các hàm phụ trợ như sao chép mảng, in mảng, đọc và ghi file ra tệp txt,...
- support.cpp chứa các hàm hỗ trợ xử lý dữ liệu đầu vào và dữ liệu đầu ra sau đó trả ra tham số tương ứng được định dạng sẵn để biết được nó là chế độ nào, thuật toán nào, kích cỡ và kiểu dữ liệu cũng như là yêu cầu in ra thời gian thực thi hay đếm phép so sánh tùy vào người dùng nhập
- menu.cpp là hàm tổng hợp thực thi các command được yêu cầu
- main.cpp là hàm main() của chương trình
- Trong bài có dùng thư viện ctime hàm clock() để đo thời gian thực thi thuật toán, thư viện fstream để hỗ trợ việc đọc ghi, thư viện string để xử lý chuỗi, thư viện iomanip và hàm setw() để thực hiện việc in ra màn hình console giống đề yêu cầu.

CÁC NGUỒN TÀI LIỆU THAM KHẢO

1. <https://www.stdio.vn/giai-thuat-lap-trinh/bubble-sort-va-shaker-sort-01Si3U>
2. <https://codelearn.io/learning/data-structure-and-algorithms/856660>
3. <https://en.wikipedia.org/wiki/Flashsort>
4. <https://iq.opengenus.org/radix-sort/>
5. <https://iq.opengenus.org/counting-sort/>
6. <https://nguyenvanhieu.vn/thuat-toan-sap-xep-quick-sort/>
7. <https://cafedev.vn/thuat-toan-quicksort-gioi-thieu-chi-tiet-va-code-vi-du-tren-nhieu-ngon-ngu-lap-trinh/>
8. <https://www.geeksforgeeks.org/quick-sort/>
9. <https://codelearn.io/sharing/flash-sort-thuat-toan-sap-xep-than-thanh>
10. <https://www.stdio.vn/giai-thuat-lap-trinh/merge-sort-u1Ti3U>
11. <https://nguyenvanhieu.vn/thuat-toan-sap-xep-merge-sort/>
12. <https://www.programiz.com/dsa/counting-sort>
13. <https://tek4.vn/khoa-hoc/cau-truc-du-lieu-va-giai-thuat/thuat-toan-sap-xep-vun-dong-heap-sort>
14. <https://cafedev.vn/thuat-toan-heapsort-gioi-thieu-chi-tiet-va-code-vi-du-tren-nhieu-ngon-ngu-lap-trinh/>
15. <https://www.geeksforgeeks.org/heap-sort/>
16. <https://hoclaptrinh.vn/tutorial/cau-truc-du-lieu-amp-giai-thuat-55-bai/shell-sort-trong-cau-truc-du-lieu-va-giai-thuat>
17. <https://www.stdio.vn/giai-thuat-lap-trinh/distribution-sort-radix-sort-vqu1H1>
18. <https://viblo.asia/p/radix-sort-Do7542eWZM6>
19. <https://nguyenvanhieu.vn/thuat-toan-sap-xep-chen/>
20. <https://freetuts.net/thuat-toan-sap-xep-chon-selection-sort-2931.html>
21. <https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/>
22. <https://www.cprogramming.com/tutorial/lesson14.html>
23. <https://www.stdio.vn/giai-thuat-lap-trinh/bubble-sort-va-shaker-sort-01Si3U>
24. https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_s%E1%BA%AFp_x%E1%BA%BFp_cocktail
25. <https://www.geeksforgeeks.org/shellsort/>

-HẾT-