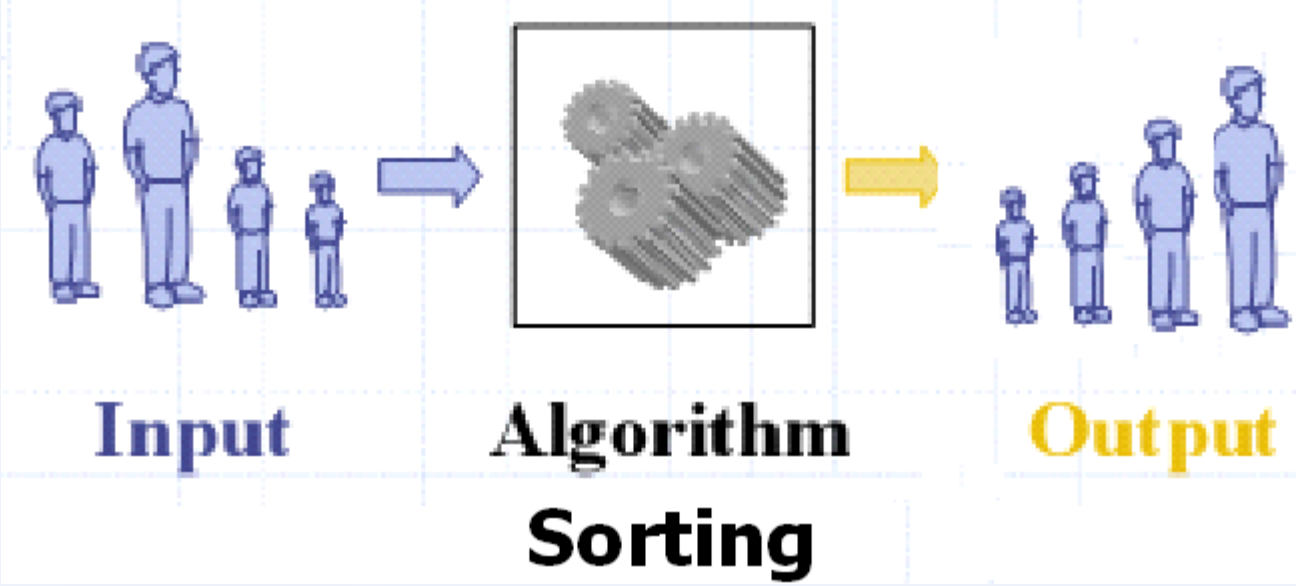
Bài 11.
Sắp xếp (Sorting)



Sorting

Tại sao cần sắp xếp

- Sắp xếp một danh sách các phần tử theo một thứ tự nào đó là một bài toán có ý nghĩa trong thực tiễn
- Sắp xếp là một yêu cầu không thể thiếu trong thiết kế, phát triển các phần mềm ứng dụng
- Nghiên cứu phương pháp sắp xếp là rất cần thiết

Sorting

Khái niệm

- **Sắp xếp** là quá trình bố trí lại các phần tử trong một tập hợp theo một trình tự nào đó nhằm mục đích giúp quản lý và tìm kiếm các phần tử dễ dàng và nhanh chóng hơn
- **Sắp xếp trong** là sự sắp xếp dữ liệu được tổ chức trong bộ nhớ trong của máy tính
- **Sắp xếp ngoài** là sự sắp xếp được sử dụng khi số lượng phần tử cần sắp xếp lớn không thể lưu trữ trong bộ nhớ trong mà phải lưu trữ trên bộ nhớ ngoài

Sorting

Bài toán

- **Input:**
  - Dãy các phần tử (và một thứ tự)
  - (Dãy các phần tử thường được lưu bằng mảng.)
- **Output:**
  - Dãy các phần tử được sắp theo thứ tự tăng hoặc giảm dần theo một hoặc một vài thuộc tính của nó (các thuộc tính này gọi là thuộc **tính khóa**).
  - Thuộc tính khóa được sắp xếp theo một hàm logic, ví dụ (<=) hoặc các toán tử so sánh khác.

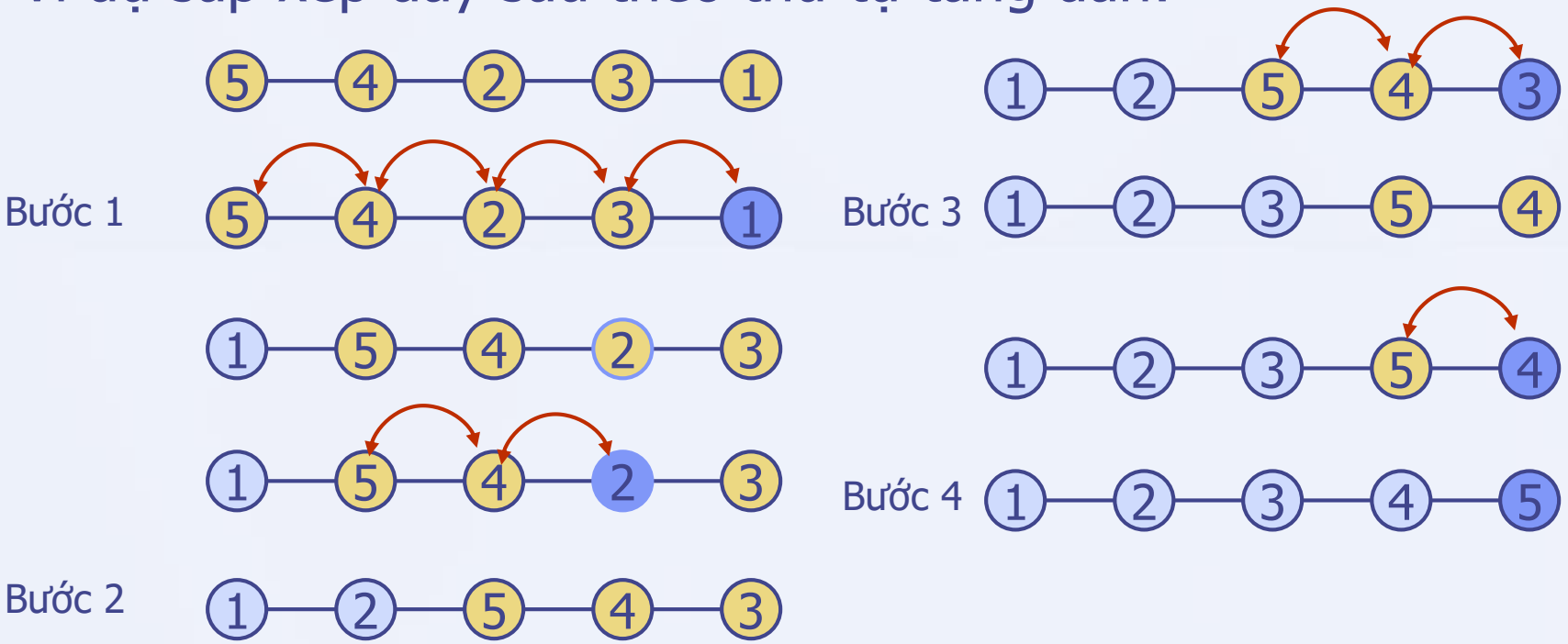
Các thuật toán với thời gian chạy $O(n^2)$

- Nổi bọt – Bubble sort
- Chèn – Insertion sort
- Chọn – Selection sort

Sắp xếp nổi bọt – Bubble sort

Ý tưởng: Thực hiện chuyển dần các phần tử có giá trị khóa nhỏ về đầu dãy, các phần tử có khóa lớn về cuối dãy.

Ví dụ sắp xếp dãy sau theo thứ tự tăng dần:



Thuật toán

Algorithm ***BubbleSort(Array A, n)***
Input: Mảng A có n phần tử
Output: Mảng A được sắp theo thứ tự tăng dần của khóa

```
for i ← 0 to n-2 do
    for j ← n-1 downto i+1 do
        if A[j] < A[j-1] then
            swap(A[j-1], A[j]);
```

- Trong đó swap là thủ tục tráo đổi vị trí của hai phần tử

```
void Swap(object &a, object &b){
Object tg;
        tg = a;  a = b;   b = tg;
}
```

Chứng minh thời gian chạy của thuật toán trong trường hợp xấu nhất là $O(n^2)$

?

Ví dụ

- SX dãy: 80 76 10 15 75 20 9

Thời gian chạy

Algorithm ***BubbleSort(Array A, n)***
Input: Mảng A có n phần tử
Output: Mảng A được sắp theo thứ tự tăng dần của khóa

```
for i ← 0 to n-2 do                          n+2
    for j ← n-1 downto i+1 do                 n-i+3
        if A[j] < A[j-1] then                 4
            swap(A[j-1], A[j]);               6
```

- Thời gian chạy:

$T(n) = (n+2) + (n-i+3)*(n-2) + 10*[(n-1) +(n-2)+..+2+1]$

- Thời gian chạy của thuật toán là $O(n^2)$

Ví dụ:

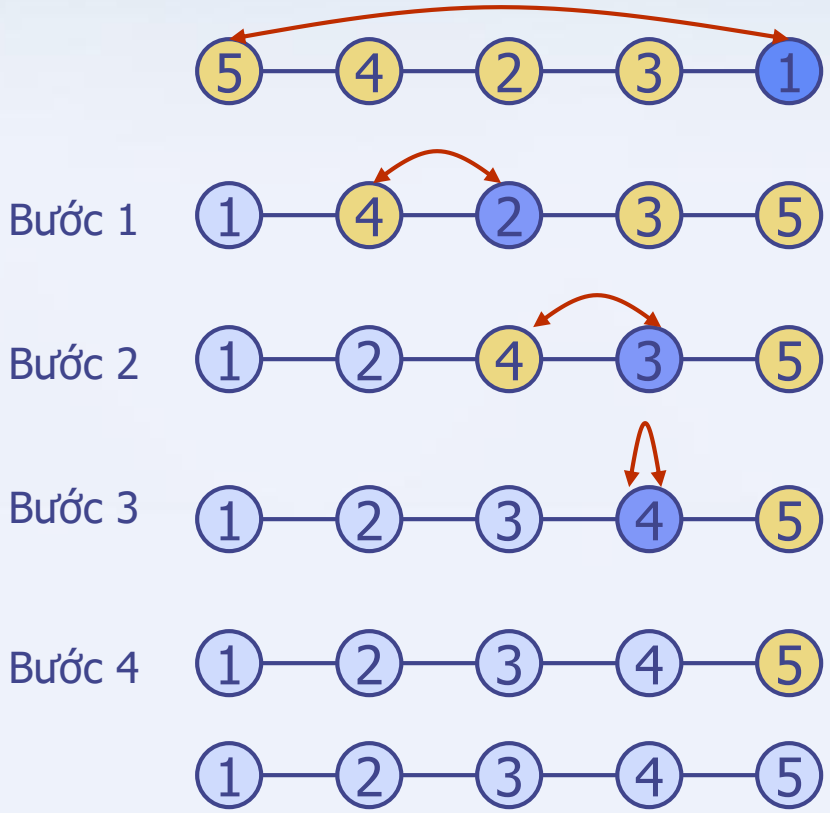
Mô tả quá trình sắp xếp của dãy số

12 43 11 34 23 43

Sắp xếp chọn - Selection sort

- **Ý tưởng:** Chọn phần tử có khóa nhỏ nhất trong các phần tử còn lại chuyển nó về đầu và loại bỏ nó khỏi dãy.

- Ví dụ sắp xếp dãy sau theo thứ tự tăng dần:



Thuật toán

Algorithm ***SelectionSort(Array A, n)***
Input: Mảng A có n phần tử
Output: Mảng A được sắp theo thứ tự tăng dần của khóa

```
for i ← 0 to n-2  do
        posmin ← i ;
        for j ← i+1 to n-1 do
            if A[posmin] > A[j] then
                posmin ← j ;
        if posmin ≠ i then
                swap (A[i], A[posmin]);
```

Chứng minh thời gian chạy của thuật toán trong trường hợp xấu nhất là $O(n^2)$

?

Thời gian chạy

```
for i ← 0 to n-2  do                          n+2
        posmin ← i ;                          n-1
        for j ← i+1 to n -1  do               2(n-1)
            if A[posmin] > A[j] then          3
                posmin ← j ;                  1
        if posmin ≠ i then                    n-1
                swap (A[i], A[posmin]);       6(n-1)
```

Thời gian chạy của thuật toán

$T(n) = (n+2) + 4* [(n-1)+(n-2)+..+1] + 10*(n-1)$

Thời gian chạy của thuật toán là $O(n^2)$

Ví dụ:

Mô tả quá trình sắp xếp của dãy số

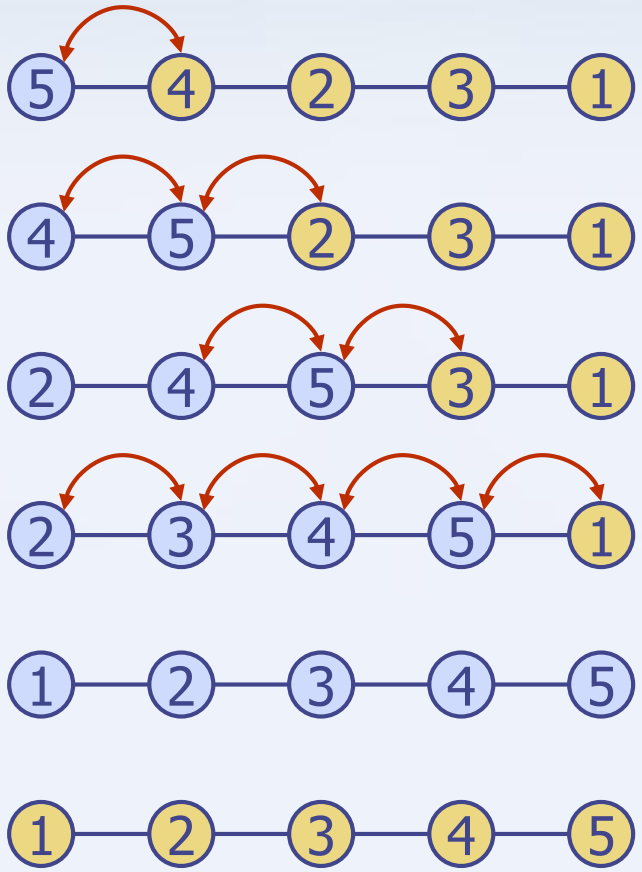
12 43 11 34 23 435

Sắp xếp chèn – Insertion sort

- **Ý tưởng:** Lấy phần tử thứ j (A[j]) chèn vào dãy đã được sắp gồm các phần tử từ A[0], …, A[j-1] sao cho ta được dãy A[0], …, A[j] cũng được sắp.

***Lưu ý*: dãy A[0], …, A[j-1] là dãy đã được sắp.**

Ví dụ sắp xếp dãy sau theo thứ tự tăng dần:



Thuật toán

Algorithm ***InsertionSort(Array A, n)***
Input: Mảng A có n phần tử
Output: Mảng A được sắp theo thứ tự tăng dần của khóa

```
for i ← 1 to n-1 do
    j ← i-1;
    x ← A[i];
    while (A[j] > x) and (j>=0) do
        A[j+1] ← A[j];
        j ← j-1;
    A[j+1] ← x;
```

Chứng minh thời gian chạy của thuật toán trong trường hợp xấu nhất là $O(n^2)$

?

Chứng minh thời gian chạy của thuật toán trong trường hợp xấu nhất là $O(n^2)$

- Đánh giá độ phức tạp
  - Vòng lặp ngoài (chỉ số i) của thuật toán được thực hiện n – 1 lần
  - Trong trường hợp xấu nhất (mảng có thứ tự đảo ngược), vòng lặp trong được thực thi với số lần:

$(n-1) + (n-2) + \ldots + 1 = n(n-1)/2$

→ $T(n) = O(n^2)$

Ví dụ:

Mô tả quá trình sắp xếp của dãy số

12 43 11 34 23 43