SORTING ALGORITHMS

DATA STRUCTURES AND ALGORITHMS

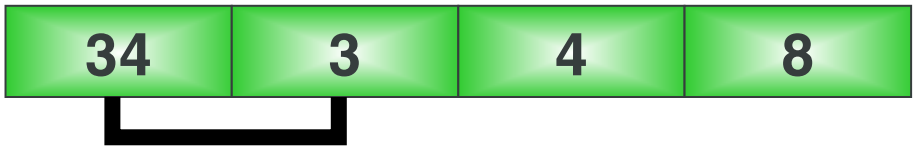
ThS Nguyễn Thị Ngọc Diễm
diemntn@uit.edu.vn

Bài toán Sắp xếp

- Cho danh sách có n phần tử $a_0, a_1, a_2 \ldots, a_{n-1}$.
- Sắp xếp là quá trình xử lý các phần tử trong danh sách để đặt chúng theo một thứ tự thỏa mãn một số tiêu chuẩn nào đó dựa trên thông tin lưu tại mỗi phần tử, như:
  - Sắp xếp danh sách lớp học tăng theo điểm trung bình.
  - Sắp xếp danh sách sinh viên tăng theo tên.
  - …
- Để đơn giản trong việc trình bày giải thuật ta dùng mảng 1 chiều **a** để lưu danh sách trên trong bộ nhớ chính.

Bài toán Sắp xếp (tt)

- a: là dãy các phần tử dữ liệu
- Để sắp xếp dãy a theo thứ tự (giả sử theo thứ tự tăng), ta tiến hành triệt tiêu tất cả các nghịch thế trong a.
  - Nghịch thế:
    - Cho dãy có n phần tử $a_0, a_1, \ldots, a_{n-1}$
    - Nếu $i<j$ và $a_i > a_j$



a[0], a[1] là cặp nghịch thế

- Đánh giá độ phức tạp của giải thuật, ta tính

  C_{ss}: Số lượng phép so sánh cần thực hiện
  C_{HV}: Số lượng phép hoán vị cần thực hiện

Nội dung

- Các giải thuật sắp xếp nội:

1. **Chọn trực tiếp - Selection Sort**
2. **Đổi chỗ trực tiếp - Interchange Sort**
3. Nổi bọt - Bubble Sort
4. Shaker Sort
5. **Chèn trực tiếp - Insertion Sort**
6. **Chèn nhị phân - Binary Insertion Sort**
7. Shell Sort
8. **Quick Sort**
9. **Heap Sort**
10. **Merge Sort**
11. Counting Sort
12. Radix Sort

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort
3. Nổi bọt - Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp - Insertion Sort
6. Chèn nhị phân - Binary Insertion Sort
7. Shell Sort
8. Quick Sort
9. Heap Sort
10. Merge Sort
11. Counting Sort
12. Radix Sort

Selection Sort: Ý tưởng

- Chọn phần tử nhỏ nhất trong N phần tử trong dãy hiện hành ban đầu.
- Đưa phần tử này về vị trí đầu dãy hiện hành.
- Xem dãy hiện hành chỉ còn N-1 phần tử của dãy ban đầu (bắt đầu từ vị trí thứ 2)
- Lặp lại quá trình trên cho dãy hiện hành ... đến khi dãy hiện hành chỉ còn 1 phần tử

Selection Sort

Từ khóa: Selection Sort

Phân tích: Giả sử danh sách $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\Re$.Khi đó:

- a_0 là phần tử nhỏ nhất theo $\Re$ trong A
- a_1 là phần tử nhỏ nhất theo $\Re$ trong $A \setminus \{a_0\}$
- a_2 là phần tử nhỏ nhất theo $\Re$ trong $A \setminus \{a_0, a_1\}$
- …

Selection Sort

Thuật toán:

Input: $A=\{a_0, a_1, .., a_{n-1}\}$ chưa có thứ tự $\mathfrak{R}$

Output: $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\mathfrak{R}$

Selection Sort

Mã giả:

- Bước 1: Khởi gán i = 0;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]
- Bước 3: Đổi chỗ a[min] và a[i]
- Bước 4: Nếu i < N-1 thì i = i+1; Lặp lại Bước 2;

  Ngược lại: Dừng.

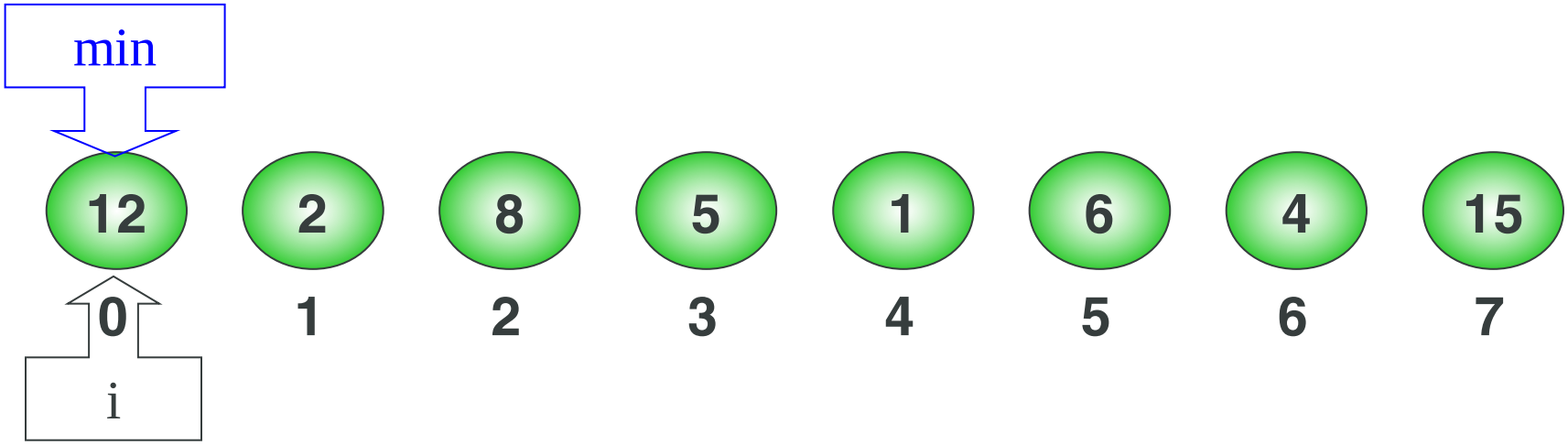
Selection Sort : Code C/C++

```
template <class Object>

void Swap(Object &lhs, Object &rhs) {
    Object tmp = lhs;
    lhs = rhs;
    rhs = tmp;
}

template <class Item>

void SelectionSort(Item a[], int n) {
    for (int i = 0; i < n-1; i++) {
        int min = i;
        for (int j = i+1; j < n; j++)
            if (a[j] < a[min]) min = j;
        if (min != i ) Swap(a[i], a[min]);
    }
```

Selection Sort: Minh họa

Vị trí nhỏ nhất(0,7)

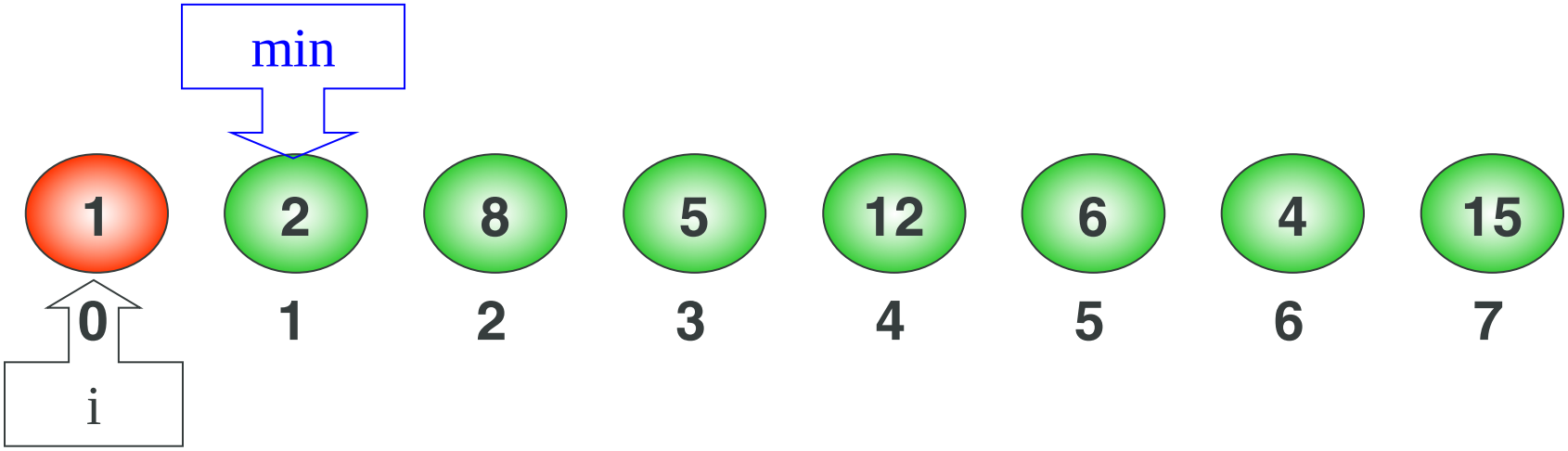
Swap(a[0], a[4])



Selection Sort: Minh họa (tt)

Vị trí nhỏ nhất(1,7)

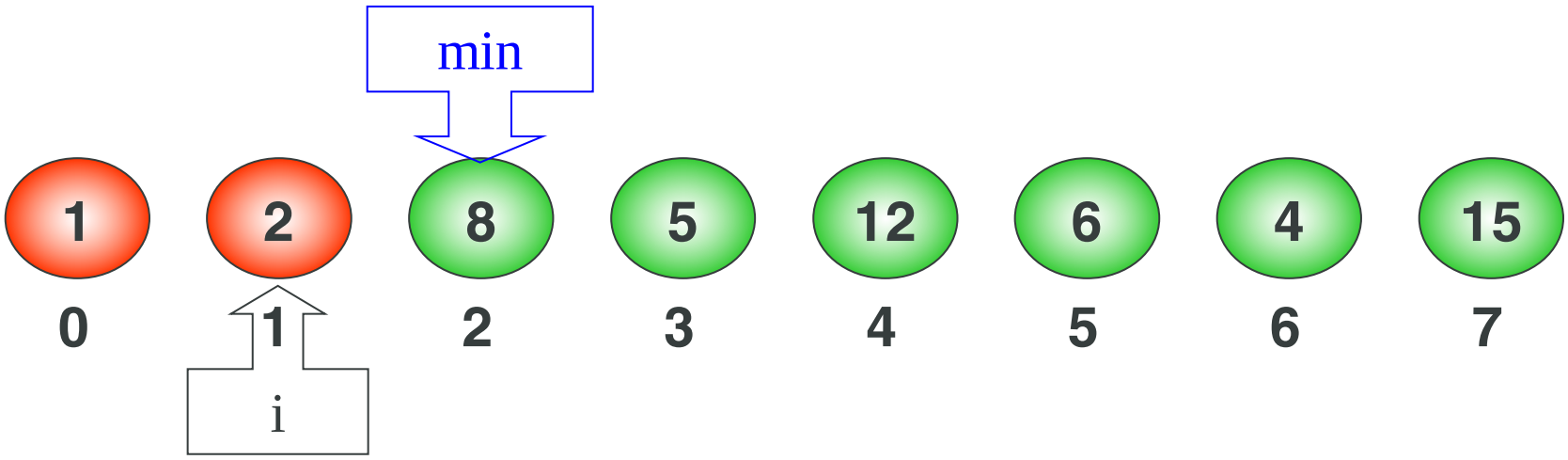
Swap(a[1], a[1])



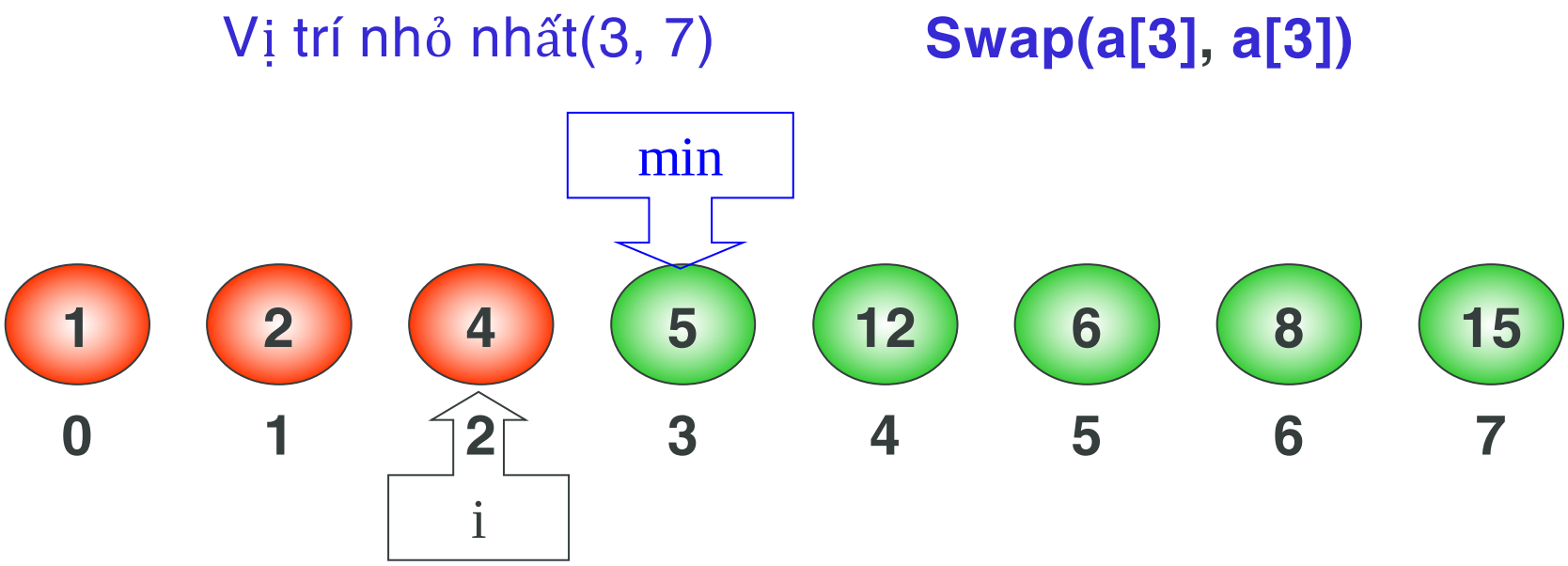
Selection Sort: Minh họa (tt)

Vị trí nhỏ nhất(2,7)

Swap(a[2], a[6])



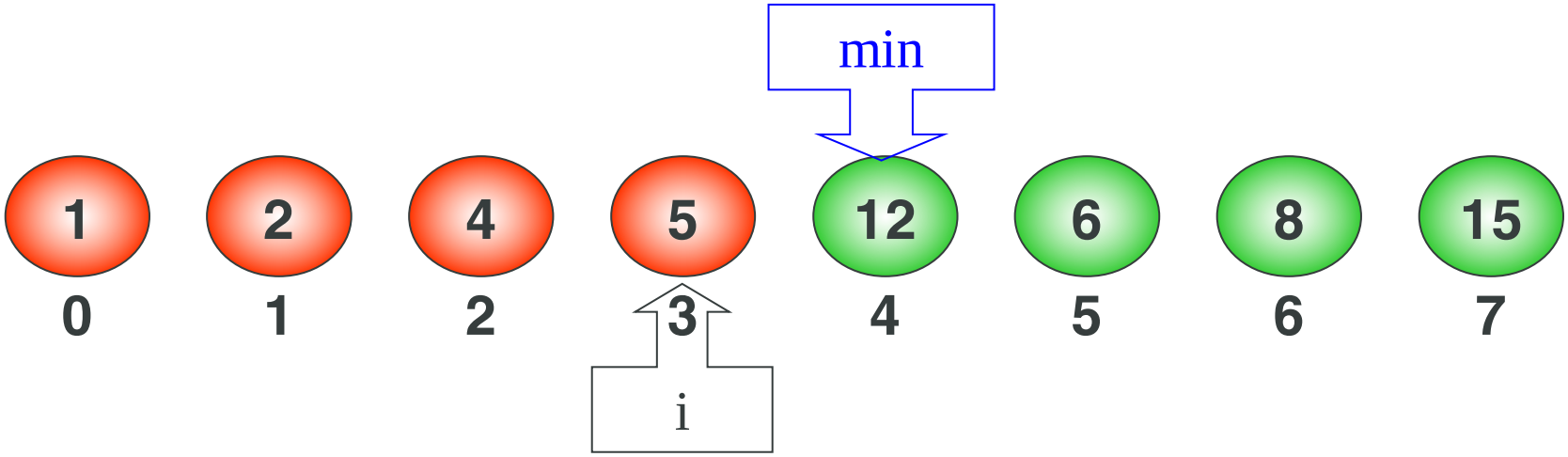
Selection Sort: Minh họa (tt)



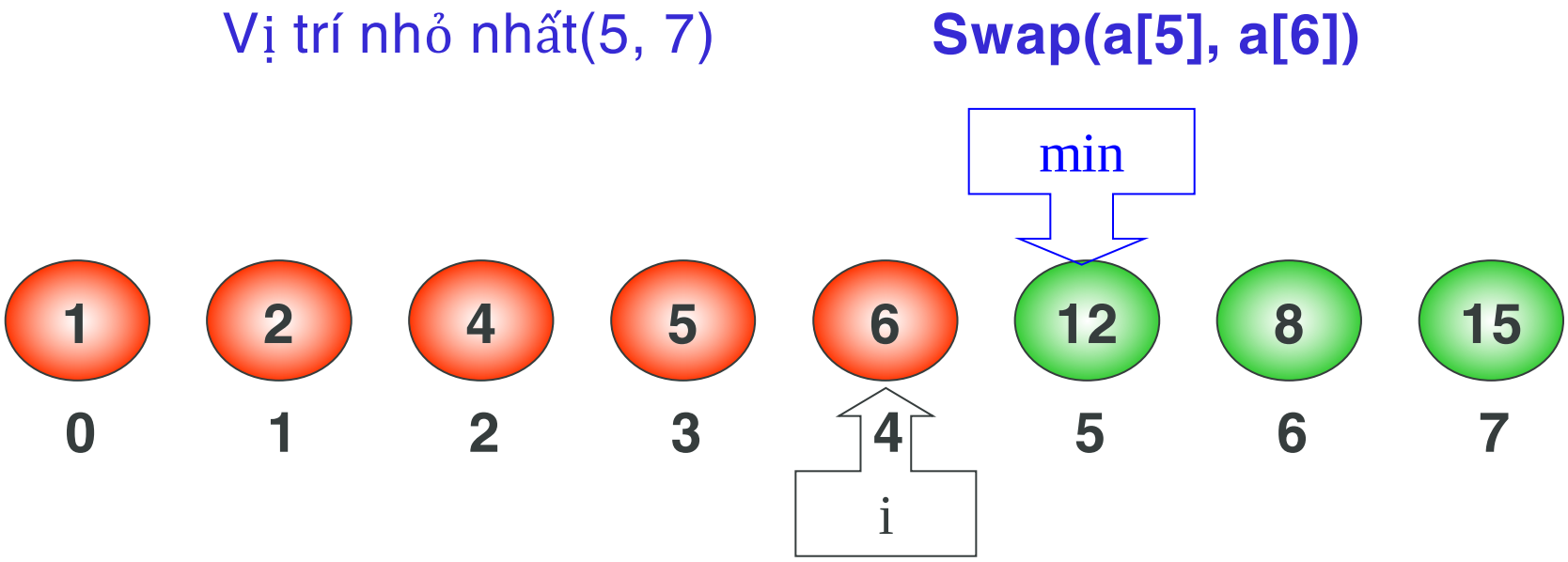
Selection Sort: Minh họa (tt)

Vị trí nhỏ nhất(4, 7)

Swap(a[4], a[5])

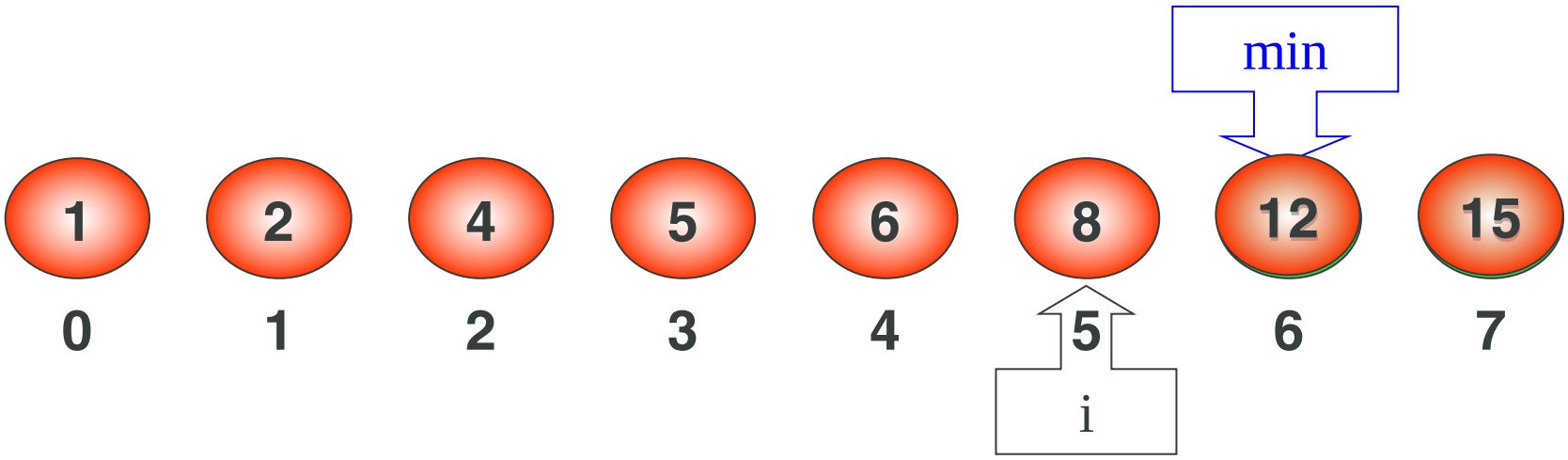


Selection Sort: Minh họa (tt)



Selection Sort: Minh họa (tt)

Vị trí nhỏ nhất(6, 7)



Selection Sort -- Analysis

- In general, we compare keys and move items (or exchange items) in a sorting algorithm (which uses key comparisons).

  ➔ **So, to analyze a sorting algorithm we should count the number of key comparisons and the number of moves.**

  - Ignoring other operations does not affect our final result.

- In selectionSort function, the outer for loop executes n-1 times.
- We invoke swap function once at each iteration.

  ➔ Total Swaps: n-1

  ➔ Total Moves: 3*(n-1) (Each swap has three moves)

Selection Sort – Analysis (cont.)

- The inner for loop executes the size of the unsorted part minus 1 (from 1 to n-1), and in each iteration we make one key comparison.

  ➔ # of key comparisons = 1+2+...+n-1 = n*(n-1)/2

  ➔ **So, Selection sort is $O(n^2)$**

- The best case, the worst case, and the average case of the selection sort algorithm are same. ➔ all of them are **$O(n^2)$**
  - This means that the behavior of the selection sort algorithm does not depend on the initial organization of data.
  - Since $O(n^2)$ grows so rapidly, the selection sort algorithm is appropriate only for small n.
  - Although the selection sort algorithm requires $O(n^2)$ key comparisons, it only requires $O(n)$ moves.
  - A selection sort could be a good choice if data moves are costly but key comparisons are not costly (short keys, long records).

Selection Sort: Độ phức tạp

	Tốt nhất (đúng thứ tự)	**Trung bình** (chưa có thứ tự)	**Xấu nhất** (thứ tự ngược)
Theo phép so sánh	$O(n^2)$	$O(n^2)$	$O(n^2)$
Theo phép gán giá trị khóa	O(n)	O(n^2)	O(n^2)

Thuật toán có độ phức tạp: $O(n^2)$

Selection Sort: Comparison of *N*, *logN* and N^2

N	O(logN)	$O(N^2)$
16		4 256
64		6 4K
256	8	64K
1,024	10	1M
16,384	14	256M
131,072	17	16G
262,144	18	6.87E+10
524,288	19	2.74E+11
1,048,576	20	1.09E+12
1,073,741,824		30 1.15E+18

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort

3. Nổi bọt - Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp - Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Quick Sort

9. Heap Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Interchange Sort: Ý tưởng

- Xuất phát từ đầu dãy, tìm tất các các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy cho đến khi dãy còn 1 phần tử.

Interchange Sort: Mã giả

- Bước 1: Khởi gán i = 0; *// bắt đầu từ đầu dãy*
- Bước 2: j = i+1; *//tìm các nghịch thế với a[i]*
- Bước 3:

  Trong khi j < N thực hiện

  * Nếu a[j]<a[i] *//xét cặp a[i], a[j]*

    Swap(a[i], a[j]);

  * j = j+1;

- Bước 4: i = i+1;

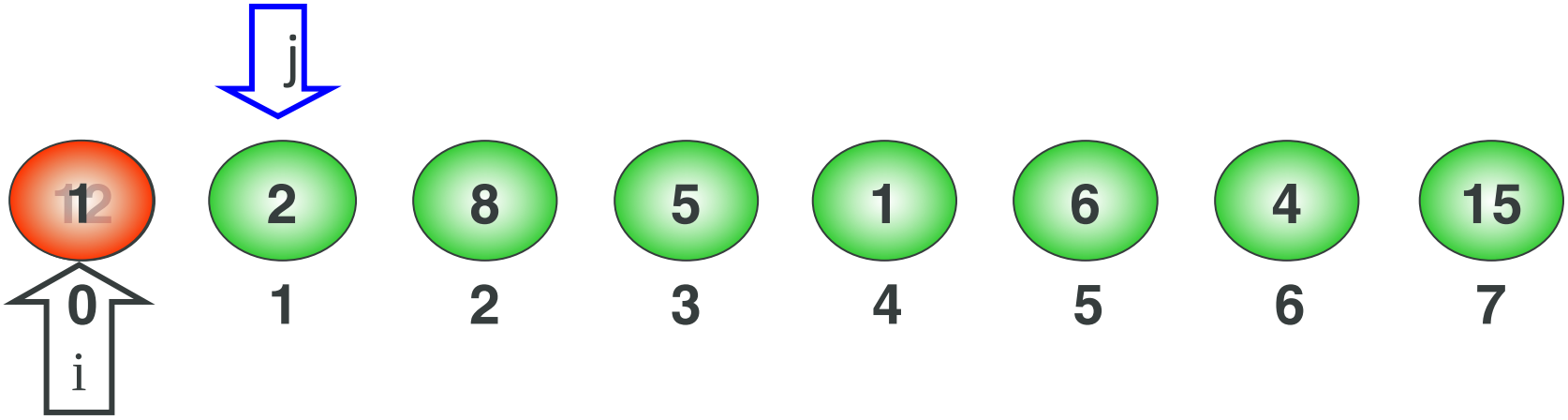
  Nếu i < N-1: Lặp lại Bước 2.

  Ngược lại: Dừng.

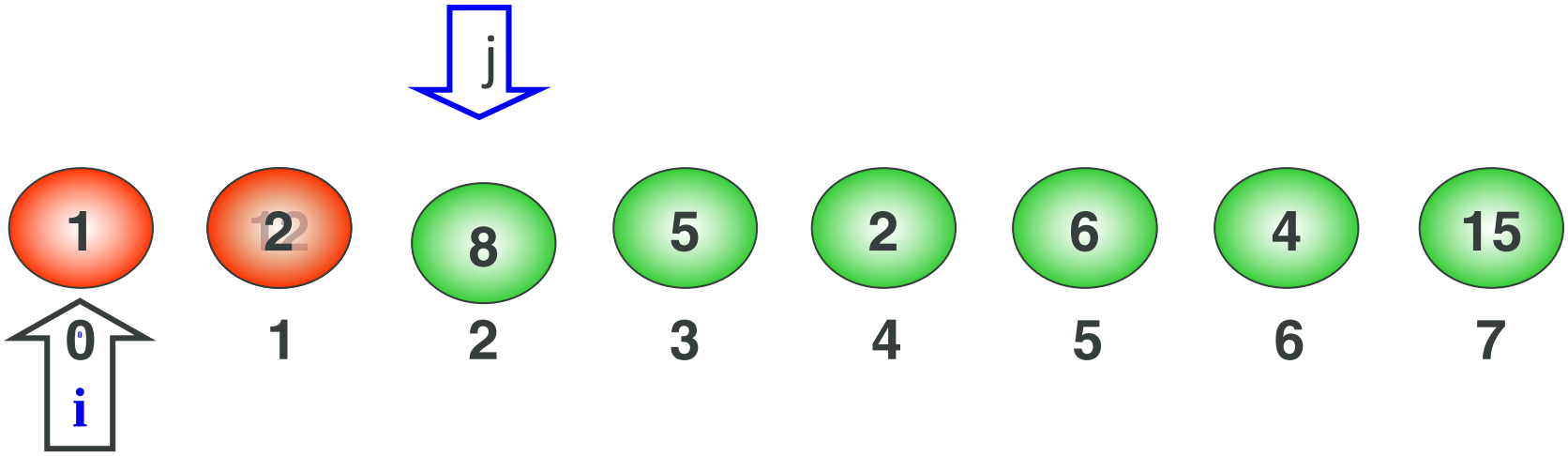
Interchange Sort: Code C/C++

```
template <class Object>

void Swap(Object &lhs, Object &rhs) {
    Object tmp = lhs;
    lhs = rhs;
    rhs = tmp;
}

template <class Item>

void InterchangeSort(Item a[], int n) {
    for (int i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++)
            if (a[i] < a[j])
             Swap(a[i], a[j]);
        }
```

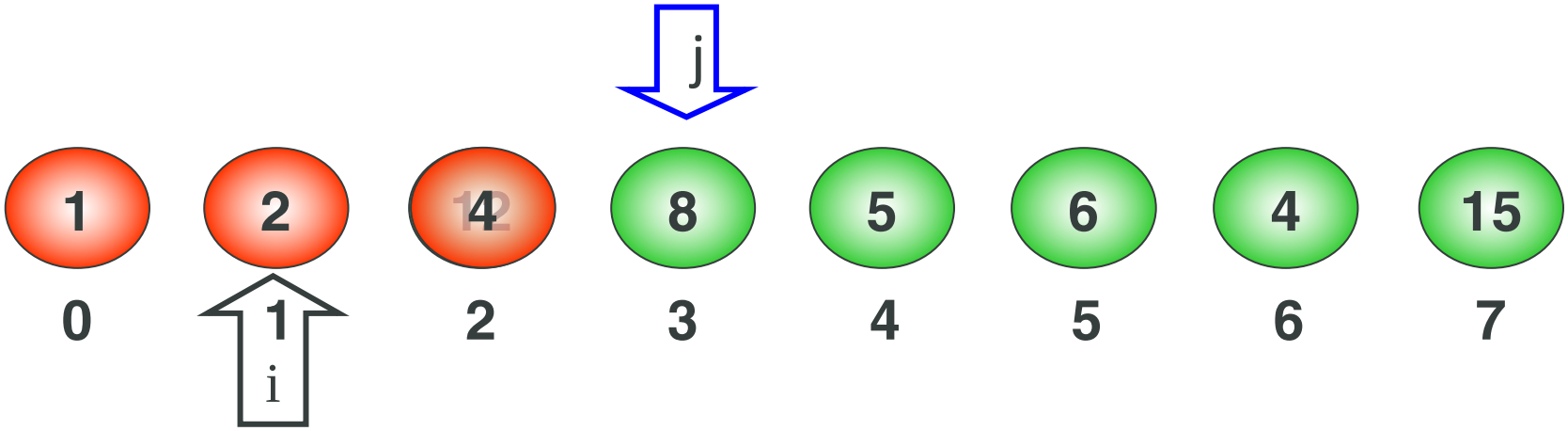
Interchange Sort: Minh họa



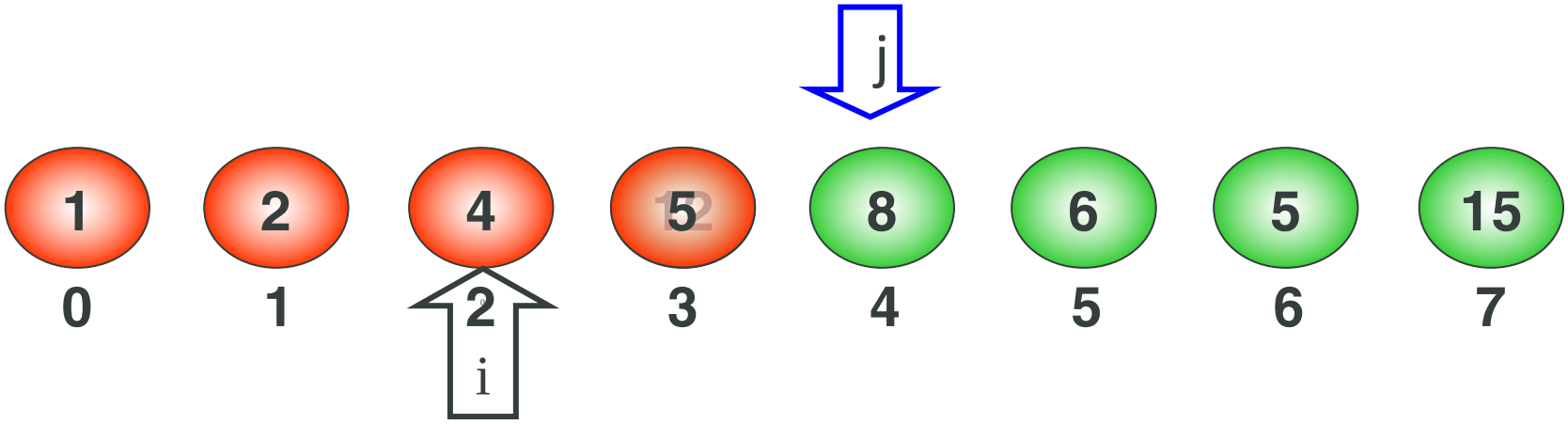
Interchange Sort: Minh họa (tt)



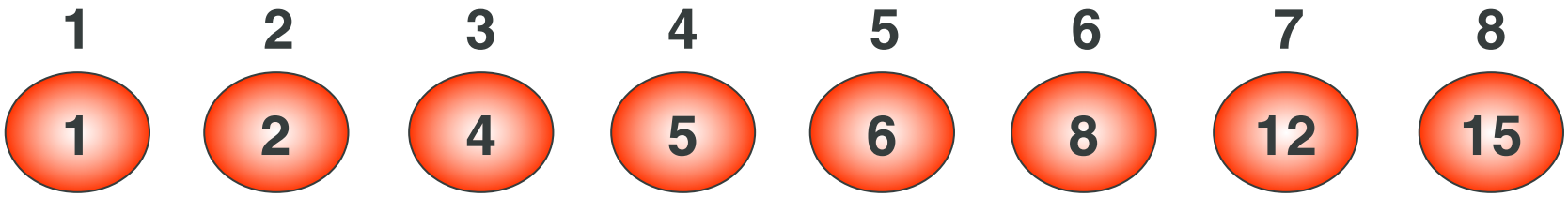
Interchange Sort: Minh họa (tt)



Interchange Sort: Minh họa (tt)



Interchange Sort: Minh họa (tt)



Interchange Sort: Độ phức tạp

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

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort

3. Nổi bọt - Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp - Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Quick Sort

9. Heap Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Bubble Sort: Ý tưởng

- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đúng đầu dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i.

- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

Bubble Sort: Mã giả

- Bước 1: Khởi gán i = 0; // lần xử lý đầu tiên
- Bước 2: Khởi gán j = N-1;//Duyệt từ cuối dãy ngược về vị trí i

  Trong khi (j > i) thực hiện:

  Nếu a[j]<a[j-1] thì

  Swap(a[j], a[j-1]);

  j = j-1;
- Bước 3: i = i+1; // lần xử lý kế tiếp

  Nếu i >=N-1: Hết dãy. Dừng

  Ngược lại: Lặp lại Bước 2.

Bubble Sort: Code C/C++

```
void Swap(int &lhs, int &rhs) {
   int tmp = lhs;
   lhs = rhs;
   rhs = tmp;
}

void BubbleSort(int a[], int n) {
   for (int i = 0; i < n-1; i++)
       for (int j = n-1; j > i; j--)
          if (a[j] < a[j-1]) // nếu sai vị trí thì
    đổi chỗ
           Swap(a[j], a[j-1]);
}
```

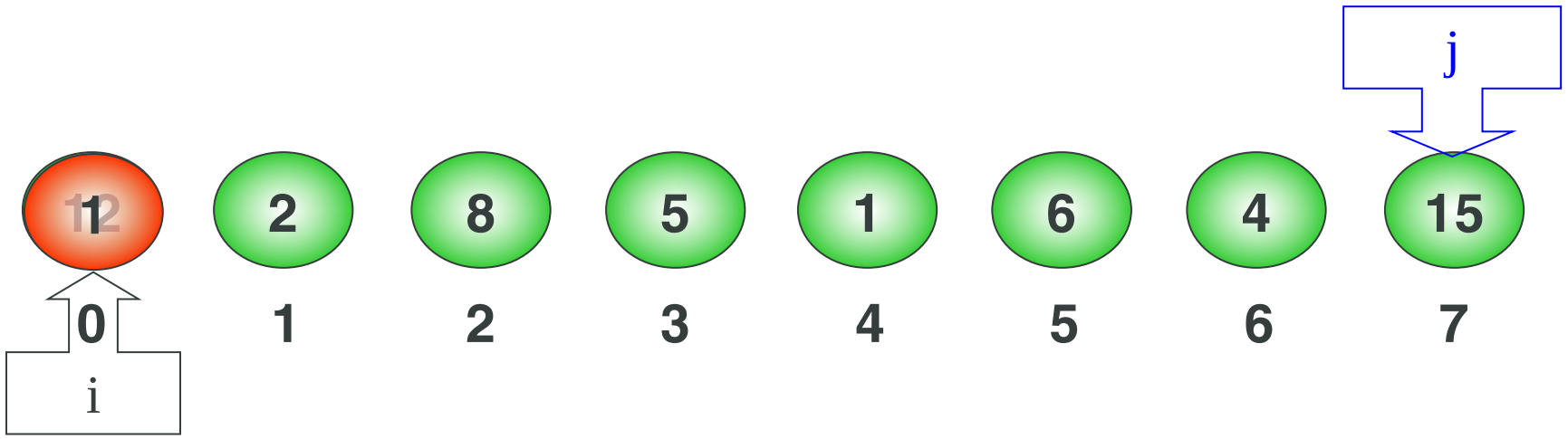
Bubble Sort: Code C/C++ (Cải tiến)

```
void Swap(int &lhs, int &rhs) {
   int tmp = lhs;
   lhs = rhs;
   rhs = tmp;

}
```

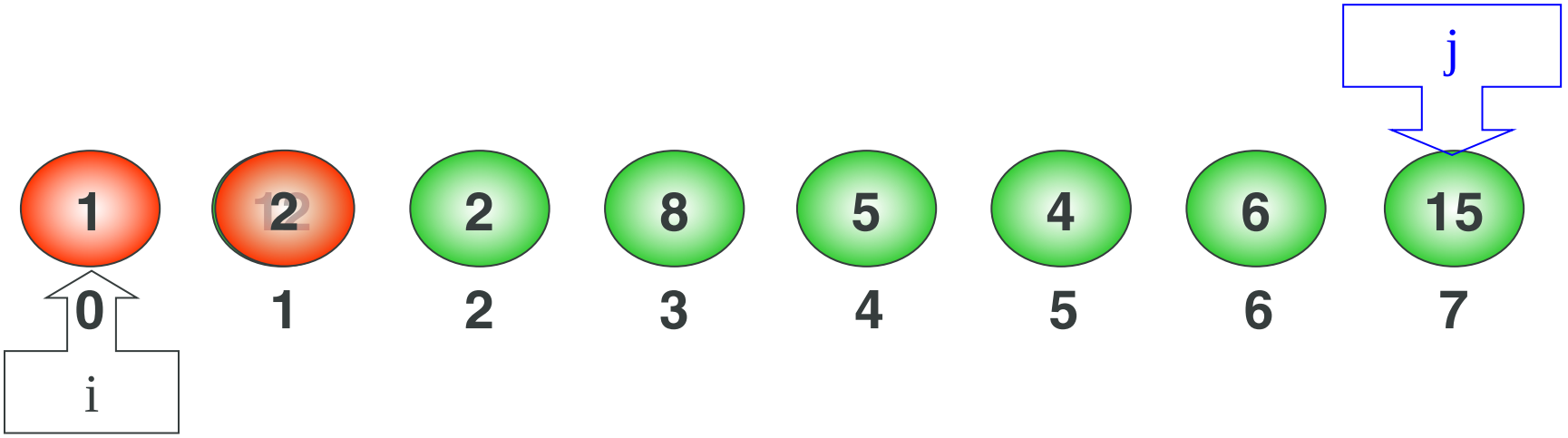
```
void BubbleSort(int a[], int n) {
   bool isSwap;
   for (int i = 0; i < n - 1; i++) {
      isSwap = false;
      for (int j = n - 1; j > i; j--)
         if (a[j] < a[j - 1]) {
            Swap(a[j], a[j - 1]);
            isSwap = true;
         }
      if (!isSwap) break;
   }

}
```

VD minh họa (http://techieme.in/improving-bubble-sort/)

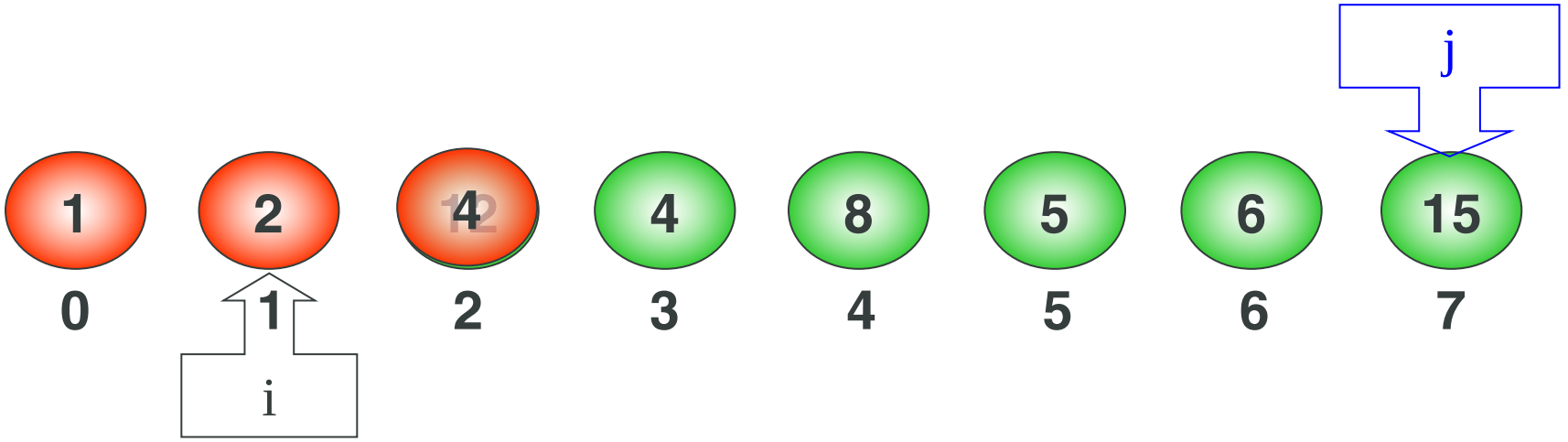
Bubble Sort: Minh họa



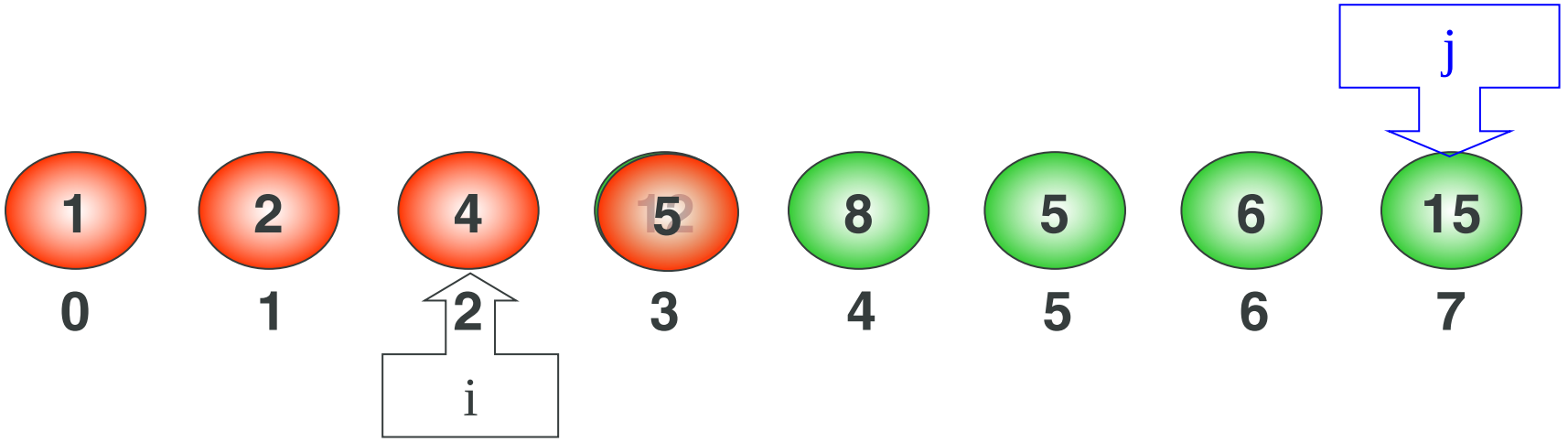
Bubble Sort: Minh họa (tt)



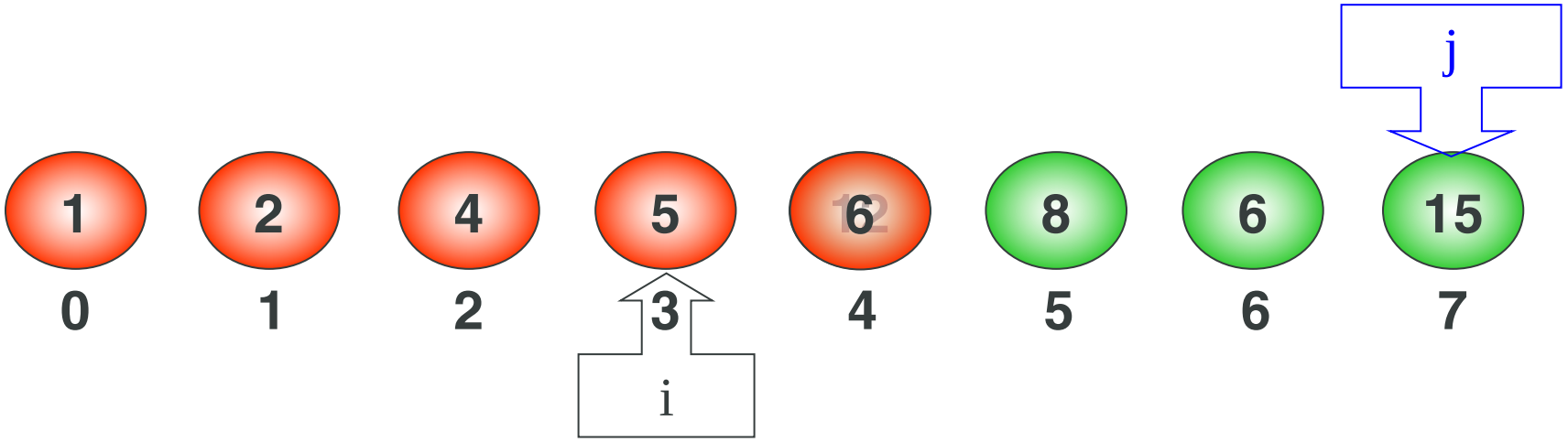
Bubble Sort: Minh họa (tt)



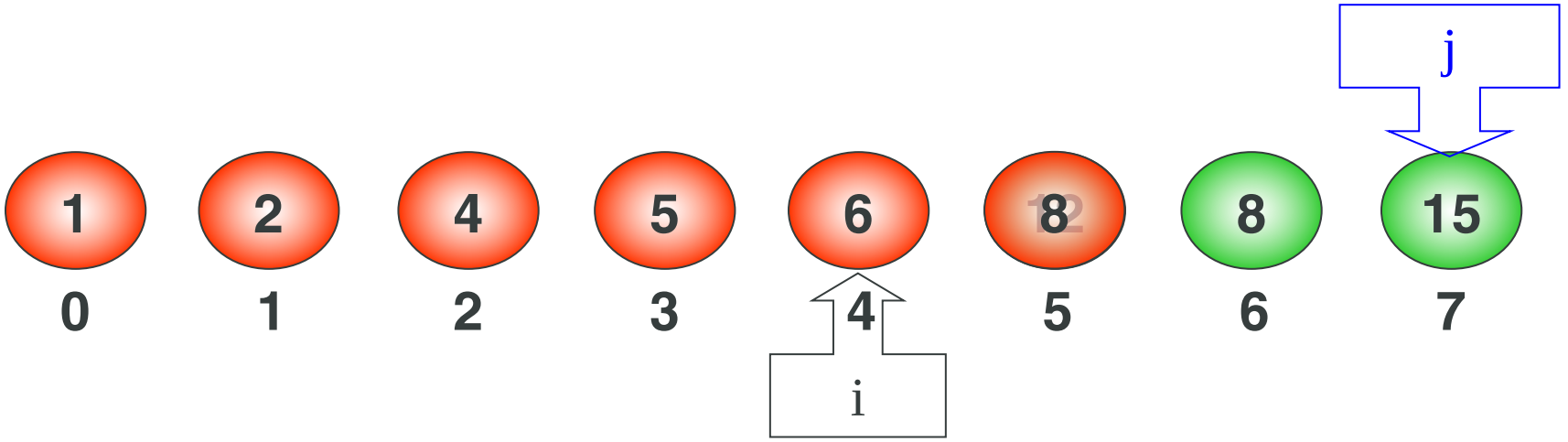
Bubble Sort: Minh họa (tt)



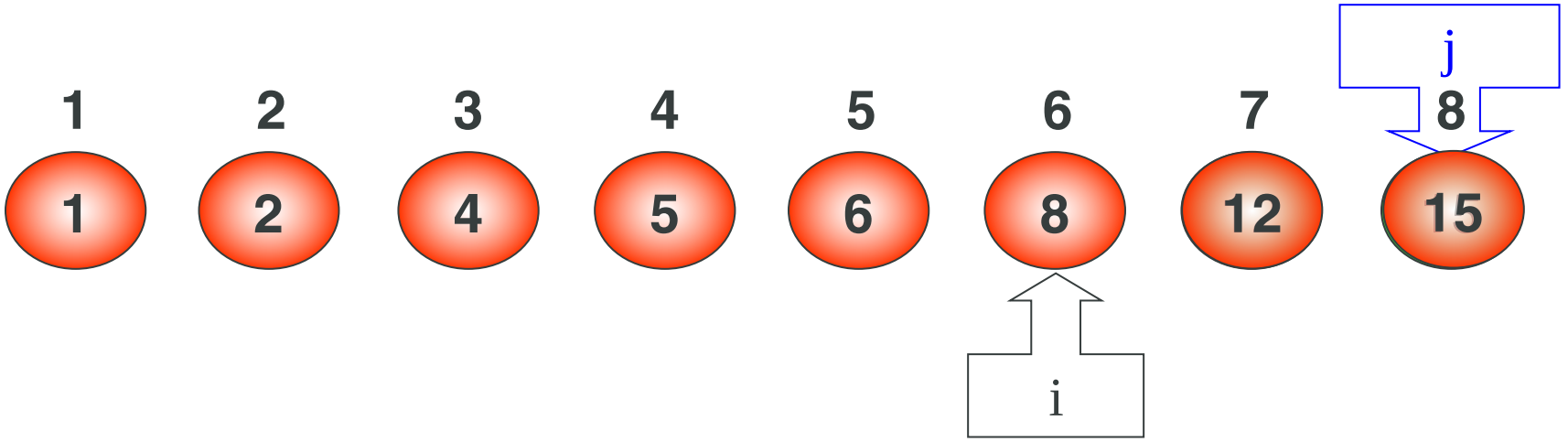
Bubble Sort: Minh họa (tt)



Bubble Sort: Minh họa (tt)



Bubble Sort: Minh họa (tt)



Bubble Sort: Độ phức tạp

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

Bubble Sort – Analysis

- **Best-case:** ➔ **O(n)**
  - Array is already sorted in ascending order.
  - The number of moves: 0 ➔ O(1)
  - The number of key comparisons: (n-1) ➔ O(n)
- **Worst-case:** ➔ **$O(n^2)$**
  - Array is in reverse order:
  - Outer loop is executed n-1 times,
  - The number of moves: $3*(1+2+...+n-1) = 3 * n*(n-1)/2$ ➔ $O(n^2)$
  - The number of key comparisons: $(1+2+...+n-1)= n*(n-1)/2$ ➔ $O(n^2)$
- Average-case: ➔ **$O(n^2)$**
  - We have to look at all possible initial data organizations.
- **So, Bubble Sort is $O(n^2)$**

Bubble sort: Rabbits and Turtles

Ví dụ 1: Sắp xếp mảng sau bằng bubble sort:

2 3 4 5 1

Các bước:

- Bước 1: 1 2 3 4 5

=> Số 1 trong mảng được gọi là phần tử Rabbit

Bubble sort: Rabbits and Turtles

Ví dụ 2: Sắp xếp mảng sau bằng bubble sort:

5 1 2 3 4

Các bước:

- Bước 1: 1 5 2 3 4
- Bước 2: 1 2 5 3 4
- Bước 3: 1 2 3 5 4
- Bước 4: 1 2 3 4 5

=> Số 5 trong mảng được gọi là phần tử Turtle

Bubble sort: Rabbits and Turtles

Ví dụ 3: Sắp xếp mảng sau bằng bubble sort:

5 1 2 3 4

Các bước:

- Bước 1: 1 5 2 3 4
- Bước 2: 1 2 3 4 5

=> Số 5 trong mảng được gọi là phần tử Turtle

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp – Selection Sort

2. Đổi chỗ trực tiếp – Interchange Sort

3. Nổi bọt – Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp – Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Heap Sort

9. Quick Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Shaker Sort

- Trong 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 nhằm tiết kiệm các phép so sánh thừa.

Shaker Sort: Mã giả

- Bước 1: l=0; r=n-1; //đoạn l->r là đoạn cần được sắp xếp

  k=n;//ghi nhận vị trí k xảy ra hoán vị sau cùng để làm cơ sơ thu hẹp đoạn l->r

- Bước 2:

  Bước 2a:

  j=r;//đẩy phần tử nhỏ về đầu mảng

  Trong khi j>l

  Nếu a[j]<a[j-1] thì Doicho(a[j],a[j-1])

  j--;

  l=k;//loại phần tử đã có thứ tự ở đầu dãy

  Bước 2b: j=l

  Trong khi j<r

  nếu a[j]>a[j+1] thì Doicho(a[j],a[j+1])

  j++

  r=k;//loại các phần tử đã có thứ tự ở cuối dãy

- Bước 3: Nếu l<r lặp lại Bước 2

  Ngược lại: Dừng

Shaker Sort

```
void ShakerSort(int a[], int n) {
    int left, right, k;
      left = 0; right = n-1; k = n-1;
      while (left < right) {
        for (int j = right; j > left; j--)
            if (a[j] < a[j-1]) {
                Swap(a[j], a[j-1]); k = j;
            }
        left = k;
        for (int j = left; j < right; j++)
            if (a[j] > a[j + 1]) {
                Swap(a[j], a[j - 1]); k = j;
            }
        right = k;
      }
}
```

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort

3. Nổi bọt - Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp - Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Quick Sort

9. Heap Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Chèn trực tiếp - Insertion Sort

Từ khóa: Insertion Sort

Phân tích: Giả sử danh sách $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\Re$. Khi đó, để tạo danh sách A có n+1 phần tử có thứ tự $\Re$, cần tìm vị trí **k** để chèn phần tử a_n sao cho đảm bảo

$$a_i \Re a_n \ \forall i < k$$

Chèn trực tiếp - Insertion Sort

Ý tưởng:

Insertion Sort là một thuật toán sắp xếp đơn giản thích hợp cho mảng kích thước nhỏ. Ý tưởng:

- Một mảng được chia thành 2 đoạn liền kề nhau: Đoạn đã sắp xếp và đoạn chưa được sắp xếp.
- Trong mỗi lần chạy, phần tử đầu tiên của đoạn chưa sắp xếp được chọn để chèn vào vị trí thích hợp trong đoạn đã được sắp xếp.
- Mảng có n phần tử sẽ tốn n-1 lần chạy để sắp xếp xong mảng.

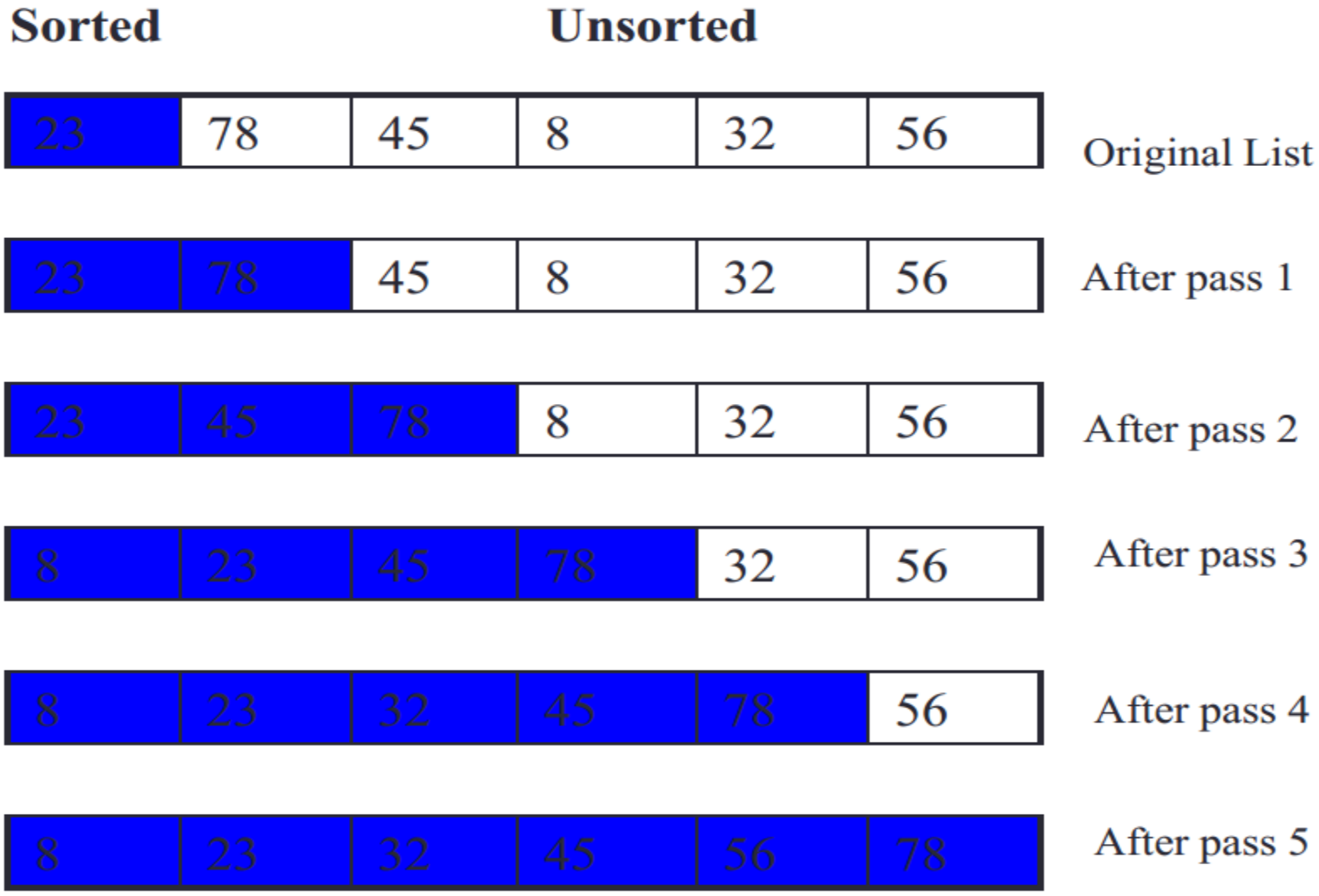
Chèn trực tiếp - Insertion Sort

Thuật toán:

Đầu vào: $A=\{a_0, a_1, .., a_{n-1}\}$ chưa có thứ tự $\Re$

Đầu ra: $A=\{a_0, a_1, .., a_{n-1}\}$ đã có thứ tự $\Re$

Insertion Sort:Ví dụ



Insertion Sort: Mã giả

- Bước 1: Khởi gán i = 1; //giả sử có đoạn a[1] đã được sắp
- Bước 2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào
- Bước 3: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]
- Bước 4: a[pos] = x; //có đoạn a[1]..a[i] đã được sắp
- Bước 5: i = i+1;

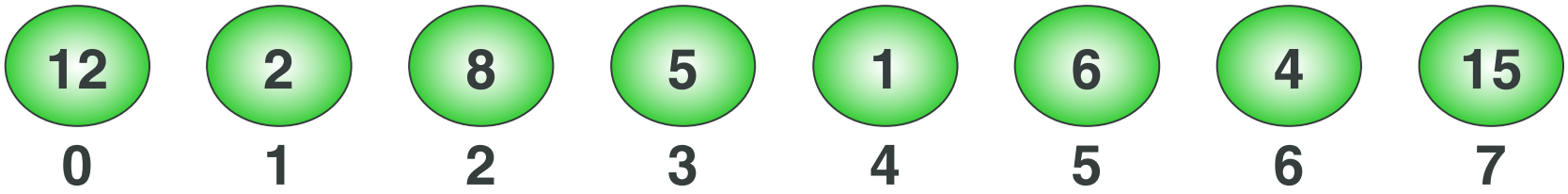
  Nếu i < n: Lặp lại Bước 2

  Ngược lại: Dừng

Insertion Sort: Code C/C++

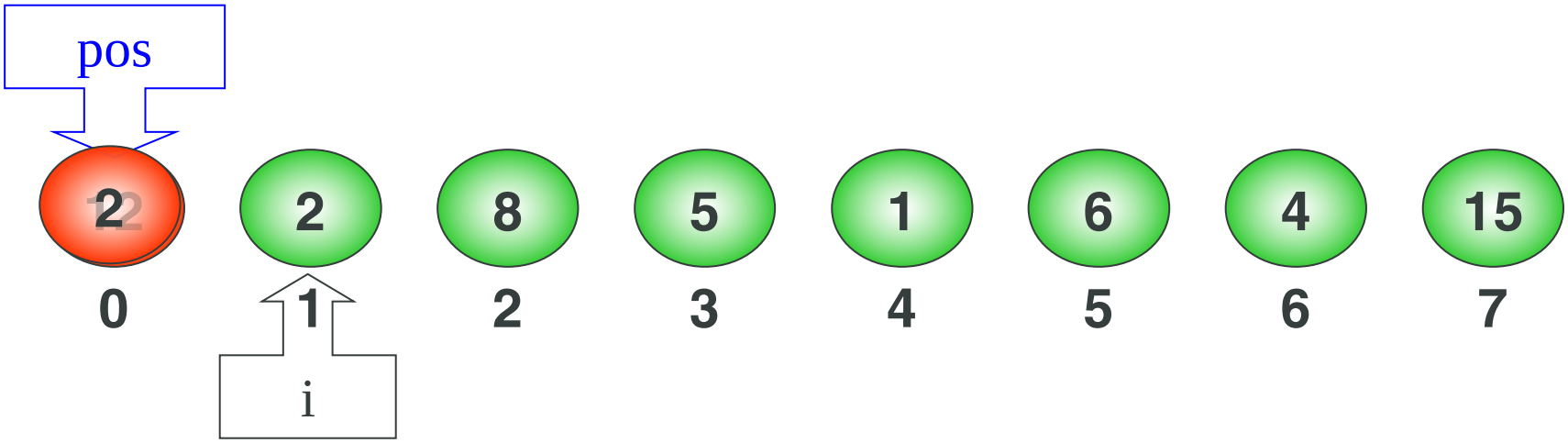
```
void InsertionSort(int a[], int n) {
  int pos;
  int x; //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
  for (int i=1; i<n; i++) { //đoạn a[0] đã sắp
      x = a[i]; pos = i-1; // tìm vị trí chèn x
      while ((pos >= 0) && (a[pos] > x)) {
          //kết hợp dời chỗ các phần tử sẽ đứng sau x trong dãy mới
          a[pos + 1] = a[pos];
          pos--;
      }
      a[pos+1] = x; // chèn x vào dãy
  }
}
```

Insertion Sort: Minh họa



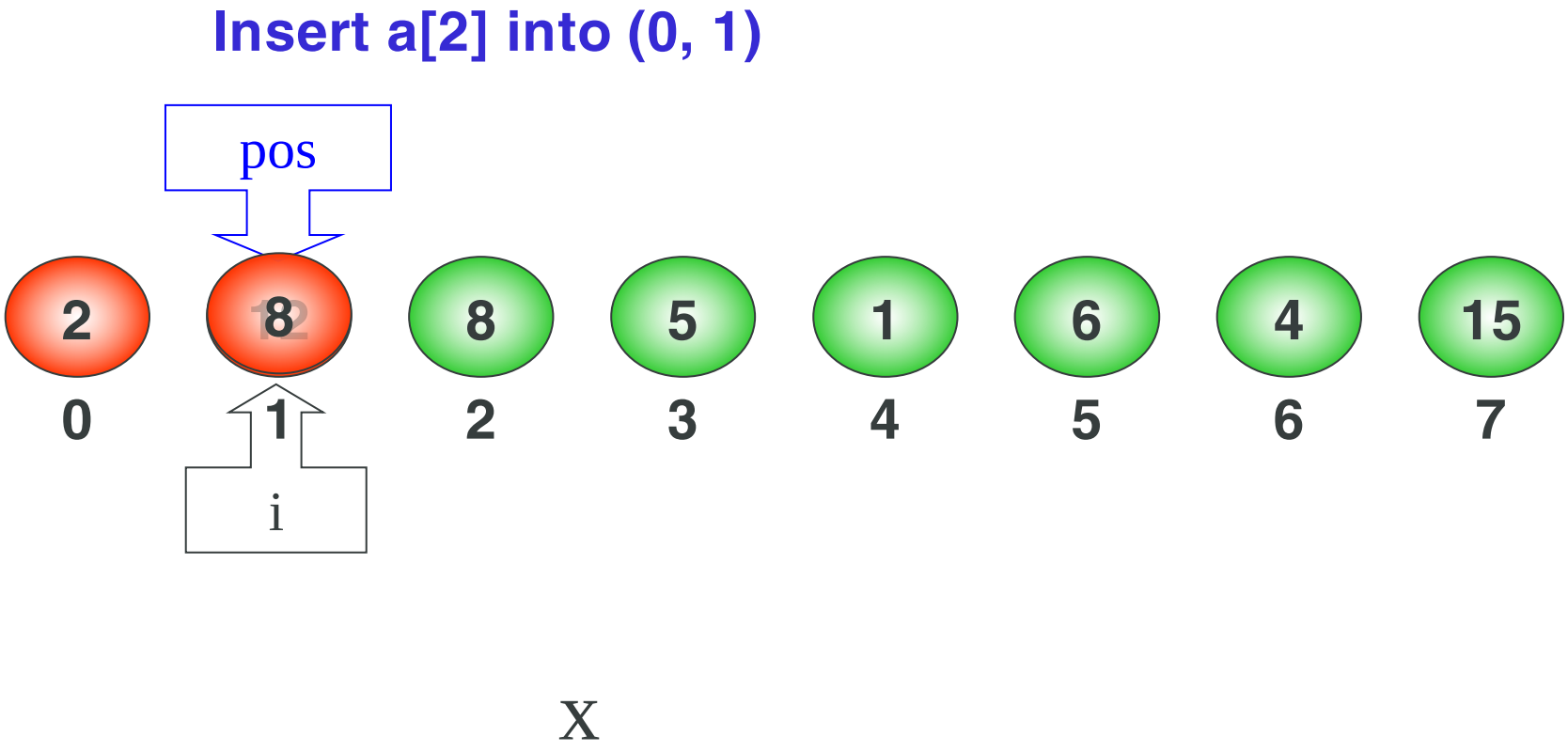
Insertion Sort: Minh họa (tt)

Insert a[1] into (0,0)



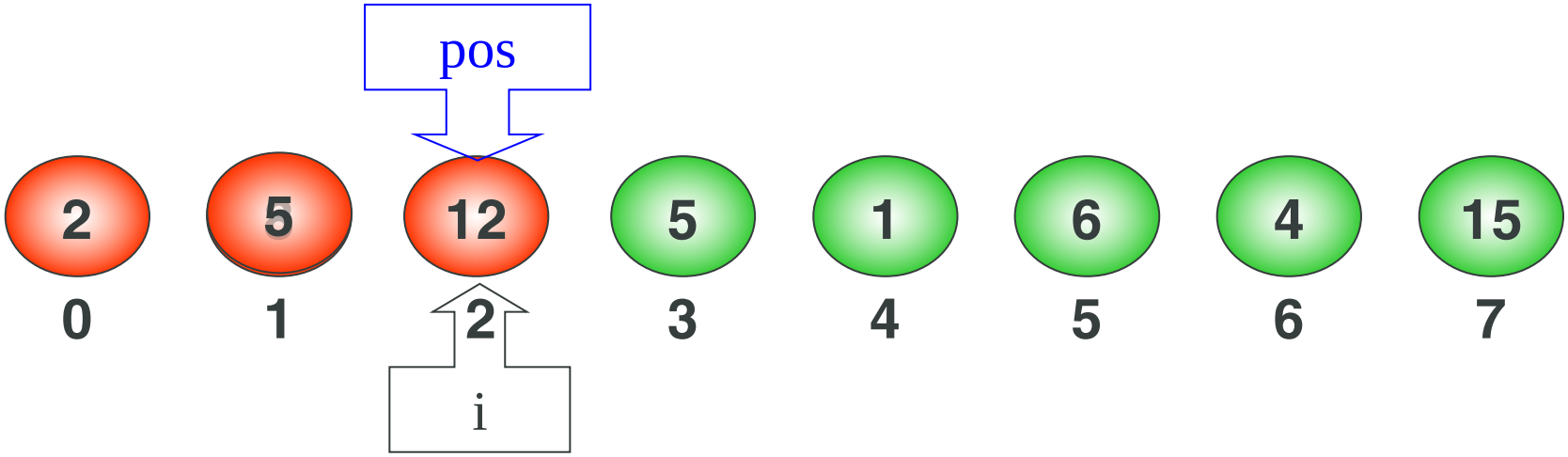
X

Insertion Sort: Minh họa (tt)



Insertion Sort: Minh họa (tt)

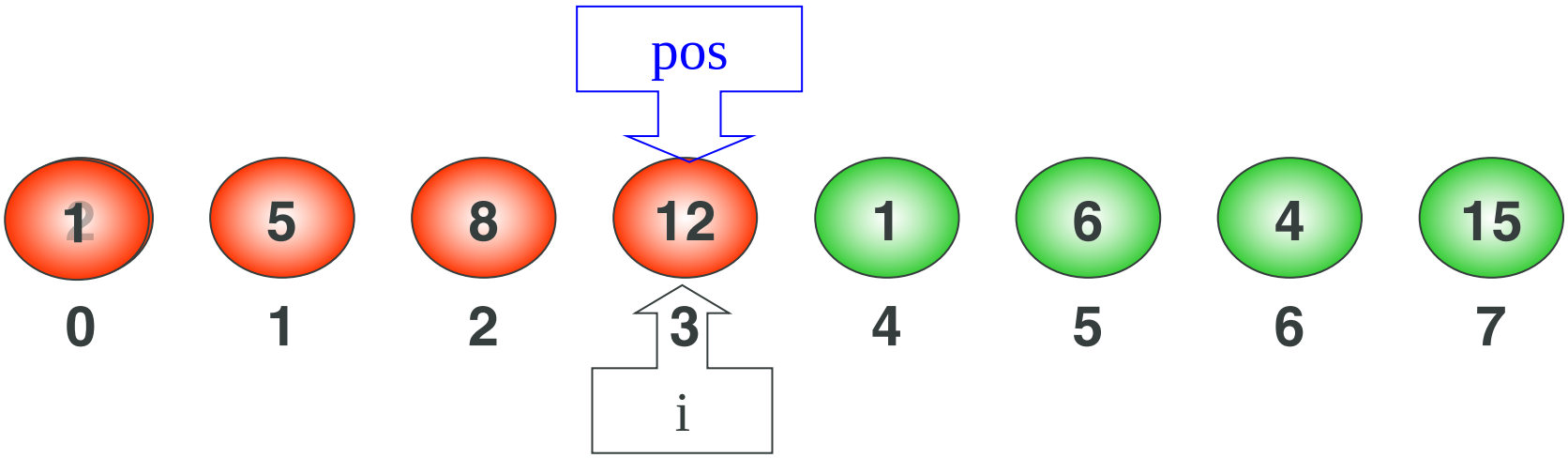
Insert a[3] into (0, 2)



X

Insertion Sort: Minh họa (tt)

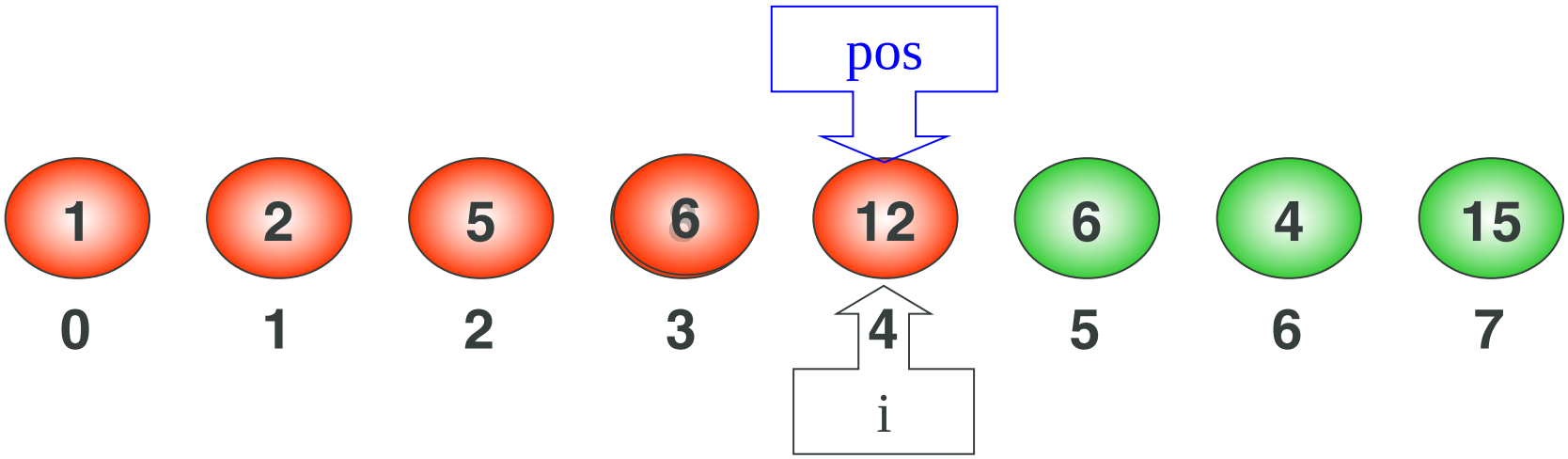
Insert a[4] into (0, 3)



X

Insertion Sort: Minh họa (tt)

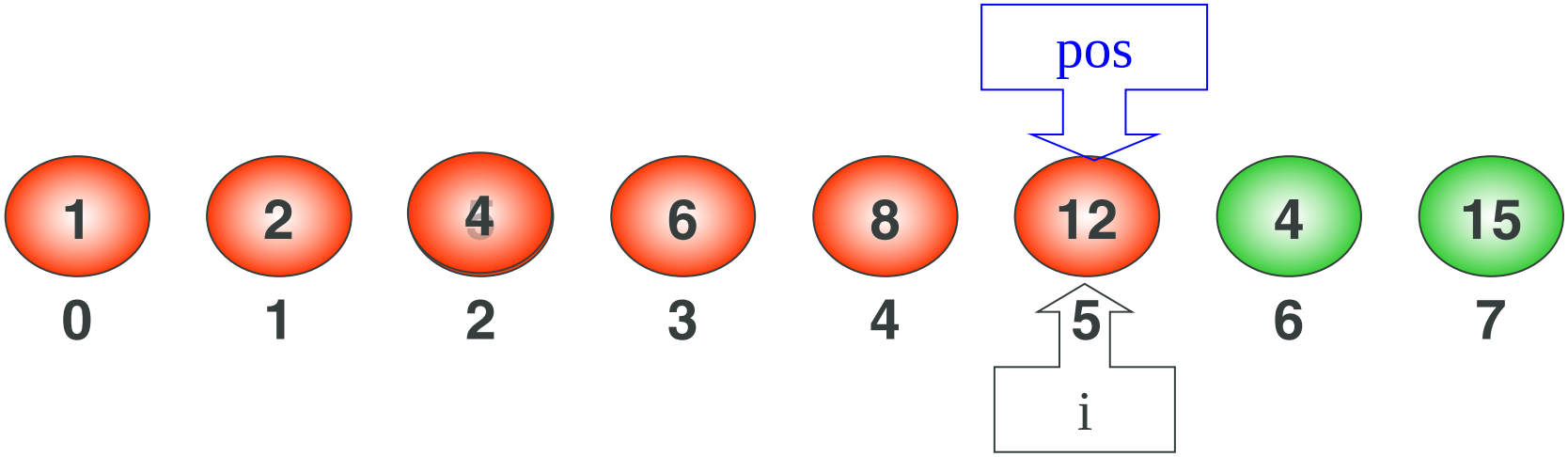
Insert a[5] into (0, 4)



X

Insertion Sort: Minh họa (tt)

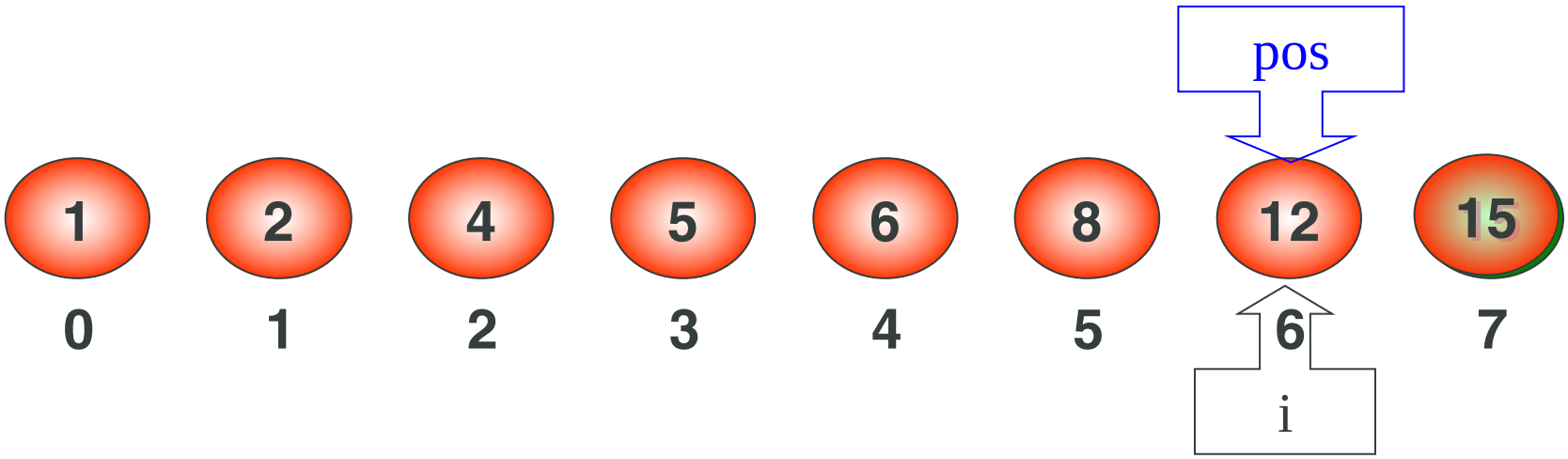
Insert a[6] into (0, 5)



X

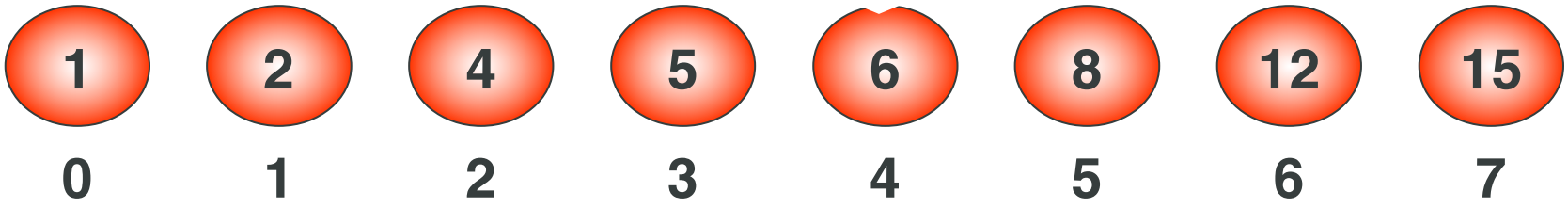
Insertion Sort: Minh họa (tt)

Insert a[8] into (0, 6)



X

Insertion Sort: Minh họa (tt)



Insertion Sort: Ví dụ

- Chạy ví dụ sau:

1 2 3 6 5 7 8 9

Insertion Sort: Độ phức tạp

Trường hợp	Số lần 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	$\sum_{i=1}^{n-1} (i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1$

Trường hợp	Độ phức tạp
Tốt nhất	O(n)
Trung bình	O(n^2)
Xấu nhất	O(n^2)

Insertion Sort – Analysis

- Running time depends on not only the size of the array but also the contents of the array.
- ***Best-case:*** ➔ **O(n)**
  - Array is already sorted in ascending order.
  - Inner loop will not be executed.
  - The number of moves: 2*(n-1) ➔ O(n)
  - The number of key comparisons: (n-1) ➔ O(n)
- ***Worst-case:*** ➔ **$O(n^2)$**
  - Array is in reverse order:
  - Inner loop is executed i-1 times, for i = 2,3, …, n
  - The number of moves: 2*(n-1)+(1+2+…+n-1)= 2*(n-1)+ n*(n-1)/2 ➔ $O(n^2)$
  - The number of key comparisons: (1+2+…+n-1)= n*(n-1)/2 ➔ $O(n^2)$
- Average-case: ➔ **$O(n^2)$**
  - We have to look at all possible initial data organizations.
- **So, Insertion Sort is $O(n^2)$**

Analysis of insertion sort

- Which running time will be used to characterize this algorithm?
  - Best, worst or average?
- Worst:
  - Longest running time (this is the upper limit for the algorithm)
  - It is guaranteed that the algorithm will not be worse than this.
- Sometimes we are interested in average case. But there are some problems with the average case.
  - It is difficult to figure out the average case. i.e. what is average input?
  - Are we going to assume all possible inputs are equally likely?
  - In fact for most algorithms average case is same as the worst case.

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort

3. Nổi bọt - Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp - Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Heap Sort

9. Quick Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Binary Insertion Sort - Chèn Nhị phân: Code C/C++

```
void BInsertionSort(int a[], int n) {
  int l, r, m, i;
  int x; //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
  for (int i=1; i<n; i++) {
      x = a[i]; l = 0; r = i-1;
      while (l <= r) { // tìm vị trí chèn x
          m = (l+r)/2;
          if (x < a[m]) r = m-1;
          else l = m+1;
      }
      for (int j = i-1; j >= l; j--)
        a[j+1] = a[j];
      a[l] = x;
  }
}
```

// Ý nghĩa đoạn code:
// Tìm vị trí để chèn x bằng phương pháp tìm nhị phân
// Vị trí tìm được chính là giá trị l (left)

// Ý nghĩa đoạn code:
// Dời các phần tử lớn hơn x trong
// đoạn sorted sang phải 1 đơn vị.
// Sau đó chèn x vào vị trí l (left) tìm được ở đoạn trên.

```
// A binary search based function to find the position
// where x should be inserted in a[low..high]
int binarySearch(int a[], int low, int high, int x) {
    if (high <= low) return (x > a[low])?  (low + 1): low;
    int mid = (low + high)/2;
    if(x == a[mid]) return mid+1;
    if(x > a[mid])
        return binarySearch(a, mid+1, high, x);
    return binarySearch(a, low, mid-1, x);
}
void BInsertionSort(int a[], int n) {
    int i, j, k, x;
    for (i = 1; i < n; ++i) {
        x = a[i];
        pos = i - 1;
        // find location where selected sould be inseretd
        k = binarySearch(a, 0, pos, x);
        // Move all elements after location to create space
        while (pos >= k) { a[pos+1] = a[pos]; pos--; }
        a[pos+1] = x;
    }
}
```

Câu hỏi và Bài tập

1. Trình bày ý tưởng của 2 thuật toán sắp xếp chọn trực tiếp và chèn trực tiếp?

2. Hãy trình bày những ưu khuyết điểm của 2 thuật toán sắp xếp chọn trực tiếp và chèn trực tiếp? Theo bạn cách khắc phục những nhược điểm này là như thế nào?

3. Nêu những điểm giống nhau và khác nhau của 3 thuật toán Selection sort, Interchange sort, Bubble sort.

4. Sử dụng hàm tạo ngẫu nhiên trong C/C++ để tạo ra một dãy có 100,000 số nguyên. Vận dụng các thuật toán sắp xếp đã học để sắp xếp các phần tử của mảng theo thứ tự tăng dần về mặt giá trị. Với cùng một dữ liệu như nhau, cho biết thời gian thực hiện các thuật toán?

Nội dung

- Các giải thuật sắp xếp nội:

1. Chọn trực tiếp - Selection Sort

2. Đổi chỗ trực tiếp - Interchange Sort

3. Nổi bọt - Bubble Sort

4. Shaker Sort

5. Chèn trực tiếp - Insertion Sort

6. Chèn nhị phân - Binary Insertion Sort

7. Shell Sort

8. Quick Sort

9. Heap Sort

10. Merge Sort

11. Counting Sort

12. Radix Sort

Shell Sort: Ý tưởng

- Cải tiến của phương pháp Chèn trực tiếp

- Ý tưởng:
  - Phân hoạch dãy thành các dãy con
  - Sắp xếp các dãy con theo phương pháp chèn trực tiếp
  - Dùng phương pháp chèn trực tiếp sắp xếp lại cả dãy.

Shell Sort

- Phân chia dãy ban đầu thành những dãy con gồm các phần tử ở cách nhau **h** vị trí
- Dãy ban đầu : $a_0, a_1, ..., a_{N-1}$ được xem như sự xen kẽ của các dãy con sau :
  - Dãy con thứ nhất : $a_0\ a_h\ a_{2h}$ ...
  - Dãy con thứ hai : $a_1\ a_{h+1}\ a_{2h+1}$ ...
  - ....
  - Dãy con thứ h : $a_{h-1}\ a_{2h-1}\ a_{3h-1}$ ...

Shell Sort

- Tiến hành sắp xếp các phần tử trong cùng dãy con sẽ làm cho các phần tử được đưa về vị trí đúng tương đối
- Giảm khoảng cách **h** để tạo thành các dãy con mới
- Dừng khi h=1

Shell Sort

- Giả sử quyết định sắp xếp **k** bước, các khoảng cách chọn phải thỏa điều kiện:

  $h_i > h_{i+1}$ và $h_k = 1$

- $h_i = (h_{i-1} - 1)/3$ và $h_k = 1, k = \log_3 n - 1$

  Ví dụ: 127, 40, 13, 4, 1

- $h_i = (h_{i-1} - 1)/2$ và $h_k = 1, k = \log_2 n - 1$

  Ví dụ: 15, 7, 3, 1

Shell Sort

- h có dạng 3i+1: 364, 121, 40, 13, 4, 1
- Dãy fibonaci: 34, 21, 13, 8, 5, 3, 2, 1
- h là dãy các số nguyên tố giảm dần đến 1: 13, 11, 7, 5, 3, 1.

Shell Sort: Mã giả

- Bước 1: Chọn **k** khoảng cách h[0], h[1], ..., h[k-1];

  i = 1;

- Bước 2: Phân chia dãy ban đầu thành các dãy con cách nhau h[i] khoảng cách.

  Sắp xếp từng dãy con bằng phương pháp chèn trực tiếp;

- Bước 3: i = i+1;

  Nếu i > k: Dừng

  Ngược lại: Lặp lại Bước 2.

Shell Sort: Minh họa

- Cho dãy số a:

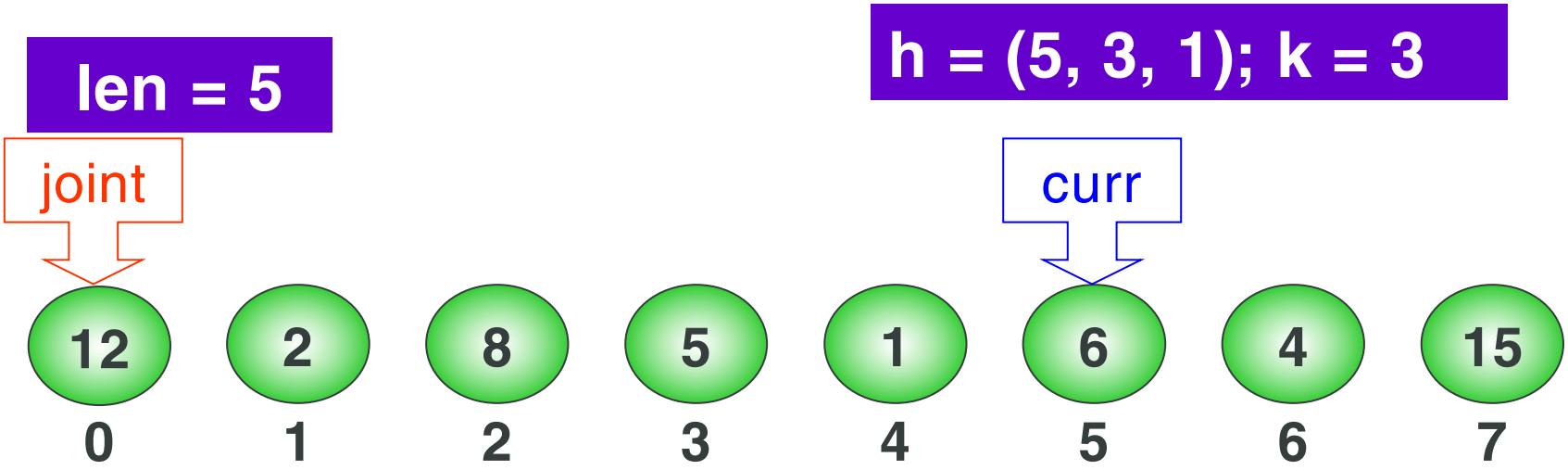
12 2 8 5 1 6 4 15

- Giả sử chọn các khoảng cách là 5, 3, 1

Shell Sort: Code C/C++

```
void ShellSort(int a[], int n, int h[], int k) {
  int step, i, j, x, len;
  for (step = 0; step < k; step++) {
      len = h[step];
      for (i = len; i < n; i++) {
          x = a[i];
          j = i-len; // a[j] đứng kề trước a[i] trong cùng dãy con
          while (j >= 0 && x<a[j]) { // sắp xếp dãy con chứa  x
              // bằng phương pháp chèn trực tiếp
              a[j+len] = a[j];
              j = j-len;
          }
          a[j+len] = x;
      }
  }
}
```

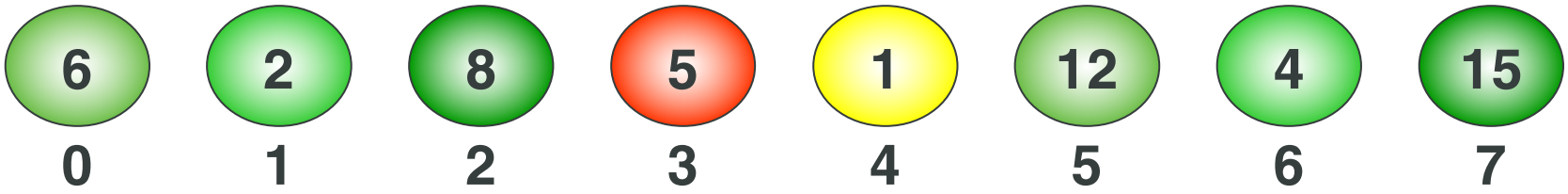
Shell Sort: Minh họa



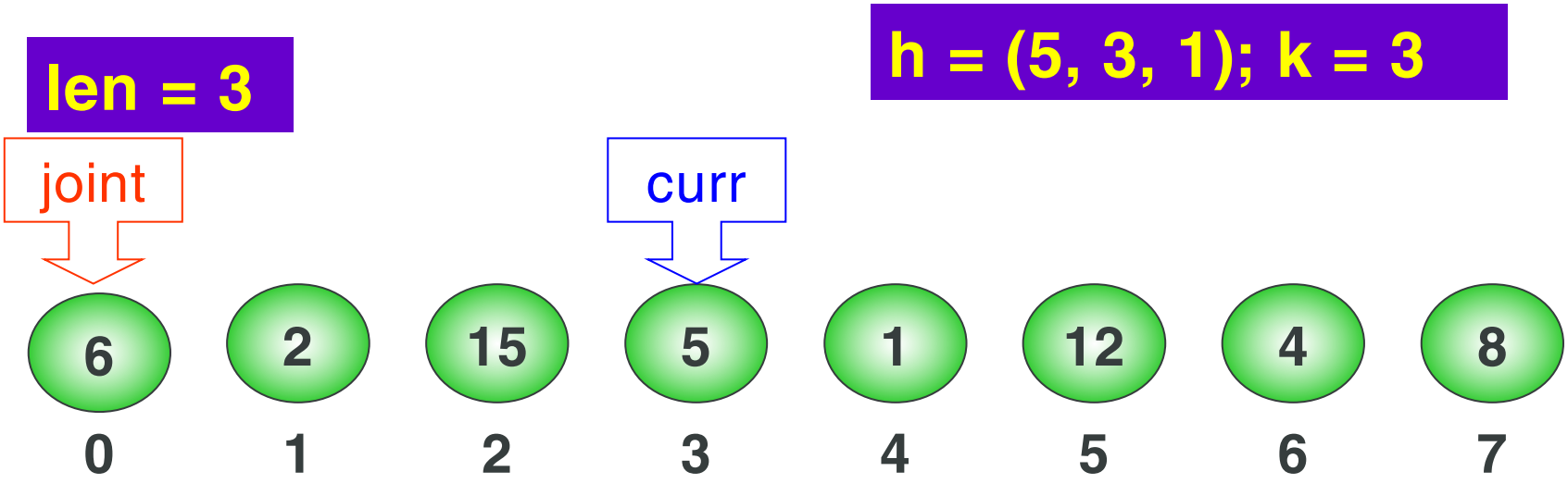
Shell Sort: Minh họa (tt)

len = 5;

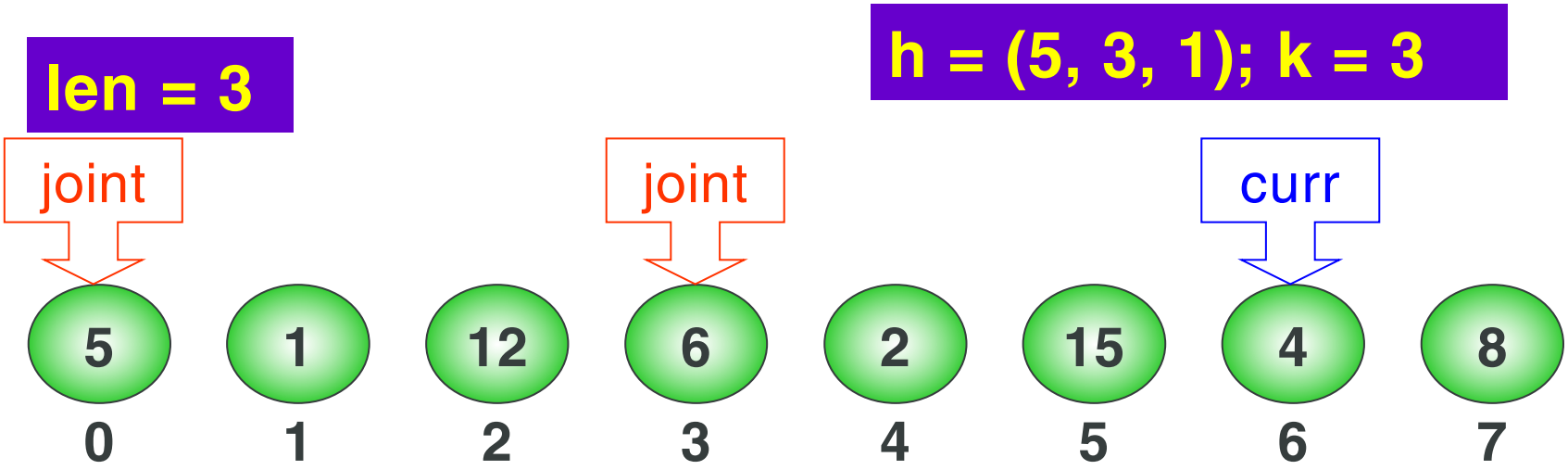
h = (5, 3, 1); k = 3



Shell Sort: Minh họa (tt)



Shell Sort: Minh họa (tt)

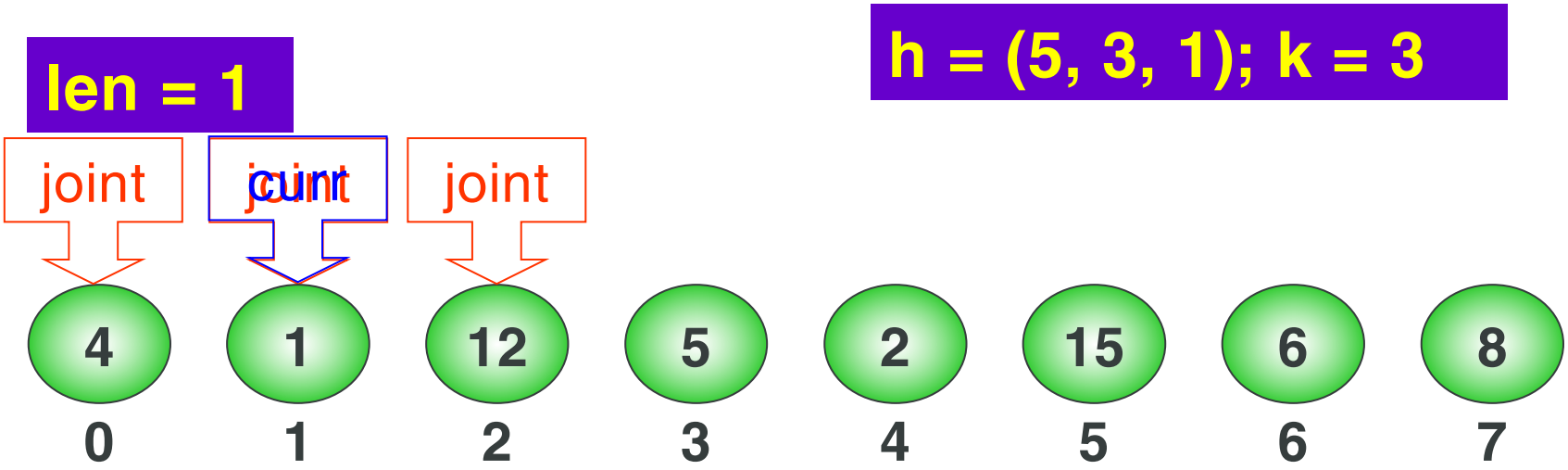


Shell Sort: Minh họa (tt)

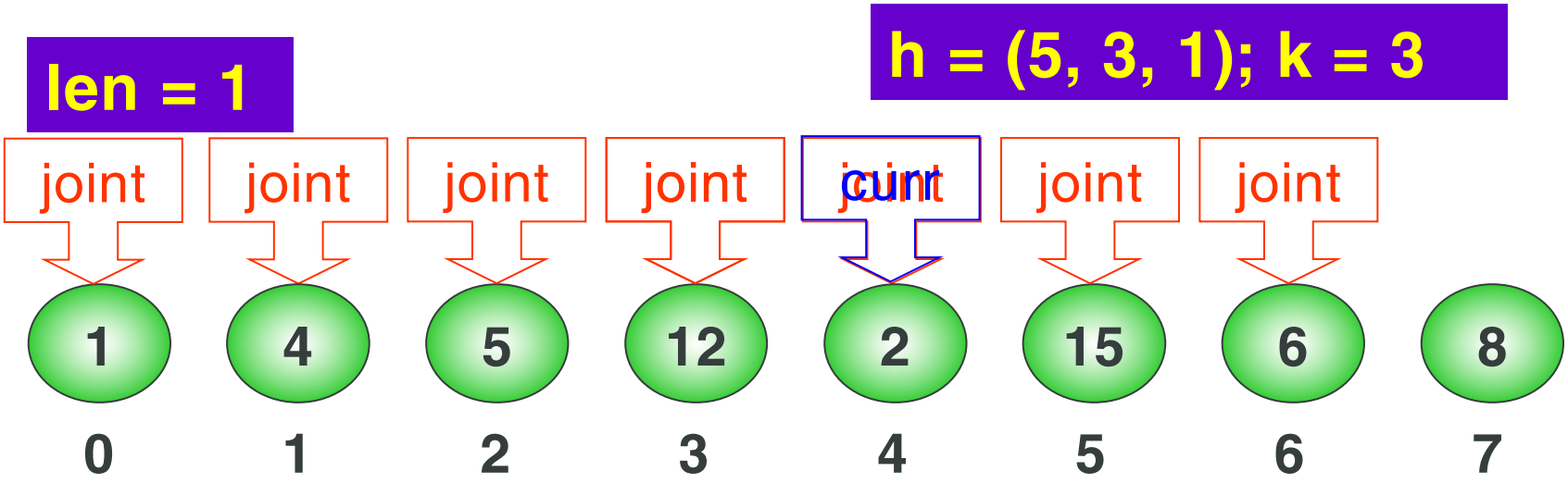
len = 3

h = (5, 3, 1); k = 3

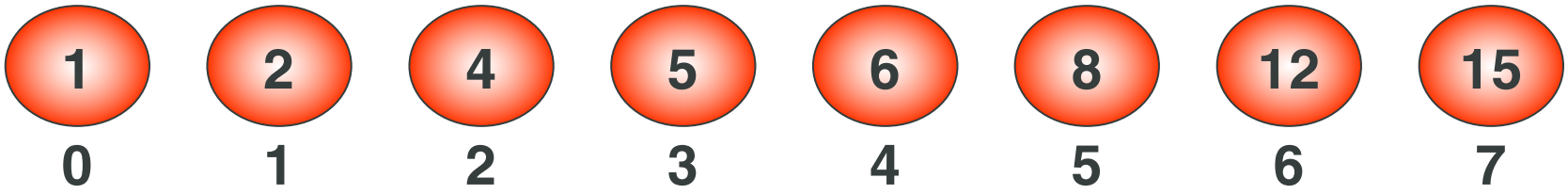
Shell Sort: Minh họa (tt)



Shell Sort: Minh họa (tt)



Shell Sort: Minh họa (tt)



Chúc các em học tốt!

Chúc các em học tốt!