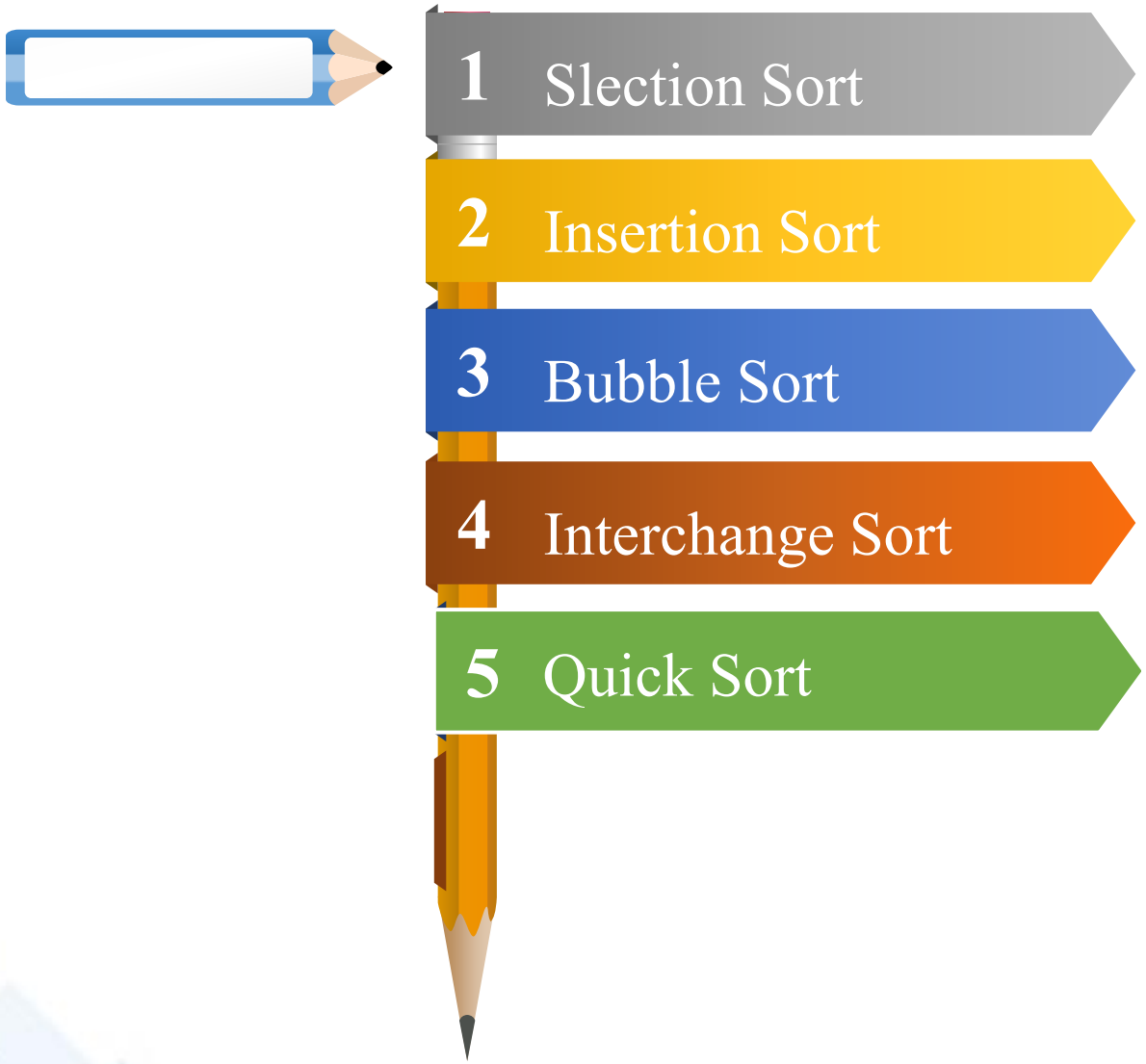
HCMUTE

CHƯƠNG 3 : MỘT SỐ THUẬT TOÁN SẮP XẾP

Nội dung


1 Slection Sort
2 Insertion Sort
3 Bubble Sort
4 Interchange Sort
5 Quick Sort

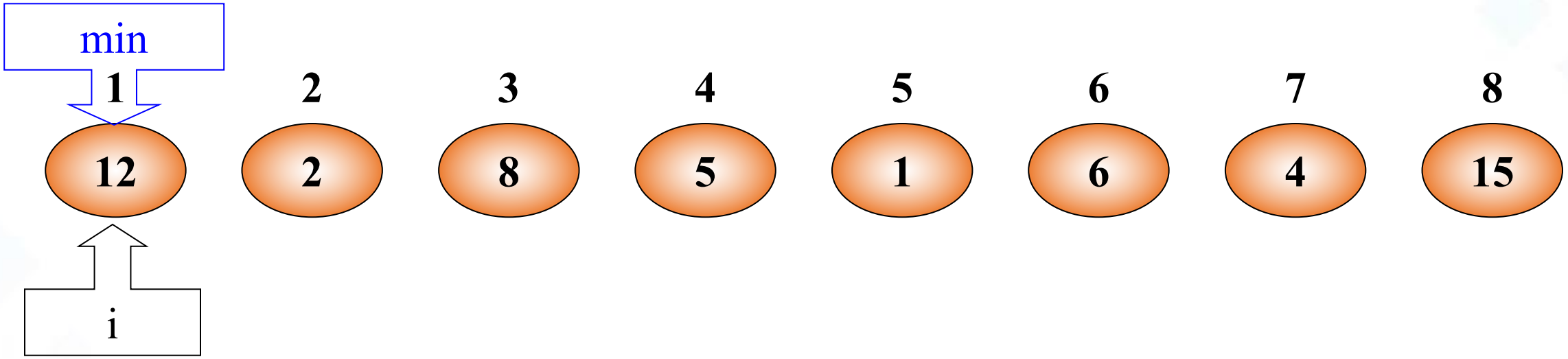
HCMUTE

1. Selection Sort – Ý tưởng

- **Chọn trực tiếp** (Selection Sort) là một cách sắp xếp gần với tự nhiên khi cho một dãy bất kỳ.
  - ✓ Đầu tiên ta **lựa chọn** phần tử ***nhỏ nhất*** trong dãy hiện hành.
  - ✓ Sau đó hoán đổi phần tử nhỏ nhất với phần tử đầu dãy sau đó không xét đền nó nữa trong lần thực hiện tiếp theo.
  - ✓ Lặp lại thao tác trên cho tới khi danh sách hiện hành chỉ còn 1 phần tử.

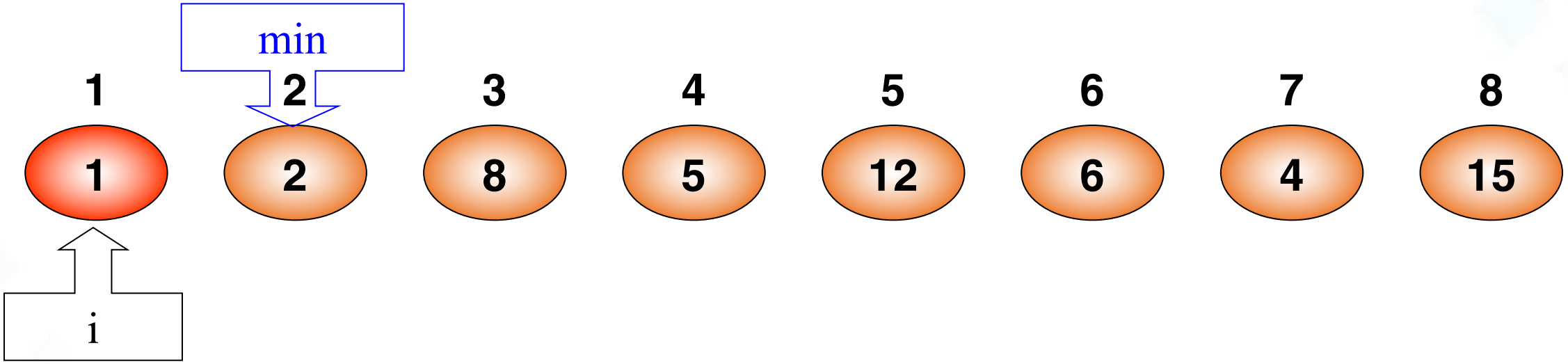
1. Selection sort

Find MinPos(1, 8) **Swap(a_i, a_{min})**



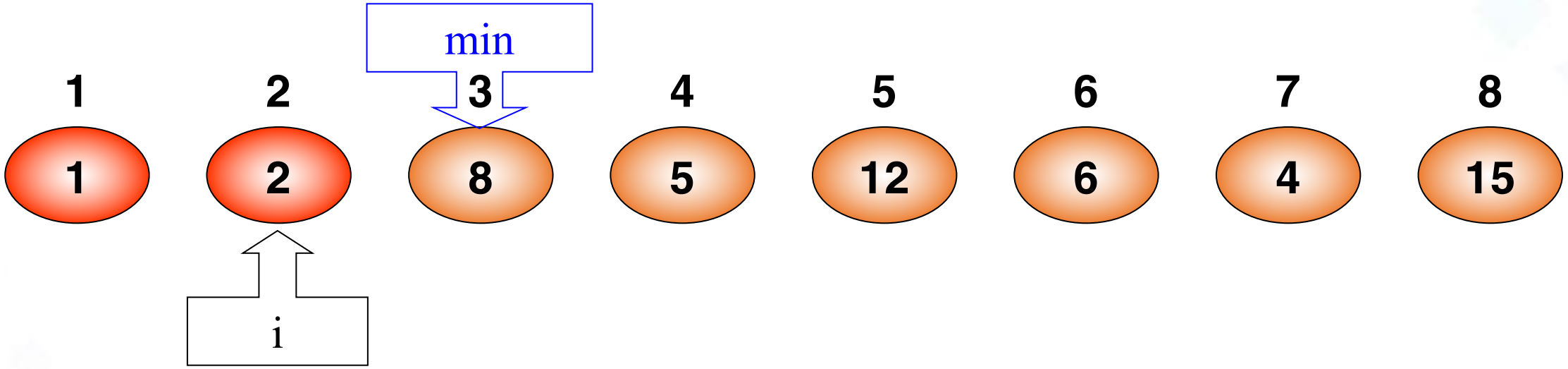
1. Selection sort

Find MinPos(2, 8) **Swap(a_i, a_{min})**



1. Selection sort

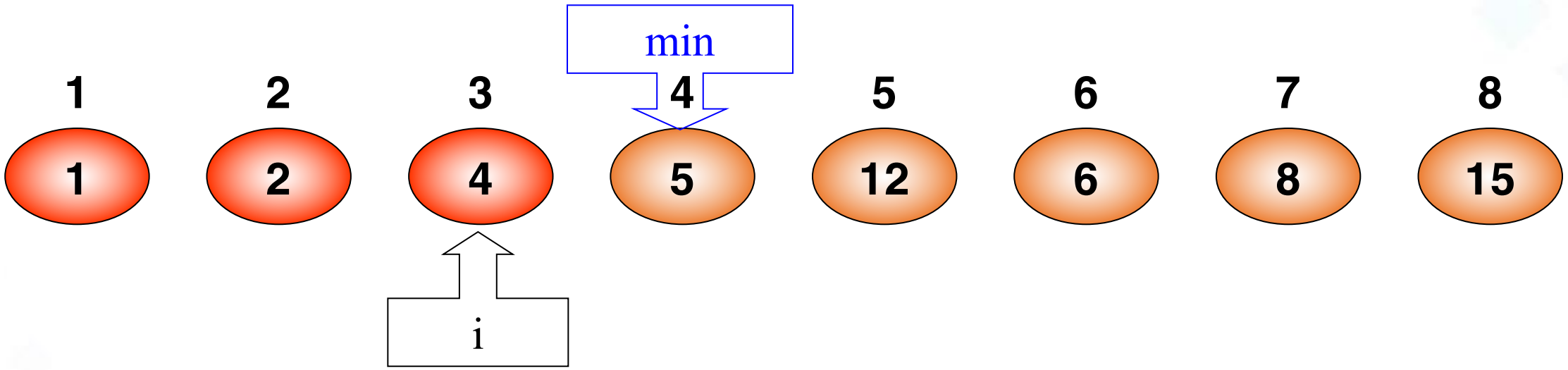
Find MinPos(3, 8) **Swap(a_i, a_{min})**



1. Selection sort

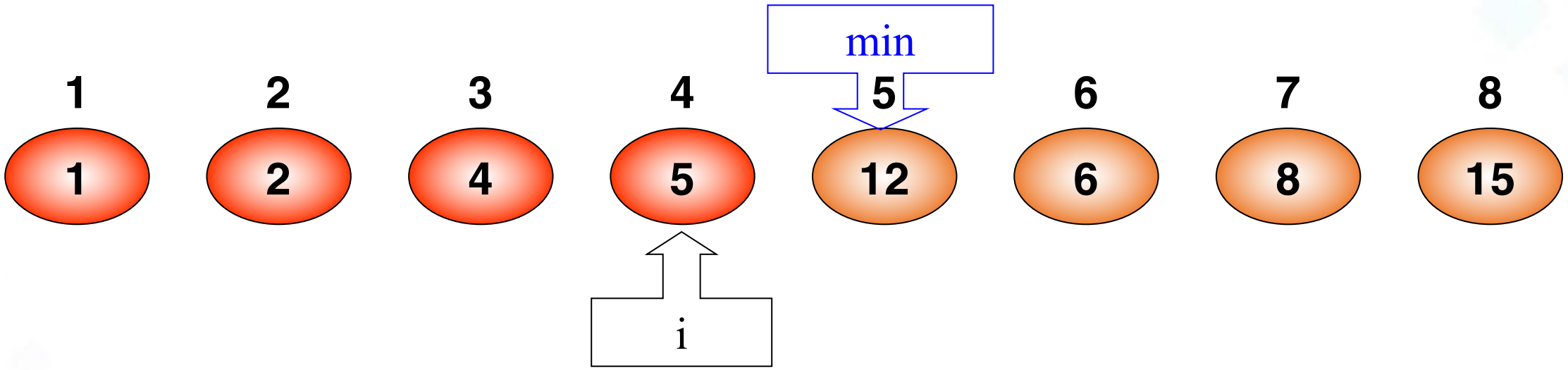
Find MinPos(4, 8)

Swap(a_i, a_{min})



1. Selection sort

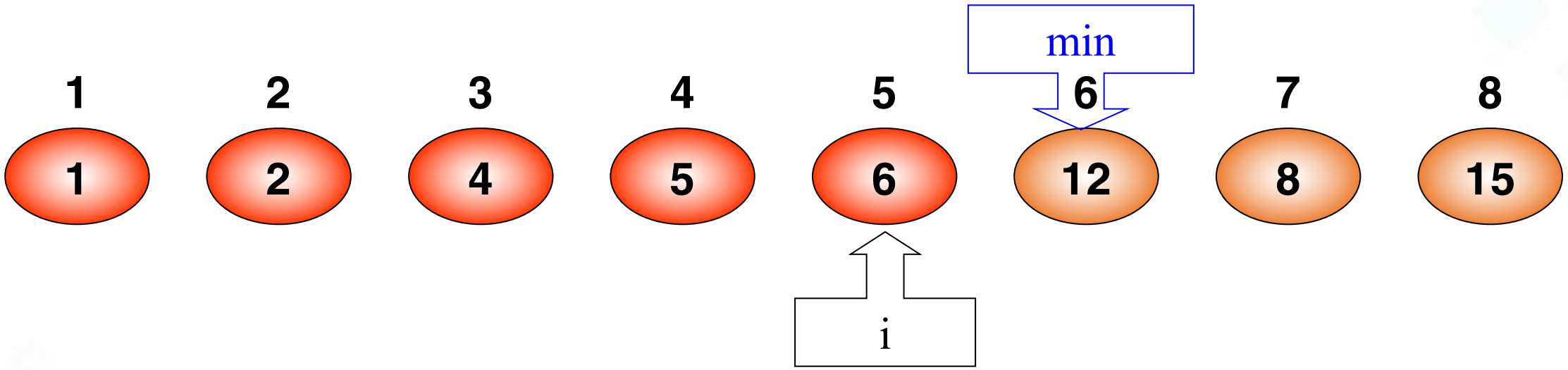
Find MinPos(5, 8) **Swap(a_i, a_{min})**



1. Selection sort

Find MinPos(6, 8)

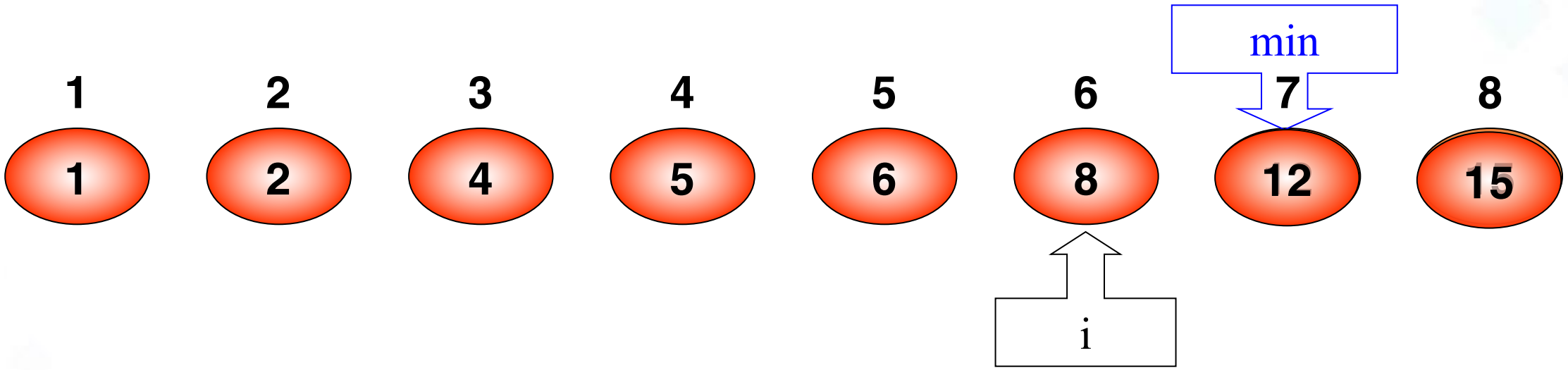
Swap(a_i, a_{min})



1. Selection sort

Find MinPos(7, 8)

Swap(a_i, a_{min})



1. Selection sort - Quá trình phát sinh code

Dựa trên mô phỏng: Yêu cầu 2 sinh viên lên tái hiện thành những đoạn code

1. Selection sort – Cài đặt

```
static void SelectionSort(int[] arrInt)
{
    int min;
    for (int i = 0; i < arrInt.Length - 1; i++)
    {
        min = i;
        for (int j = i + 1; j < arrInt.Length; j++)
        {
            if (arrInt[j] < arrInt[min])
                min = j; //lưu vị trí phần tử nhỏ nhất
        }
        //hoán vị arrInt[i] và arrInt[min]
        Swap(arrInt[min], arrInt[i]);
    }

}
```

1. Selection sort - Đánh giá giải thuật

Độ phức tạp

Trường hợp tốt nhất: O (n^2)

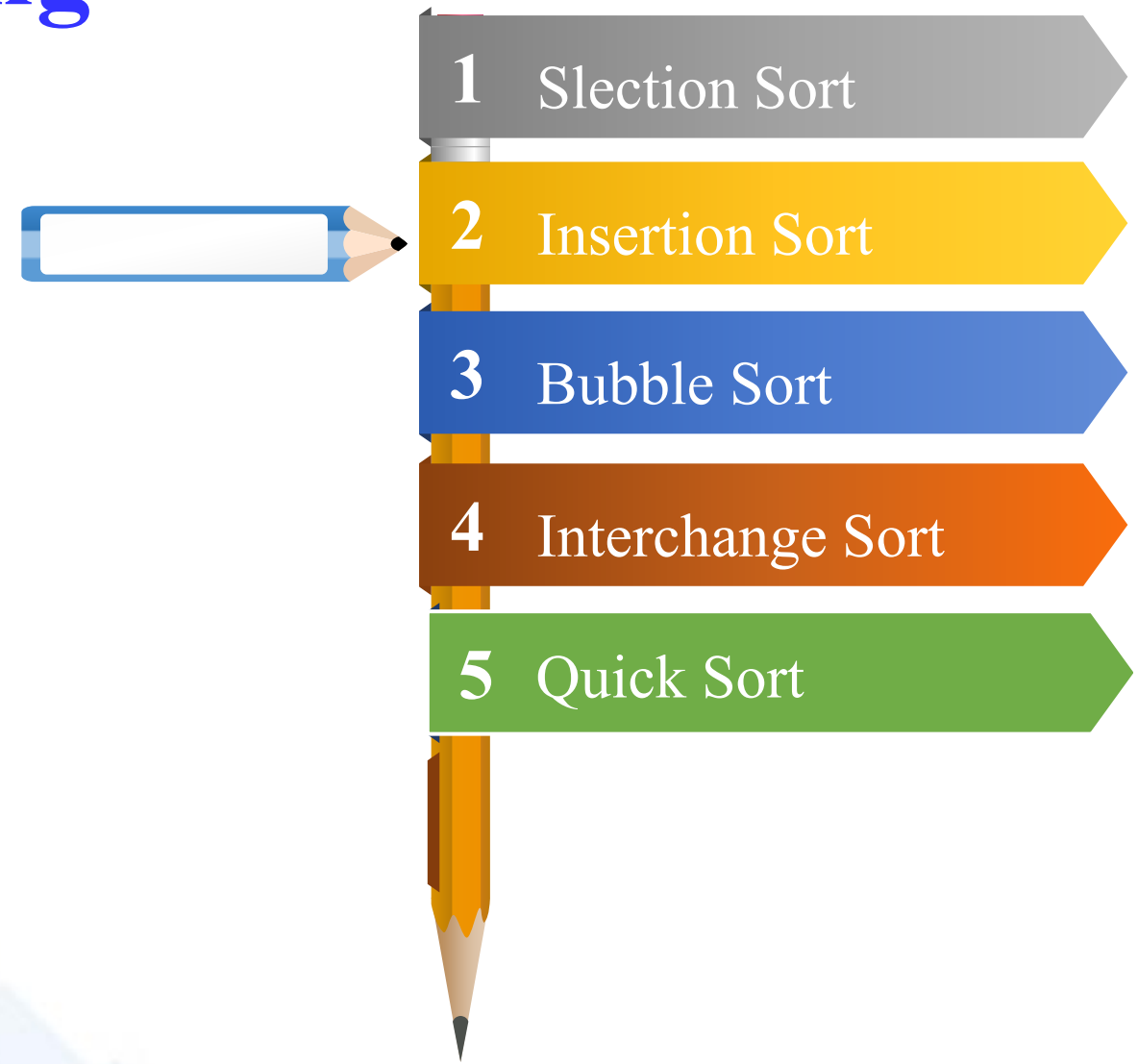
Trường hợp xuất nhất: O (n^2)

Trường hợp trung bình O (n^2)

1. Selection sort - Sắp xếp trên dữ liệu có cấu trúc

- Code sẽ thay đổi như thế nào?
- Có nhận xét gì kiểu dữ liệu của khóa tìm kiếm và kiểu dữ liệu của danh sách?

Nội dung


1 Slection Sort
2 Insertion Sort
3 Bubble Sort
4 Interchange Sort
5 Quick Sort

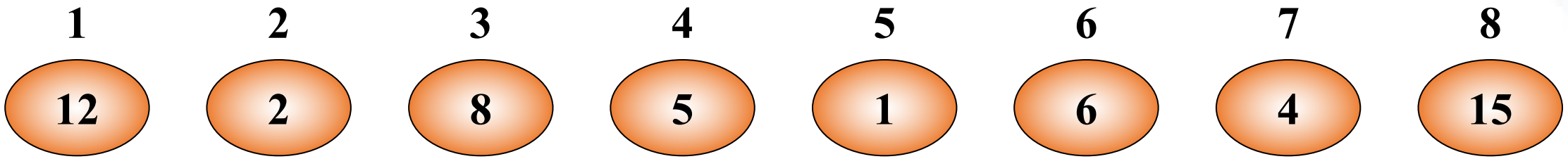
2. Insertion Sort

Hãy cho biết phương pháp sắp bài

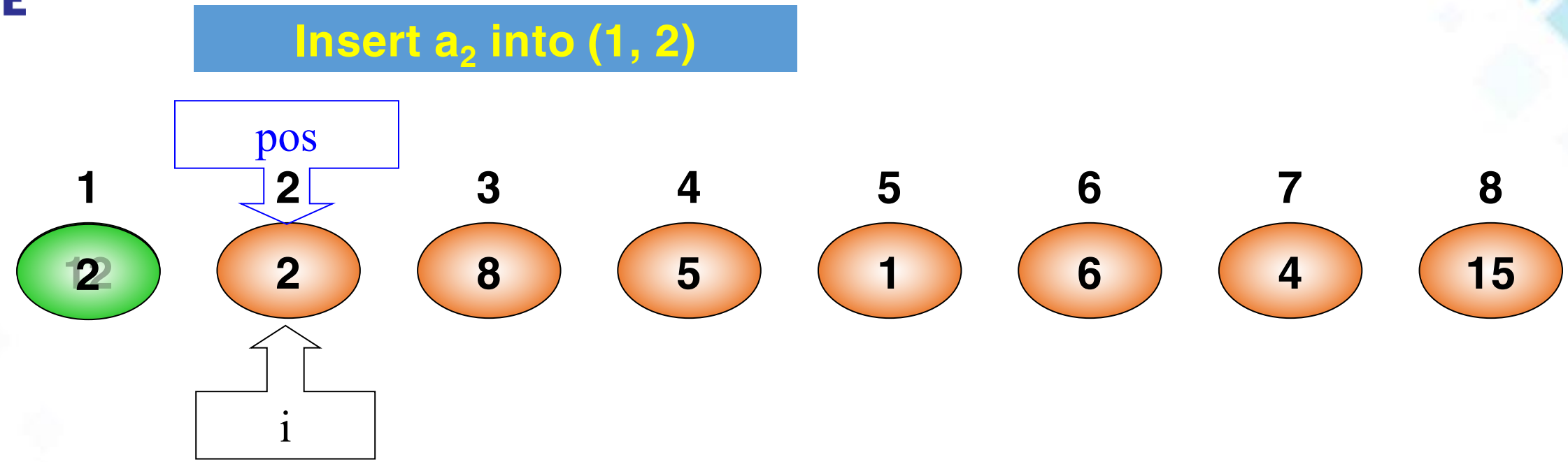
2. Insertion Sort - Ý tưởng

- **Sắp xếp chèn** (Insertion Sort) được thực hiện dựa trên ý tưởng **chèn** từng phần tử vào đoạn đã có thứ tự. Với giả sử ban đầu phần tử đầu tiên là đoạn ban đầu có 1 phần tử, đã có thứ tự. Sau đó mỗi phần tử còn lại sẽ phải tìm vị trí thích hợp để chèn vào sao cho sau khi chèn danh sách vẫn giữ được thứ tự. Thuật toán kết thúc khi tất cả phần tử của danh sách được chèn hết.

2. Insertion Sort



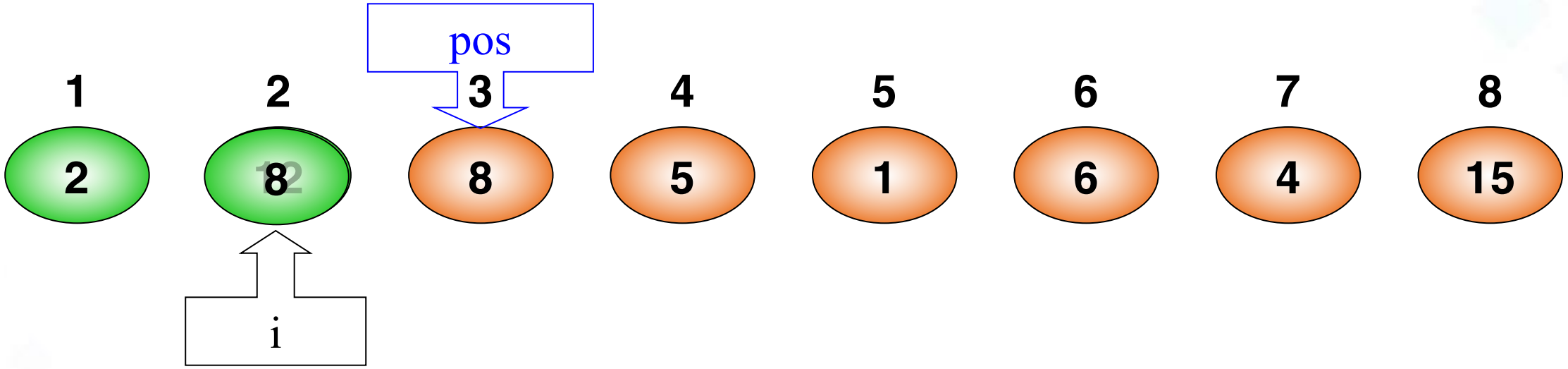
2. Insertion Sort



x

2. Insertion Sort

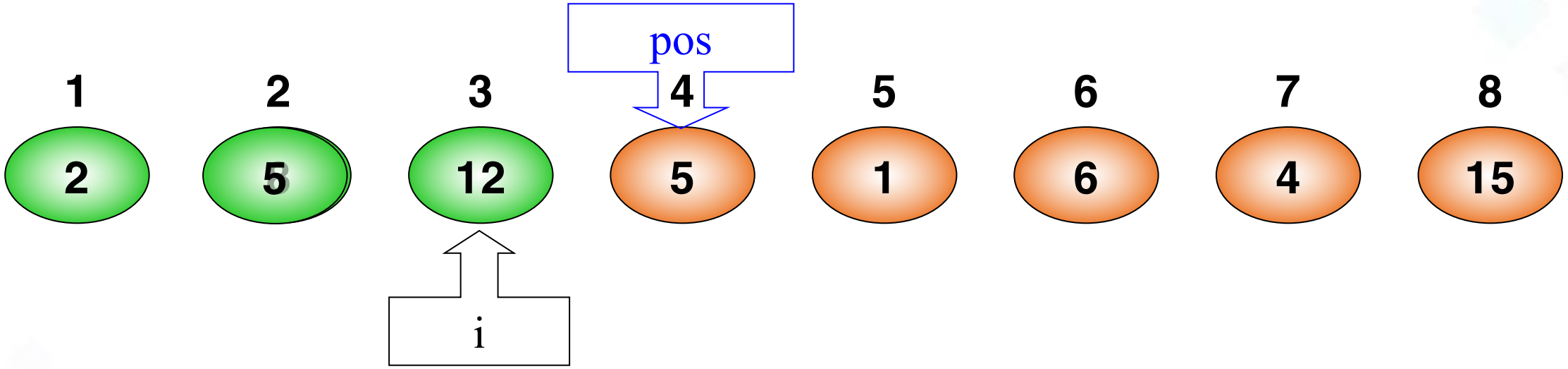
Insert a_3 into (1, 3)



x

2. Insertion Sort

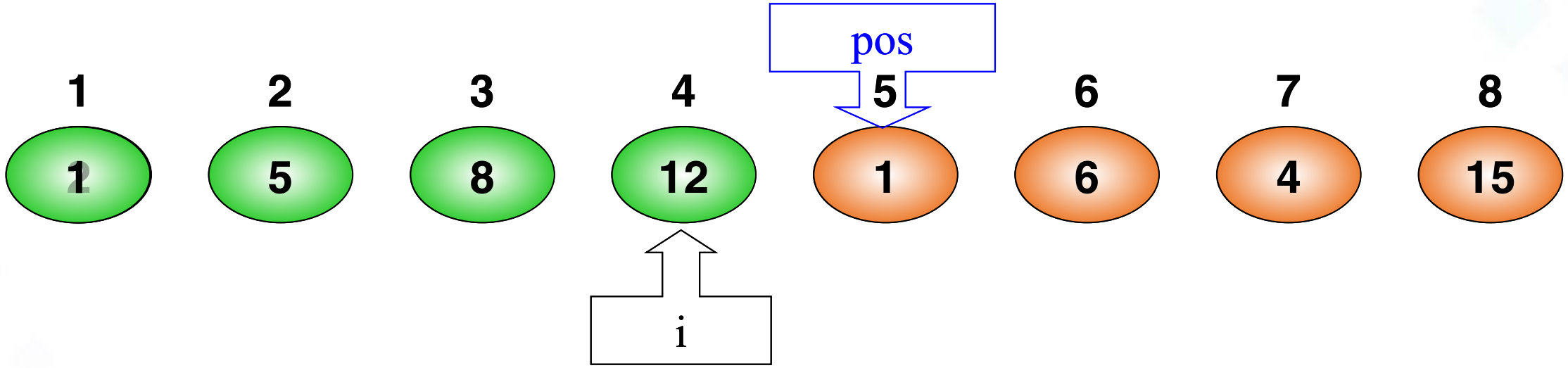
Insert a_4 into (1, 4)



x

2. Insertion Sort

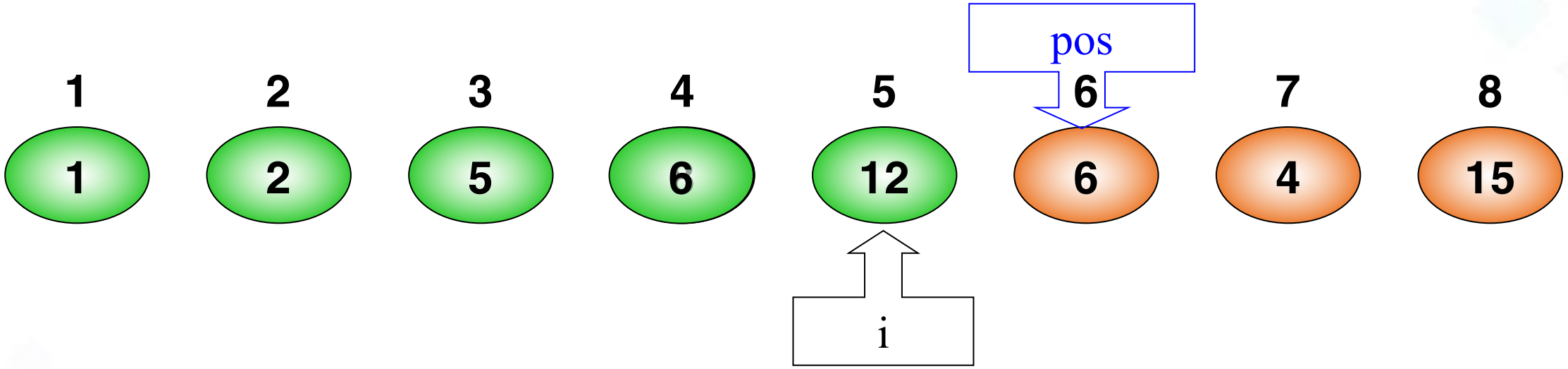
Insert a_5 into (1, 5)



x

2. Insertion Sort

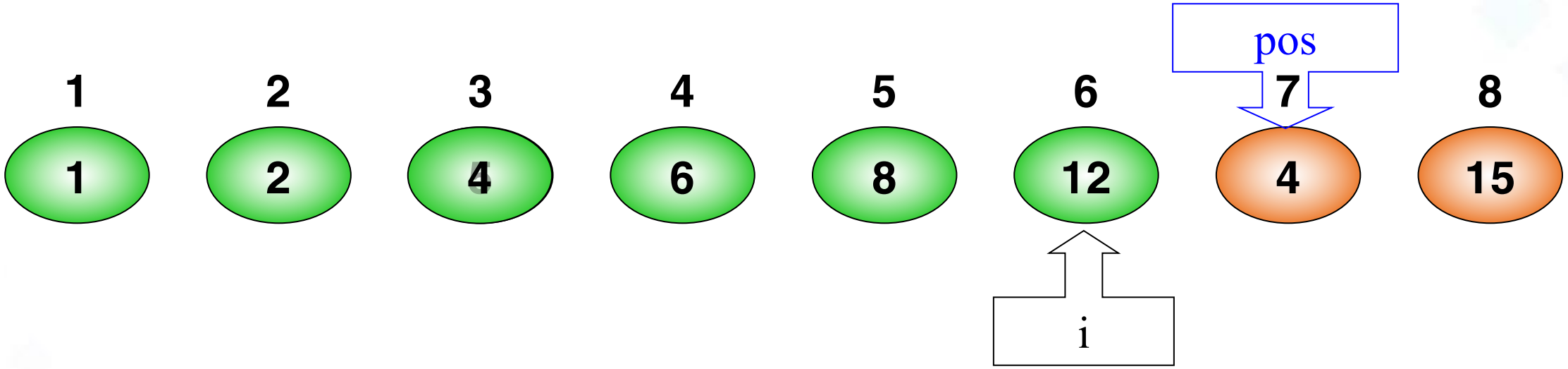
Insert a_6 into (1, 6)



x

2. Insertion Sort

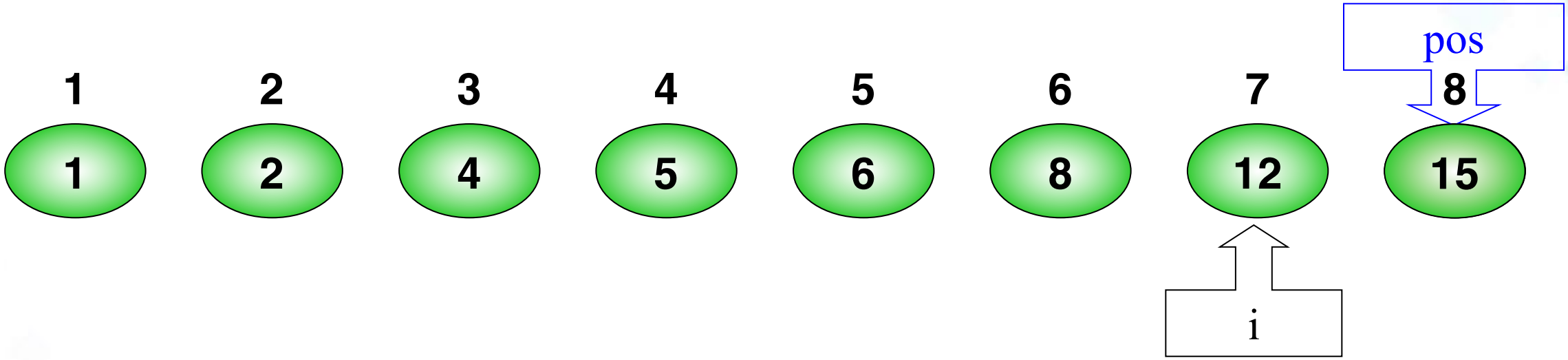
Insert a_7 into (1, 7)



x

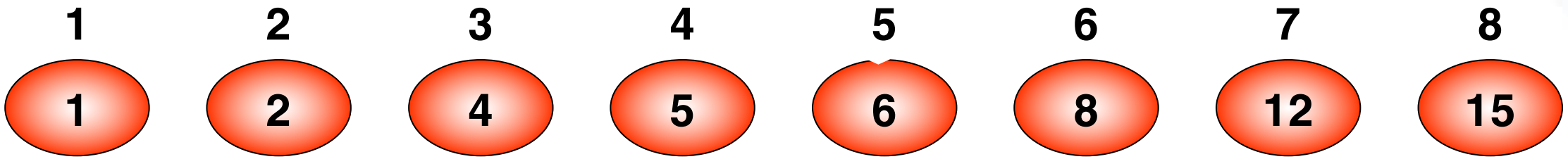
2. Insertion Sort

Insert a_8 into (1, 8)



x

2. Insertion Sort



2. Insertion Sort - Quá trình phát sinh code

Dựa trên mô phỏng: Yêu cầu 2 sinh viên lên tái hiện thành những đoạn code

2. Insertion sort – Cài đặt

```
static void InsertionSort(int[] arrInt)
{
    int i, pos;
    int v; //lưu giá trị arrInt[i] tránh bị ghi đè khi dời chỗ
    for (i = 1; i < arrInt.Length; i++)
    {
        v = arrInt[i];
        pos = i - 1;//tìm vị trí chèn v
        while (pos >= 0 && v < arrInt[pos])
        {
            arrInt[pos + 1] = arrInt[pos];
            pos--;
        }
        arrInt[pos + 1] = v;//chèn v vào dãy
    }
}
```

2. Insertion sort - Đánh giá giải thuật

Độ phức tạp

Trường hợp xấu nhất xảy ra khi mỗi i phải chuyển tất cả phần tử từ vị trí 0 tới i – 1 (tức khi arrInt[i] nhỏ hơn tất cả phần tử của dãy đã có thứ tự đó). $T(n) = O(n^2)$

Trường hợp tốt nhất: $T(n) = O(n)$

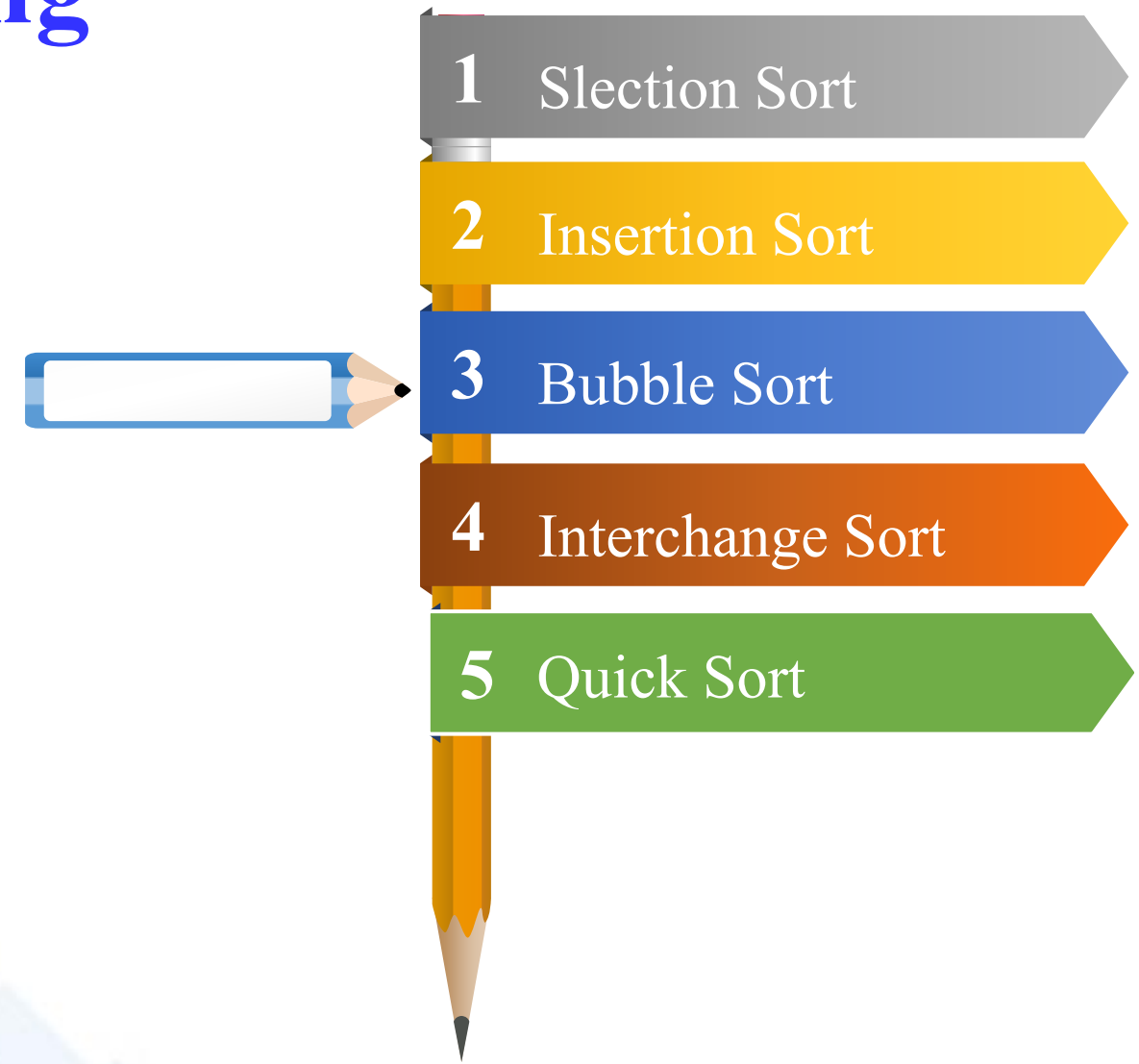
Trường hợp trung bình: $T(n) = O(n^2)$

2. Insertion Sort - Sắp xếp trên dữ liệu có cấu trúc

- Code sẽ thay đổi như thế nào?
- Có nhận xét gì kiểu dữ liệu của khóa tìm kiếm và kiểu dữ liệu của danh sách?

HCMUTE

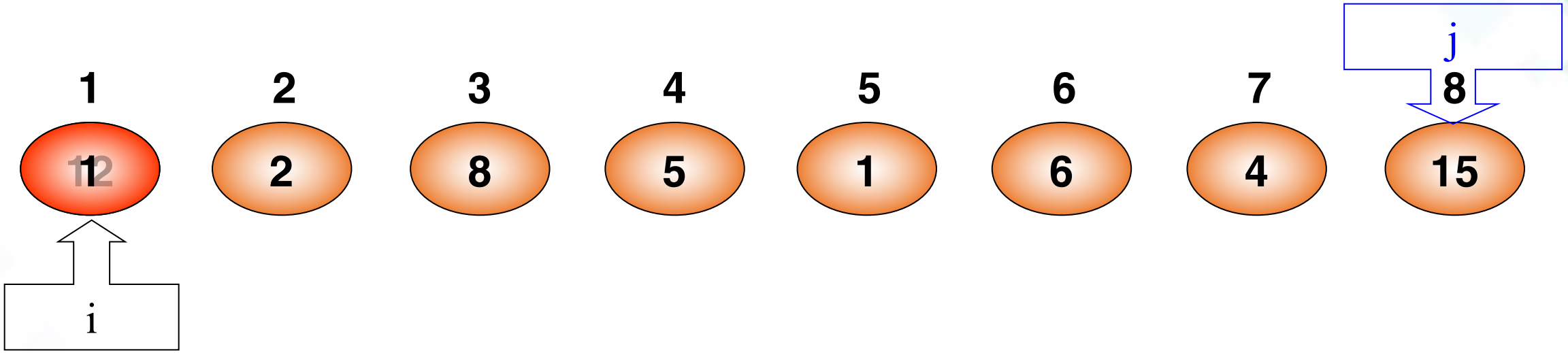
Nội dung



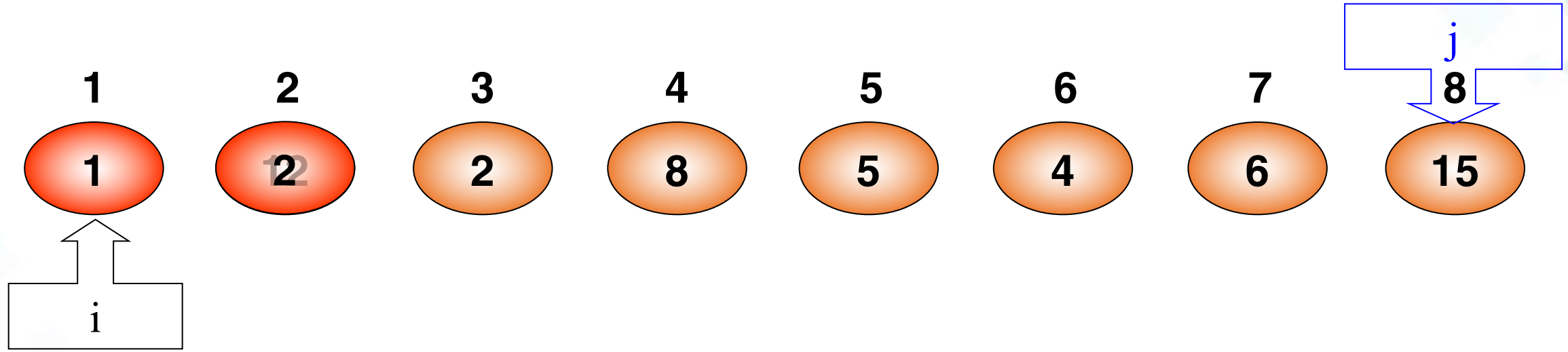
3. Bubble Sort - Ý tưởng

- **Sắp xếp nổi bọt** (*bubble sort*) là một thuật toán sắp xếp đơn giản, với thao tác cơ bản là so sánh hai phần tử **kề** nhau, nếu chúng chưa đứng đúng thứ tự thì đổi chỗ (*swap*). Có thể tiến hành từ trên xuống (bên trái sang) hoặc từ dưới lên (bên phải sang)

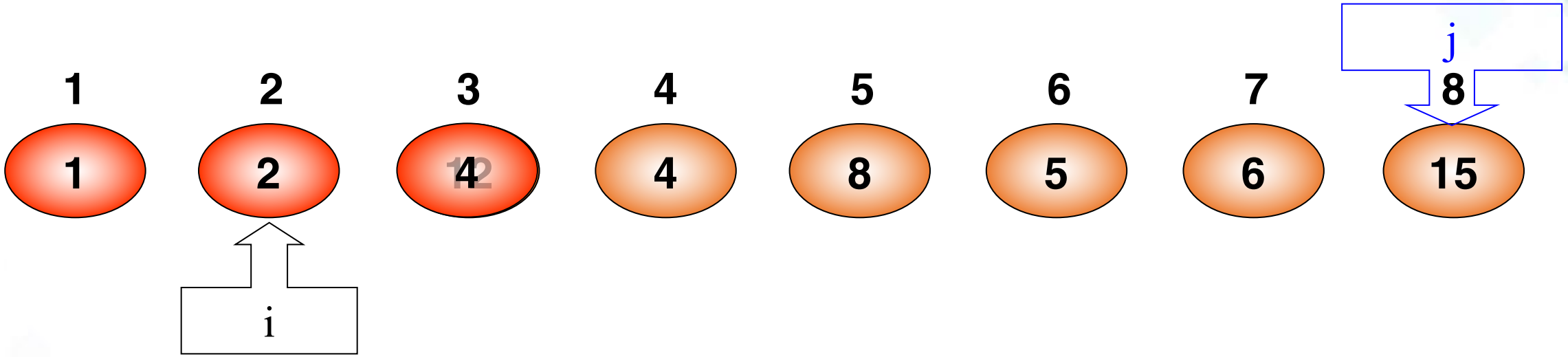
3. Bubble Sort



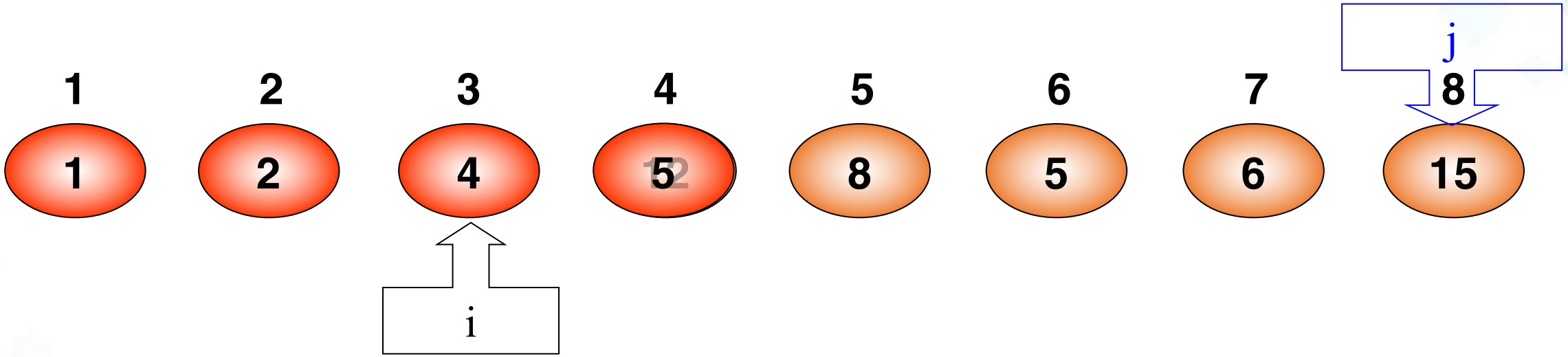
3. Bubble Sort



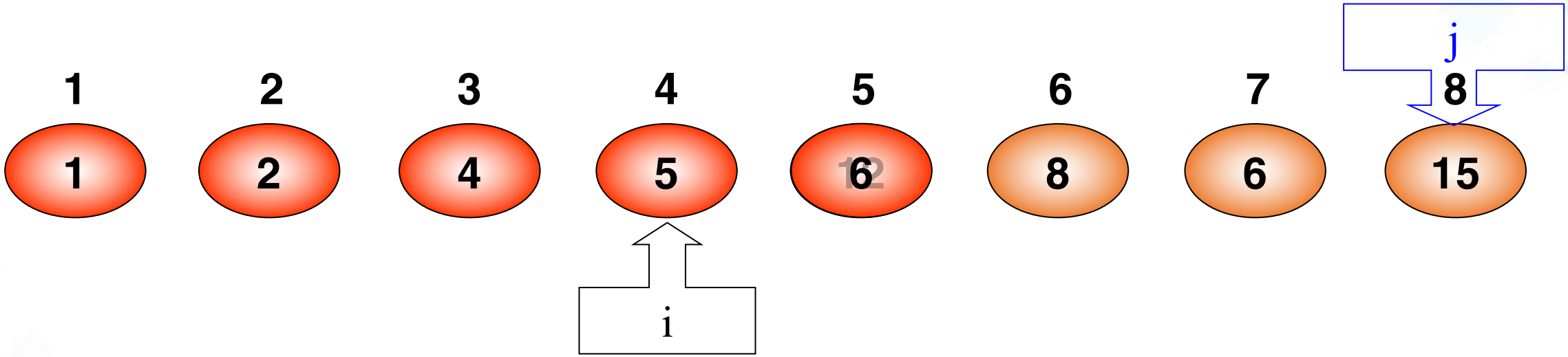
3. Bubble Sort



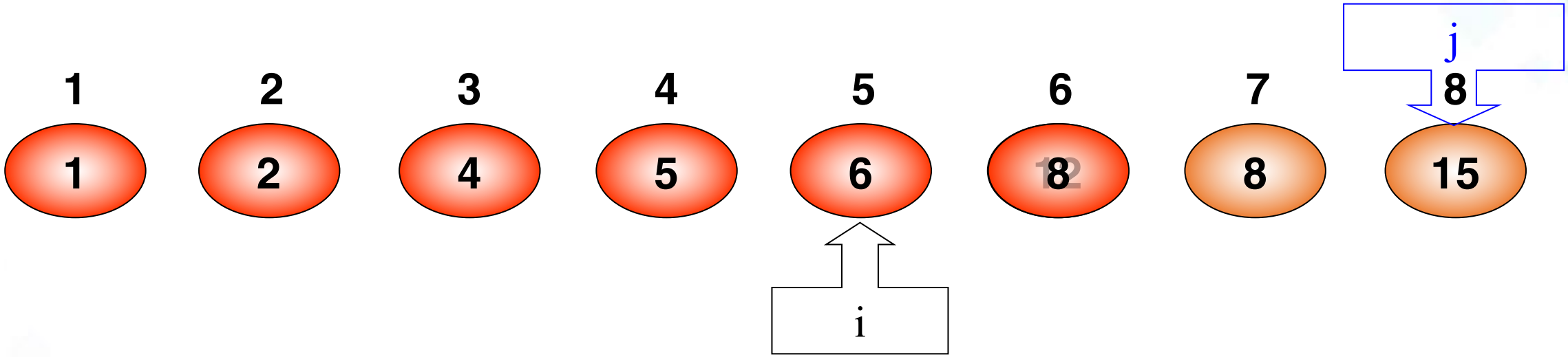
3. Bubble Sort



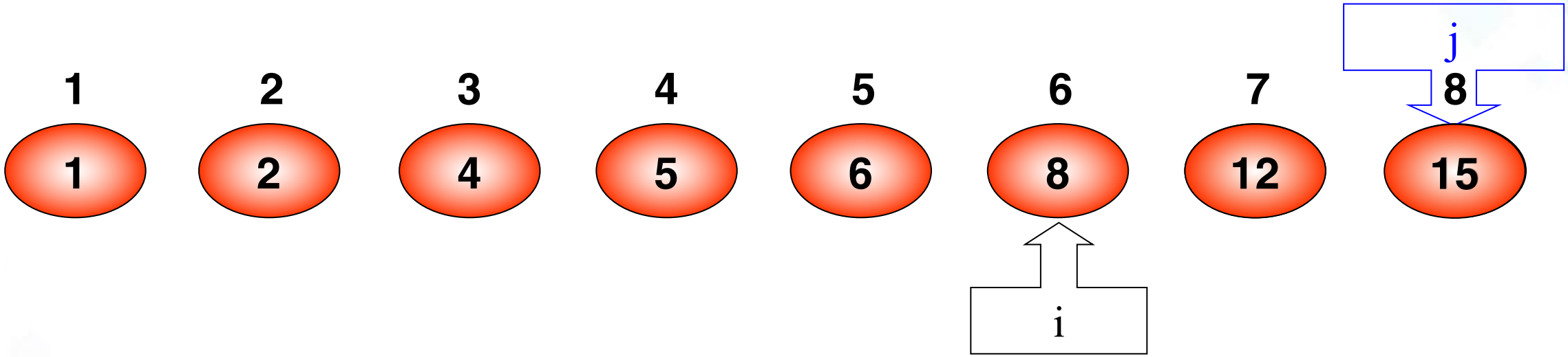
3. Bubble Sort



3. Bubble Sort



3. Bubble Sort



3. Bubble Sort - Quá trình phát sinh code

Dựa trên mô phỏng: Yêu cầu 2 sinh viên lên tái hiện thành những đoạn code

3. Bubble Sort – Cài đặt

```
static void BubbleSort(int[] arrInt)
{
    int i, j;
    for (i = 0; i < arrInt.Length - 1; i++)
    {
        for (j = arrInt.Length - 1; j > i; j--)
            if (arrInt[j] < arrInt[j - 1])
            {//hoán vị arrInt[j] và arrInt[i]
                Swap(arrInt[j], arrInt[j - 1]);
             }
    }
}
```

3. Bubble Sort - Đánh giá giải thuật

Độ phức tạp

Trường hợp xấu nhất: $T(n) = O(n^2)$

Trường hợp tốt nhất: $T(n) = O(n^2)$

Trường hợp trung bình: $T(n) = O(n^2)$

HCMUTE

3. Bubble Sort Cải tiến

Sử dụng thêm một biến boolean là check

- Biến check nhận giá trị false nếu arrInt[0..i] chưa được sắp xếp.
- Biến check nhận giá trị true nếu arrInt[0..i] đã được sắp xếp.

Nếu check nhận giá trị true thì vòng for đầu tiên sẽ dừng lại. Mục đích là ở lệnh lặp đầu tiên, nếu đến chỉ số i nào đó mà đoạn đầu arrInt[0..i] đã được sắp xếp thì ta có thể dừng không lặp nữa, giảm thiểu được thời gian chạy.

3. Bubble Sort Cải tiến – Cài đặt

```
static void BubbleSortImproved(int[] arrInt)
{
    int i, j;
    bool check = false;
    for (i = 0; i < arrInt.Length - 1 && !check; i++)
    {
        check = true;
        for (j = arrInt.Length - 1; j > i; j--)
            if (arrInt[j] < arrInt[j - 1])
            {//hoán vị arrInt[j] và arrInt[i]
                Swap(arrInt[j], arrInt[j - 1]);
                check = false;
            }
    }
}
```

3. Bubble Sort Cải tiến – Đánh giá giải thuật

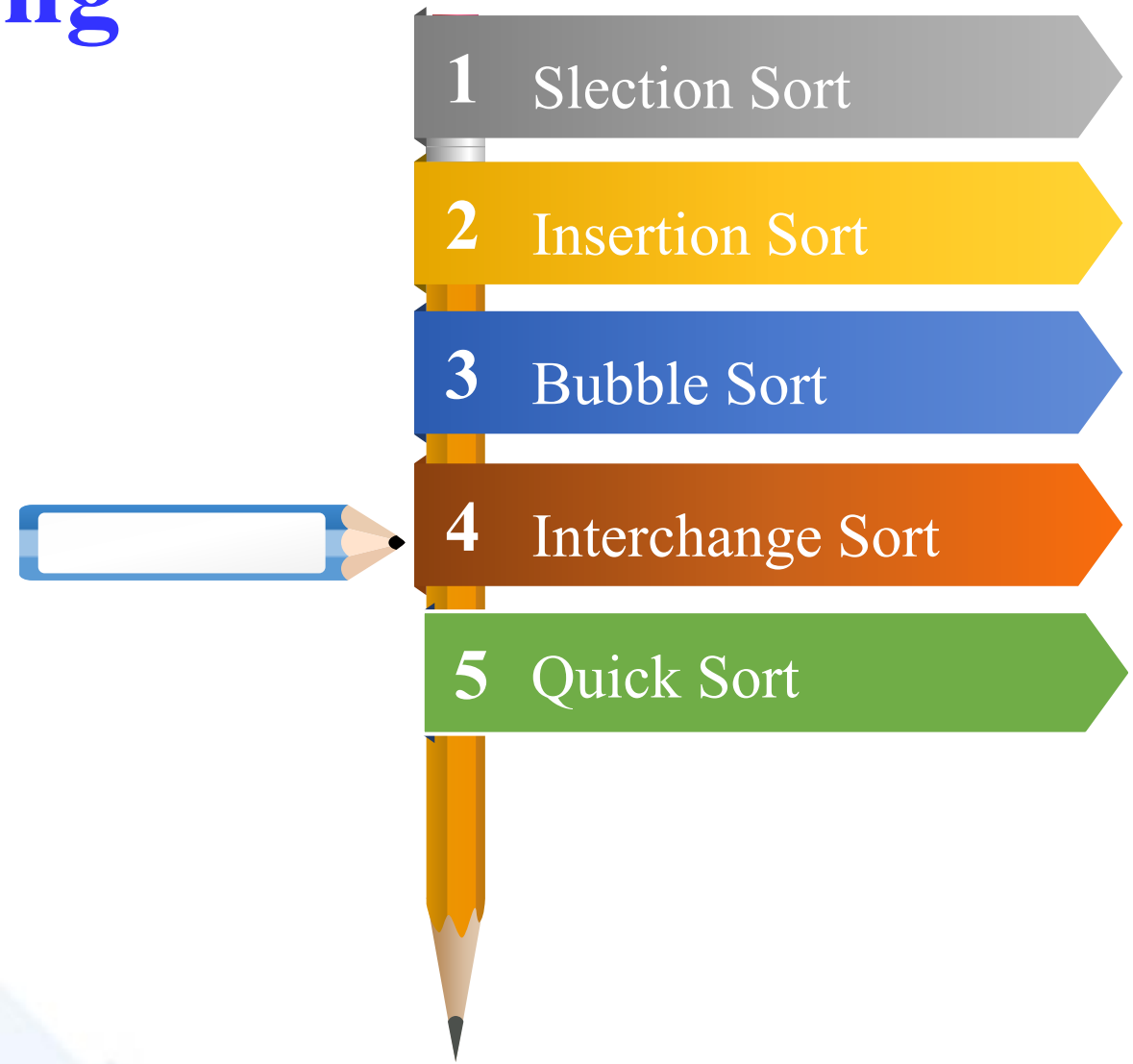
- Phiên bản này cải thiện trường hợp tốt nhất của giải thuật bubble sort từ $O(n^2)$ xuống còn $O(n)$.
- Trường hợp xấu nhất: $T(n) = O(n^2)$
- Trường hợp trung bình: $T(n) = O(n^2)$

3. Bubble Sort - Sắp xếp trên dữ liệu có cấu trúc

- Code sẽ thay đổi như thế nào?
- Có nhận xét gì kiểu dữ liệu của khóa tìm kiếm và kiểu dữ liệu của danh sách?

HCMUTE

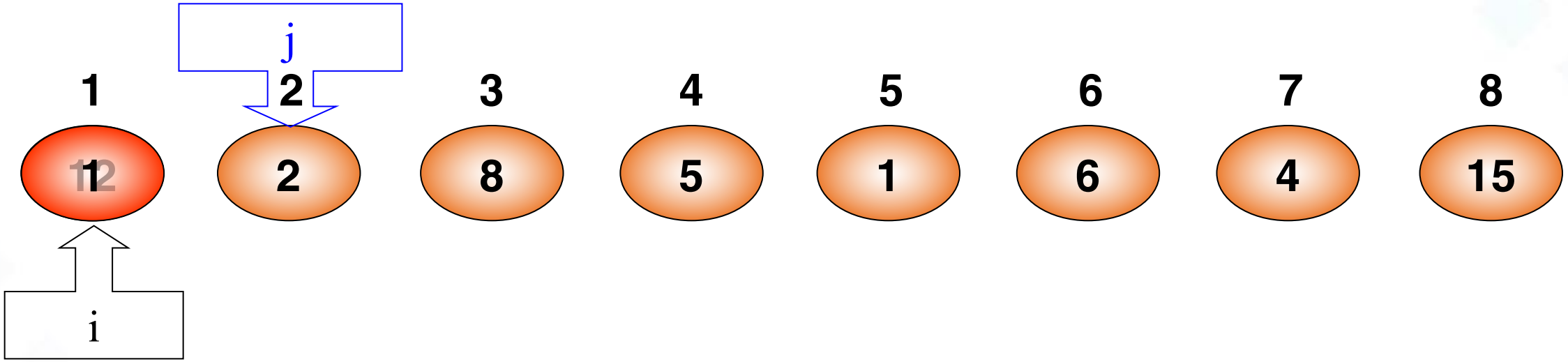
Nội dung



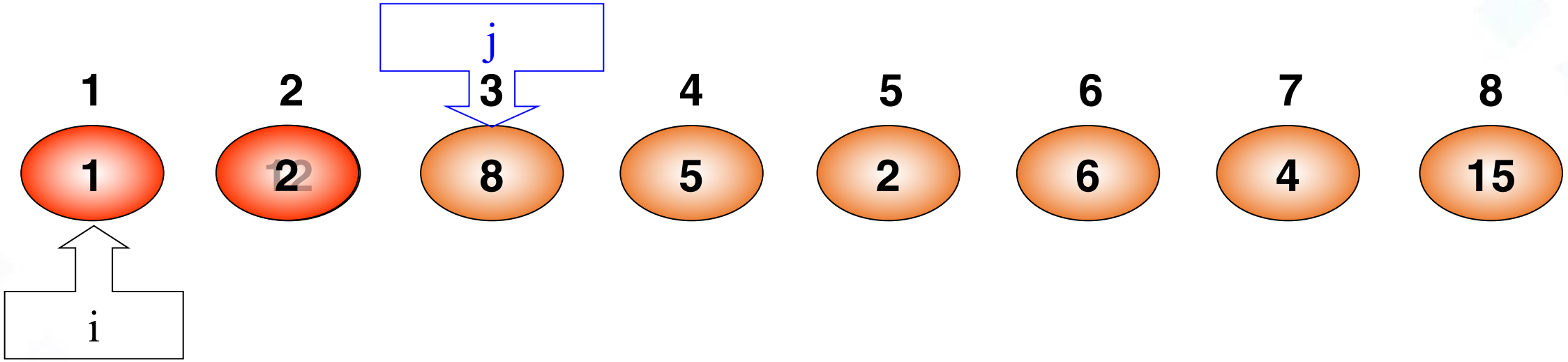
4. Interchange sort - Ý tưởng

- Ý tưởng chính của giải thuật là xuất phát từ đầu dãy, tìm tất cả nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chổ phần tử này với phần tử tương ứng trong cặp nghịch thế. Lặp lại xử lý trên với các phần tử kế tiếp theo trong dãy.

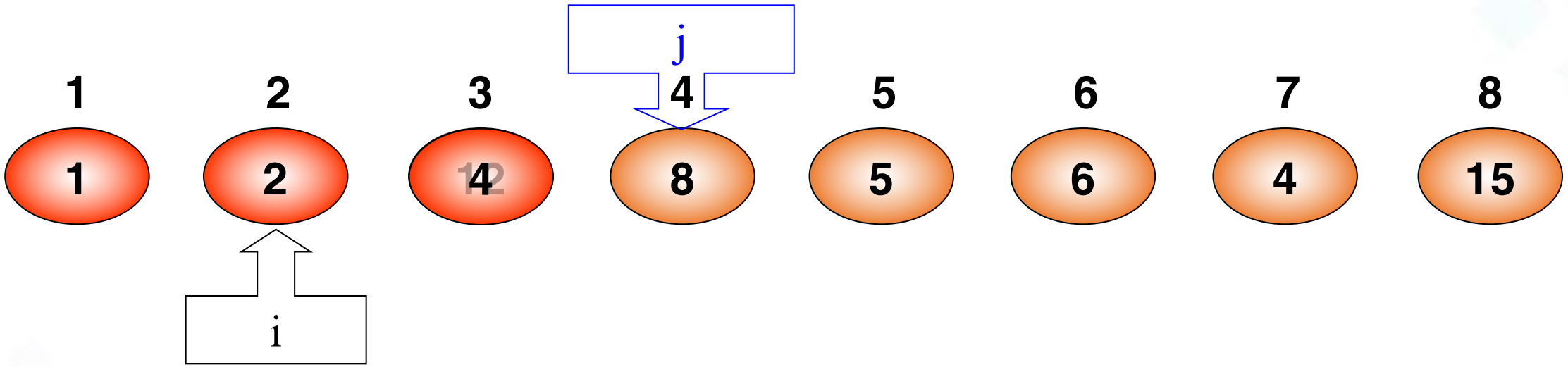
4. Interchange sort



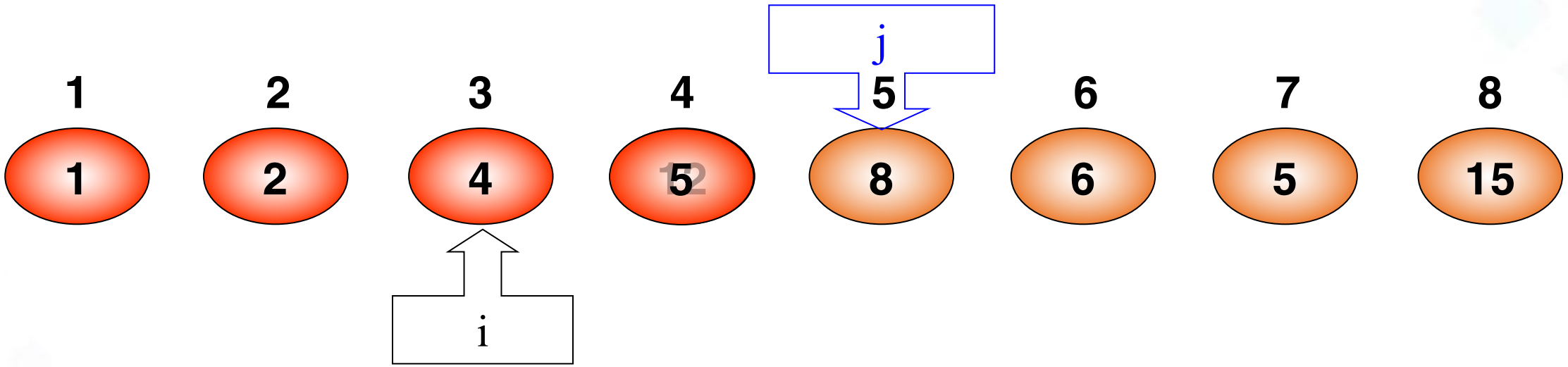
4. Interchange sort



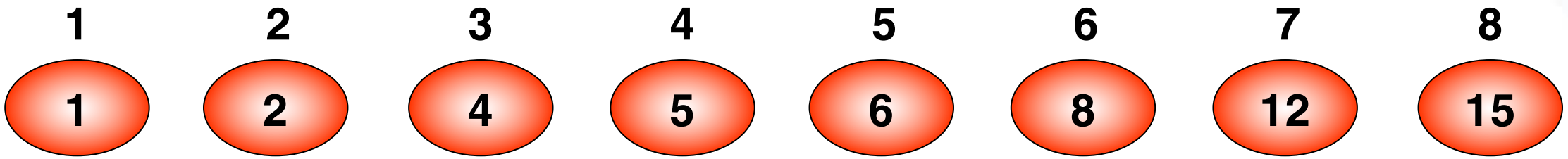
4. Interchange sort



4. Interchange sort



4. Interchange sort



4. Interchange sort - Quá trình phát sinh code

Dựa trên mô phỏng: Yêu cầu 2 sinh viên lên tái hiện thành những đoạn code

4. Interchange sort - Cài đặt

```
static void InterchangeSort(int[] arrInt)
{
    int i, j;
    for (i = 0; i < arrInt.Length - 1; i++)
    {
        for (j = i + 1; j < arrInt.Length; j++)
            if (arrInt[j] < arrInt[i])//thỏa 1 cặp nghịch thế
            {//hoán vị arrInt[j] và arrInt[i]
                int temp = arrInt[j];
                arrInt[j] = arrInt[i];
                arrInt[i] = temp;
            }
    }
}
```

4. Interchange sort - Đánh giá giải thuật

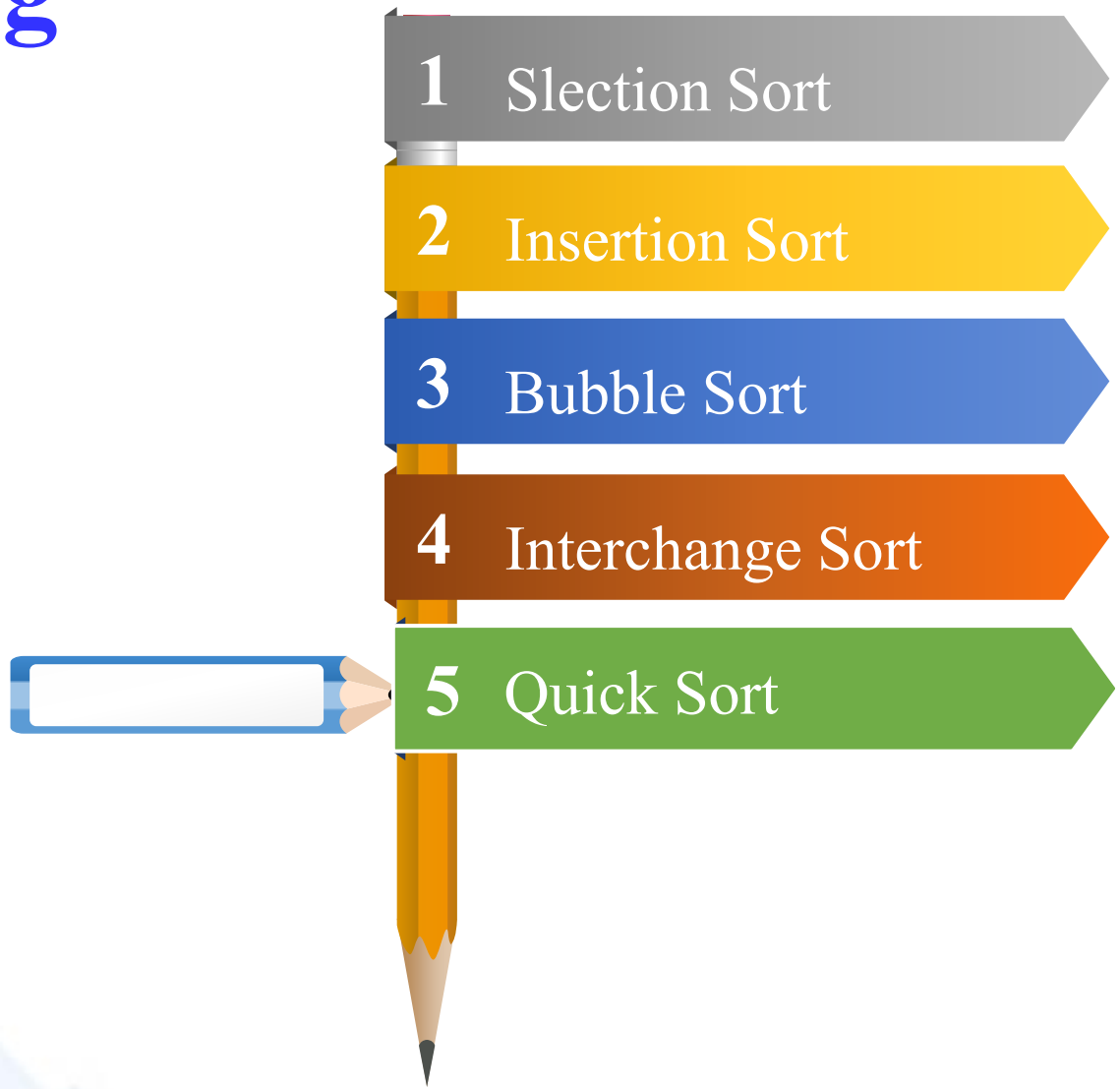
Độ phức tạp

Trường hợp xấu nhất: $T(n) = O(n^2)$

Trường hợp tốt nhất: $T(n) = O(n^2)$

Trường hợp trung bình: $T(n) = O(n^2)$

Nội dung


1 Slection Sort
2 Insertion Sort
3 Bubble Sort
4 Interchange Sort
5 Quick Sort

5. Quick Sort - Ý tưởng

- Giải thuật QuickSort sắp xếp dãy $\mathbf{a_1}, \mathbf{a_2} ..., \mathbf{a_N}$ dựa trên việc phân hoạch dãy ban đầu thành 3 phần
  - Phần 1: Gồm các phần tử có giá trị $\leq \mathbf{x}$
  - Phần 2: Gồm các phần tử có giá trị $= \mathbf{x}$
  - Phần 3: Gồm các phần tử có giá trị $\geq \mathbf{x}$

với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

5. Quick Sort - Ý tưởng

- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:
  - 1. $a_k \leq x$, với k = 1 .. j
  - 2. $a_k = x$, với k = j+1 .. i-1
  - 3. $a_k \geq x$, với k = i..N
- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 chỉ có 1 phần tử : đã có thứ tự

  → khi đó dãy con ban đầu đã được sắp

5. Quick Sort - Mô tả thuật toán Quick Sort

- Bước 1: Nếu left ≥ right //dãy có ít hơn 2 phần tử

  Kết thúc; //dãy đã được sắp xếp

- Bước 2: Phân hoạch dãy $a_{left} \dots a_{right}$ thành các đoạn: $a_{left}.. a_j$, $a_{j+1}.. a_{i-1}$, $a_i.. a_{right}$

  Đoạn 1: $a_{left}.. a_j \le x$

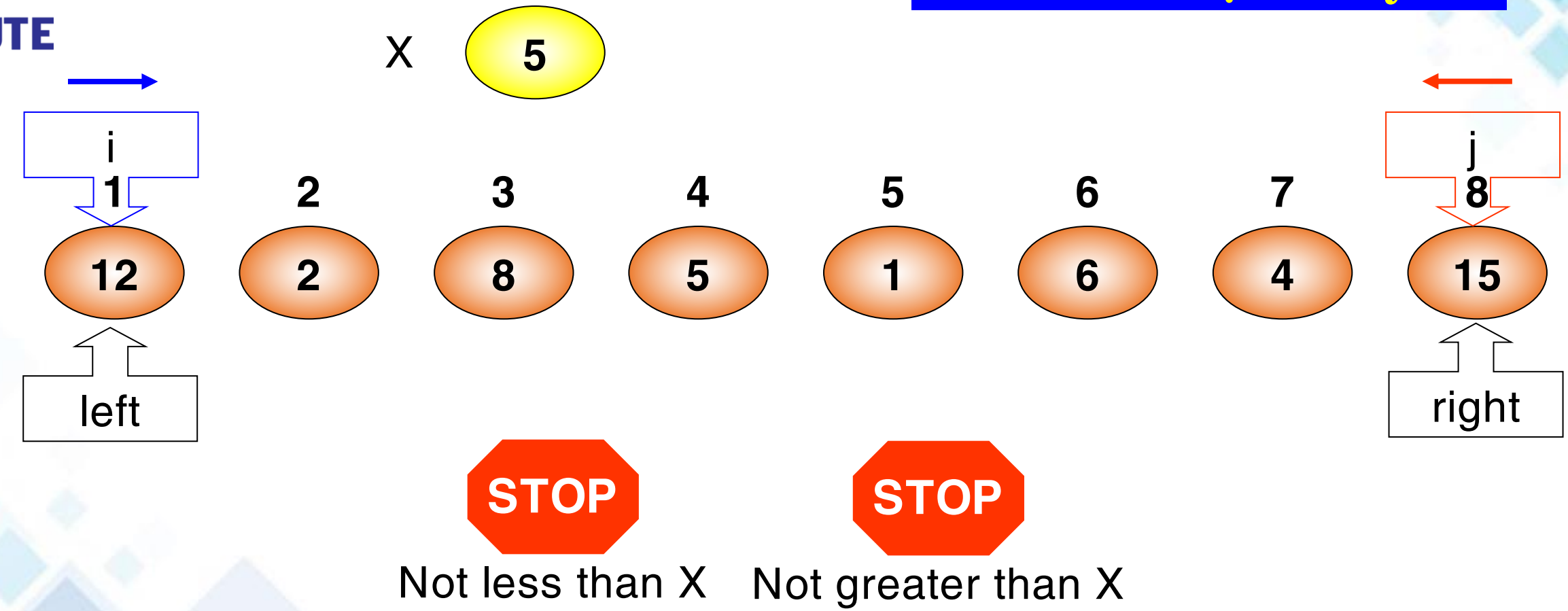
  Đoạn 2: $a_{j+1}.. a_{i-1} = x$

  Đoạn 3: $a_i.. a_{right} \ge x$

- Bước 3: **Sắp xếp đoạn 1**: $a_{left}.. a_j$
- Bước 4: **Sắp xếp đoạn 3**: $a_i.. a_{right}$

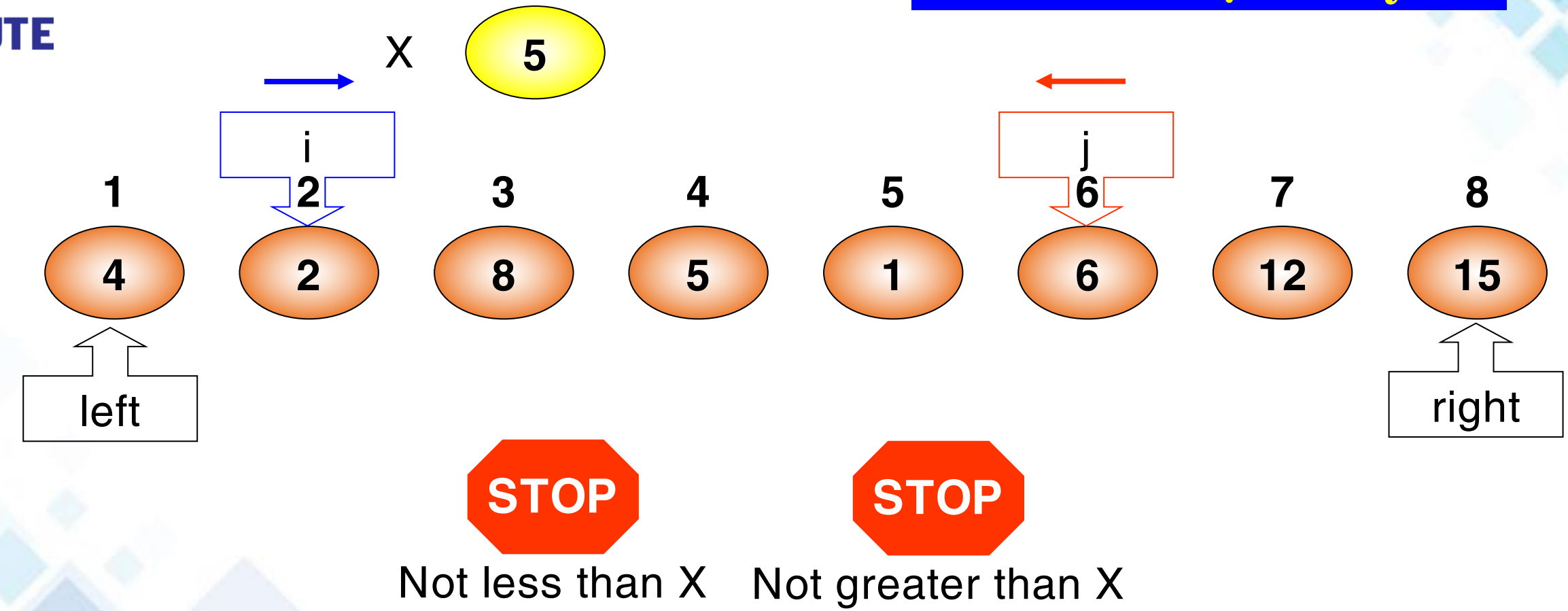
5. Quick Sort

Phân hoạch dãy

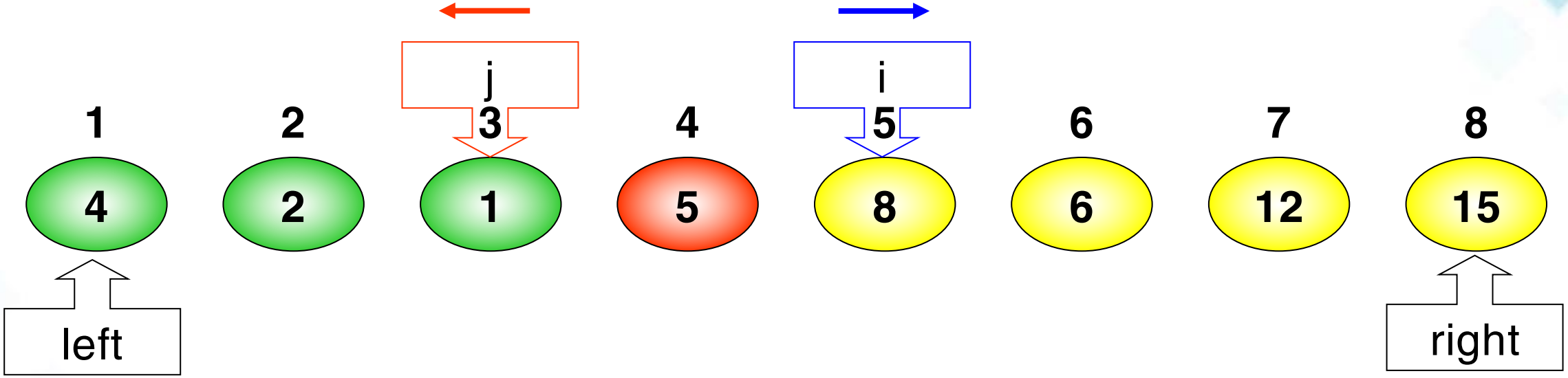


5. Quick Sort

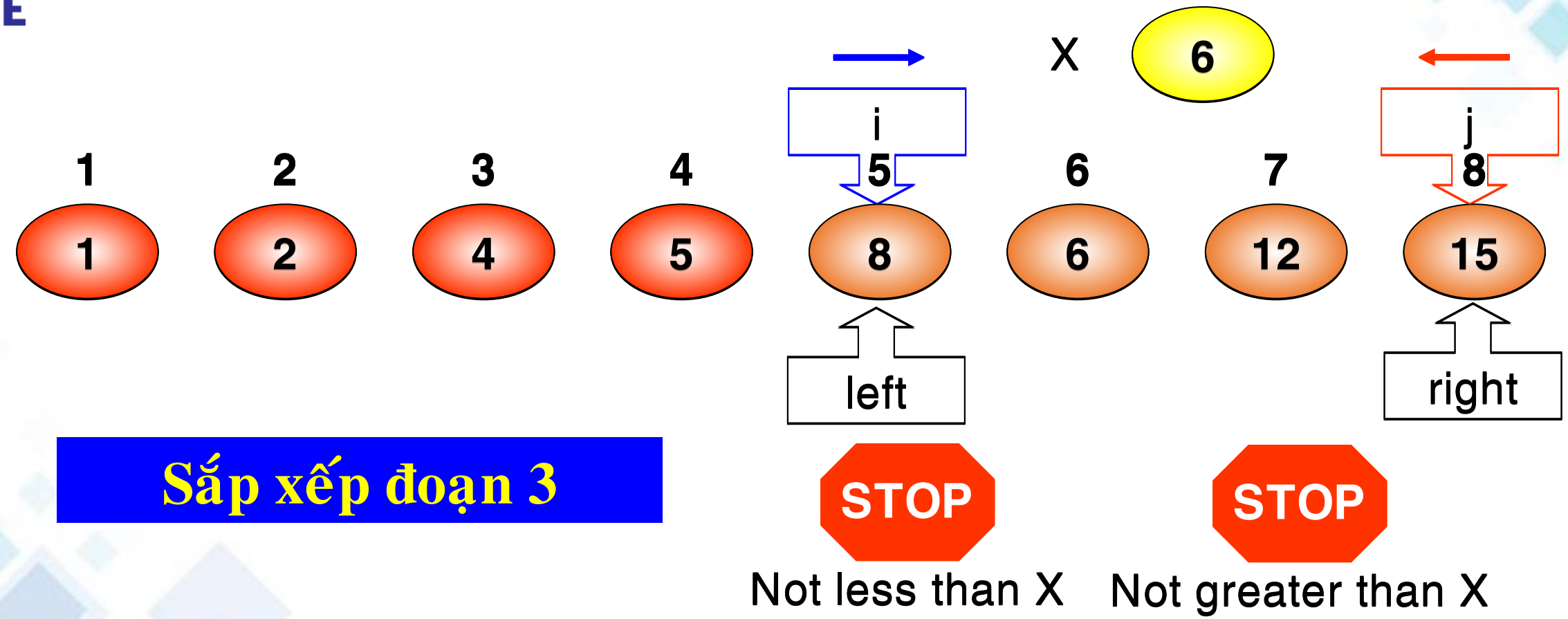
Phân hoạch dãy



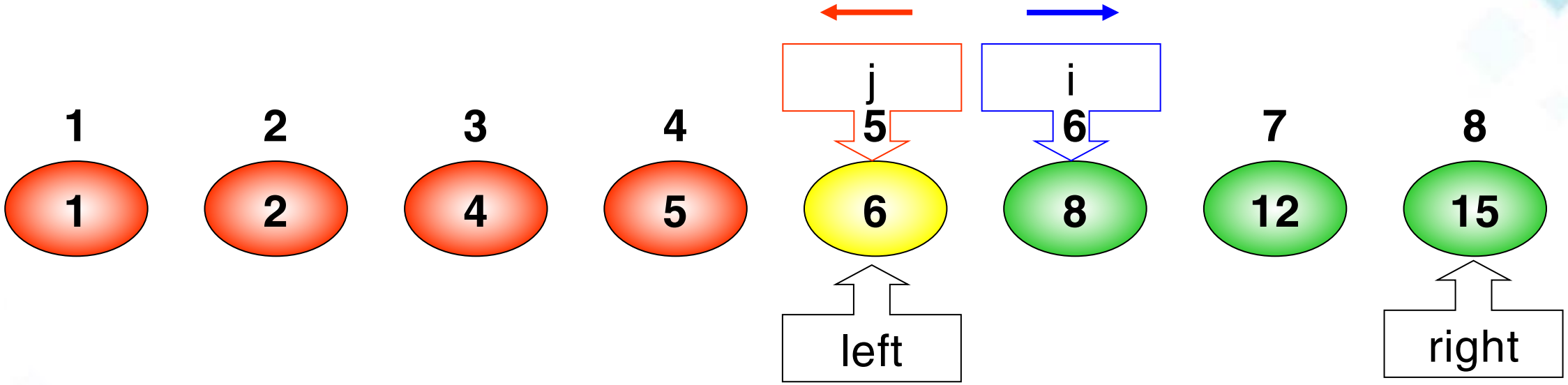
5. Quick Sort



5. Quick Sort

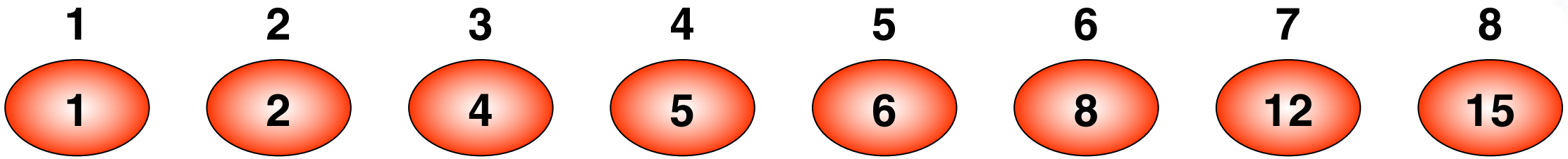

Phân hoạch dãy
X
6
i
j
1
2
3
4
5
6
7
8
1
2
4
5
8
6
12
15
left
right
STOP
STOP
Not less than X
Not greater than X
Sắp xếp đoạn 3

5. Quick Sort



Sắp xếp đoạn 3

5. Quick Sort



5. Quick Sort – Cài đặt

```
static void QuickSort(int[] arrInt, int left, int right)
{
    if (left >= right) return;

    int i = left, j = right, mid = arrInt[(left + right) / 2];
    while (i <= j)
    {
        while (arrInt[i] < mid) i++;
        while (arrInt[j] > mid) j--;
        if (i <= j)
        {//hoán vị arrInt[j] và arrInt[i]
            int temp = arrInt[j];
            arrInt[j] = arrInt[i];
            arrInt[i] = temp;
            i++; j--;
        }
    }
    QuickSort(arrInt, left, j);
    QuickSort(arrInt, i, right);

}
```