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


# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

LÊ YẾN NHI – 19127498 – 19CLC3

BÁO CÁO ĐỒ ÁN

# LAB 2 – SORTING

**GIẢNG VIÊN BỘ MÔN**

- PhD. Nguyễn Hải Minh
- M.Sc. Bùi Huy Thông
- M.Sc. Trần Thị Thảo Nhi

Thành phố Hồ Chí Minh – 11/2020

# 1

# TỔNG QUAN

## THÔNG TIN HỌC SINH

| MSSV | Họ Tên | Email |
|---|---|---|
| 19127498 | Lê Yến Nhi | 19127498@student.hcmus.edu.vn |

## THÔNG TIN ĐỒ ÁN

| Tên đồ án | LAB 2 – SORTING | |
|---|---|---|
| **Công cụ và chức năng** | Code Blocks, Xcode | Viết code |
| | Google Documents, Office 365 | Viết báo cáo đồ án |
| **Option đã chọn** | **Set 1 (7 algorithms)** Selection Sort, Insertion Sort, Bubble Sort, Heap Sort, Merge Sort, Quick Sort, và Radix Sort. | |
| **Hoàn thành** | 100% | |

# 2

# TRÌNH BÀY CÁC THUẬT TOÁN

## 1. SELECTION SORT

### 1.1 Ý TƯỞNG THUẬT TOÁN

- Thuật toán **Selection Sort** sắp xếp một mảng tăng dần bằng cách:
    + Đi tìm phần tử có giá trị nhỏ nhất trong mảng chưa được sắp xếp
    + Đổi phần tử có giá trị nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp
- Thuật toán sẽ chia mảng làm hai mảng con: một mảng con đã được sắp xếp và một mảng con chưa được sắp xếp.
- Tại mỗi bước lặp, phần tử nhỏ nhất ở mảng con chưa được sắp xếp sẽ di chuyển về đoạn đã sắp xếp.

### 1.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

**Bước 1:** Đặt ***min*** ở vị trí 0.

**Bước 2:** Tìm kiếm phần tử nhỏ nhất trong mảng a[1… n – 1].

**Bước 3:** Đổi chỗ với giá trị tại vị trí ***min***, tăng vị trí của ***min*** lên 1.

**Bước 4:** Lặp lại cho tới khi toàn bộ mảng đã được sắp xếp.

**Ví dụ:** $\boldsymbol{a}[\,] = \{\mathbf{69}, \mathbf{25}, \mathbf{16}, \mathbf{22}, \mathbf{9}\}$

| 69 | 25 | 16 | 22 | 9 |
|---|---|---|---|---|

**BƯỚC 1.** Tìm phần tử nhỏ nhất trong trong mảng a[0...4] và đổi chỗ nó với phần tử đầu tiên

| 9 | 25 | 16 | 22 | 69 |
|---|---|---|---|---|

**BƯỚC 2**. Tìm phần tử nhỏ nhất trong trong mảng a[1...4] và đổi chỗ nó với phần tử đầu tiên của mảng a[1...4]

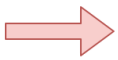
| 9 | 16 | 25 | 22 | 69 |
|---|---|---|---|---|

**BƯỚC 3.** Tìm phần tử nhỏ nhất trong trong mảng a[2...4] và đổi chỗ nó với phần tử đầu tiên của mảng a[2...4]

| 9 | 16 | 22 | 25 | 69 |
|---|---|---|---|---|

**BƯỚC 4.** Tìm phần tử nhỏ nhất trong trong mảng a[3...4] và đổi chỗ nó với phần tử đầu tiên của mảng a[3...4]

| 9 | 16 | 22 | 25 | 69 |
|---|---|---|---|---|



| 9 | 16 | 22 | 25 | 69 |
|---|---|---|---|---|

### 1.3 NHẬN XÉT

- Độ phức tạp thuật toán trong cả 3 trường hợp tốt, xấu, trung bình đều là **O ($n^2$)**.
- Không gian bộ nhớ sử dụng là **O(1)**.
- **Ưu điểm:** Thuật toán ít phải đổi chỗ các phần tử nhất trong số các thuật toán sắp xếp, nó không tạo ra nhiều hơn n lần hoán vị.
- **Nhược điểm:** Thuật toán này có hiệu suất không cao, không thích hợp đối với mảng có số lượng phần tử lớn.

## 2. INSERTION SORT

### 2.1 Ý TƯỞNG THUẬT TOÁN

- Thuật toán **Insertion Sort** hoạt động tương tự như việc xếp các lá bài Tiến Lên.

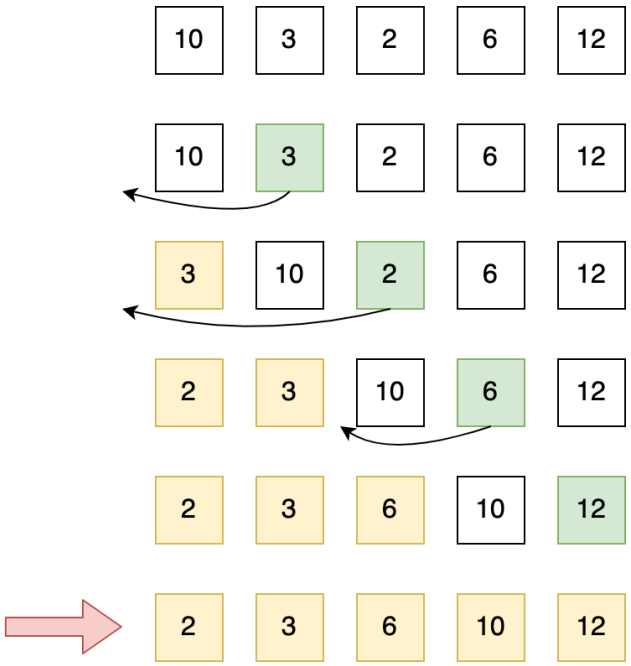
- Mảng được chia thành một phần được sắp xếp và một phần chưa được sắp xếp.

- Ta duyệt lần lượt từng phần tử chưa được sắp xếp và đặt chúng vào vị trí trong phần được sắp xếp sao cho phần được sắp xếp luôn đảm bảo tính chất của một dãy tăng dần.

### 2.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

**Bước 1:** Duyệt lần lượt các phần tử a[1] đến a[n].
**Bước 2:** So sánh phần tử key (phần tử đang duyệt) với các phần tử đứng trước nó.
**Bước 3:** Nếu giá trị của phần tử key nhỏ hơn so với các phần tử trước nó, di chuyển các phần tử lớn hơn key lên một vị trí rồi bỏ key vào vị trí trống.

**Ví dụ:** $a[\,] = \{10, 3, 2, 6, 12\}$




### 2.3 NHẬN XÉT

- Độ phức tạp thuật toán:

  + Trong trường hợp tốt nhất (mảng đã được sắp xếp) Insertsion Sort có thời gian chạy tuyến tính là **O(n)** vì trong mỗi lần lặp, phần tử key không cần phải đổi chỗ.

  + Trong trường hợp xấu nhất (mảng được sắp xếp ngược lại) và trường hợp trung bình là $\mathbf{O(n^2)}$.

- Không gian bộ nhớ sử dụng là **O(1).**
- **Ưu điểm:** Thích hợp đối với mảng đã được sắp xếp một phần hoặc mảng có kích thước nhỏ.
- **Nhược điểm:** Thuật toán này có hiệu suất thấp, không thích hợp, không đủ nhanh đối với mảng có số lượng phần tử lớn .

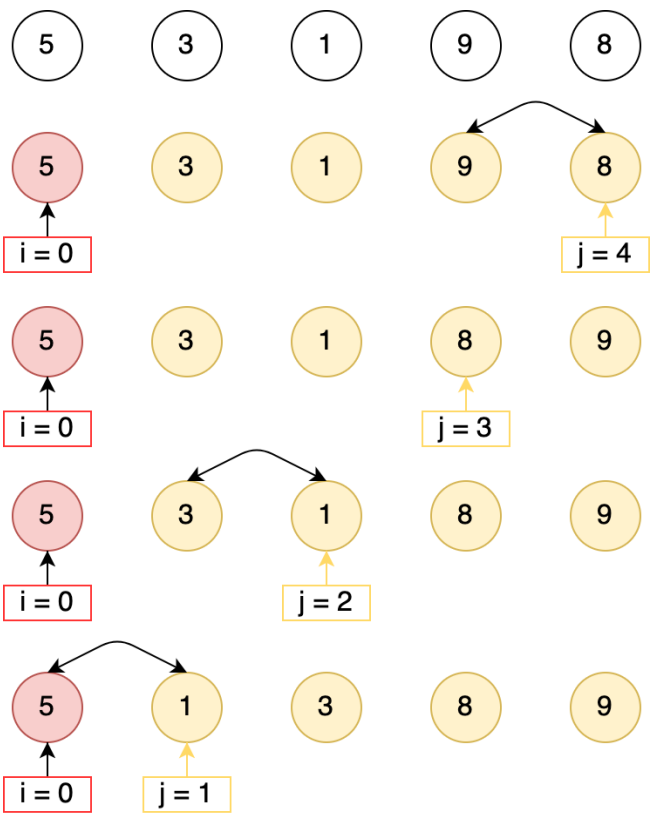
## 3. BUBBLE SORT

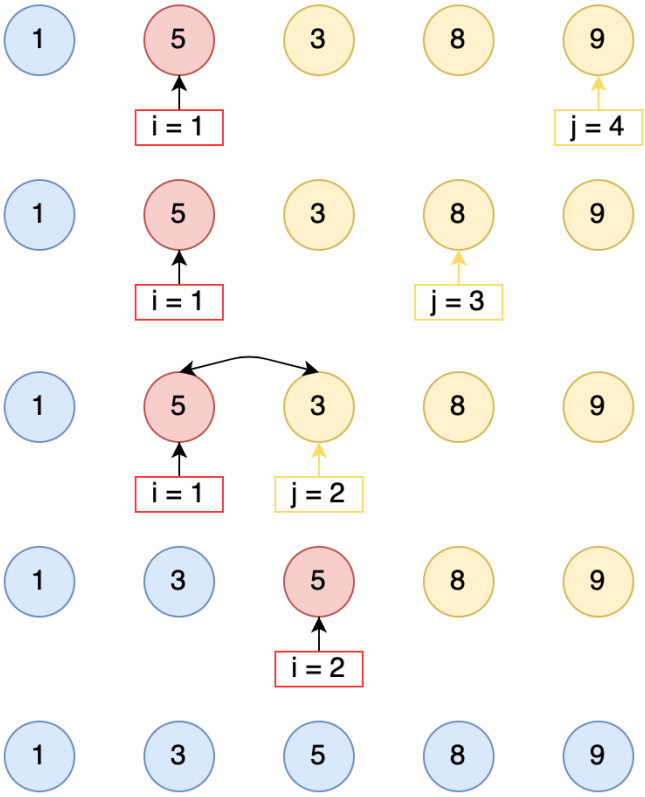
### 3.1 Ý TƯỞNG THUẬT TOÁN

- **Bubble Sort** là một thuật toán sắp xếp đơn giản, giải thuật này sắp xếp một mảng bằng cách so sánh hai phần tử liền kề nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ. Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang).
- Sắp xếp nổi bọt còn có tên khác là sắp xếp bằng so sánh trực tiếp.

## 3.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

**Ví dụ:** $a[\,] = \{5, 3, 1, 9, 8\}$




## 3.3 NHẬN XÉT

- Độ phức tạp thuật toán trong trường hợp:
  + Tốt nhất (mảng đã được sắp xếp) là **O(n)**
  + Xấu nhất (mảng được sắp xếp ngược lại) và trung bình là **O($n^2$)**.
- Không gian bộ nhớ sử dụng là **O(1)**.
- **Ưu điểm:** Code đơn giản, dễ hiểu, không tốn thêm bộ nhớ.
- **Nhược điểm:** Thuật toán này có hiệu suất thấp nhất, không thích hợp đối với mảng có số lượng phần tử lớn.

Bubble Sort là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn Interchange Sort do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn, mặc dù số lần so sánh là bằng nhau.

# 4. HEAP SORT

## 4.1 Ý TƯỞNG THUẬT TOÁN

- **Heap Sort** là một thuật toán sắp xếp dựa trên sự so sánh, có thể được coi là một Selection Sort cải tiến. Điều cải tiến ở đây là việc sử dụng cấu trúc dữ liệu heap thay vì tìm kiếm tuyến tính (Linear-time Search) như Selection Sort để tìm ra phần tử lớn nhất.
- Heap Sort chia các phần tử của mảng thành hai mảng con: một mảng đã sắp xếp và một mảng chưa được sắp xếp. Ở mảng chưa được sắp xếp, các phần tử lớn nhất sẽ được tách ra và đưa vào mảng đã sắp xếp.
- Thuật toán Heapsort được chia thành 2 giai đoạn.

**Giai đoạn 1**

Sắp xếp các dữ liệu trong mảng thành một cây heap (là cây nhị phân bộ phận hoàn chỉnh).

Có 2 cây heap:

+ Max heap: key của mỗi node không nhỏ hơn key của các node con của nó.

+ Min heap: key của mỗi node không lớn hơn key của các node con của nó.

Ở đây ta sử dụng Max heap để sắp xếp mảng tăng dần, node gốc của heap sẽ là phần tử lớn nhất (node gốc ở chỉ số 0)

**Giai đoạn 2** Giai đoạn này gồm các thao tác được lặp đi lặp lại cho đến khi các phần tử của heap đều được đưa vào mảng kết quả.

+ Đưa phần tử lớn nhất của heap được tạo vào cuối mảng kết quả (mảng này sẽ chứa các phần tử đã được sắp xếp)

+ Sắp xếp lại heap sau khi loại bỏ node gốc (có giá trị lớn nhất) để tìm phần tử có giá trị lớn nhất tiếp theo.

## 4.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

Sắp xếp các dữ liệu trong mảng thành một cây heap

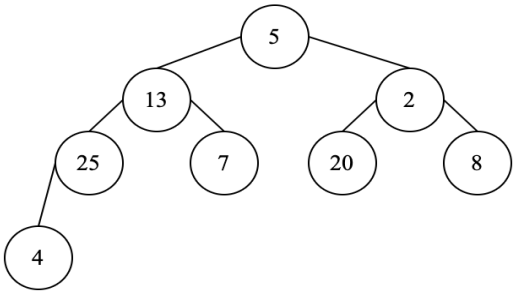
**Bước 1:** Build Heap (sắp xếp sao cho với mọi node cha đều có giá trị lớn hơn node con, trong đó node gốc có giá trị lớn nhất)

**Bước 2:** Swap node gốc với node cuối cùng (node thứ n – 1)

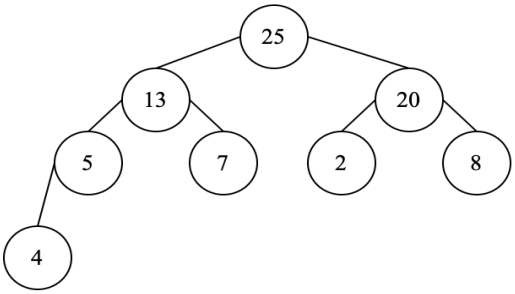
**Bước 3:** Heapify, lệnh này giống với Build Heap nhưng loại bỏ node n – 1 đi. Node thứ n – 1 lúc này có giá trị lớn nhất, ta bỏ nó vào cuối mảng kết quả.

**Bước 4:** Lặp lại các bước 2 và 3 cho đến khi hoàn thành, ta được một mảng sắp xếp tăng dần.

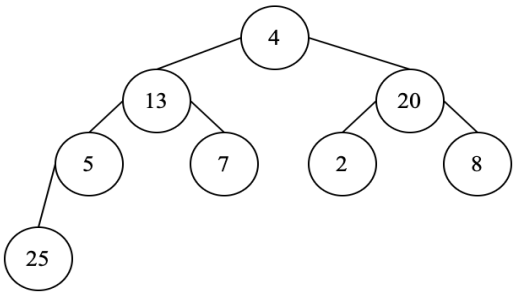
**Ví dụ:** $a[\,] = \{5, 13, 2, 25, 7, 20, 8, 4\}$



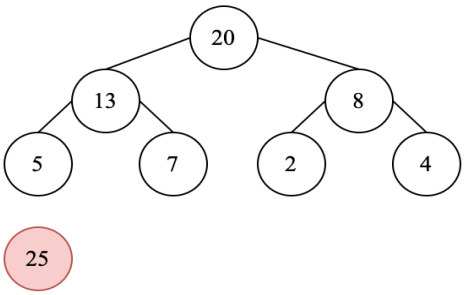

**1. Build Heap**



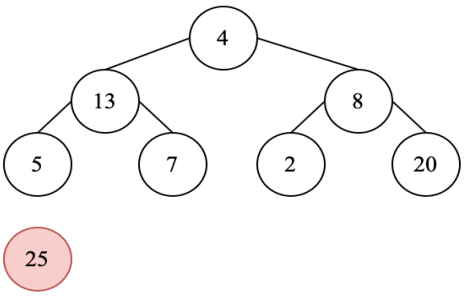

**2. Swap root node with (n - 1)th node**

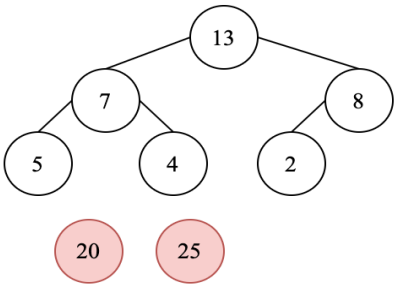


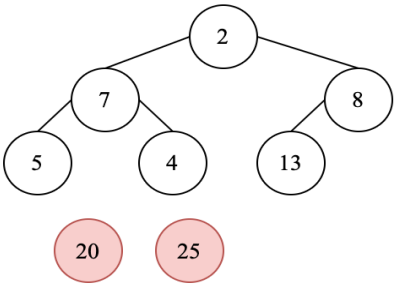

**3. Heapify**

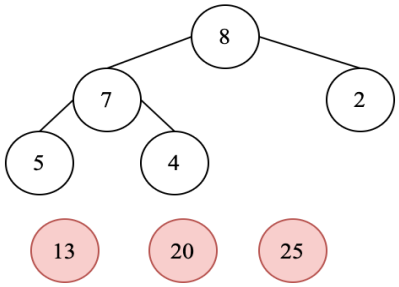


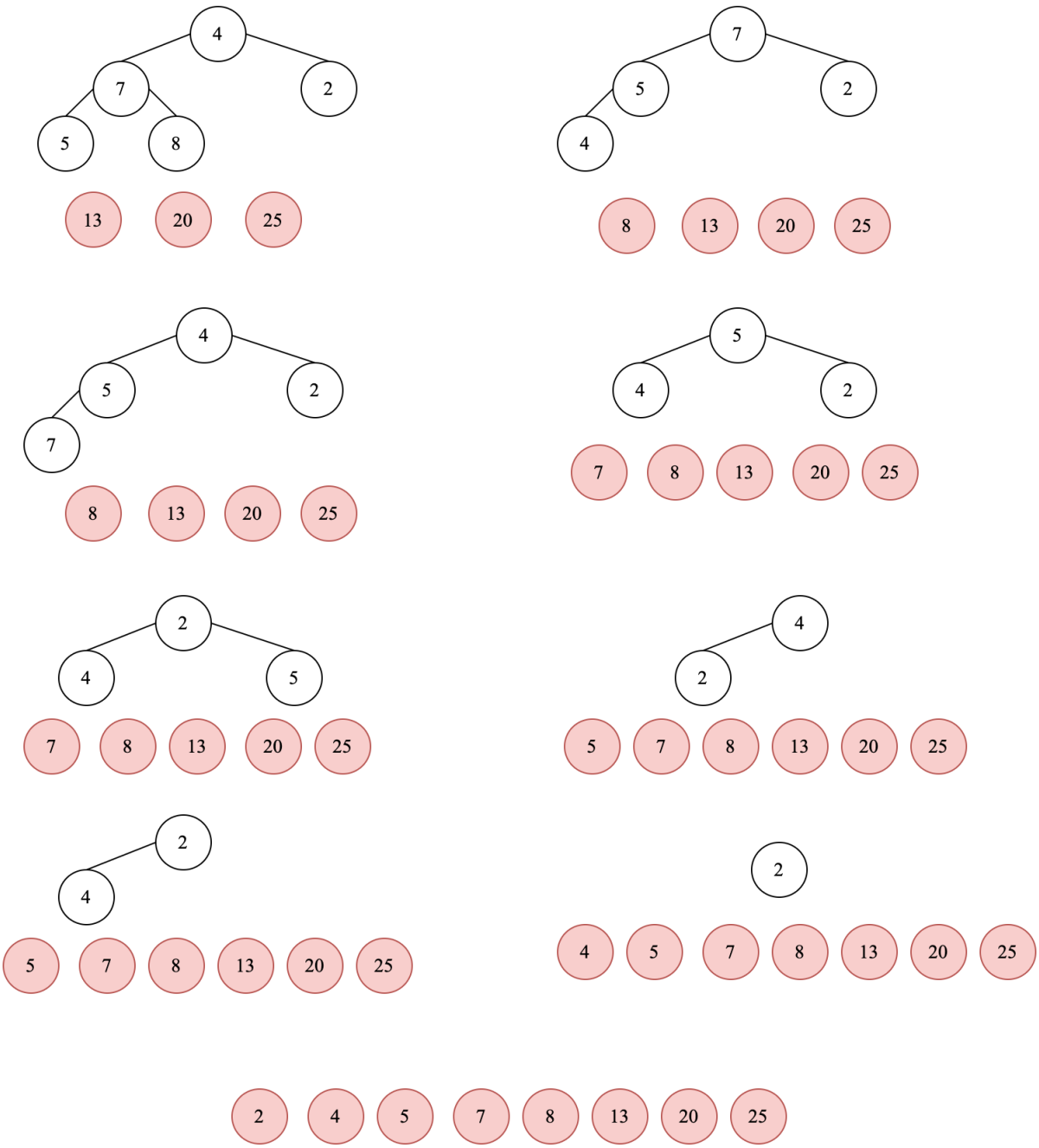

**4. Repeat step 2 and 3 while size of heap is greater than 1**






## 4.3 NHẬN XÉT

- Độ phức tạp thuật toán trong trường hợp:
  + Tốt nhất, xấu nhất và trung bình là $\boldsymbol{O(n \log n)}$
  + Đối với trường hợp tốt nhất, nếu các khóa có giá trị bằng nhau, độ phức tạp: **O(n)**.

- **Ưu điểm:** Heapsort là một thuật toán tại chỗ (in-place) không cần thêm bất cứ cấu trúc dữ liệu phụ trợ trong quá trình chạy thuật toán, chạy nhanh.
- **Nhược điểm:** Thuật toán này không có sự ổn định, code phức tạp.

# 5. MERGE SORT

## 5.1 Ý TƯỞNG THUẬT TOÁN

**- Merge Sort** (sắp xếp trộn) là một trong những thuật toán sắp xếp các mảng, các danh sách (hoặc bất kỳ cấu trúc dữ liệu nào có thể truy cập tuần tự) theo một trật tự nào đó. Merge Sort là một thuật toán chia để trị, nó được xếp vào thể loại sắp xếp so sánh.

- Merge Sort sắp xếp một mảng tăng dần bằng cách chia mảng thành hai nửa cho tới khi không thể chia được nữa. Nếu một mảng mà chỉ có một phần tử thì mảng này coi như đã được sắp xếp. Sau đó, kết hợp với các mảng đã được sắp xếp tạo thành một mảng mới (mảng này cũng đã được sắp xếp).

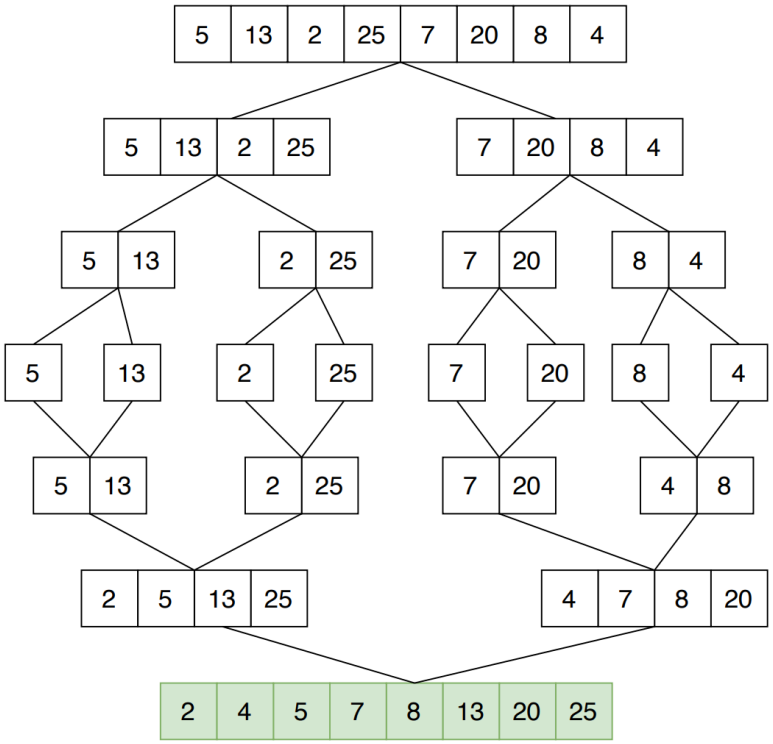
## 5.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

**SẮP XẾP TRỘN ĐỆ QUY**

**Bước 1:** Tách mảng lớn thành những mảng con nhỏ hơn bằng cách chia đôi mảng lớn và tiếp tục chia đôi các mảng con cho đến khi mảng con nhỏ nhất chỉ còn 1 phần tử.

**Bước 2:** So sánh 2 mảng con có cùng mảng cơ sở (khi chia đôi mảng lớn thành 2 mảng con thì mảng lớn đó gọi là mảng cơ sở của 2 mảng con đó).

**Bước 3:** Khi so sánh chúng vừa sắp xếp vừa ghép 2 mảng con đó lại thành mảng cơ sở, tiếp tục so sánh và ghép các mảng con lại đến khi còn lại mảng duy nhất, đó là mảng đã được sắp xếp.




**Ví dụ:**

$\boldsymbol{a}[\,] = \{\mathbf{5}, \mathbf{13}, \mathbf{2}, \mathbf{25}, \mathbf{7}, \mathbf{20}, \mathbf{8}, \mathbf{4}\}$

## 5.3 NHẬN XÉT

- Độ phức tạp thuật toán trong cả 3 trường hợp tốt, xấu, trung bình đều là $\boldsymbol{O}(\boldsymbol{n}\log\boldsymbol{n})$
- Không gian bộ nhớ sử dụng là **O(n).**

- **Ưu điểm:** Thuật toán này có tính ổn định, chạy nhanh. Đối với các chương trình cần tối ưu, Merge Sort là một lựa chọn tốt.
- **Nhược điểm:** Code phức tạp, nó cần dùng thêm nhiều bộ nhớ để lưu giá trị mảng.

# 6. QUICK SORT

## 6.1 Ý TƯỞNG THUẬT TOÁN

- **Quick Sort** là một thuật toán chia để trị, đạt hiệu quả cao và dựa trên việc chia mảng dữ liệu thành các mảng nhỏ hơn.

- Quick Sort chia mảng thành hai phần bằng cách so sánh từng phần tử của mảng với một phần tử được chọn làm **phần tử chốt (Pivot)**: những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong mảng con thứ nhất, các phần tử lớn hơn chốt được đưa về phía sau và nằm trong mảng con thứ hai. Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

Việc lựa chọn pivot ảnh hưởng rất nhiều tới tốc độ sắp xếp. Dưới đây là một số cách để chọn pivot thường được sử dụng:

1. Luôn chọn phần tử đầu tiên của mảng.
2. Luôn chọn phần tử cuối cùng của mảng.
3. Chọn một phần tử random.
4. Chọn một phần tử có giá trị nằm giữa mảng.

## 6.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

***Các bước chọn pivot và chia mảng:***

**Bước 1:** Chọn pivot là phần tử có vị trí ở giữa mảng.

**Bước 2:** Khai báo hai biến để trỏ tới bên trái và bên phải của danh sách, ngoại trừ pivot

**Bước 3:** Biến bên trái trỏ tới mảng con bên trái**,** biến bên phải trỏ tới mảng con bên phải

**Bước 4:** Khi giá trị tại biến bên trái là nhỏ hơn pivot thì di chuyển sang phải

**Bước 5:** Khi giá trị tại biến bên phải là lớn hơn pivot thì di chuyển sang trái

**Bước 6:** Nếu left $\leq$ right ta hóan đổi giá trị biến trái và phải, sau đó tăng biến trái lên 1, giảm biến phải xuống 1.

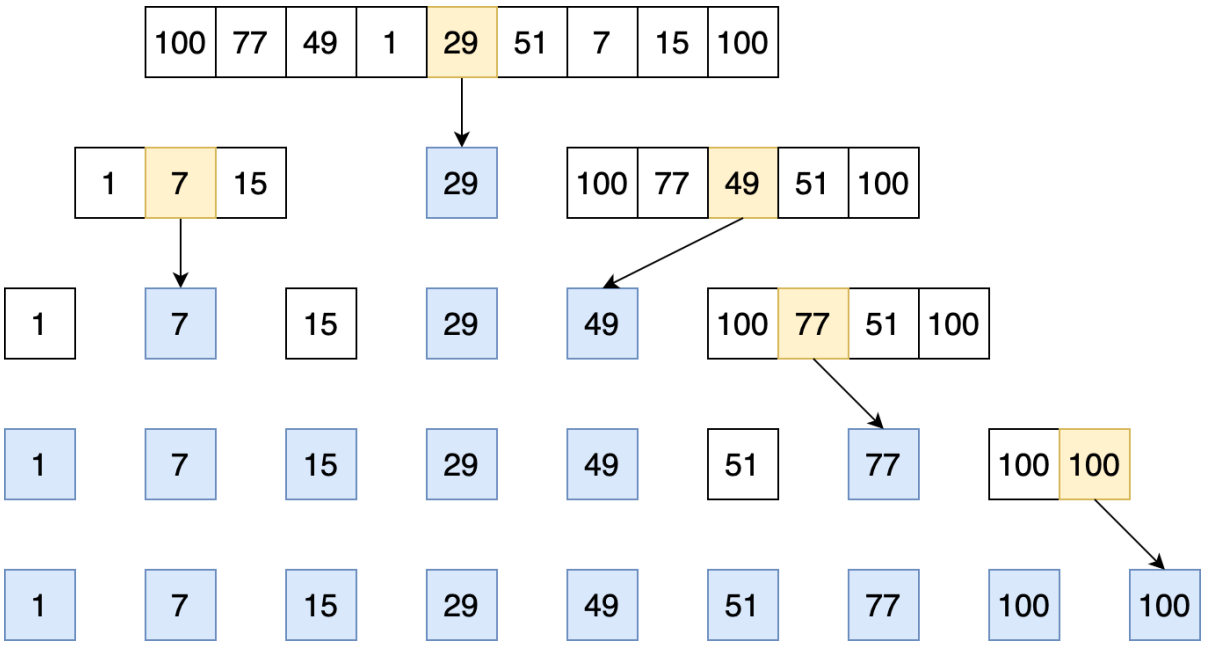
***Các bước thực hiện thuật toán:***

**Bước 1:** Lấy pivot là phần tử ở giữa danh sách.

**Bước 2:** Chia mảng theo pivot.

**Bước 3:** Sử dụng đệ qui với mảng con bên trái và đệ qui với mảng con bên phải.

**Ví dụ:** $a[\,] = \{100, 77, 49, 1, 29, 51, 7, 15, 100\}$




### 6.3 NHẬN XÉT

- Độ phức tạp thuật toán trong trường hợp:
  + Tốt nhất (mảng đã được sắp xếp) và trung bình là $\boldsymbol{O(n \log n)}$
  + Xấu nhất (mảng được sắp xếp ngược lại) $\mathbf{O(n^2)}$.
- Không gian bộ nhớ sử dụng là $\boldsymbol{O(\log n)}$
- **Ưu điểm**: Thuật toán này chạy nhanh (nhanh nhất trong các thuật toán sắp xếp dựa trên việc só sánh các phần tử) và tỏ ra khá hiệu quả với các tập dữ liệu lớn. Không lo lắng dữ liệu đầu vào kể cả trường hợp xấu nhất
- **Nhược điểm:** Tùy thuộc vào cách chọn pivot mà độ phức tạp khác nhau, thuật toán này không ổn định, code phức tạp.

## 7. RADIX SORT

### 7.1 Ý TƯỞNG THUẬT TOÁN

- Cơ sở để sắp xếp của **Radix Sort** dựa theo nguyên tắc phân loại thư của bưu điện (Postman's Sort), phân loại thư theo tỉnh thành, quận huyện, phường xã,...
- Radix Sort được thực hiện dựa trên ý tưởng nếu một dãy số đã được sắp xếp hoàn chỉnh thì từng chữ số cũng sẽ được sắp xếp hoàn chỉnh dựa trên giá trị của các chữ số đó. Thuật

toán này yêu cầu dãy cần được sắp xếp có thể so sánh thứ tự các vị trí vì thế sắp xếp theo cơ số không giới hạn ở tập số nguyên.

## 7.2 TỪNG BƯỚC VẬN HÀNH (STEP BY STEP)

Cụ thể, để sắp xếp dãy $a_1, a_2, \ldots, a_n$ giải thuật Radix Sort thực hiện như sau:

**Bước 1:** Trước tiên, giả sử mỗi phần tử $a_i$ trong dãy $a_1, a_2, \ldots, a_n$ là một số nguyên có tối đa m chữ số.

**Bước 2:** Ta phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm,…

**Ví dụ:** $\boldsymbol{a}[\,] = \{\mathbf{186}, \mathbf{90}, \mathbf{58}, \mathbf{112}, \mathbf{309}, \mathbf{108}, \mathbf{3}, \mathbf{89}\}$

| Hàng đơn vị | 186 | 90 | 58 | 112 | 309 | 108 | 3 | 89 |
|---|---|---|---|---|---|---|---|---|
| Hàng chục | 90 | 112 | 3 | 186 | 58 | 108 | 309 | 89 |
| Hàng trăm | 3 | 108 | 309 | 112 | 58 | 186 | 89 | 90 |
| **KẾT QUẢ** | **3** | **58** | **89** | **90** | **108** | **112** | **186** | **309** |

## 7.3 NHẬN XÉT

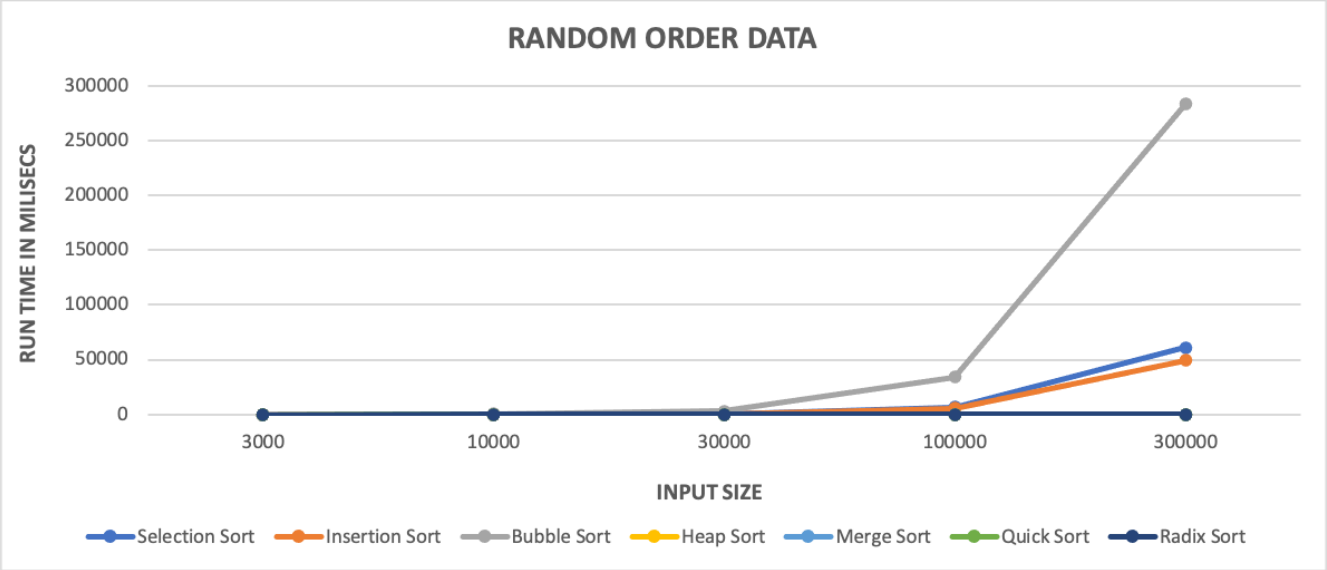
- Độ phức tạp thuật toán là $\boldsymbol{O(d\,n)}$
- Độ phức tạp không gian là $\boldsymbol{O(d+n)}$
- **Ưu điểm:** Có thể chạy nhanh hơn các thuật toán sắp xếp sử dụng so sánh.
- **Nhược điểm:** Không thể sắp xếp số thực, chỉ sắp xếp được cho số nguyên, phải biết miền giá trị của số nguyên.

# 3

# TRÌNH BÀY CÁC BIỂU ĐỒ

# 1. RANDOM ORDER DATA

## 1.1 BIỂU ĐỒ




| RANDOM ORDER DATA | | | | | |
|---|---|---|---|---|---|
| | 3000 | 10000 | 30000 | 100000 | 300000 |
| Selection Sort | 12,808 | 79,66 | 639,131 | 6348,4 | 60902 |
| Insertion Sort | 5,019 | 64,31 | 473,396 | 5608,83 | 49446 |
| Bubble Sort | 19,435 | 258,075 | 2820,79 | 33854,5 | 283495 |
| Heap Sort | 0,55 | 2,214 | 8,22 | 26,162 | 92,713 |
| Merge Sort | 0,38 | 1,479 | 5,059 | 17,239 | 55,873 |
| Quick Sort | 0,444 | 1,241 | 3,935 | 21,452 | 45,108 |
| Radix Sort | 0,249 | 0,657 | 2,608 | 12,834 | 25,572 |

## 1.2 NHẬN XÉT BIỂU ĐỒ

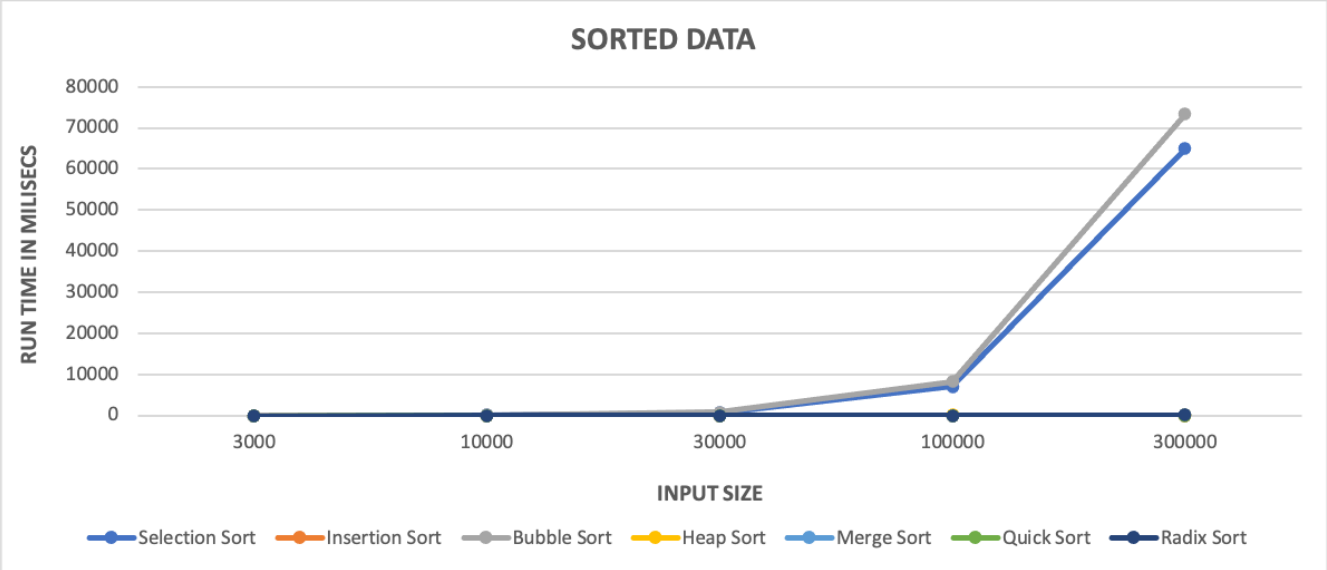
Với kiểu dữ liệu ***ngẫu nhiên (Random data)***

+ Thuật toán **Bubble Sort** chạy **chậm nhất**
+ **Nhanh nhất** là **Radix Sort** rồi đến **Quick Sort**
+ Các thuật toán còn lại chạy bình thường.

Vì trong trường hợp xấu nhất, độ phức tạp thuật toán của Radix Sort là $O(d\,n)$ bé hơn bất kì thuật toán nào trong 7 thuật toán. Riêng Bubble Sort, chạy chậm nhất vì độ phức tạp thuật toán là $O(n^2)$ và nó còn là giải thuật chậm nhất, hiệu xuất thấp nhất trong các thuật toán.

# 2. SORTED DATA

## 2.1 BIỂU ĐỒ




| SORTED DATA | | | | | |
|---|---|---|---|---|---|
| | 3000 | 10000 | 30000 | 100000 | 300000 |
| Selection Sort | 6,487 | 69,951 | 632,282 | 6967,51 | 64913,1 |
| Insertion Sort | 0,011 | 0,034 | 0,11 | 0,392 | 1,622 |
| Bubble Sort | 7,096 | 75,467 | 696,432 | 8232,3 | 73368,1 |
| Heap Sort | 0,431 | 2,025 | 5,497 | 20,448 | 68,006 |
| Merge Sort | 0,224 | 0,815 | 2,448 | 12,072 | 26,076 |
| Quick Sort | 0,147 | 0,234 | 0,825 | 5,802 | 9,632 |
| Radix Sort | 0,202 | 0,584 | 2,652 | 12,121 | 24,157 |

## 2.2 NHẬN XÉT BIỂU ĐỒ

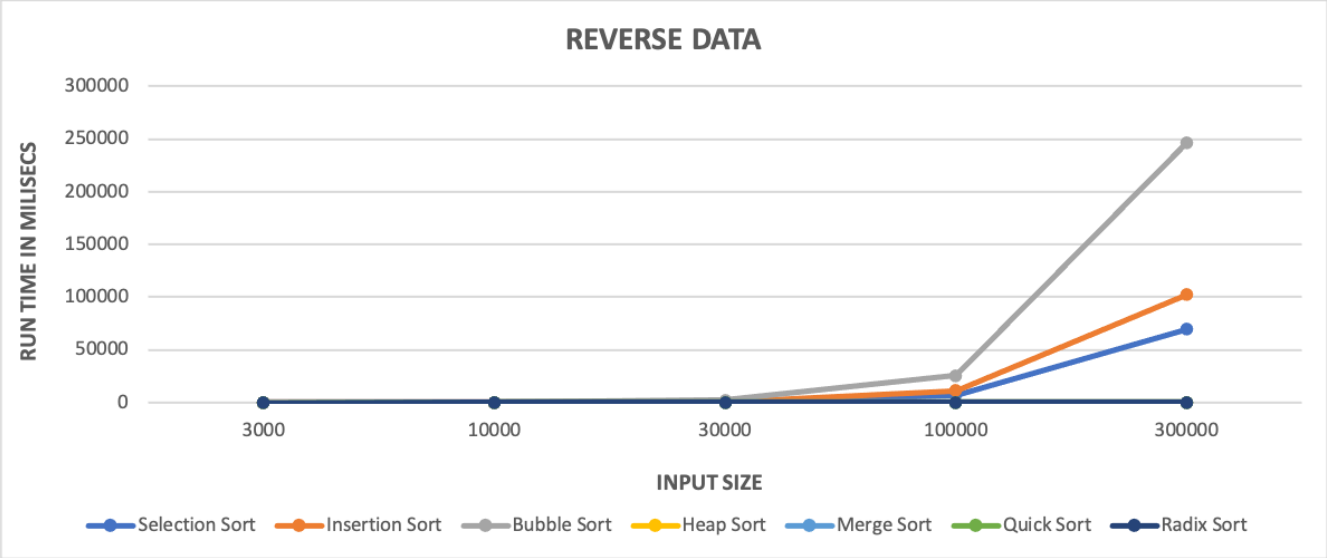
Với kiểu dữ liệu ***đã được sắp xếp tăng dần (Sorted data)***

+ Thuật toán **Bubble Sort** chạy **chậm nhất**
+ **Nhanh nhất** là **Insertion Sort** (nhanh hơn cả **Quick Sort**)
+ Các thuật toán còn lại chạy bình thường, **Merge Sort** và **Radix Sort** thời gian chạy xấp xỉ nhau.

Vì kiểu dữ liệu được sắp xếp sẵn nên đây là trường hợp tốt nhất, độ phức tạp thuật toán của Insertion Sort là $\boldsymbol{O(n)}$ bé hơn độ phức tạp thuật toán của Quick Sort là $\boldsymbol{O(n \log n)}$.

# 3. REVERSE DATA

## 3.1 BIỂU ĐỒ




| REVERSE DATA | | | | | |
|---|---|---|---|---|---|
| | 3000 | 10000 | 30000 | 100000 | 300000 |
| Selection Sort | 10,935 | 75,334 | 624,625 | 7127,88 | 69646,6 |
| Insertion Sort | 10,156 | 109,895 | 989,524 | 11331,3 | 102295 |
| Bubble Sort | 24,334 | 260,018 | 2404,44 | 25791,9 | 246038 |
| Heap Sort | 0,931 | 1,51 | 5,759 | 20,33 | 65,964 |
| Merge Sort | 0,242 | 1,456 | 2,551 | 9,135 | 29,247 |
| Quick Sort | 0,092 | 0,319 | 1,218 | 6,266 | 15,587 |
| Radix Sort | 0,283 | 0,627 | 2,205 | 12,698 | 24,864 |

## 3.2 NHẬN XÉT BIỂU ĐỒ

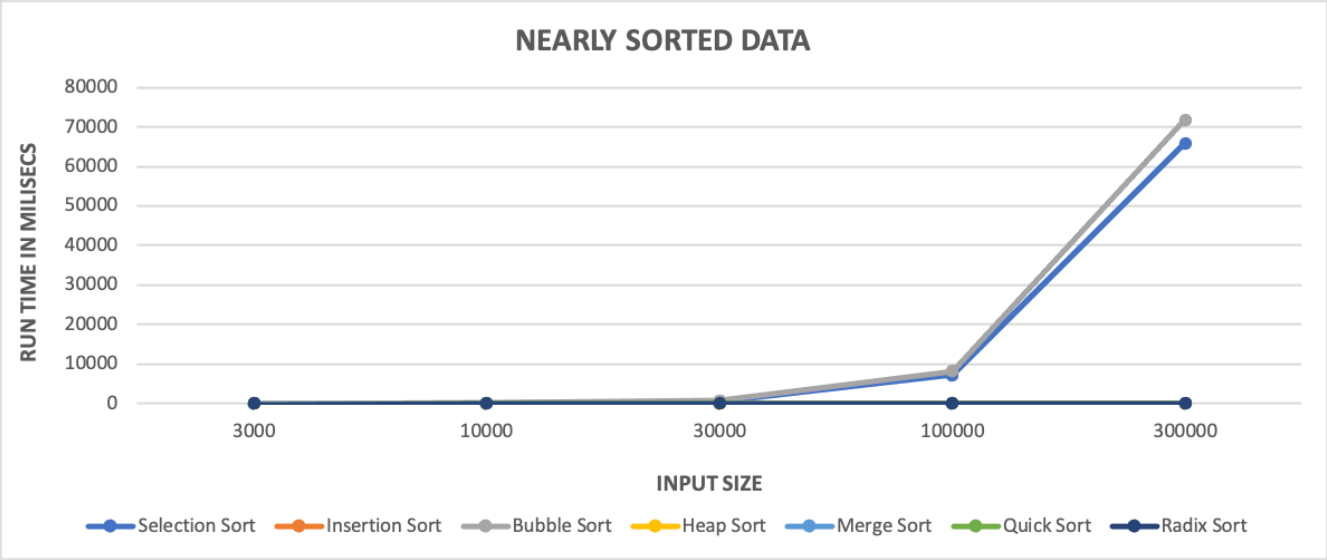
Với kiểu dữ liệu ***được sắp xếp ngược (Reverse data)***

+ Thuật toán **Bubble Sort** chạy **chậm nhất** rồi đến **Insertion Sort** và **Selection Sort**
+ **Nhanh nhất** là **Quick Sort**
+ Các thuật toán còn lại chạy bình thường, **Merge Sort** và **Radix Sort** thời gian chạy xấp xỉ nhau.

Đúng theo lí thuyết, Quick Sort là thuật toán sắp xếp nhanh nhất trong phần lớn các trường hợp.

# 4. NEARLY SORTED DATA

## 4.1 BIỂU ĐỒ




| NEARLY SORTED DATA | | | | | |
|---|---|---|---|---|---|
| | 3000 | 10000 | 30000 | 100000 | 300000 |
| Selection Sort | 8,026 | 70,365 | 586,35 | 7165,41 | 65861 |
| Insertion Sort | 0,056 | 0,224 | 0,496 | 3,364 | 10,017 |
| Bubble Sort | 7,273 | 78,293 | 672,296 | 8151,38 | 71859,7 |
| Heap Sort | 0,531 | 1,685 | 5,775 | 19,952 | 63,938 |
| Merge Sort | 0,294 | 0,817 | 2,715 | 8,889 | 27,969 |
| Quick Sort | 0,1 | 0,28 | 1,745 | 5,185 | 13,103 |
| Radix Sort | 0,276 | 0,687 | 2,381 | 12,665 | 34,483 |

## 4.2 NHẬN XÉT BIỂU ĐỒ

Với kiểu dữ liệu ***gần như đã được sắp xếp tăng dần (Nearly Sorted data)***

+ Thuật toán **Bubble Sort** chạy **chậm nhất** rồi đến **Selection Sort**
+ **Nhanh nhất** là **Insertion Sort** (nhanh hơn cả **Quick Sort**)
+ Các thuật toán còn lại chạy bình thường.

Insertion Sort tỏ ra hiệu quả hơn, chạy nhanh hơn Quick Sort khi mảng đầu vào gần như được sắp xếp, vì chỉ có một số phần tử bị đặt sai vị trí trong một mảng lớn hoàn chỉnh. Đây là trường hợp gần như tốt nhất, độ phức tạp thuật toán của Insertion Sort là **O(n)** bé hơn độ phức tạp thuật toán của Quick Sort là $\boldsymbol{O(n \log n)}$.

# NHẬN XÉT TỔNG THỂ CÁC THUẬT TOÁN TRONG BỐN KIỂU DỮ LIỆU

- **Seclection Sort** chạy ổn định trong cả bốn kiểu dữ liệu, thời gian chạy giữa các dữ liệu trên từng kích thước mảng không chênh lệch nhiều.

- **Insertion Sort** chạy nhanh nhất ở hai kiểu dữ liệu: đã được sắp xếp và gần như được sắp xếp, chạy chậm nhất ở kiểu dữ liệu được sắp xếp ngược.

- **Bubble Sort** chạy nhanh hơn đối với hai kiểu dữ liệu: đã được sắp xếp và gần như được sắp xếp, với hai kiểu dữ liệu còn lại chạy rất chậm. Nhìn chung, với kiểu dữ liệu nào thì Bubble Sort vẫn chạy chậm nhất so với các giải thuật còn lại (đúng như lí thuyết).

- **Heap Sort** nhìn chung chạy khá nhanh, kết quả thời gian chạy giữa các kiểu dữ liệu xấp xỉ nhau.

- **Merge Sort** chạy nhanh, theo kết quả thí nghiệm thì nhanh hơn Heap Sort và kết quả thời gian chạy giữa các kiểu dữ liệu xấp xỉ nhau (vì nó không quan tâm thứ tự sắp xếp các phần tử trong mảng).

- **Quick Sort** theo lí thuyết là thuật toán chạy nhanh nhất, nhưng thực tế đối với hai kiểu dữ liệu: đã được sắp xếp và gần như được sắp xếp Quick Sort chạy chậm hơn Insertion Sort. Đối với kiểu dữ liệu được sắp xếp ngược (trường hợp xấu nhất) thì Quick Sort là giải thuật chạy nhanh nhất.

- **Radix Sort** chạy nhanh trong bốn kiểu dữ liệu, ở kiểu dữ liệu ngẫu nhiên Radix Sort chạy nhanh nhất trong các giải thuật (nhanh hơn Quick Sort) và thời gian chạy giữa các dữ liệu trên từng kích thước mảng gần như giống nhau.

# KẾT LUẬN:

Không có thuật toán nào là hoàn hảo. Thuật toán đơn giản, dễ hiểu chỉ giải quyết được những bài toán nhỏ, đơn giản. Thuật toán phức tạp thì khó hiểu, không nhanh bằng các thuật toán đơn giản trong trường hợp dữ liệu có ít phần tử (5 - 20 phần tử)…

Khi lựa chọn thuật toán để giải quyết vấn đề, ta cần nhìn nhận vấn đề gặp phải ở nhiều góc độ khác nhau để lựa chọn giải pháp hợp lí, hiệu quả. Góc độ ở đây là dữ liệu đầu vào (kích thước mảng, kiểu dữ liệu…), dung lượng bộ nhớ hay cấu hình máy…

# 4

# TÀI LIỆU THAM KHẢO

https://en.wikipedia.org/wiki/Selection_sort
https://www.geeksforgeeks.org/selection-sort/?ref=leftbar-rightbar

https://en.wikipedia.org/wiki/Insertion_sort
https://www.geeksforgeeks.org/insertion-sort/

https://en.wikipedia.org/wiki/Bubble_sort
https://www.geeksforgeeks.org/bubble-sort/?ref=lbp

https://www.stdio.vn/giai-thuat-lap-trinh/heap-sort-thuat-toan-sap-xep-vun-dong-9kQFE1
https://www.geeksforgeeks.org/heap-sort/?ref=lbp

https://en.wikipedia.org/wiki/Merge_sort
https://www.stdio.vn/giai-thuat-lap-trinh/merge-sort-u1Ti3U
https://www.geeksforgeeks.org/merge-sort/?ref=leftbar-rightbar

https://en.wikipedia.org/wiki/Quicksort
https://nguyenvanhieu.vn/thuat-toan-sap-xep-quick-sort/
Slide bài học của giảng viên

https://en.wikipedia.org/wiki/Radix_sort
https://www.geeksforgeeks.org/radix-sort/