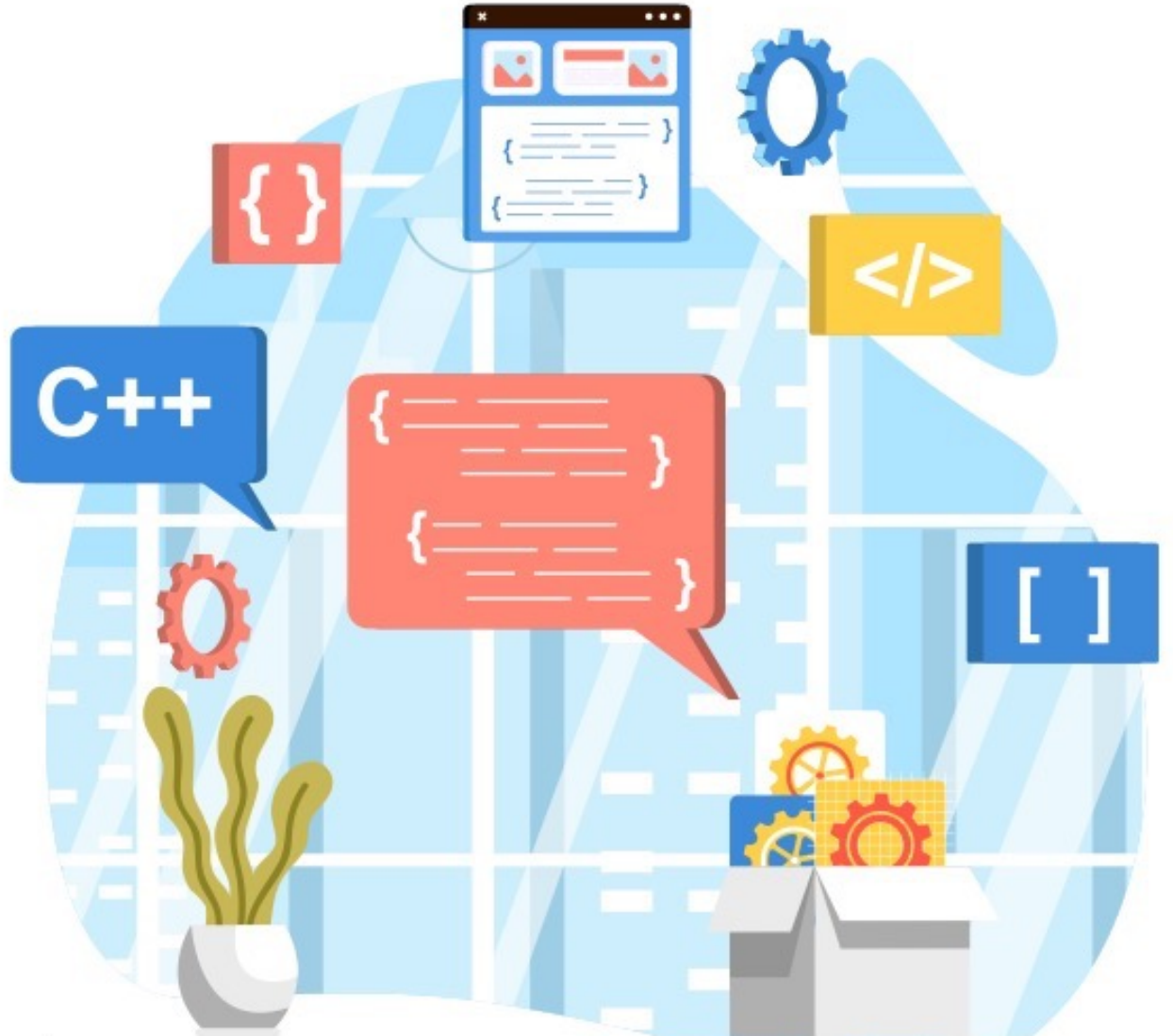
CÁC GIẢI THUẬT SẮP XẾP CƠ BẢN



VŨ NGỌC THANH SANG

KHOA CÔNG NGHỆ THÔNG TIN

ĐẠI HỌC SÀI GÒN

I - INTRODUCTION

- Sinh viên CNTT điển hình nghiên cứu các thuật toán sắp xếp cơ bản ít nhất ba lần trước khi tốt nghiệp:
  1. Cơ sở lập trình
  2. Cấu trúc dữ liệu và giải thuật
  3. Luận văn tốt nghiệp
- Sắp xếp là một thuật toán quan trọng trong khoa học máy tính. Sắp xếp phù hợp có thể làm giảm đáng kể độ phức tạp của một vấn đề và thường được sử dụng cho các thuật toán và tìm kiếm cơ sở dữ liệu.

I - INTRODUCTION

Các bài toán sẽ trở nên dễ dàng nếu dữ liệu được sắp xếp

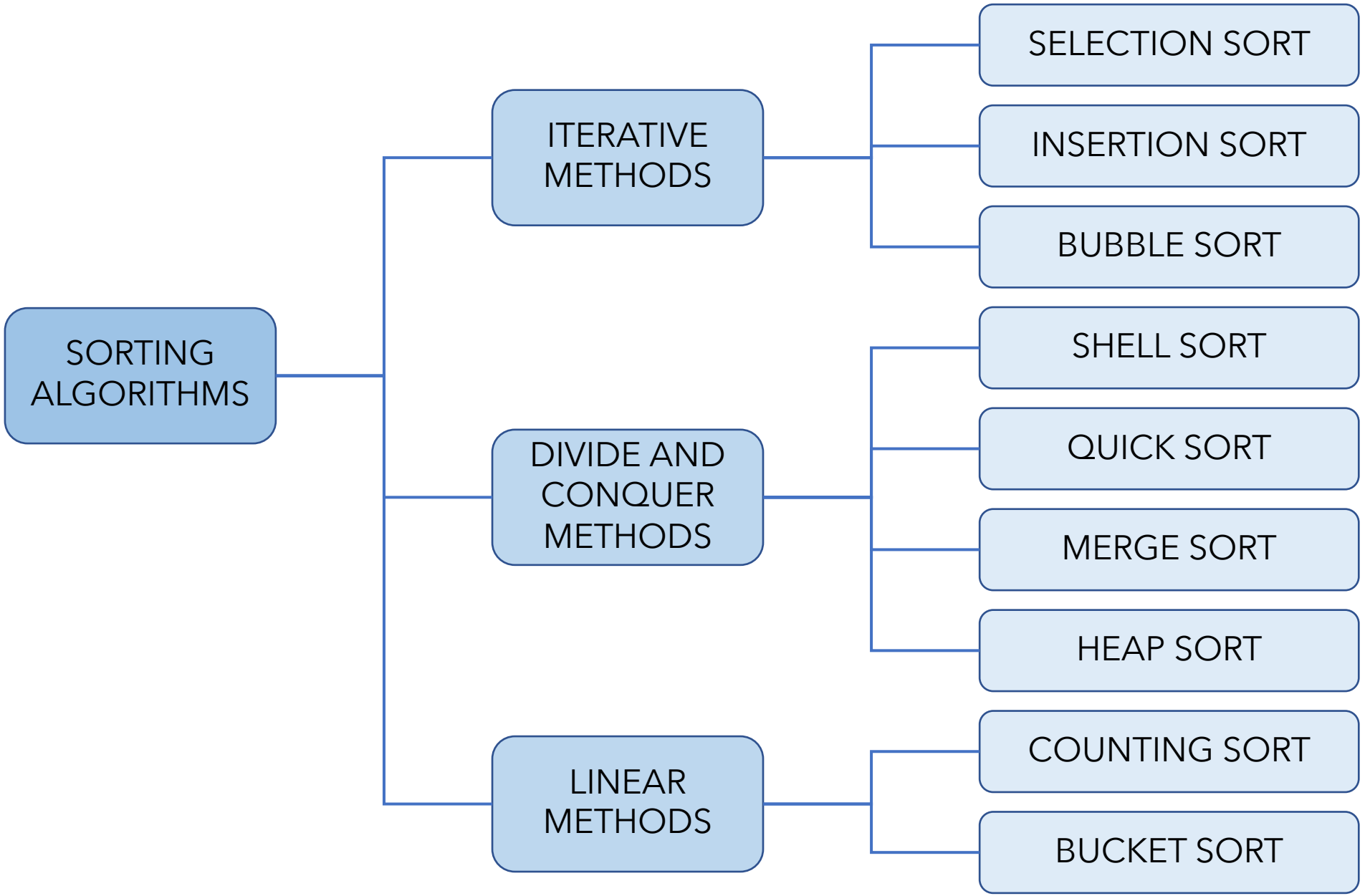
- **Tìm kiếm**: Tìm một giá trị trong một tập dữ liệu. Đây là ứng dụng quan trọng nhất của sắp xếp.
- **Cặp số gần nhất**: Cho một tập hợp n số, làm sao để tìm cặp số có hiệu nhỏ nhất? Sau khi các số được sắp xếp, cặp số gần nhất sẽ nằm cạnh nhau.

I - INTRODUCTION

Các bài toán sẽ trở nên dễ dàng nếu dữ liệu được sắp xếp

- **Tính duy nhất của phần tử**: Có bất kì bản sao nào trong một tập hợp n phần tử cho trước hay không?
- **Số yếu vị**: Trong một tập hợp n phần tử, phần tử nào có số lần xuất hiện lớn nhất trong tập hợp?
- **Lựa chọn**: Làm sao để lựa chọn k vật phẩm sao cho tổng giá trị của các vật phẩm là lớn nhất?

I - INTRODUCTION



I - INTRODUCTION

Các vấn đề cần xác định trong bài toán sắp xếp

- **Kích thước của dữ liệu đầu vào**
- **Sắp xếp theo chiều tăng dần hay giảm dần**: tùy thuộc vào yêu cầu của bài toán
- **Sắp xết theo khóa (key) hay toàn bộ bản ghi (record)**: sự đồng bộ của dữ liệu có được bảo toàn?
- **Xử lý các trường hợp bằng nhau**: ý nghĩa của các trường hợp bằng nhau, sự cần thiết của tiêu chí sắp xếp thứ cấp.
- **Xử lý dữ liệu chuỗi ký tự**: viết hoa, viết thường, ký tự đặc biệt.

I - INTRODUCTION

- Các cách sắp xếp thứ tự được sử dụng thường xuyên nhất là sắp xếp theo số (numerical order) và bảng chữ cái (alphabetical order).
- Sắp xếp hiệu quả rất quan trọng trong việc tối ưu hóa hiệu quả của các thuật toán khác.
- Một mảng được sắp xếp theo **thứ tự tăng dần** nếu

$$A[i] \leq A[i+1], 0 \leq i \leq n-2$$

- Một mảng được sắp xếp theo **thứ tự giảm dần** nếu

$$A[i] \geq A[i+1], 0 \leq i \leq n-2$$

II – SORTING ALGORITHMS – SELECTION SORT

Sắp xếp tăng dần một mảng có n phần tử.

- **Ý tưởng**: tìm liên tục phần tử nhỏ nhất trong phần mảng chưa được sắp xếp và hoán đổi nó với phần tử đầu tiên của phần mảng chưa được sắp xếp. Quá trình này tiếp tục cho đến khi tất cả các phần tử được sắp xếp.

Mã giả - Sắp xếp chọn trực tiếp

```
1 Inputs:
2 - array A: an array of integers to be sorted
3 - int n: the length of the array
4
5 Outputs:
6 - array A: the sorted array
```

II – SORTING ALGORITHMS – SELECTION SORT

Mã giả - Sắp xếp chọn trực tiếp

```
01 function selectionSort(A, n)
02     for i = 0 to n-1
03         # Set the first unsorted element as the minimum
04         minIndex = i
05         for j = i+1 to n
06             // Check if an unsorted element is smaller than
07             // the current minimum
08             if A[j] < A[minIndex] then
09                 # If so, set it as the new minimum
10                 minIndex = j
11             end if
12         end for
13         // Swap the current minimum with the first unsorted element
14         swap(A[minIndex], A[i])
15     end for
16 end function
```

II – SORTING ALGORITHMS – SELECTION SORT FUNCTION

```
01 void selectionSort(int arr[], int n) {
02     // Loop through all the elements in the array
03     for (int i = 0; i < n-1; i++) {
04         // Find the index of the minimum element
05         // in the unsorted part of the array
06         int minIndex = i;
07         for (int j = i+1; j < n; j++) {
08             if (arr[j] < arr[minIndex]) {
09                 minIndex = j;
10             }
11         }
12         // Swap the minimum element with the first element
13         // in the unsorted part of the array
14         // only if they are different
15         if (arr[minIndex] != arr[i]) {
16             swap(&arr[minIndex], &arr[i]);
17         }
18     }
19 }
```

II – SORTING ALGORITHMS – SELECTION SORT

```
// Function to swap two elements in an array
1 void swap(int *a, int *b) {
2     int temp = *a;
3     *a = *b;
4     *b = temp;
5 }
```

II – SORTING ALGORITHMS – SELECTION SORT

Minh họa quá trình sắp xếp tăng dần mảng sau 70, 6, 7, 13, 65, 48, 29 sử dụng thuật toán sắp xếp chọn trực tiếp (selection sort).

```
Mảng ban đầu:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  70 |  6  |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – SELECTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  60 |  6  |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 1:
   * Index của giá trị nhỏ nhất hiện tại: 0
   * Giá trị nhỏ nhất hiện tại: 60
   * Index của giá trị nhỏ nhất thực tế: 1
   * Giá trị nhỏ nhất thực tế: 6
   * Đổi chỗ 60 và 6
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 2:
   * Index của giá trị nhỏ nhất hiện tại: 1
   * Giá trị nhỏ nhất hiện tại: 60
   * Index của giá trị nhỏ nhất thực tế: 2
   * Giá trị nhỏ nhất thực tế: 7
   * Đổi chỗ 60 và 7
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 3:
   * Index của giá trị nhỏ nhất hiện tại: 2
   * Giá trị nhỏ nhất hiện tại: 60
   * Index của giá trị nhỏ nhất thực tế: 3
   * Giá trị nhỏ nhất thực tế: 13
   * Đổi chỗ 60 và 13
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – SELECTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 4:
   * Index của giá trị nhỏ nhất hiện tại: 3
   * Giá trị nhỏ nhất hiện tại: 60
   * Index của giá trị nhỏ nhất thực tế: 6
   * Giá trị nhỏ nhất thực tế: 29
   * Đổi chỗ 60 và 29
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  65 |  48 |  60 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – SELECTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  65 |  48 |  60 |
+-----------------------------------------+
Lần lặp 5:
   * Index của giá trị nhỏ nhất hiện tại: 4
   * Giá trị nhỏ nhất hiện tại: 65
   * Index của giá trị nhỏ nhất thực tế: 5
   * Giá trị nhỏ nhất thực tế: 48
   * Đổi chỗ 65 và 48
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  65 |  60 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – SELECTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  65 |  60 |
+-----------------------------------------+
Lần lặp 6:
   * Index của giá trị nhỏ nhất hiện tại: 5
   * Giá trị nhỏ nhất hiện tại: 65
   * Index của giá trị nhỏ nhất thực tế: 6
   * Giá trị nhỏ nhất thực tế: 60
   * Đổi chỗ 65 và 60
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – SELECTION SORT

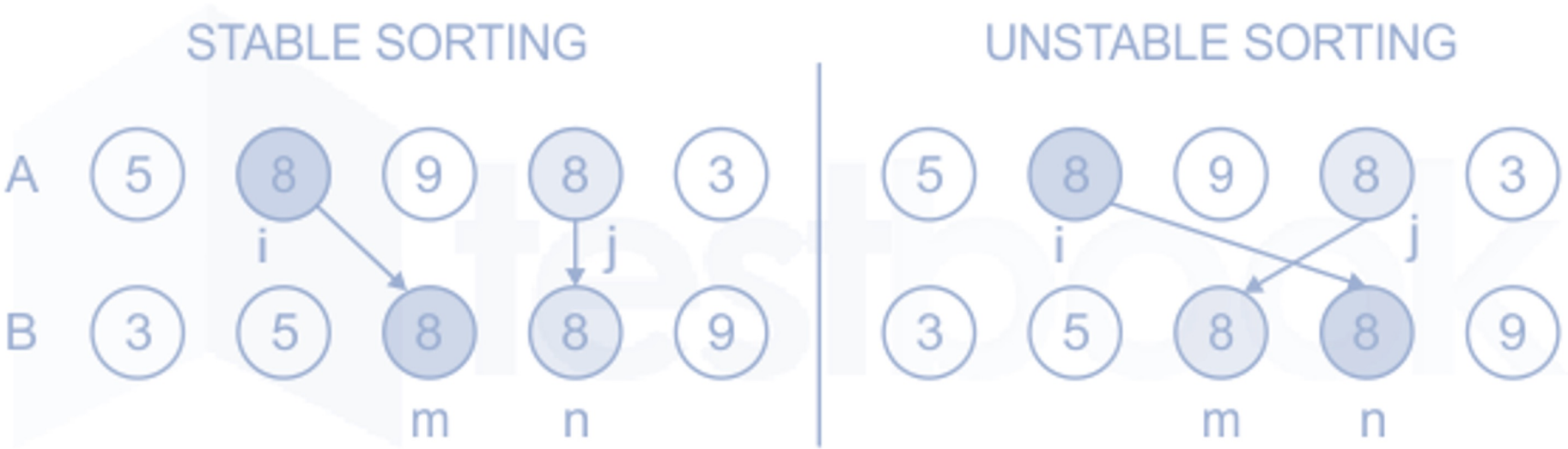
```
Mảng sau khi được sắp xếp
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

Ưu điểm

1. **Dễ hiểu**: Thuật toán đơn giản và dễ hiểu.
2. **Sắp xếp tương đối ổn định**: Sắp xếp lựa chọn duy trì thứ tự tương đối của các phần tử bằng nhau trong mảng được sắp xếp, làm cho nó trở thành thuật toán sắp xếp ổn định.

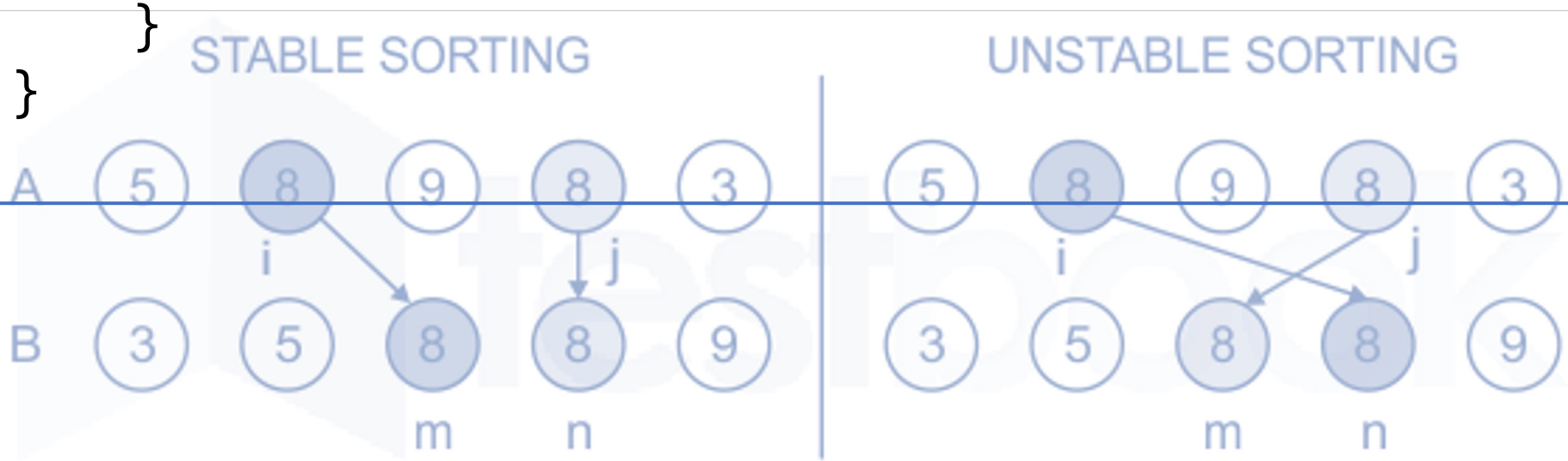
II – SORTING ALGORITHMS – SELECTION SORT

```
01 void selectionSort(int arr[], int n) {
...
03      for (int i = 0; i < n-1; i++) {
...
12          // Swap the minimum element with the first element
13          // in the unsorted part of the array
14          swap(&arr[minIndex], &arr[i]);
15      }
16 }
```



II – SORTING ALGORITHMS – SELECTION SORT

```
01 void selectionSort(int arr[], int n) {
...
03      for (int i = 0; i < n-1; i++) {
...
12          // Swap the minimum element with the first element
13          // in the unsorted part of the array
14          // only if they are different
15          if (minIndex != i) {
16              swap(&arr[minIndex], &arr[i]);
17          }
18      }
19 }
```



Ưu điểm

3. **Sắp xếp tại chỗ**: Sắp xếp lựa chọn là thuật toán sắp xếp tại chỗ, nghĩa là nó sắp xếp các phần tử của mảng đầu vào mà không sử dụng bất kỳ bộ nhớ bổ sung nào. Điều này có thể hữu ích khi bộ nhớ bị hạn chế.
4. **Hiệu quả đối với các tập dữ liệu nhỏ**: Đối với các tập dữ liệu nhỏ, sắp xếp lựa chọn nhanh hơn các thuật toán sắp xếp khác.

Nhược điểm

1. **Chậm đối với các tập dữ liệu lớn**: Sắp xếp lựa chọn có độ phức tạp về thời gian là $O(n^2)$, khiến nó không hiệu quả đối với các tập dữ liệu lớn.
2. **Không hiệu quả đối với dữ liệu được sắp xếp một phần**: Nếu mảng đầu vào được sắp xếp một phần, sắp xếp lựa chọn hoạt động kém hơn so với các thuật toán sắp xếp khác, vì nó không tận dụng được tính chất được sắp xếp một phần của dữ liệu.

Nhược điểm

3. **Tối ưu hóa phần cứng không cải thiện hiệu suất:** Tối ưu hóa phần cứng, chẳng hạn như tính song song ở cấp độ phần cứng, không thể cải thiện hiệu suất của sắp xếp lựa chọn nhiều vì đây là thuật toán tuần tự.

II – SORTING ALGORITHMS – INSERTION SORT

Sắp xếp tăng dần một mảng có N phần tử.

- **Ý tưởng** tạo ra một tập hợp các phần tử đã sắp xếp từ một tập hợp các phần tử chưa sắp xếp. Mỗi lần, chúng ta lấy một phần tử từ tập hợp các phần tử chưa sắp xếp và chèn nó vào vị trí thích hợp trong tập hợp các phần tử đã sắp xếp. Quá trình này được lặp lại cho đến khi tất cả các phần tử đã được sắp xếp.

II – SORTING ALGORITHMS – INSERTION SORT

Mã giả - Sắp xếp chèn

```
1 Inputs:
2 - array A: an array of integers to be sorted
3 - int n: the length of the array
4
5 Outputs:
6 - array A: the sorted array
```

II – SORTING ALGORITHMS – INSERTION SORT

Mã giả - Sắp xếp chèn

```
01 Algorithm:
02 function insertionSort(array A, int n)
03     // Loop through all the elements in the array
04     for i from 1 to n - 1
05         // Save the current element as a value
06         value = A[i]
07         // Start comparing from the previous element
08         j = i - 1
09         // Keep moving elements to the right
10         // until the correct position is found
11         while j >= 0 and A[j] > value
12             A[j + 1] = A[j]
13             j = j - 1
14         end while
15         // Insert the current element into the correct position
16         A[j + 1] = value
17     end for
18 end function
```

II – SORTING ALGORITHMS – INSERTION SORT

```
01 void insertionSort(int arr[], int n) {
02     // Loop through all the elements in the array
03     for (int i = 1; i < n; i++) {
04         // Save the current element as a value
05         int value = arr[i];
06         // Start comparing from the previous element
07         int j = i - 1;
08         // Keep moving elements to the right
09         // until the correct position is found
10         while (j >= 0 && arr[j] > value) {
11             arr[j + 1] = arr[j];
12             j = j - 1;
13         }
14         // Insert the current element into the correct position
15         arr[j + 1] = value;
16     }
17 }
```

II – SORTING ALGORITHMS – INSERTION SORT

Minh họa quá trình sắp xếp tăng dần mảng sau 70, 6, 7, 13, 65, 48, 29 sử dụng thuật toán sắp xếp chèn (insertion sort).

Mảng ban đầu:

0	1	2	3	4	5	6
70	6	7	13	65	48	29

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
 0     1     2     3     4     5     6
+-----------------------------------------+
|  60 |  6  |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 1:
   * Index của đang xét: 1
   * Phần tử đang xét: 6
   * Cần chèn tại index: 0
   * Phần tử tại vị trí cần chèn: 60
   * Phần tử 6 được chèn vào vị trí 0
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 2:
   * Index của đang xét: 2
   * Phần tử đang xét: 7
   * Cần chèn tại index: 1
   * Phần tử tại vị trí cần chèn: 60
   * Phần tử 7 được chèn vào vị trí 1
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 3:
   * Index của đang xét: 3
   * Phần tử đang xét: 13
   * Cần chèn tại index: 2
   * Phần tử tại vị trí cần chèn: 60
   * Phần tử 13 được chèn vào vị trí 2
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 4:
   * Index của đang xét: 4
   * Phần tử đang xét: 65
   * Cần chèn tại index: 3
   * Phần tử tại vị trí cần chèn: 60
   * Phần tử 65 được chèn vào vị trí 4
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
Lần lặp 5:
   * Index của đang xét: 5
   * Phần tử đang xét: 48
   * Cần chèn tại index: 4
   * Phần tử tại vị trí cần chèn: 65
   * Phần tử 48 được chèn vào vị trí 3
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  60 |  65 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  60 |  65 |  29 |
+-----------------------------------------+
Lần lặp 6:
   * Index của đang xét: 6
   * Phần tử đang xét: 29
   * Cần chèn tại index: 5
   * Phần tử tại vị trí cần chèn: 65
   * Phần tử 29 được chèn vào vị trí 3
Kết quả sau lần lặp này
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – INSERTION SORT

```
Mảng sau khi được sắp xếp
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

Ưu điểm

1. **Đơn giản và dễ hiểu**: Insertion sort là một thuật toán rất đơn giản và dễ hiểu, giúp cho việc sử dụng và áp dụng dễ dàng hơn.
2. **Thực hiện nhanh trên tập hợp dữ liệu có sẵn**: Nếu tập hợp dữ liệu có một số phần tử đã được sắp xếp, Insertion sort có thể hoạt động nhanh hơn so với các thuật toán sắp xếp khác.
3. **Nhẹ tải trên bộ nhớ**: Insertion sort chỉ yêu cầu một số biến tạm để hoàn tất việc sắp xếp, do đó nó sử dụng ít bộ nhớ hơn so với các thuật toán sắp xếp khác.

Nhược điểm

1. **Tốc độ chậm trên tập hợp dữ liệu lớn:** Insertion sort có thể chậm trên các tập hợp dữ liệu lớn vì nó cần phải di chuyển nhiều phần tử để sắp xếp.
2. **Không hiệu quả trên tập hợp dữ liệu đảo ngược:** Nếu tập hợp dữ liệu ban đầu đảo ngược, Insertion sort cần phải di chuyển tất cả các phần tử cho mỗi phần tử được chèn.
3. **Chỉ phù hợp với tập hợp dữ liệu nhỏ**: vì tốc độ chậm và sử dụng bộ nhớ cao.

II – SORTING ALGORITHMS – INSERTION SORT

Vòng lặp for bên ngoài thực hiện n – 1 lần lặp. Tuy nhiên, số lần các phần tử lớn hơn data[i] được di chuyển sang phải không cố định.

- **Trường hợp tốt nhất** là khi dữ liệu đã được sắp xếp sẵn. Chỉ một so sánh được thực hiện với mỗi giá trị của i. Vậy có tổng cộng n – 1 so sánh (comparison), 2(n -1) di chuyển (movement) và độ phức tạp thuật toán là O(n)

- **Trường hợp xấu nhất** là khi dữ liệu được sắp xếp theo chiều ngược lại. Mỗi lần lặp i sẽ có i phép so sánh. Vậy tổng cộng ta có

$$\sum_{i=1}^{n-1} i = 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} [\text{so sánh}] = O(n^2)$$

$$\frac{n(n-1)}{2} + 2(n-1) = \frac{n^2 + 3n - 4}{2} [\text{di chuyển}] = O(n^2)$$

II – SORTING ALGORITHMS – BUBBLE SORT

Sắp xếp tăng dần một mảng có N phần tử.

- Ý tưởng của thuật toán sắp xếp nổi bọt (bubble sort) là sử dụng một vòng lặp để so sánh các cặp phần tử liền kề và hoán đổi chúng nếu cần.
- Quá trình này sẽ diễn ra một số lần cho đến khi tập hợp dữ liệu được sắp xếp. Mỗi lần qua vòng lặp, phần tử lớn nhất sẽ "nổi bọt" lên đầu tập hợp dữ liệu. Kết quả cuối cùng sẽ là tập hợp dữ liệu được sắp xếp từ nhỏ đến lớn.

II – SORTING ALGORITHMS – BUBBLE SORT

Mã giả - Sắp xếp nổi bọt

```
1 Inputs:
2 - array A: an array of integers to be sorted
3 - int n: the length of the array
4
5 Outputs:
6 - array A: the sorted array
```

II – SORTING ALGORITHMS – BUBBLE SORT

Mã giả - Sắp xếp nổi bọt

```
01 function bubbleSort(array arr, int n)
02     // Loop through all elements of the array
03     // (except the last element)
04     for i = 0 to n-2
05         // Compare the unsorted elements
06         for j = 0 to n-i-2
07             // If the current element is greater
08             // than the next element, swap them
09             if arr[j] > arr[j+1] then
10                 swap arr[j] and arr[j+1]
11             end if
12         end for
13     end for
14 end function
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
01 void bubbleSort(int arr[], int n) {
02     // Loop through all elements of the array
03     // (except the last element)
04     for (int i = 0; i < n - 1; i++) {
05         // Compare the unsorted elements
06         for (int j = 0; j < n - i - 1; j++) {
07             // If the current element is greater than
08             // the next element, swap them
09             if (arr[j] > arr[j + 1]) {
10                 swap(arr[j], arr[j + 1]);
11             }
12         }
13     }
14 }
```

II – SORTING ALGORITHMS – BUBBLE SORT

Minh họa quá trình sắp xếp tăng dần mảng sau 70, 6, 7, 13, 65, 48, 29 sử dụng thuật toán sắp xếp nổi bọt (bubble sort).

Mảng ban đầu:

0	1	2	3	4	5	6
70	6	7	13	65	48	29

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
 0     1     2     3     4     5     6
+-----------------------------------------+
|  60 |  6  |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+

Lần lặp 0, vòng lặp 0:
   * Phần tử 6 được hoán vị với 60
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  60 |  7  |  13 |  65 |  48 |  29 |
+-----------------------------------------+

Lần lặp 0, vòng lặp 1:
   * Phần tử 7 được hoán vị với 60
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  60 |  13 |  65 |  48 |  29 |
+-----------------------------------------+

Lần lặp 0, vòng lặp 2:
   * Phần tử 13 được hoán vị với 60
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-------------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-------------------------------------------+

Lần lặp 0, vòng lặp 3:
Không có hoán vị trong lần lặp này
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  65 |  48 |  29 |
+-----------------------------------------+

Lần lặp 0, vòng lặp 4:
   * Phần tử 48 được hoán vị với 65
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  48 |  65 |  29 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  48 |  65 |  29 |
+-----------------------------------------+

Lần lặp 0, vòng lặp 5:
   * Phần tử 29 được hoán vị với 65
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  48 |  29 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  48 |  29 |  65 |
+-----------------------------------------+
Lần lặp 1, vòng lặp 0:
Không có hoán vị trong lần lặp này

Lần lặp 1, vòng lặp 1:
Không có hoán vị trong lần lặp này

Lần lặp 1, vòng lặp 2:
Không có hoán vị trong lần lặp này
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  60 |  48 |  29 |  65 |
+-----------------------------------------+

Lần lặp 1, vòng lặp 3:
   * Phần tử 48 được hoán vị với 60
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  60 |  29 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  60 |  29 |  65 |
+-----------------------------------------+

Lần lặp 1, vòng lặp 4:
   * Phần tử 29 được hoán vị với 60
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  29 |  60 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  29 |  60 |  65 |
+-----------------------------------------+
Lần lặp 2, vòng lặp 0:
Không có hoán vị trong lần lặp này

Lần lặp 2, vòng lặp 1:
Không có hoán vị trong lần lặp này

Lần lặp 2, vòng lặp 2:
Không có hoán vị trong lần lặp này
```

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  48 |  29 |  60 |  65 |
+-----------------------------------------+

Lần lặp 2, vòng lặp 3:
   * Phần tử 29 được hoán vị với 48
Kết quả sau lần lặp này:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
Lần lặp 3, vòng lặp 0:
Không có hoán vị trong lần lặp này

Lần lặp 3, vòng lặp 1:
Không có hoán vị trong lần lặp này

Lần lặp 3, vòng lặp 2:
Không có hoán vị trong lần lặp này
```

```
Mảng trước đó
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
Lần lặp 4, vòng lặp 0:
Không có hoán vị trong lần lặp này

Lần lặp 4, vòng lặp 1:
Không có hoán vị trong lần lặp này

Lần lặp 5, vòng lặp 0:
Không có hoán vị trong lần lặp này
```

II – SORTING ALGORITHMS – BUBBLE SORT

```
Mảng sau khi được sắp xếp:
   0     1     2     3     4     5     6
+-----------------------------------------+
|  6  |  7  |  13 |  29 |  48 |  60 |  65 |
+-----------------------------------------+
```

Ưu điểm

1. **Dễ hiểu và thực hiện**: Bubble Sort là một trong những thuật toán sắp xếp dễ hiểu và thực hiện.
2. **Sắp xếp ổn định**: duy trì thứ tự tương đối của các phần tử bằng nhau trong danh sách.
3. **Sắp xếp tại chỗ**: không yêu cầu bộ nhớ bổ sung để sắp xếp các phần tử.

Nhược điểm

1. **Hiệu suất chậm**: Bubble Sort có độ phức tạp thời gian là O (n^2), điều này khiến nó rất chậm cho các tập dữ liệu lớn.
2. **Ứng dung hạn chế**: không phù hợp để ứng dung thực tế với dữ liệu lớn.

EXERCISES

Bài 1: Một số chương trình sẽ có hiệu quả tốt hơn khi dữ liệu được sắp xếp và ngược lại. Xác định việc sắp xếp dữ liệu là có cần thiết hay không cho các trường hợp sau:

1. Cho 2 từ, xác định xem 2 từ đó có phải là đảo ngữ của nhau hay không?

   Ví dụ 2 từ là đảo ngữ: elbow và below; night và thing; listen và silent
2. Tìm vật phẩm có giá trị thấp nhất
3. Tính giá trị trung bình (mean) của mảng
4. Tính giá trung vị (median) của mảng
5. Tìm giá trị xuất hiện nhiều nhất trong mảng

EXERCISES

Bài 2: Trong sắp xếp insertion sort, số lần so sánh và số di chuyển của trường hợp trung bình sẽ gần với trường hợp tốt nhất hay trường hợp xấu nhất hơn?

Bài 3: Viết chương trình thực hiện selection sort, insertion sort và bubble sort.

THANK YOU FOR YOUR ATTENTIONS