# SẮP XẾP

Nguyễn Thị Hải Bình

nth.binh@hutech.edu.vn

# Mục tiêu

- Hiểu được các khái niệm về thuật toán sắp xếp.
- Nắm được thuật toán đổi chỗ trực tiếp (Interchange sort) cài đặt được thuật toán.
- Nắm được thuật toán nổi bọt (Bubble sort) và cài đặt được thuật toán.
- Nắm được thuật toán nổi bọt đệ quy (Recursive Bubble sort) và cài đặt được thuật toán.
- Nắm được thuật toán chèn (Insertion sort) và cài đặt được thuật toán.
- Nắm được thuật toán chèn đệ quy (Recursive Insertion sort) và cài đặt được thuật toán
- Nắm được thuật toán sắp xếp chọn (Selection sort) và cài đặt được thuật toán.

# NỘI DUNG

1. Khái niệm về thuật toán sắp xếp.
2. Các thuật toán sắp xếp cơ bản:
   a) Sắp xếp đổi chỗ trực tiếp (Interchange Sort)
   b) Sắp xếp nổi bọt (Bubble Sort)
   c) Sắp xếp nổi bọt đệ quy (Recursive Bubble Sort)
   d) Sắp xếp chèn (Insertion Sort)
   e) Sắp xếp chèn đệ quy (Recursive Insertion Sort)
   f) Sắp xếp chọn (Selection Sort)
   g) Sắp xếp chọn đệ quy (Recursive Selection Sort)

# Khái niệm về thuật toán sắp xếp?

- Sắp xếp = sắp xếp các phần tử trong một dãy (**collection**) theo thứ tự tăng dần hoặc giảm dần.
- Thuật toán sắp xếp đã được nghiên cứu từ rất lâu  Nhiều thuật toán sắp xếp đã được phát triển.
- Một số thuật toán sắp xếp đơn giản:
  - Sắp xếp đổi chỗ trực tiếp (Interchange Sort)
  - Sắp xếp nổi bọt (Bubble Sort)
  - Sắp xếp chọn (Selection Sort)
  - Sắp xếp chèn (Insertion Sort)
- Một số thuật toán sắp xếp nhanh và phổ biến:
  - Sắp xếp trộn (Merge Sort)
  - Sắp xếp nhanh (Quick Sort)
- *Ghi chú: bài giảng sẽ thực hiện minh họa với sắp xếp theo thứ tự tăng dần.*

# Sắp xếp đổi chỗ trực tiếp (Exchange Sort)

- Lần lượt so sánh các phần tử của dãy với phần tử đầu tiên. Đổi chỗ phần tử đang xét với phần tử đầu tiên nếu phần tử đầu tiên lớn hơn phần tử đang xét.
- Tiếp tục so sánh các phần tử của dãy (không tính phần tử đầu tiên) với phần tử thứ hai. Đổi chỗ phần tử đang xét với phần tử thứ hai nếu phần tử thứ hai lớn hơn phần tử đang xét.
- Quá trình trên lặp lại tới khi phần tử cuối cùng của dãy đã được xét. Khi đó, dãy sẽ được sắp xếp thành công.

# Exchange Sort Algorithm

```
1.  n = len(A)
2.  i = 0
3.  While i < n-1:
    3.1. j = i + 1
    3.2. While j < n:
3.2.1. If A[i] > A[j]:
swap A[i] and A[j]
       3.2.2. j = j + 1
    3.3. i = i + 1
```

| **Example** | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|

# Another Example: 4, 2, 5, 9, 10, 5, 7, 11

# Another Example: 46, 44, 20, 5, 13, 35, 21, 33

# Another Example: 2,44,38,5,47,15,36,26,27,46

# Exchange Sort Algorithm Implementation

```
1.  n = len(A)
2.  i = 0
3.  While i < n-1:
    3.1. j = i + 1
    3.2. While j < n:
 3.2.1. If A[i] > A[j]:
swap A[i] and A[j]
       3.2.2. j = j + 1
    3.3. i = i + 1
```

# Exchange Sort Algorithm Implementation

1. n = len(A)
2. i = 0
3. While i < n-1:

    3.1. j = i + 1

    3.2. While j < n:

3.2.1. If A[i] > A[j]:
swap A[i] and A[j]

        3.2.2. j = j + 1

    3.3. i = i + 1

```python
# Exchange Sort
def exchange_sort(A):
    i = 0
    while i < len(A) - 1:
        j = i + 1
        while j < len(A):
            if A[i] > A[j]:
                A[i], A[j] = A[j], A[i]
            j += 1
        i += 1
```

# Sắp xếp nổi bọt (Bubble Sort)

- The bubble sort algorithm got its name from the way bubbles rises to the surface.
  - The largest bubble will **reach** the surface first.
  - The second largest bubble will **reach** the surface next.
  - And so on.

- In the **bubble sort** algorithm:
  - The largest value will **reach** its correct position first.
  - The second largest bubble will **reach** its correct position next.
  - And so on.

# Bubble Sort Algorithm

```
1.  n = len(A)
2.  i = 0
3.  While i < n-1:
    3.1. j = 0
    3.2. While j < n-i-1:
 3.2.1. If A[j] > A[j+1]:
swap A[j] and A[j+1]
       3.2.2. j = j + 1
    3.3. i = i + 1
```

| **Example** | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

# Another Example: 4, 2, 5, 9, 10, 5, 7, 11

# Another Example: 46, 44, 20, 5, 13, 35, 21, 33

# Another Example: 2,44,38,5,47,15,36,26,27,46

# Bubble Sort Algorithm Implementation

```
1. n = len(A)
2. i = 0
3. While i < n-1:
   3.1. j = 0
      3.2. While j < n-i-1:
         3.2.1. If A[j] > A[j+1]:
                swap A[j] and A[j+1]
         3.2.2. j = j + 1
      3.3. i = i + 1
```

# Bubble Sort Algorithm Implementation

```python
# Bubble Sort
def bubble_sort(A):
    n = len(A)
    i = 0
    while i < n - 1:
        j = 0
        while j < n-i-1:
            if A[j] > A[j+1]:
                A[j], A[j+1] = A[j+1], A[j]
            j += 1
        i += 1

# Test the implementation
A = [3, 1, -8, 2, 0, -1, 5]
bubble_sort(A)
print(A)
```

# Sắp xếp nổi bọt đệ quy (Recursive Bubble Sort)

- Điểm dừng (base case):
  - Nếu len(A) = 1 □ Sắp xếp xong.

- Đệ quy (recursive case):
  - Đưa phần tử lớn nhất về vị trí cuối cùng của dãy số.
  - Gọi đệ quy thực hiện sắp xếp các phần tử của dãy số ngoại trừ phần tử ở vị trí cuối cùng.

| Example | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|

```
# Recursive Bubble Sort
def recursive_bubble_sort(A, n):
    # base case
    ____________________________

    # move the largest element to end
    ____________________________

    # recursive call
    recursive_bubble_sort(A, n-1)
```

# Recursive Bubble Sort Algorithm Implementation

```python
# Recursive Bubble Sort
def recursive_bubble_sort(A, n):
    # base case
    if n == 1:
        return
    # move the largest element to end
    for i in range(n-1):
        if A[i] > A[i + 1]:
            A[i], A[i + 1] = A[i + 1], A[i]
    # recursive call
    recursive_bubble_sort(A, n-1)

# Test the implementation
A = [3, 1, -8, 2, 0, -1, -5]
recursive_bubble_sort(A, len(A))
print(A)
```

# Sắp xếp chèn (Insertion Sort)

**Ý tưởng:**

- Phương pháp này sẽ lần lược chèn các nút vào danh sách đã có thứ tự.

**Mô tả phương pháp:**

- Xem danh sách đầu tiên đã có thứ tự chỉ là 1 nút A[0].
- Lần chèn 1: chèn A[1] vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng hai nút là A[0] và A[1].
- Lần chèn 2: chèn A[2] vào đúng vị trí chúng ta được danh sách đã có thứ tự có đúng 3 nút là A[0], A[1] và A[2].
- …
- Lần chèn n-1: chèn A[n-1] vào đúng vị trí chúng ta được danh sách cuối cùng đã có thứ tự có n nút là A[0], A[1], …, A[n-1].

# Insertion Sort Algorithm

1. `i = 1`
2. a) `X = A[i]`
   b) `j = i-1`
   c) `while A[j] > X and j >=0:`
      `move A[j] to the right`
      `by 1, then j = j-1.`
   d) `Insert X to the position`
      `j+1`
3. `i = i + 1`
   a) `If i < n, repeat step #2`
   b) `Otherwise, stop.`

| **Example** | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|

# Another Example: 4, 2, 5, 9, 10, 5, 7, 11

# Another Example: 46, 44, 20, 5, 13, 35, 21, 33

# Another Example: 2,44,38,5,47,15,36,26,27,46

# Insertion Sort Algorithm Implementation

1. `i = 1`
2. a) `X = A[i]`
   b) `j = i-1`
   c) `while A[j] > X and j >=0: move A[j] to the right by 1, then j = j-1.`
   d) `Insert X to the position j+1`
3. `i = i + 1`
   a) `If i < n, repeat step #2`
   b) `Otherwise, stop.`

# Insertion Sort Algorithm Implementation

1. i = 1
2. a) X = A[i]
   b) j = i-1
   c) while A[j] > X and j >=0: move A[j] to the right by 1, then j = j-1.
   d) Insert X to the position j+1
3. i = i + 1
   a) If i < n, repeat step #2
   b) Otherwise, stop.

```python
# Insertion Sort
def insertion_sort(A):
    i = 1
    while i < len(A):
        x = A[i]
        j = i - 1
        while A[j] > x and j >= 0:
            A[j+1] = A[j]
            j -= 1
        A[j+1] = x
        i += 1
```

# Sắp xếp chèn đệ quy (Recursive Insertion Sort)

- Điểm dừng (base case):
  - Nếu len(A) = 1 □ Sắp xếp xong.

- Đệ quy (recursive case):
  - Gọi đệ quy sắp xếp n-1 phần tử đầu tiên.
  - Chèn phần tử cuối cùng vào đúng vị trí của nó trong mảng đã được sắp xếp.

| **Example** | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

```
# Recursive Insertion Sort
def recursive_insertion_sort(A, n):
    # base case

    ____________________________

    # recursive call

    ____________________________

    # move the last element to the
    # correct position

    ____________________________
```

# Recursive Insertion Sort Algorithm Implementation

```
# Recursive Insertion Sort
def recursive_insertion_sort(A, n):
    # base case
    if n == 1:
        return
    # recursive call
    recursive_insertion_sort(A, n - 1)
    # move the last element to the correct position
    x = A[n-1]
    j = n - 2
    while A[j] > x and j >= 0:
        A[j+1] = A[j]
        j -= 1
    A[j+1] = x
```

# Sắp xếp chọn (Selection Sort)

**Mô tả phương pháp:**

- Phương pháp này lần lượt chọn nút nhỏ nhất cho các vị trí 0, 1, 2, …, n-1. Cụ thể:
- Lần chọn thứ 0:
  - Dò tìm trong khoảng vị trí từ 0 đến n-1 để xác định nút nhỏ nhất tại vị trí $min_0$.
  - Đổi chỗ hai nút tại vị trí $min_0$ và vị trí 0.
- Lần chọn thứ 1:
  - Dò tìm trong khoảng vị trí từ 1 đến n-1 để xác định nút nhỏ nhất tại vị trí $min_1$.
  - Đổi chỗ hai nút tại vị trí $min_1$ và vị trí 1.

# Sắp xếp chọn (Selection Sort)

**Mô tả phương pháp:**

- Lần chọn thứ i:
  - Dò tìm trong khoảng vị trí từ i đến n-i để xác định nút nhỏ nhất tại vị trí $min_i$.
  - Đổi chỗ hai nút tại vị trí $min_i$ và vị trí i.
- ……
- Lần chọn thứ n-2 (lần chọn cuối cùng):
  - Dò tìm trong khoảng từ vị trí n-2 đến n-1 để xác định nút nhỏ nhất tại vị trí $min_{n-2}$.
  - Đổi chỗ hai nút tại vị trí $min_{n-2}$ và vị trí n-2.

# Selection Sort Algorithm

1. `i = 0`

2. a) `pMin = position of minimum value in the list containing A[i], A[i+1], … A[n-1]`

   b) `Swap A[i] and A[pMin]`

3. `i = i +1;`

   a) `If i < n-1, repeat step #2`

   b) `Otherwise, stop.`

| **Example** | 40 | 25 | 55 | 45 | 10 |
|---|---|---|---|---|---|

# Another Example: 4, 2, 5, 9, 10, 5, 7, 11

# Another Example: 46, 44, 20, 5, 13, 35, 21, 33

# Another Example: 2,44,38,5,47,15,36,26,27,46

# Selection Sort Algorithm Implementation

1. `i = 0`

2. a) `pMin = position of minimum value in the list containing A[i], A[i+1], … A[n-1]`
   b) `Swap A[i] and A[pMin]`

3. `i = i +1;`
   a) `If i < n-1, repeat step #2`
   b) `Otherwise, stop.`

# Selection Sort Algorithm Implementation

1. `i = 0`

2. a) pMin = position of minimum value in the list containing A[i], A[i+1], … A[n-1]

   b) Swap A[i] and A[pMin]

3. `i = i +1;`

   a) If i < n-1, repeat step #2

   b) Otherwise, stop.

```
# Selection Sort
def selection_sort(A):
    i = 0
    while i < len(A)-1:
        min_value = min(A[i:])
        p_min = A[i:].index(min_value) + i
        A[i], A[p_min] = A[p_min], A[i]
        i += 1
```

# Sắp xếp chọn đệ quy (Recursive Selection Sort)

- Điểm dừng (base case):
  - Nếu len(A) = 1 □ Sắp xếp xong.

- Đệ quy (recursive case):
  - Gọi đệ quy sắp xếp n-1 phần tử đầu tiên.
  - Chèn phần tử cuối cùng vào đúng vị trí của nó trong mảng đã được sắp xếp.

| | | | | | |
|---|---|---|---|---|---|
| **Example** | 40 | 25 | 55 | 45 | 10 |

```
# Recursive Selection Sort
def recursive_selection_sort(A, start_index):
    # base case

    ________________________

    # find the minimum index swap the found
    # element and the element at start_index

    ________________________

    print(A)
    # recursive call

    ________________________
```

# Recursive Selection Sort Algorithm Implementation

```
# Recursive Selection Sort
def recursive_selection_sort(A, start_index):
    # base case
    if start_index == len(A)-1:
        return
    # find the minimum index
    min_value = min(A[start_index:])
    p_min = A[start_index:].index(min_value) + start_index
    A[start_index], A[p_min] = A[p_min], A[start_index]
    print(A)
    # recursive call
    recursive_selection_sort(A, start_index + 1)
```

# BÀI TẬP

1. Cài đặt các thuật toán trong bài. Tính thời gian thực thi trung bình của mỗi thuật toán (gợi ý: sử dụng thư viện time bằng cách import time để tính).
2. Viết chương trình (dạng hàm) thực hiện các thao tác sau:
   a) Nhập danh sách sinh viên, mỗi sinh viên gồm các thông tin sau: mã sinh viên (duy nhất), họ và tên đệm,  tên, điểm thi.
   b) Sắp xếp danh sách sinh viên theo chiều tăng dần của tên.
   c) Sắp xếp danh sách sinh viên theo chiều tăng dần (hoặc giảm dần) của điểm thi. Thứ tự sắp xếp được truyền vào là tham số của hàm.