Giải thuật Sắp Xếp
(Sort algorithm)

Lecturer: Duc-Hieu Tran

Title: MSc. Computer Science

Nội dung

- Tổng quan
- Selection Sort
- Insertion Sort
- Buble Sort
- Shaker Sort

Khái quát

❖Giới thiệu

- **Sắp thứ tự là một bài toán thường gặp trong các công việc hàng ngày cũng như trong các công việc quản lý nhằm hỗ trợ cho việc tìm kiếm dễ dàng và nhanh chóng**
- **Ví dụ**

Danh sách trước khi sắp xếp

{1, 65, 2, 45, 23, 16, 46, 83, 38}

Danh sách sau khi sắp xếp

{1, 2, 16, 23, 38, 45, 46, 65, 83}

Khái quát

- **Sắp thứ tự được chia thành 2 nhóm chính:**
  - **Sắp thứ tự nội (sắp xếp mảng)**
    - **Sắp xếp các dữ liệu được nạp vào chương trình trong quá trình thực hiện (dữ liệu này được đưa vào bộ nhớ chính – RAM) cho phép quản lý và tra cứu dữ liệu**
  - **Sắp thứ tự ngoại (sắp xếp tập tin)**
    - **Sắp xếp các dữ liệu mà có 1 phần được lưu trên bộ nhớ chính (RAM) và một phần được lưu trên bộ nhớ ngoài (HDD, SSD, USB, …)**

Khái quát

❖ Các phương pháp sắp xếp thông dụng

- Buble Sort
- Selection Sort
- Insertion Sort
- Shaker Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Radix Sort

➢ Tùy vào bài toán cần lựa chọn phương pháp sắp xếp cho phù hợp

Sắp xếp chọn
Selection Sort

Sắp xếp chọn

❖**Selection Sort**

❖**Mô phỏng cách sắp xếp tự nhiên nhất trong thực tế**

❖**Ý tưởng**

- Chọn phần tử có giá trị nhỏ nhất trong dãy và đưa về vị trí đầu tiên (sắp xếp tăng dần)
- Bỏ qua phần tử đầu tiên và xét tiếp n-1 phần tử còn lại
- Lặp lại cho đến khi dãy chỉ còn 1 phần tử duy nhất

Sắp xếp chọn

❖Thuật toán

- **Bước 1**: khởi tạo i = 0
- **Bước 2**: Thực hiện:
  - ❑Tìm index mà A[index] nhỏ nhất trong dãy (A[i], A[i+1], A[i+2],…,A[n-1])
  - ❑Hoán vị A[index] và A[i]
- **Bước 3**: So sánh i và n
  - ❑Nếu i < n thì tăng i thêm 1 và quay về Bước 2
  - ❑Ngược lại, dừng thuật toán

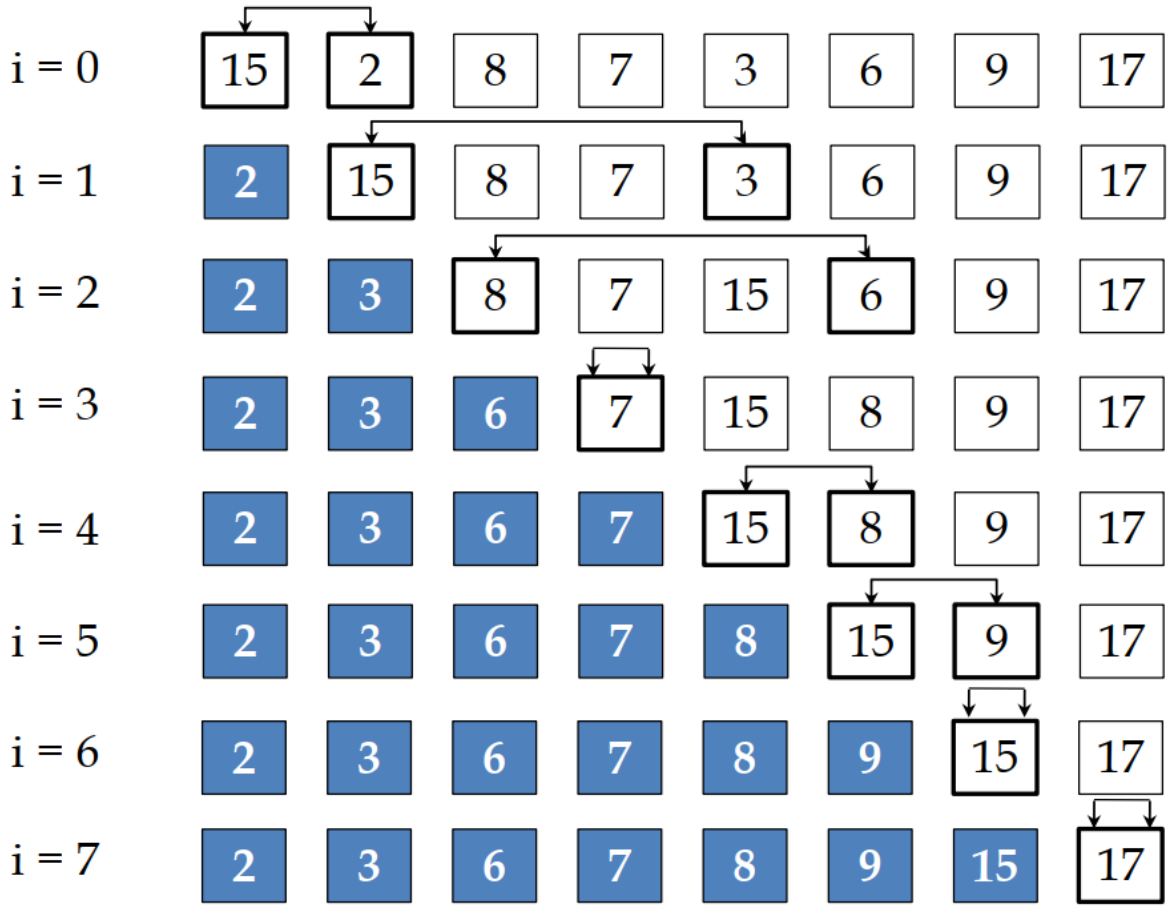
Sắp xếp chọn

❖Java

```
public static void SelectionSort(int A[], int n) {
	int index, trungGian;
	for (int i=0; i<n; i++) {
		index = i;
		for (int j=i+1; j<n; j++)
			if (A[j] < A[index])
				index = j;
		// Hoán vị A[i] và A[index]
		trungGian = A[i];
		A[i] = A[index];
		A[index] = trungGian;
	}
}
```

Sắp xếp chọn

❖Ví dụ



(Nguồn: cấu trúc dữ liệu và giải thuật – HMCUS 2016)

Sắp xếp chọn

❖Ví dụ

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

Trace of selection sort (array contents just after each exchange)

(Nguồn: Robert Sedgewick, Kevin Wayne - Algorithms 4th Edition, Princeton University, Addition Wesley)

Sắp xếp chọn

❖Độ phức tạp giải thuật

- Có n bước lặp
- Mỗi bước lặp cần (n – i) phép so sánh và 3 phép gán

⇒ f(n) = $\sum_{i=0}^{n-1}$[(n – i) + 3] = $\sum_{i=0}^{n-1}(n-i)$ + $\sum_{i=0}^{n-1} 3$

= [n + (n-1) + (n-2) + … + 2 + 1] + 3n = $\frac{n(n-1)}{2}$ + 3n

= $\frac{n^2}{2}$ - $\frac{n}{2}$ + 3n = $\frac{n^2}{2}$ + $\frac{5n}{2}$ ≤ $\frac{1}{2}$ ($n^2 + 5n^2$) = $3n^2$

Nên f(n) ≤ 3.g(n) với g(n) = n^2. Do đó, thuật toán có độ phức tạp là **O(n^2)**

(Không thay đổi trong bất kì trường hợp nào)

Sắp xếp chọn

❖Đánh giá độ phức tạp

Trường hợp	Số phép so sánh	Số phép gán
Tốt nhất	$\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{3n(n-1)}{2}$

Sắp xếp chèn
Insertion Sort

Sắp xếp chèn

❖**Insertion Sort**

❖**Mô phỏng từ cách sắp xếp các lá bài**

❖**Ý tưởng**

- Ban đầu trong dãy chỉ có 1 phần tử
- Lấy ra 1 phần tử kế tiếp đưa vào dãy
- Nếu phần tử này bé hơn 1 phần tử nào đó trong dãy thì chèn nó vào phía trước
- Ngược lại, chèn nó vào phía sau
- Lặp lại cho đến khi đã đưa hết n phần tử vào dãy

Sắp xếp chèn

❖Thuật toán

- **Bước 1**: khởi tạo i = 1
- **Bước 2**: Kiểm tra nếu i < n thì gán x = A[i]; j = i - 1;
- **Bước 3**: Kiểm tra nếu j ≥ 0 và A[j] > x thì
  - ❑Gán A[j+1] = A[j]; giảm j = j – 1;
  - ❑Lặp lại bước 3
- **Bước 4**:
  - ❑Gán A[j+1] = x; tăng i = i + 1;
  - ❑Quay lại bước 2

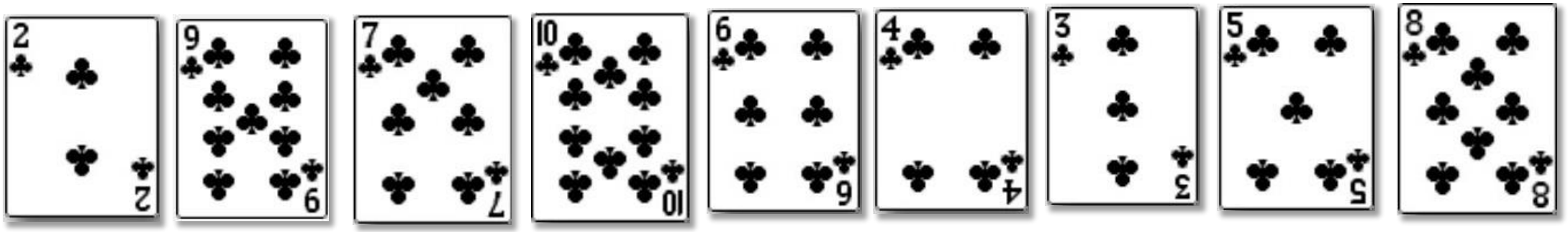
Sắp xếp chèn

❖Java

```
public static void InsertionSort(int A[], int n) {
   int x, j;
   // Phần tử A[0] của mảng xem như đã có thứ tự
   for (int i=1; i<n; i++) {
      x = A[i];
      j = i - 1;
      while (j >= 0 && A[j] > x) {
         A[j+1] = A[j];
         j = j - 1;
      }
      A[j+1] = x; // Chèn x vào dãy
   }
}
```

Sắp xếp chèn

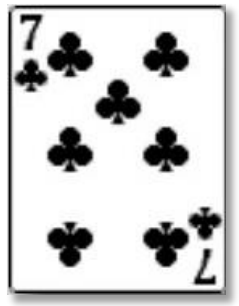
❖**Ví dụ:** Cho các lá bài, hãy sắp xếp chúng theo trật tự tiến lên



Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài

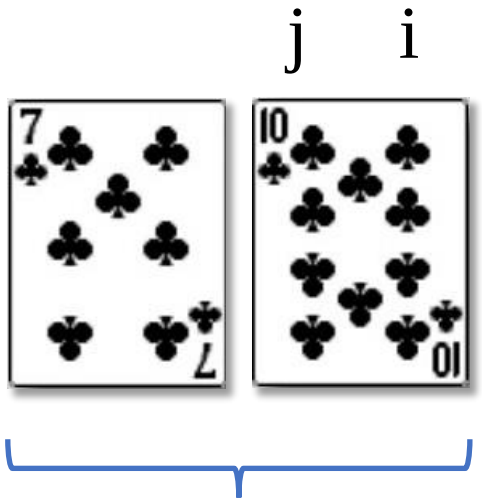
j i



Đã có thứ tự

Sắp xếp chèn

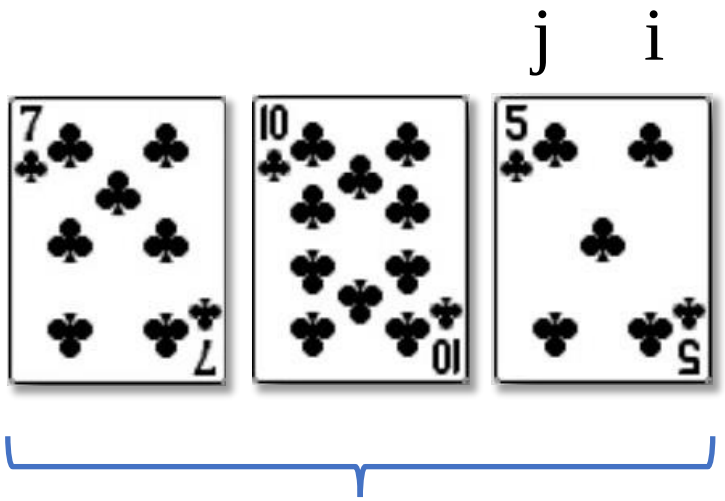
❖**Ví dụ:** sắp xếp các lá bài



Đã có thứ tự

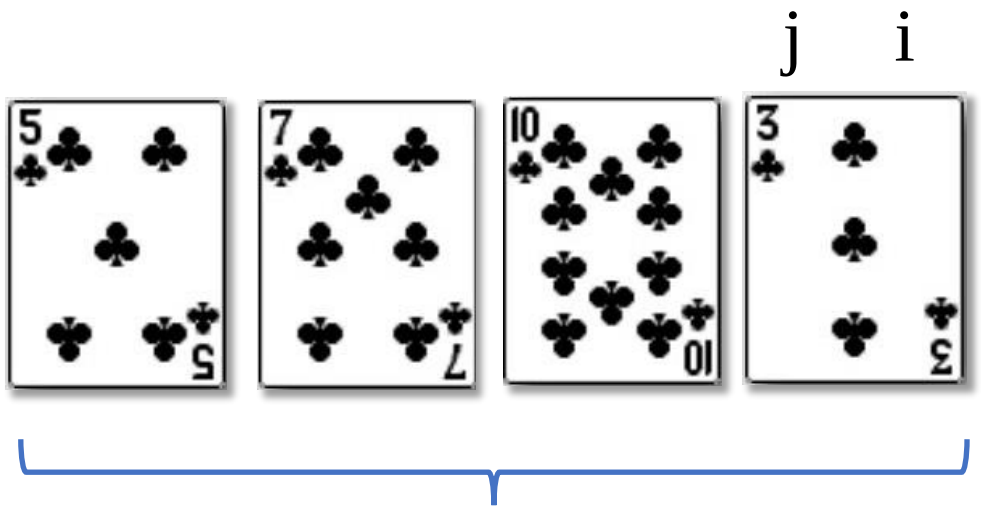
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



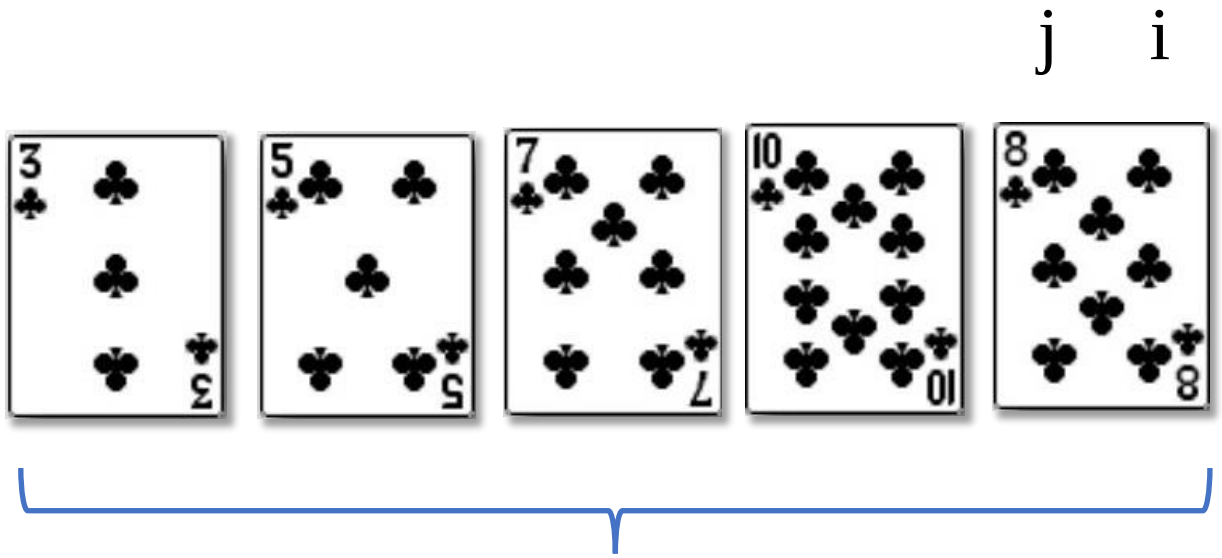
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



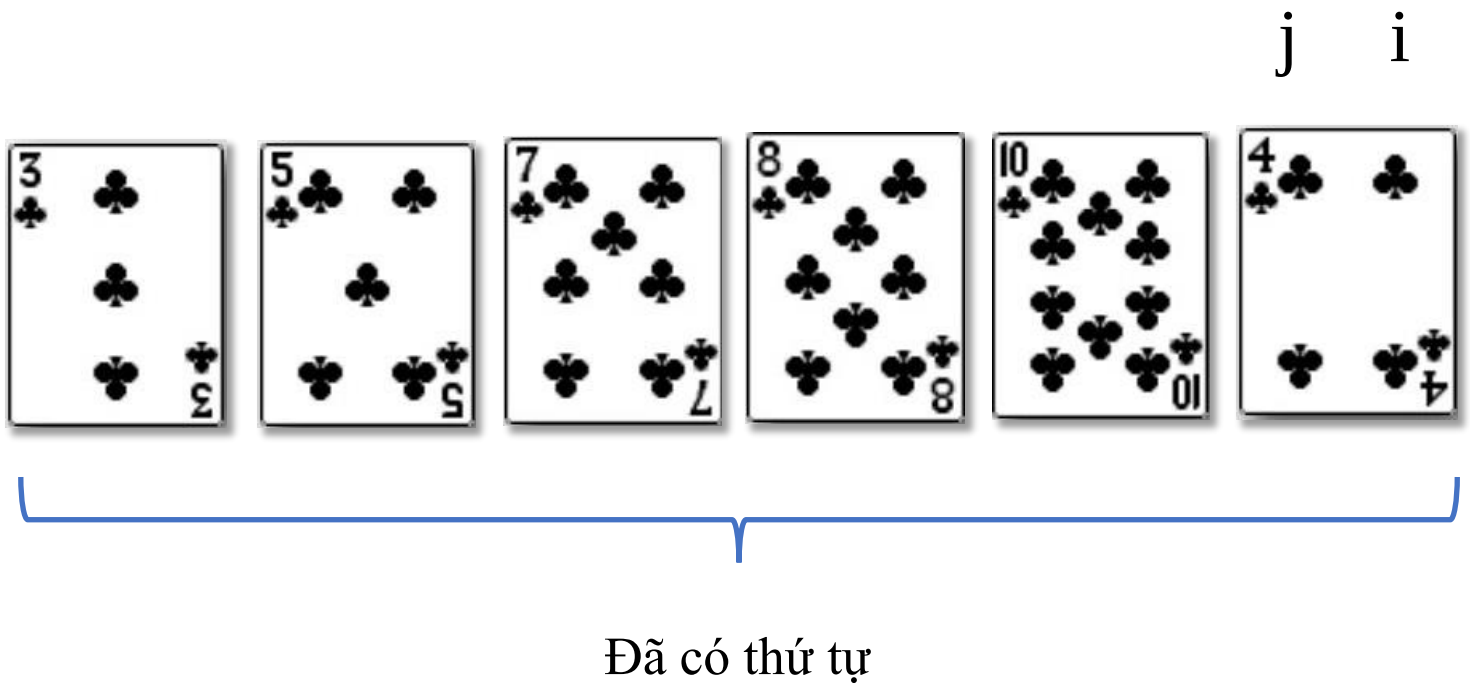
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



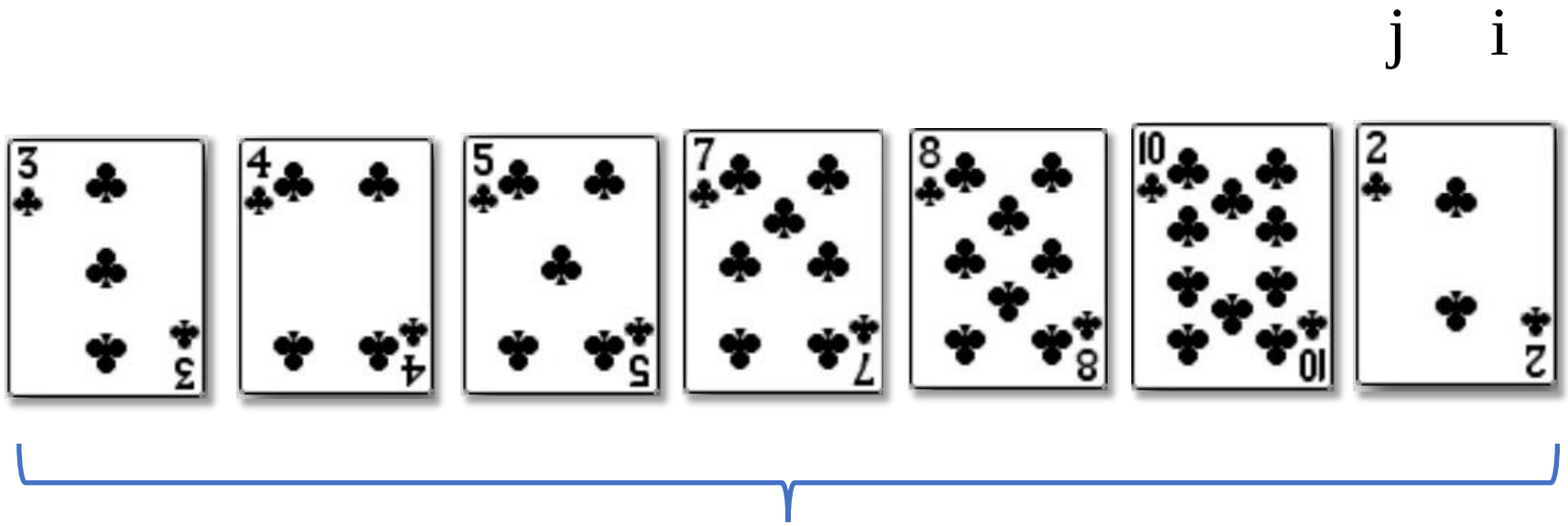
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



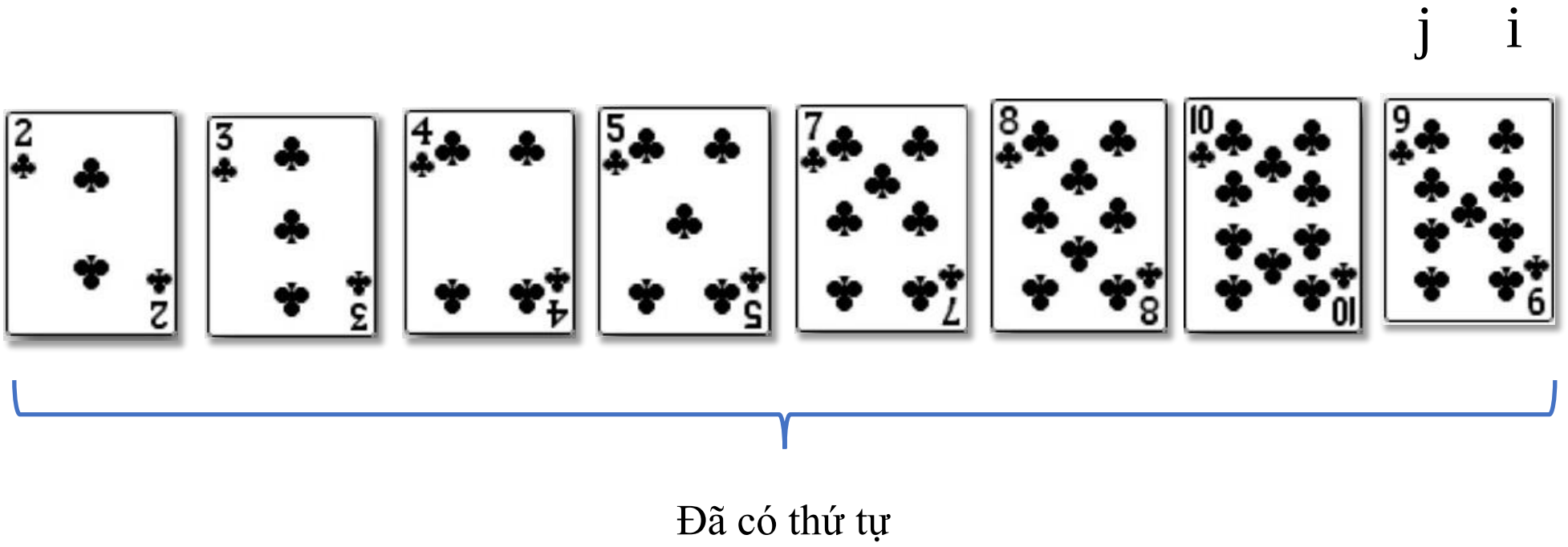
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



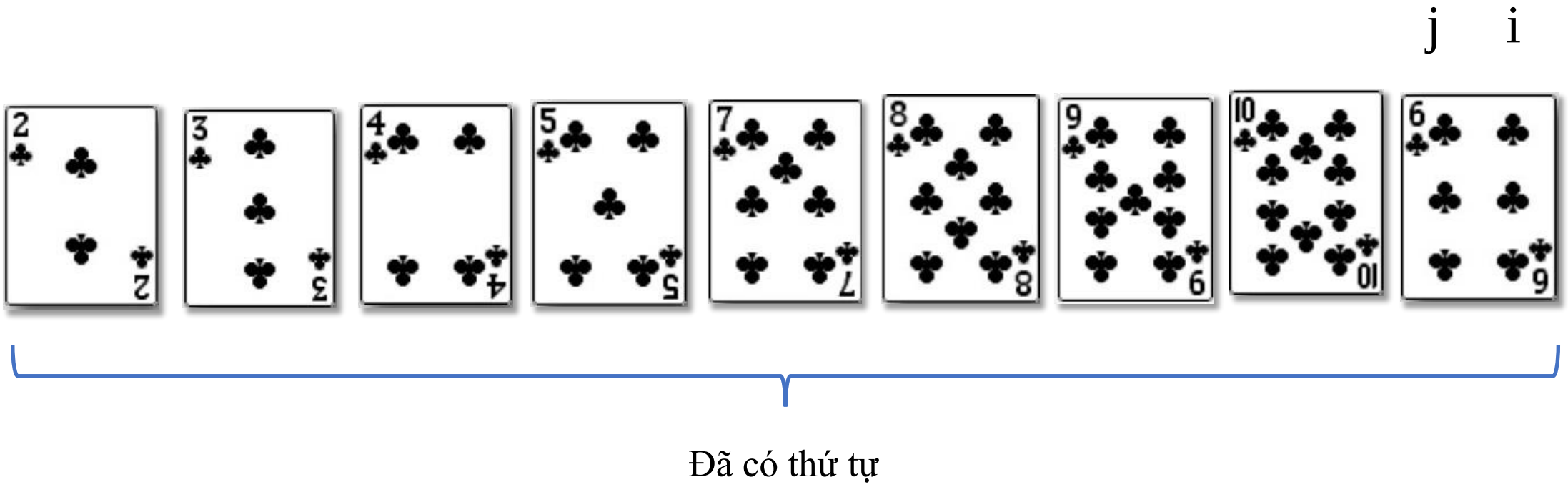
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



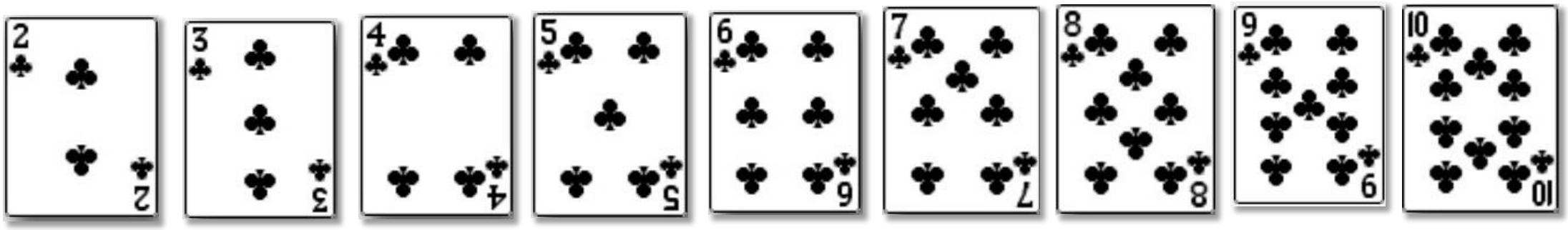
Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



Sắp xếp chèn

❖**Ví dụ:** sắp xếp các lá bài



Sắp xếp chèn

❖Độ phức tạp giải thuật

- Có n-1 bước lặp
- Mỗi bước lặp có 3 phép gán và tối đa là i phép so sánh để dời phần tử A[i] về vị trí phù hợp

$\Rightarrow$ $f(n) = \sum_{i=1}^{n-1}(3 + \sum_{j=i-1}^{0} i) = 3(n-1) + [1 + 2 + 3 + \ldots + (n-2) + (n-1)]$

$$= 3(n-1) + \frac{n(n-1)}{2} = \frac{n^2}{2} + \frac{5n}{2} - 3 \leq \frac{n^2}{2} + \frac{5n^2}{2} = 3n^2$$

➢Nên $f(n) \leq 3.g(n)$ với $g(n) = n^2$.

➢Do đó, thuật toán có độ phức tạp là $O(n^2)$

Sắp xếp chèn

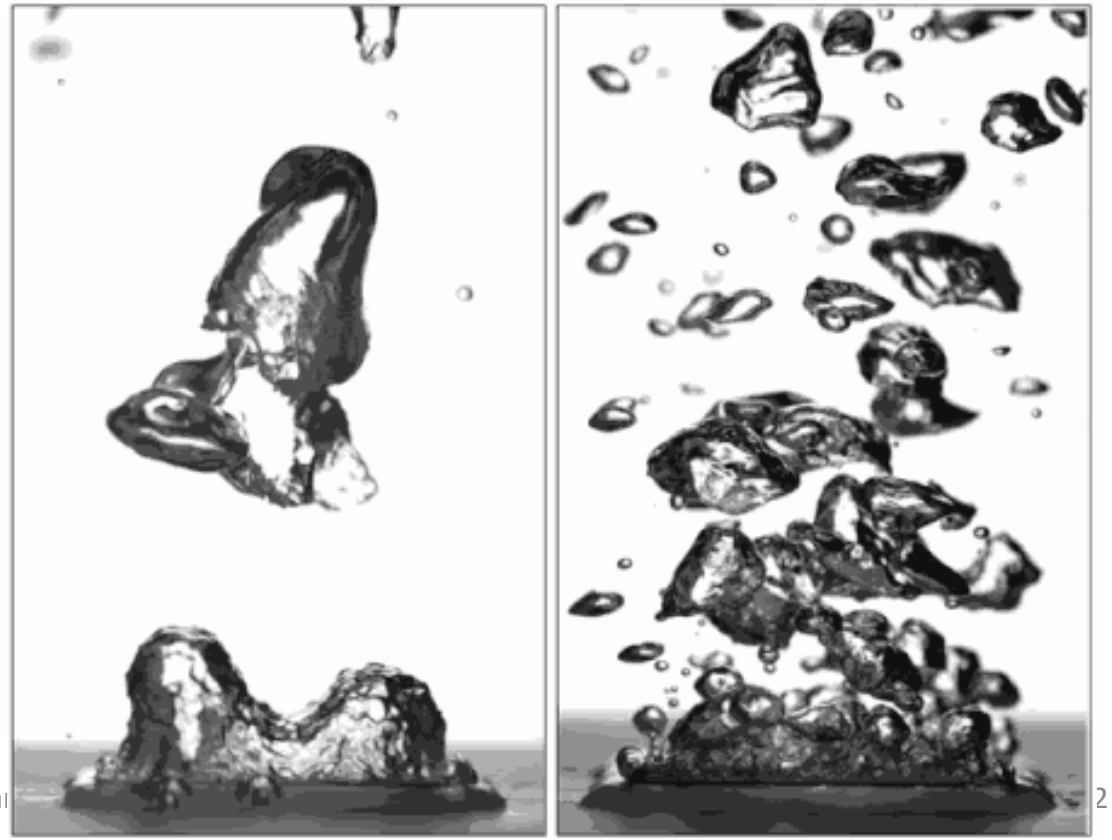
❖Độ phức tạp giải thuật

Trường hợp	Số phép so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{n-1} 1$ = n-1	$\sum_{i=1}^{n-1} 2$ = 2(n-1)
Xấu nhất	$\sum_{i=1}^{n-1}(i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1}(i+1) = \frac{n(n+1)}{2}$ - 1

Sắp xếp nổi bọt
Buble Sort

Sắp xếp nổi bọt

- Buble Sort
- Lấy ý tưởng từ trạng thái nổi bọt của không khí khi đun nước



Sắp xếp nổi bọt

❖Ý tưởng

- Sắp xếp tăng dần
  - ❑Đưa các phần tử có giá trị nhỏ về đầu dãy
  - ❑Đưa các phần tử có giá trị lớn về cuối dãy
- Sắp xếp giảm dần: ngược lại

Sắp xếp nổi bọt

❖Thuật toán

- **Bước 1**: khởi tạo i = 0;
- **Bước 2**: khởi tạo j = n – 1;
- **Bước 3**: Kiểm tra nếu (i < j) thì thực hiện:

  Nếu (A[j-1] > A[j]) thì hoán đổi A[j-1]↔A[j]

  Giảm j = j – 1; Quay về bước 3;
- **Bước 4**: Tăng i = i + 1;

  Nếu i == n - 1 (hết dãy). Dừng

  Ngược lại, quay về bước 2

Sắp xếp nổi bọt

❖Java

```
public static void BubleSort(int A[], int n) {
   int trungGian;
   for (int i=0; i<n; i++)
      for (int j=n-1; j>i; j--)
         if (A[j-1] > A[j]) {
            trungGian = A[j-1];
            A[j-1] = A[j];
            A[j] = trungGian;
         }
}
```

Sắp xếp nổi bọt

❖Java

```
// Phiên bản thường dùng của giải thuật BubleSort
public static void BubleSort(int A[], int n) {
   int trungGian;
   for (int i=0; i<n-1; i++)
      for (int j=i+1; j<n; j++)
         if (A[i] > A[j]) {
            trungGian = A[i];
            A[i] = A[j];
            A[j] = trungGian;
         }
}
```

Sắp xếp nổi bọt

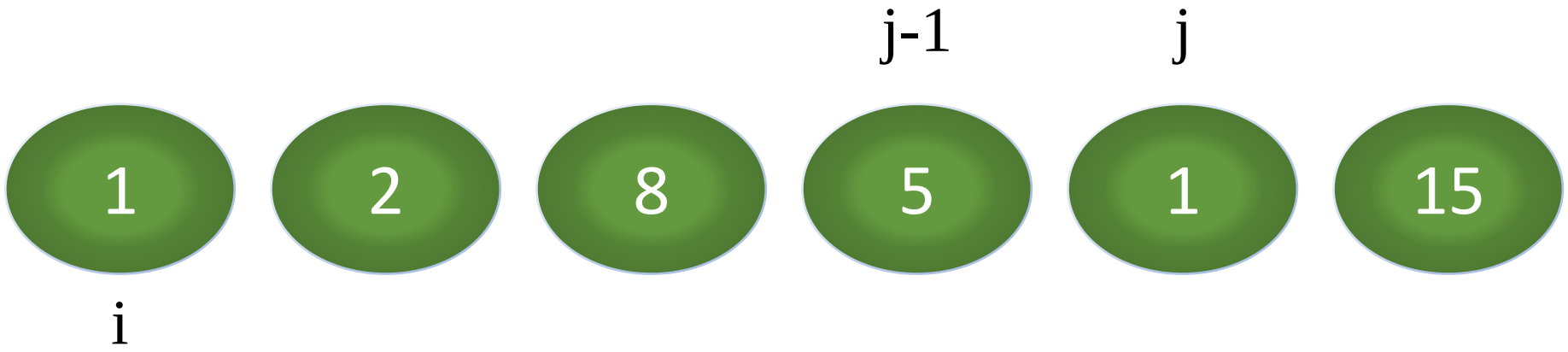
❖**Ví dụ:** sắp xếp dãy số sau theo thứ tự tăng dần

Sắp xếp nổi bọt

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

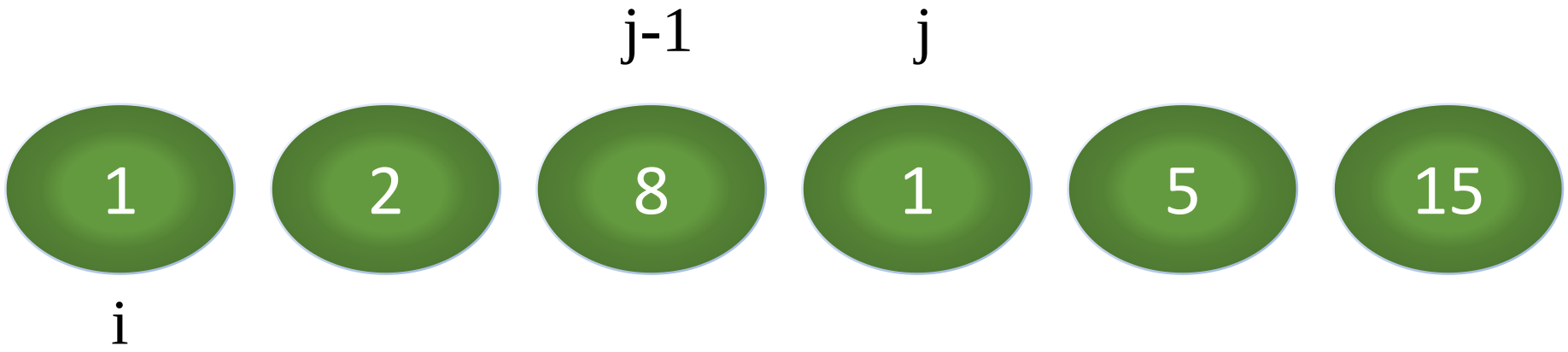
Sắp xếp nổi bọt

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



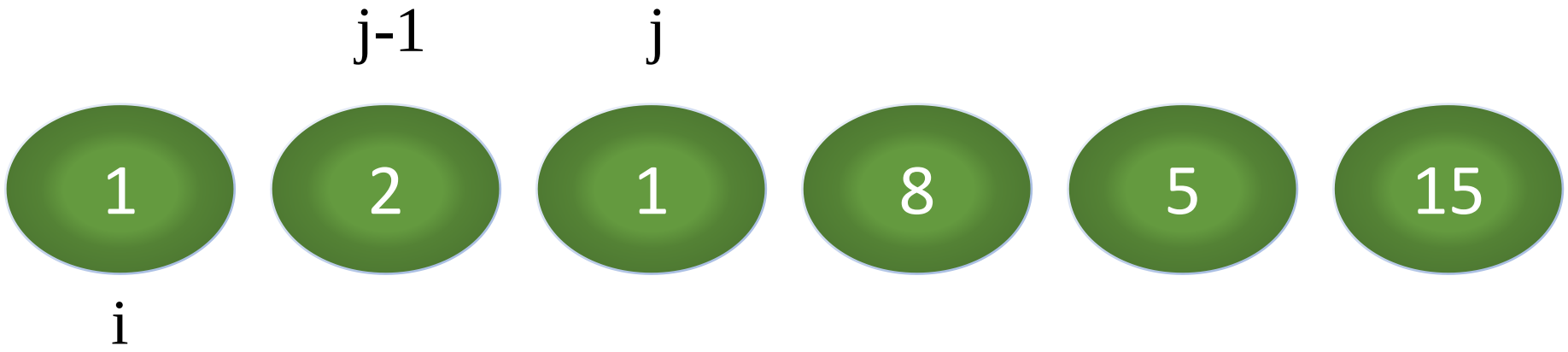
Sắp xếp nổi bọt

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



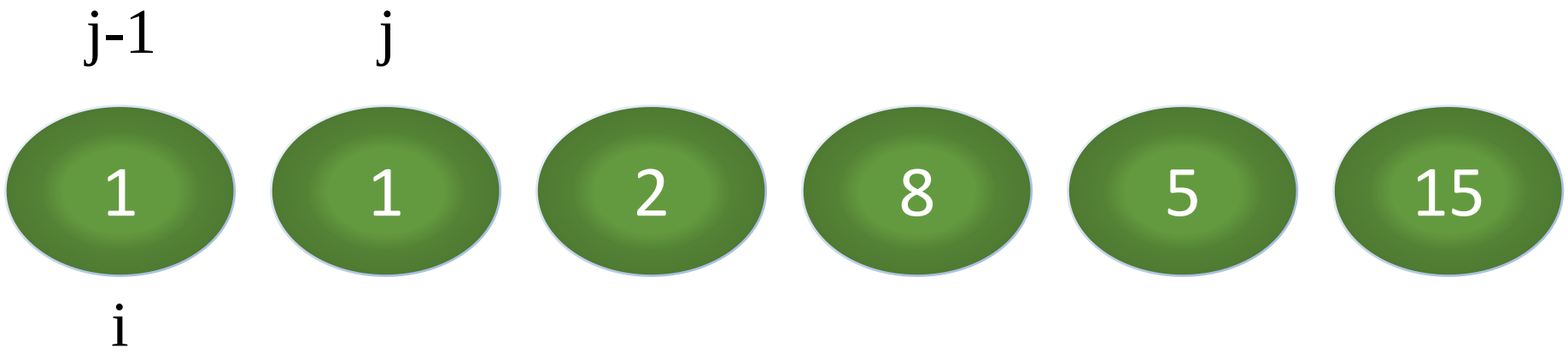
Sắp xếp nổi bọt

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



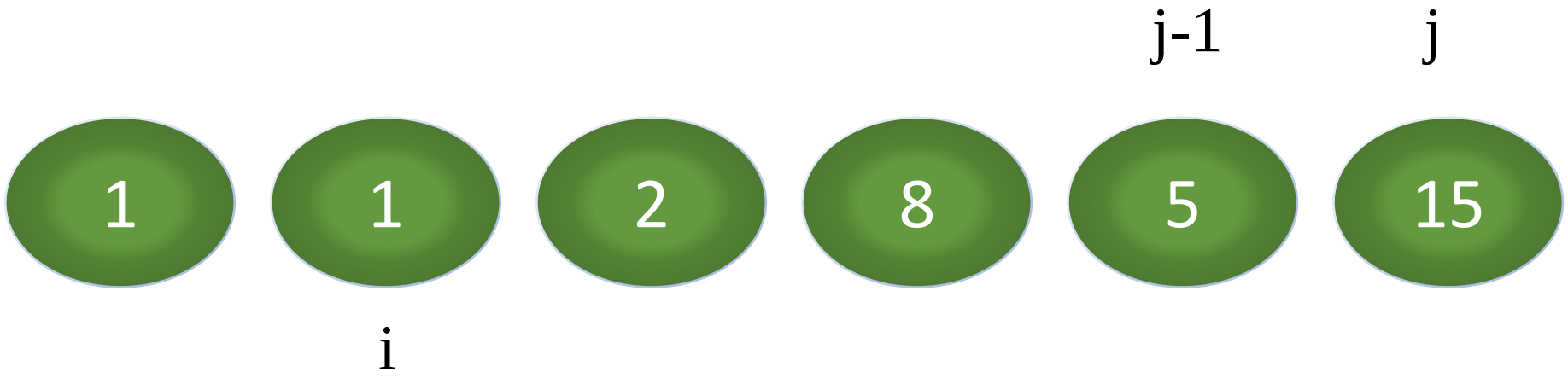
Sắp xếp nổi bọt

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



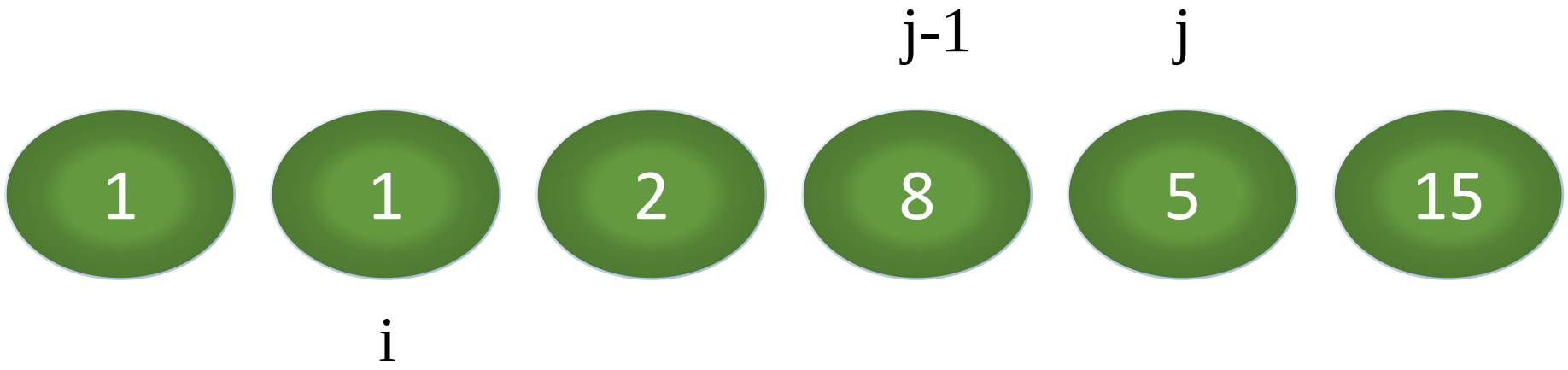
Sắp xếp nổi bọt

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



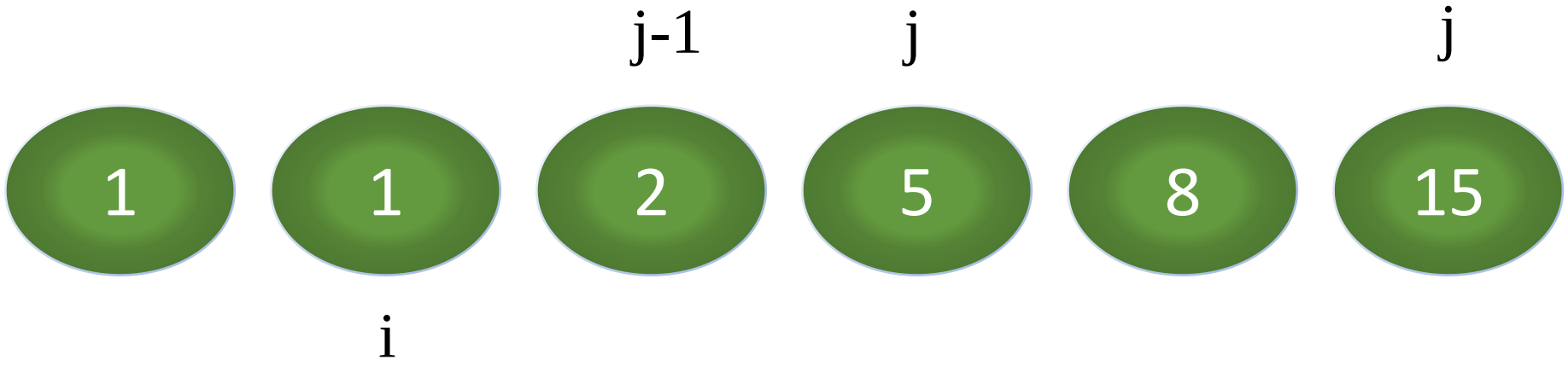
Sắp xếp nổi bọt

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



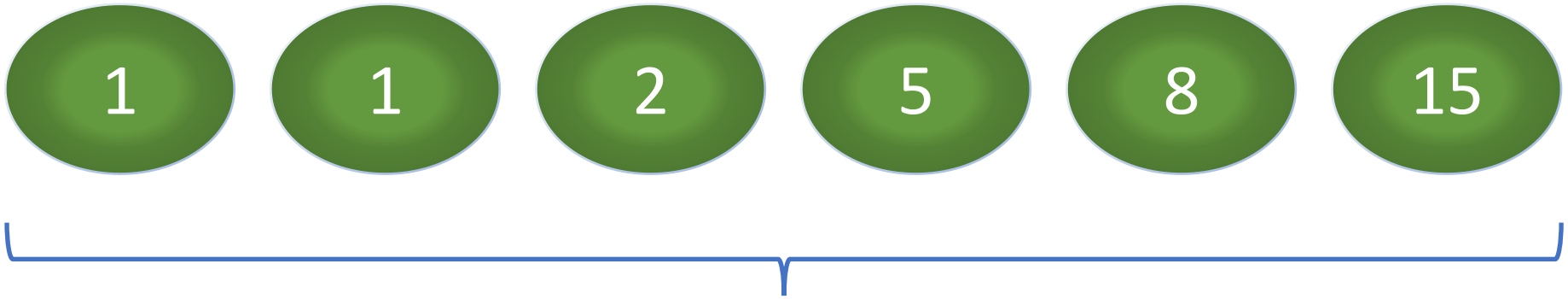
Sắp xếp nổi bọt

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



Sắp xếp nổi bọt

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



Dãy đã được sắp xếp

Sắp xếp nổi bọt

❖Độ phức tạp giải thuật

- Có n bước lặp
- Mỗi bước lặp có tối đa (n-i-1) phép so sánh để dời phần tử A[j] về vị trí phù hợp

$\Rightarrow$ $f(n) = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} 1 = (n-2) + (n-3) + \ldots + 1 = \frac{(n-2)(n-1)}{2}$

$$= \frac{n^2 - 3n + 2}{2} \leq \frac{n^2 + 2}{2} \leq \frac{n^2 + 2n^2}{2} = \frac{3}{2}n^2$$

➢ Nên $f(n) \leq \frac{3}{2}g(n)$ với $g(n) = n^2$. Do đó, thuật toán có độ phức tạp là **$O(n^2)$**

Sắp xếp nổi bọt

❖Độ phức tạp giải thuật

Trường hợp	Số phép so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{n-1}(n-i+1) = \frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1}(n-i+1) = \frac{n(n-1)}{2}$

Sắp xếp trộn
Shaker Sort

Sắp xếp trộn

❖**Shaker Sort / Cocktail Sort**

❖**Ra đời từ việc cải tiến của giải thuật Buble Sort**

❖**Ý tưởng**

- Mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau
  - ❑Lượt đi: đẩy phần tử nhỏ về đầu mảng
  - ❑Lượt về: đẩy phần tử lớn về cuối mảng
- Ghi nhận lại những đoạn đã sắp xếp để loại bỏ các phép so sánh thừa

Sắp xếp trộn

❖Thuật toán

- **Bước 1**: khởi tạo left = 0; right = n - 1; k = n - 1
- **Bước 2**: Đặt j = right; // Đẩy phần tử nhỏ về đầu mảng

Nếu (j > left) thì lặp:

NếuA[j-1] > A[j] thì

Hoán đổi A[j-1]↔A[j];

k = j; // Lưu lại nơi xảy ra hoán vị

j = j – 1;

left = k; // Loại các phần tử đã có thứ tự ở đầu dãy

Sắp xếp trộn

❖Thuật toán (T.T)

- **Bước 3**: Đặt j = left; // Đẩy phần tử lớn về cuối mảng

  Nếu (j < right) thì lặp:

  Nếu A[j] > A[j+1] thì

  Hoán đổi A[j]↔A[j+1];

  k = j; // Lưu lại nơi xảy ra hoán vị

  j = j + 1;

  right = k; // Loại các phần tử đã có thứ tự ở cuối dãy

- **Bước 4**: Nếu (left < right): Quay lại bước 2

Sắp xếp trộn

❖Java

```
public static void ShakerSort(int A[], int n) {
    int left = 0, right = n - 1, k = n - 1;
    while (left < right) {
        for (int i=left; i<right; i++) {
            if (A[i] > A[i+1]) {
                Swap(A[i], A[i+1]); // Hoán vị
                k = i;
            }
        }
        right = k;

        for (int i=right; i>left; i--) {
            if (A[i] < A[i-1]) {
                Swap(A[i], A[i-1]); // Hoán vị
                k = i;
            }
        }
        left = k;
    }
}
```

Sắp xếp chèn

❖Độ phức tạp giải thuật

- Từ left tới right có n bước lặp
- Mỗi bước lặp có 2 đoạn chương trình con chạy song song, mỗi đoạn lại có n bước lặp
- Vậy theo nguyên lý cộng thì $f(n) = n * Max(n, n) = n * n = n^2$
- Nên $f(n) \leq g(n)$ với $g(n) = n^2$. Do đó, thuật toán có độ phức tạp là **$O(n^2)$**

Kết luận

- Tất cả các thuật toán trên đều được thiết kế dựa trên nguyên lý duyệt tuần tự từng phần tử của dãy để so sánh
  - Bottom-up design
  - Vét cạn
- Tất cả các thuật toán này đều có độ phức tạp là $O(n^2)$ ⇒ phù hợp với các bài toán có kích thước dữ liệu cỡ trung bình
- Các thuật toán này trở nên phổ biến nhờ sự giản đơn trong ý tưởng và cách cài đặt (dễ lập trình)

Hỏi & Đáp

"Formal education will make you a living;
self-education will make you a fortune"

Bài học kế tiếp

Bài toán sắp xếp

- Các giải thuật sắp xếp nâng cao
  - Shell Sort
  - Quick Sort
  - Heap Sort
  - Merge Sort

Tài liệu tham khảo

➢Tài liệu môn học

- [1] Michael T. Goodrich, Roberto Tamassia, Data Structures & Algorithms in Java (6^{th} Edition)
- [2] Trần Hạnh Nhi, Dương Anh Đức, Cấu trúc dữ liệu & giải thuật, Khoa CNTT, trường ĐH KHTN ĐHQG TpHCM

➢Tài liệu tham khảo thêm

- [3] Thomas H. Cormen et al., 2009, Introduction to Algorithms, 3^{rd} Edition, ebook.
- [4] Hoàng M. L., 2002, Cấu trúc dữ liệu và giải thuật, ĐHSP Hà Nội.