

## TÌM KIẾM VÀ SẮP XẾP NỘI



- Các giải thuật tìm kiếm nội
  - 1. Tìm kiếm tuyến tính
  - 2. Tìm kiếm nhị phân
- Các giải thuật sắp xếp nội
  - 1. Đổi chỗ trực tiếp – Interchange Sort
  - 2. Chọn trực tiếp – Selection Sort
  - 3. Nổi bọt – Bubble Sort



4. Chèn trực tiếp – Insertion Sort
5. Chèn nhị phân – Binary Insertion Sort
6. Shaker Sort
7. Shell Sort
8. Heap Sort
9. Quick Sort
10. Merge Sort
11. Radix Sort



# Bài Toán Tìm Kiếm

- Cho danh sách có  $n$  phần tử  $a_0, a_1, a_2, \dots, a_{n-1}$ .
- Để đơn giản trong việc trình bày giải thuật ta dùng mảng 1 chiều  $a$  để lưu danh sách các phần tử nói trên trong bộ nhớ chính.
- Tìm phần tử có khoá bằng  $X$  trong mảng
  - Giải thuật tìm kiếm tuyến tính (tìm tuần tự)
  - Giải thuật tìm kiếm nhị phân
- ❖ **Lưu ý:** Trong quá trình trình bày thuật giải ta dùng ngôn ngữ lập trình C.



# Tìm Kiếm Tuyến Tính

- **Ý tưởng** : So sánh  $X$  lần lượt với phần tử thứ 1, thứ 2,... của mảng  $a$  cho đến khi gặp được khóa cần tìm, hoặc tìm hết mảng mà không thấy.
- **Các bước tiến hành**
  - Bước 1: Khởi gán  $i=0$ ;
  - Bước 2: So sánh  $a[i]$  với giá trị  $x$  cần tìm, có 2 khả năng
    - +  $a[i] == x$  tìm thấy  $x$ . Dừng;
    - +  $a[i] != x$  sang bước 3;
  - Bước 3:  $i=i+1$  // Xét tiếp phần tử kế tiếp trong mảng
    - Nếu  $i==N$ : Hết mảng. Dừng;
    - Ngược lại: Lặp lại bước 2;



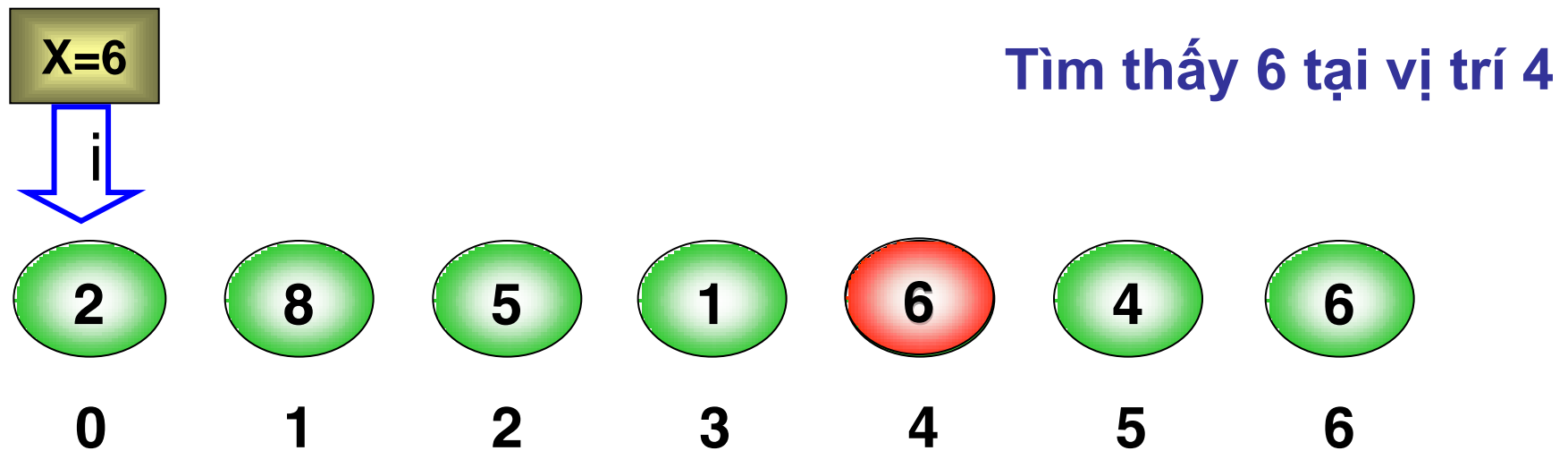
# Thuật Toán Tìm Kiếm Tuyến Tính

- Hàm trả về 1 nếu tìm thấy, ngược lại trả về 0:

```
int LinearSearch(int a[],int n, int x)
{
    int i=0;
    while((i<n)&&(a[i]!=x))
        i++;
    if(i==n)
        return 0; //Tìm không thấy x
    else
        return 1; //Tìm thấy
}
```

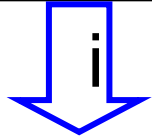


# Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính



# Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính (tt)

$X=10$



2

0

8

1

5

2

1

3

6

4

4

5

6

6

$i=7$ , không tìm thấy





# Đánh Giá Thuật Toán Tìm Tuyến Tính

Trường hợp	Css
Tốt nhất	1
Xấu nhất	N
Trung bình	$(N+1) / 2$

➤ Độ phức tạp  $O(N)$



# Cải Tiến Thuật Toán Tìm Tuyến Tính

- Nhận xét: Số phép so sánh của thuật toán trong trường hợp xấu nhất là  $2 \cdot n$ .
- Để giảm thiểu số phép so sánh trong vòng lặp cho thuật toán, ta thêm phần tử “lính canh” vào cuối dãy.

```
int LinearSearch(int a[], int n, int x)
{
    int i=0; a[n]=x; // a[n] là phần tử “lính canh”
    while(a[i]!=x)
        i++;
    if(i==n)
        return 0; // Tìm không thấy x
    else
        return 1; // Tìm thấy
}
```



# Thuật Toán Tìm Kiếm Nhị Phân

- Được áp dụng trên mảng đã có thứ tự.
- **Ý tưởng:** .
  - Giả sử ta xét mảng có thứ tự tăng, khi ấy ta có  $a_{i-1} < a_i < a_{i+1}$
  - Nếu  $X > a_i$  thì X chỉ có thể xuất hiện trong đoạn  $[a_{i+1}, a_{n-1}]$
  - Nếu  $X < a_i$  thì X chỉ có thể xuất hiện trong đoạn  $[a_0, a_{i-1}]$
  - Ý tưởng của giải thuật là tại mỗi bước ta so sánh X với phần tử đứng giữa trong dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này mà ta quyết định giới hạn dãy tìm kiếm ở nửa dưới hay nửa trên của dãy tìm kiếm hiện hành.



# Các Bước Thuật Toán Tìm Kiếm Nhị Phân

- Giả sử dãy tìm kiếm hiện hành bao gồm các phần tử nằm trong  $a_{\text{left}}$ ,  $a_{\text{right}}$ , các bước của giải thuật như sau:
- Bước 1:  $\text{left}=0$ ;  $\text{right}=N-1$ ;
- Bước 2:
  - $\text{mid}=(\text{left}+\text{right})/2$ ; //chỉ số phần tử giữa dãy hiện hành
  - So sánh  $a[\text{mid}]$  với  $x$ . Có 3 khả năng
    - $a[\text{mid}]=x$ : tìm thấy. Dừng
    - $a[\text{mid}]>x$  :  $\text{Right}=\text{mid}-1$ ;
    - $a[\text{mid}]<x$  :  $\text{Left}=\text{mid}+1$ ;
- Bước 3: Nếu  $\text{Left} \leq \text{Right}$  ; // còn phần tử trong dãy hiện hành
  - + Lặp lại bước 2
  - Ngược lại : Dừng



# Cài Đặt Thuật Toán Tìm Nhị Phân

- Hàm trả về giá trị 1 nếu tìm thấy, ngược lại hàm trả về giá trị 0

```
int BinarySearch(int a[],int n,int x)
{   int left, right, mid; left=0; right=n-1;
    do{
        mid=(left+right)/2;
        if(a[mid]==x) return 1;
        else if(a[mid]<x) left=mid+1;
        else right=mid-1;
    }while(left<=right);
    return 0;
}
```



# Đánh Giá Thuật Toán Tìm Tuyến Tính

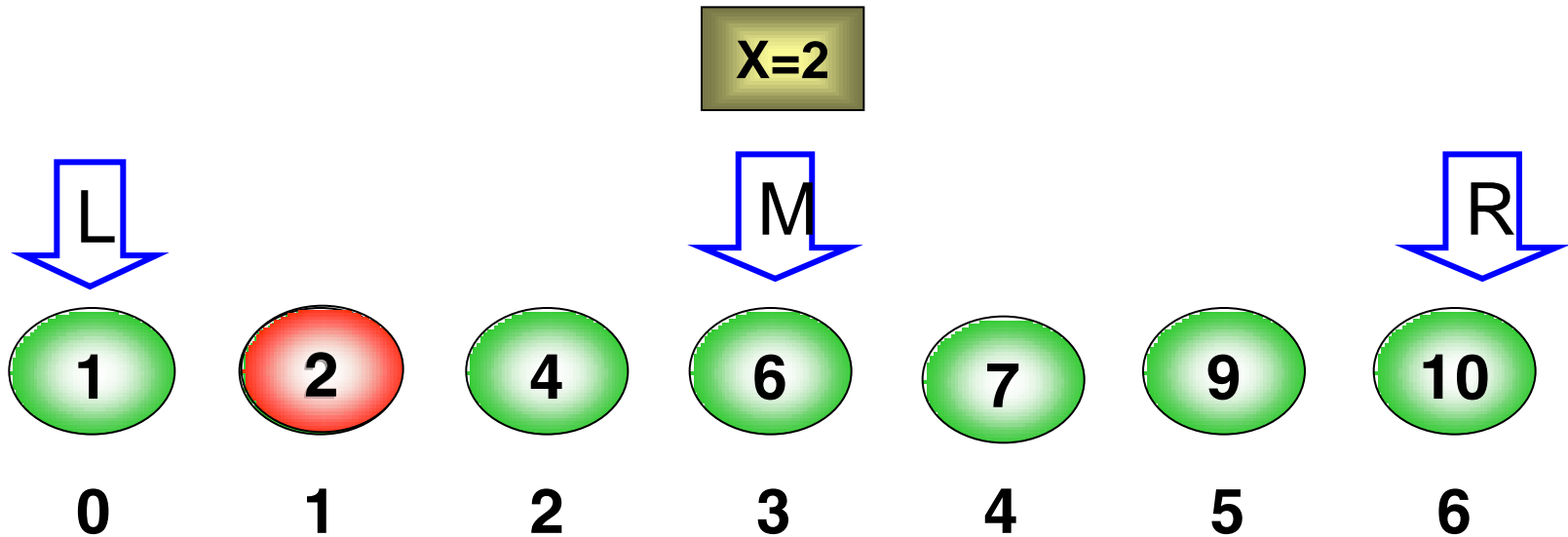
Trường hợp	Css
Tốt nhất	1
Xấu nhất	$\log_2 N$
Trung bình	$\log_2 N / 2$

➤ Độ phức tạp  $O(\log_2 N)$

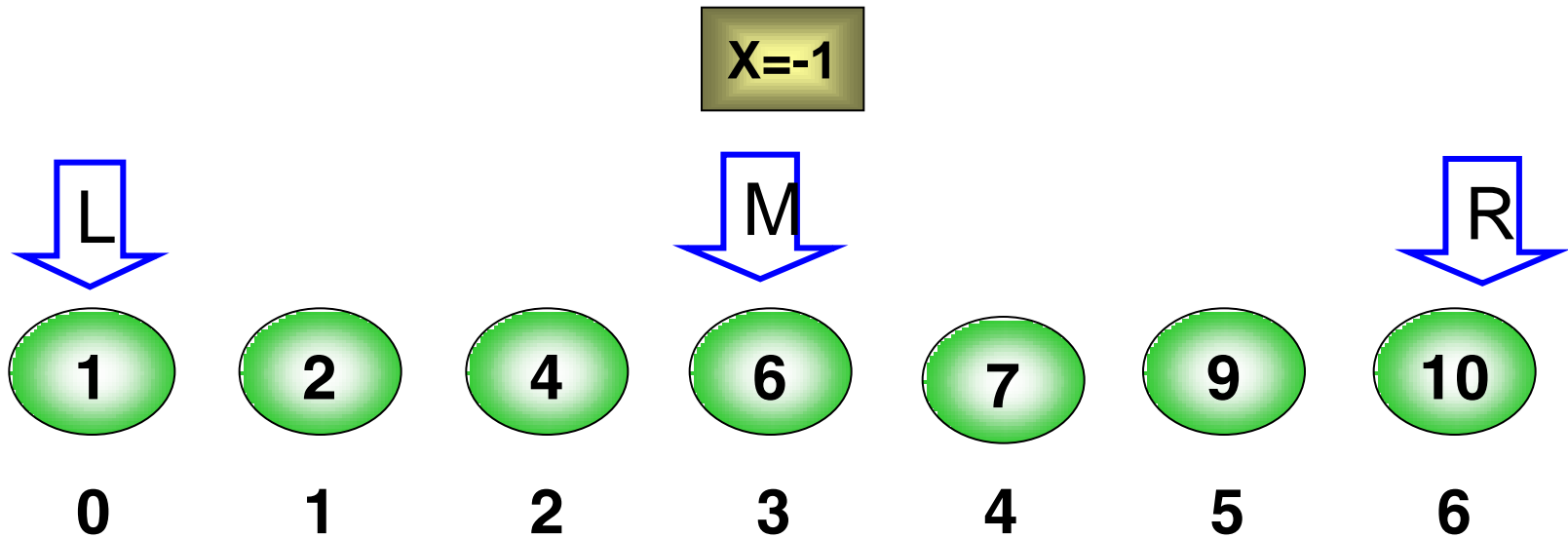


# Minh Họa Thuật Toán Tìm Nhị Phân

Tìm thấy 2 tại vị trí 1



# Minh Họa Thuật Toán Tìm Nhị Phân (tt)



$L=0$

$R=-1 \Rightarrow$  không tìm thấy  $X=-1$





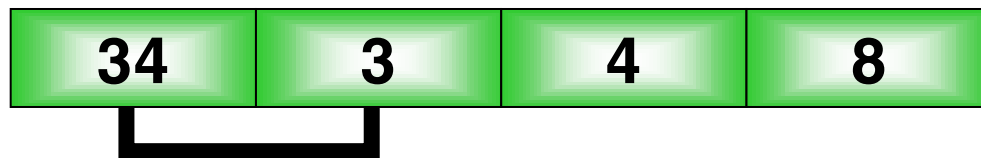
# Bài Toán Sắp Xếp

- Cho danh sách có  $n$  phần tử  $a_0, a_1, a_2, \dots, a_{n-1}$ .
- 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  $a_0, a_1, \dots, a_{n-1}$
    - 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



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Các Thuật Toán Sắp Xếp

1. **Đổi chỗ trực tiếp – Interchange Sort**
2. Chọn trực tiếp – Selection 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. Radix Sort



# Đổi Chỗ Trục Tiếp – Interchange Sort

- **Ý tưởng:** Xuất phát từ đầu dãy, tìm tất cả 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.



# Các Bước Tiến Hành

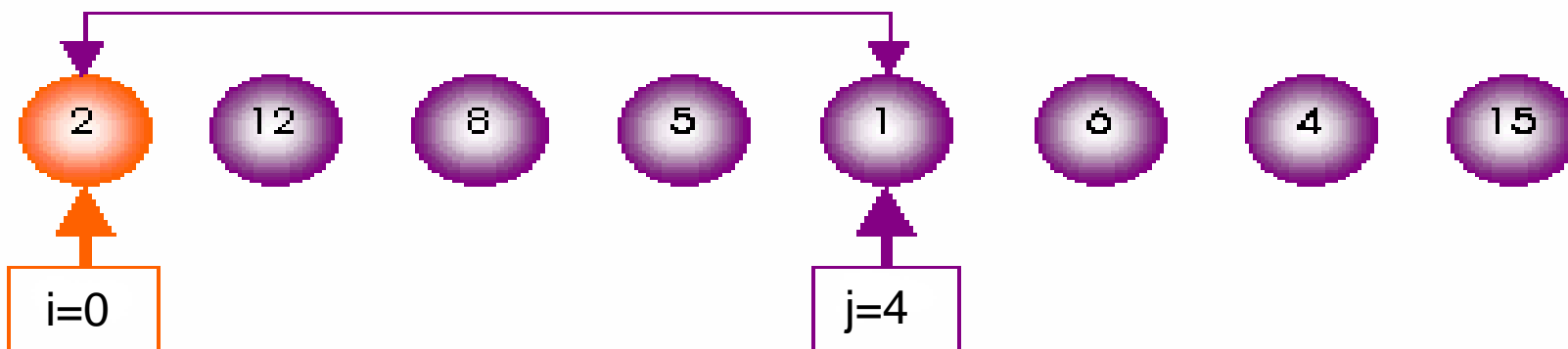
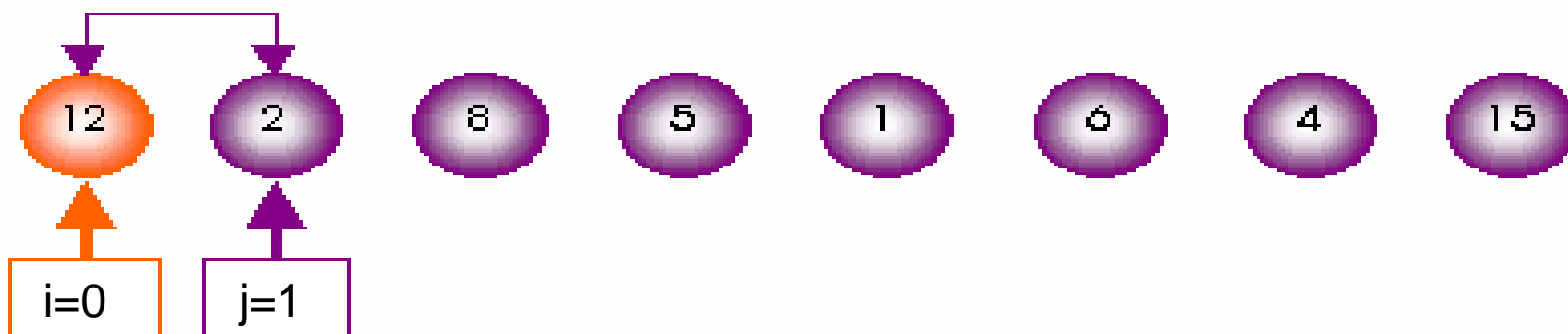
- Bước 1:  $i = 0$ ; // bắt đầu từ đầu dãy
- Bước 2:  $j = i+1$ ; // tìm các nghịch thế với  $a[i]$
- Bước 3:  
Trong khi  $j < N$  thực hiện  
Nếu  $a[j] < a[i]$  // xét cặp  $a[i], a[j]$   
Swap( $a[i], a[j]$ );  
 $j = j+1$ ;
- Bước 4:  $i = i+1$ ;  
Nếu  $i < N-1$ : Lặp lại Bước 2.  
Ngược lại: Dừng.



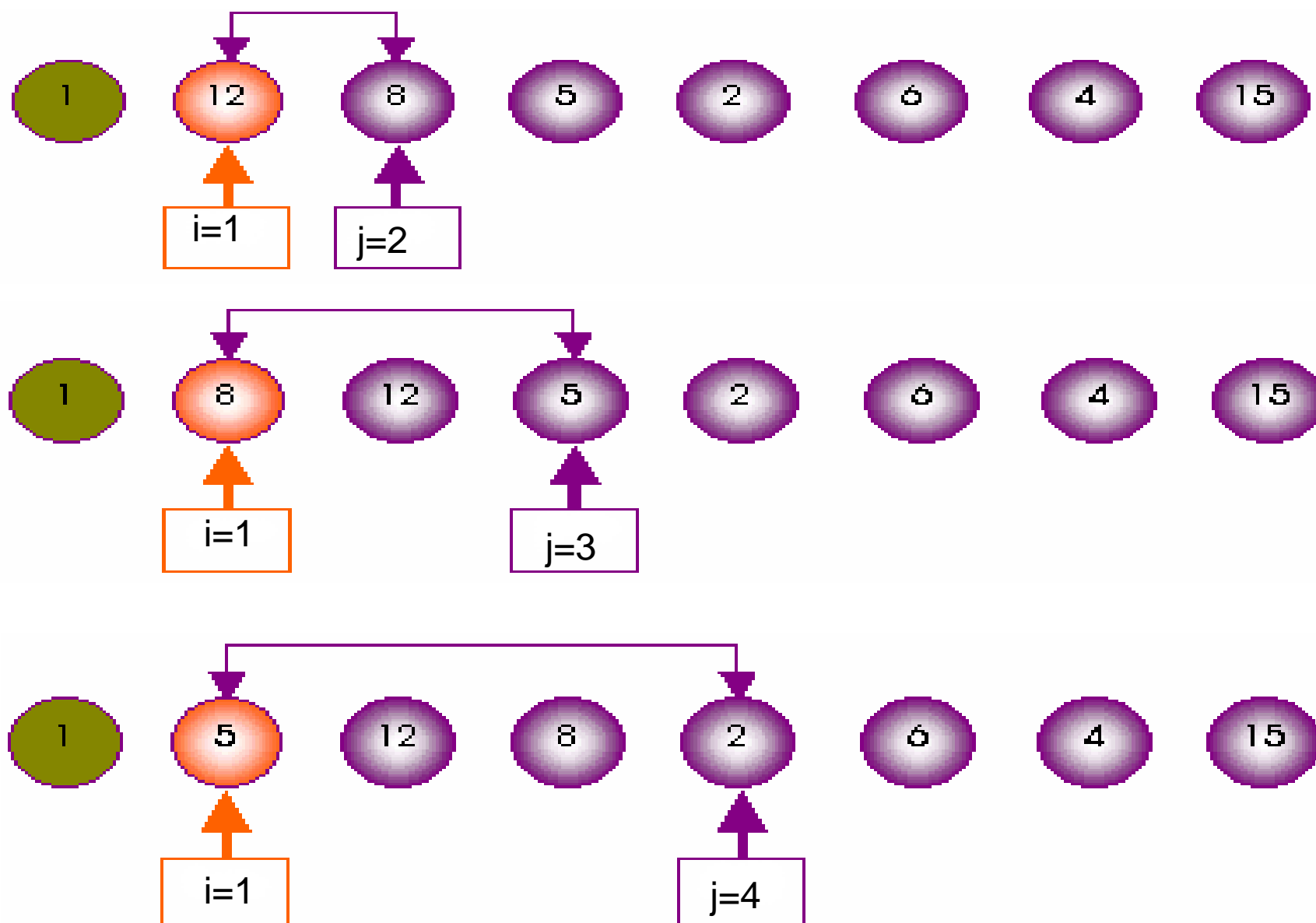
# Đổi Chỗ Trục Tiếp – Interchange Sort

➤ Cho dãy số a:

12    2    8    5    1    6    4    15

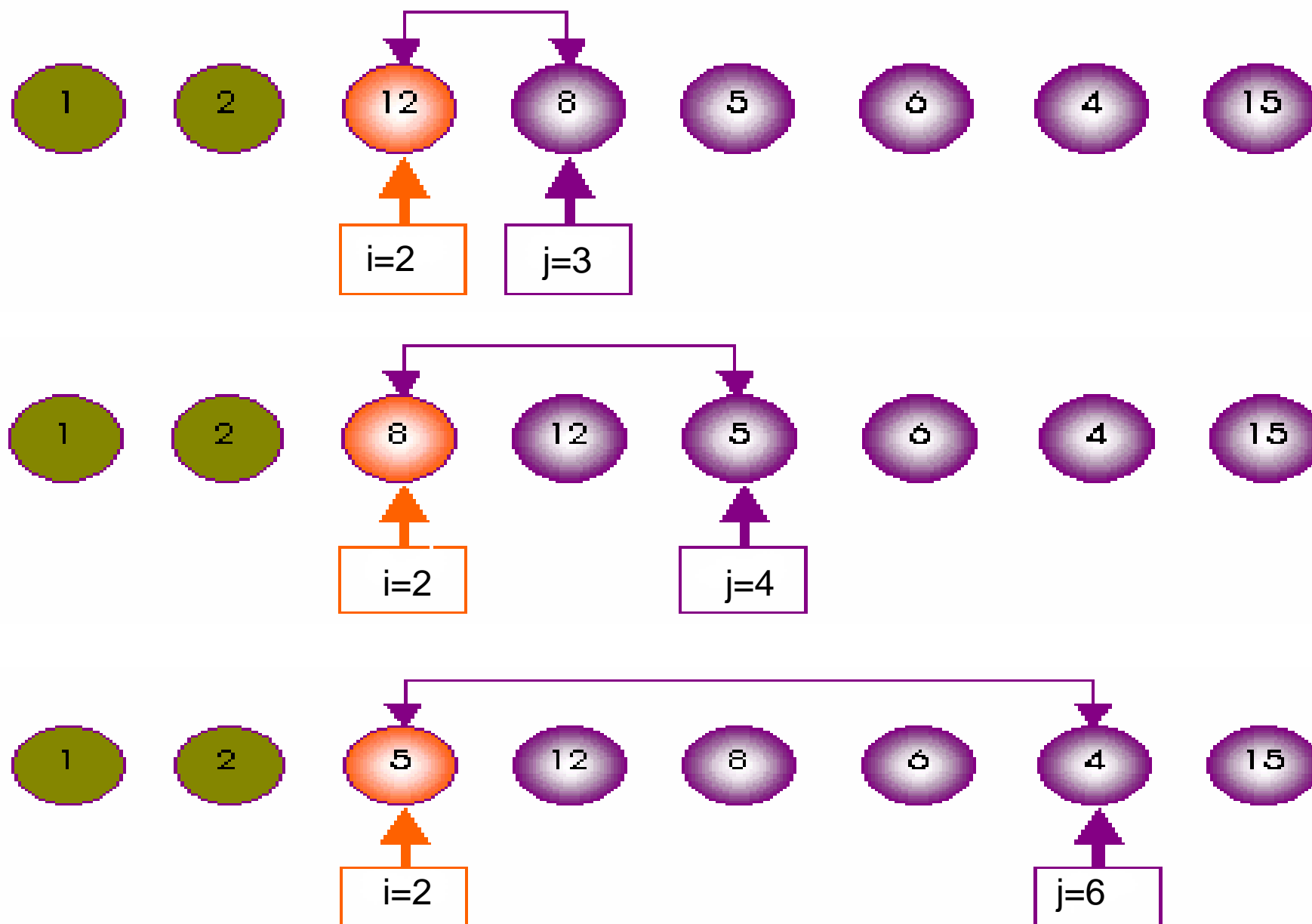


# Đổi Chỗ Trục Tiếp – Interchange Sort

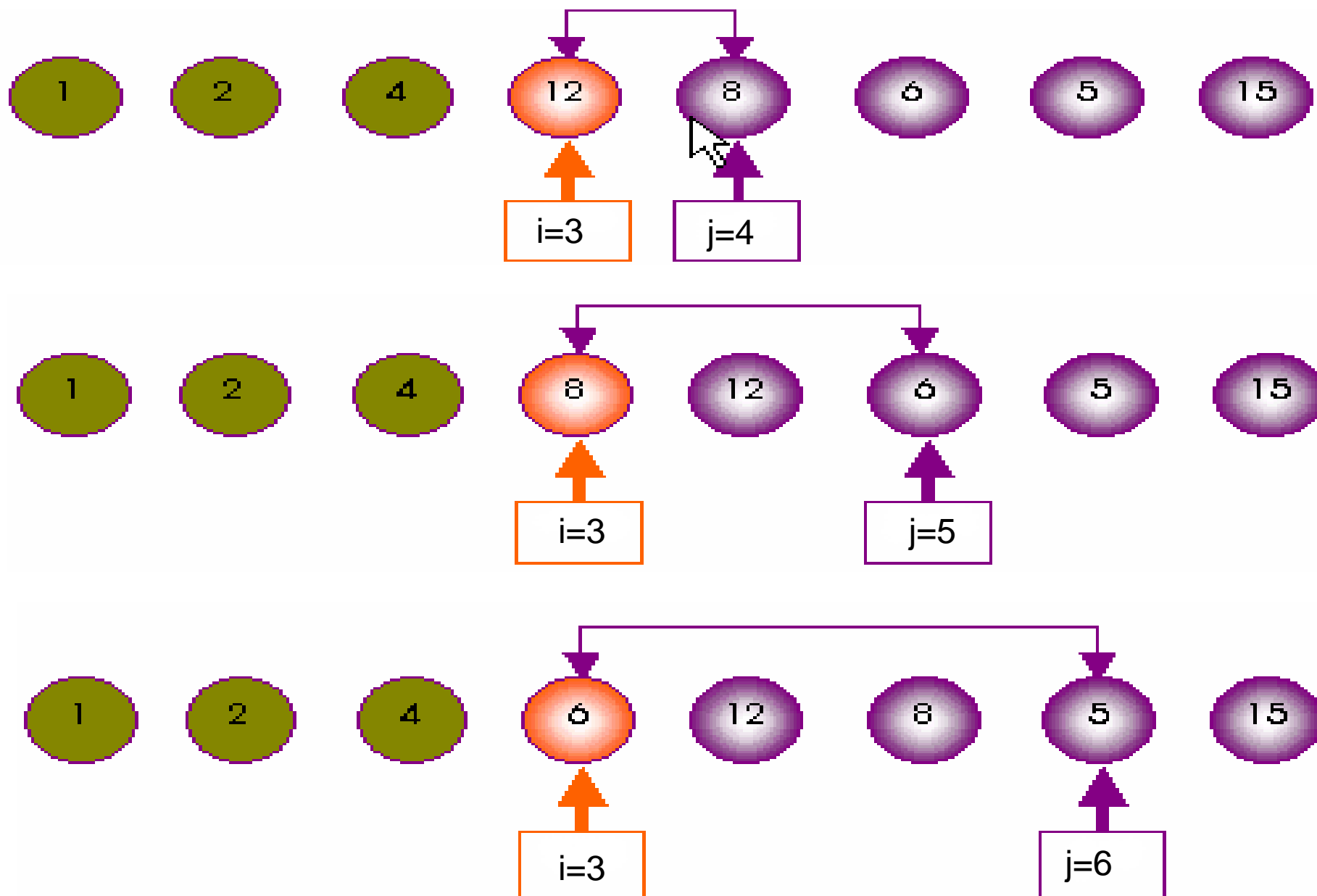




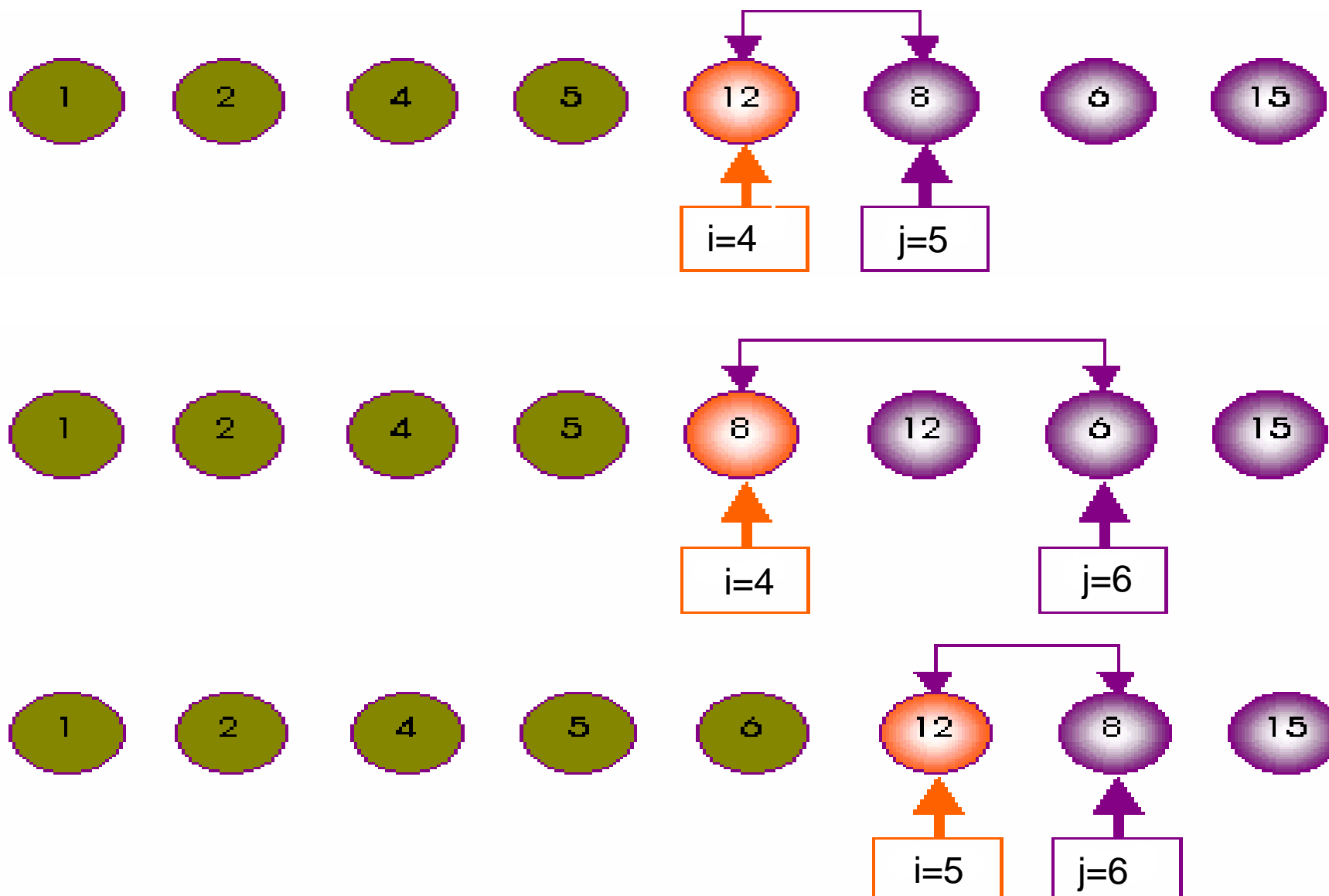
# Đổi Chỗ Trục Tiếp – Interchange Sort



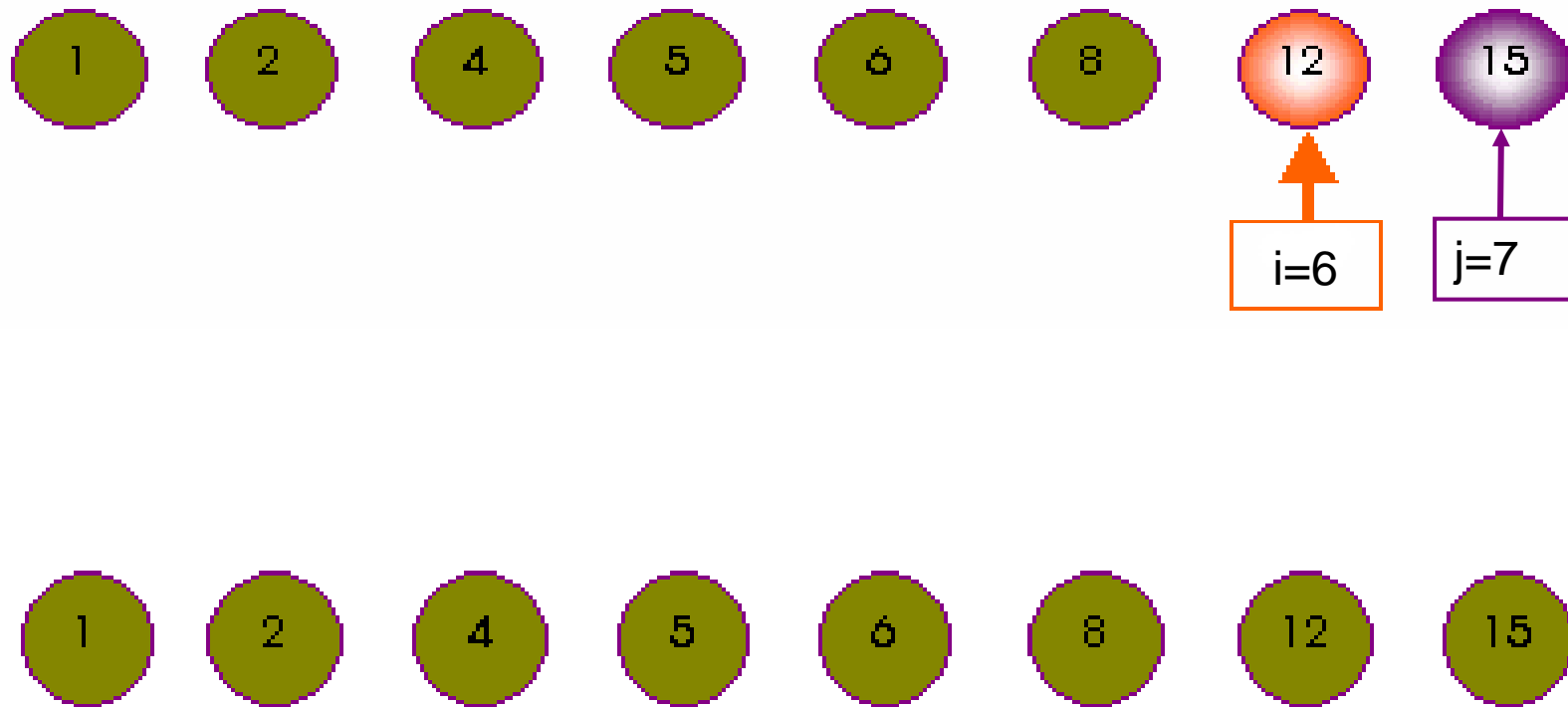
# Đổi Chỗ Trục Tiếp – Interchange Sort



# Đổi Chỗ Trực Tiếp – Interchange Sort



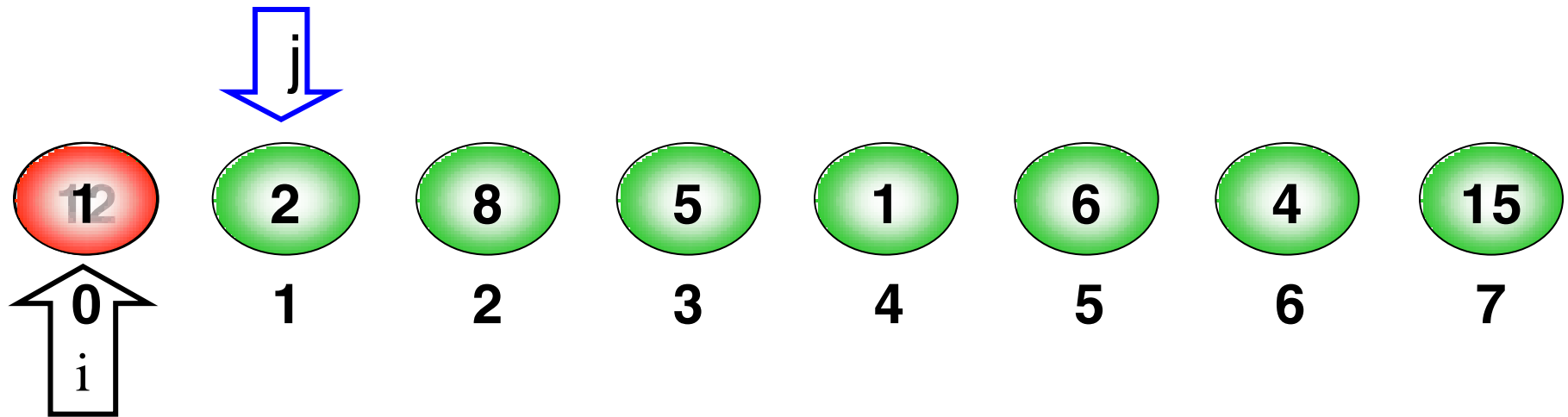
# Đổi Chỗ Trục Tiếp – Interchange Sort

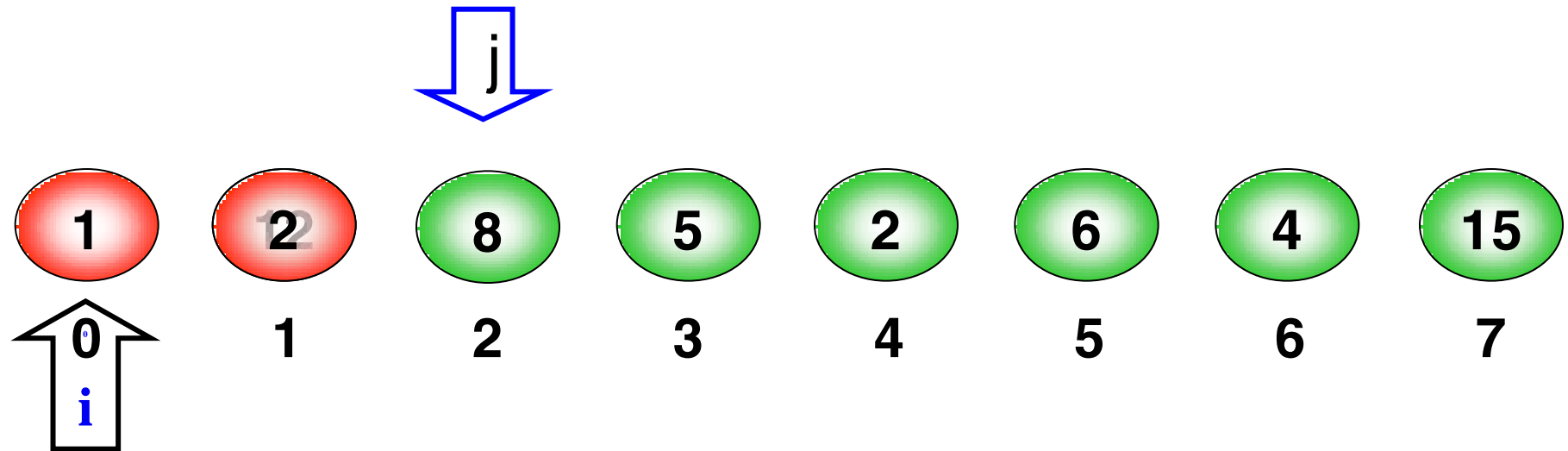


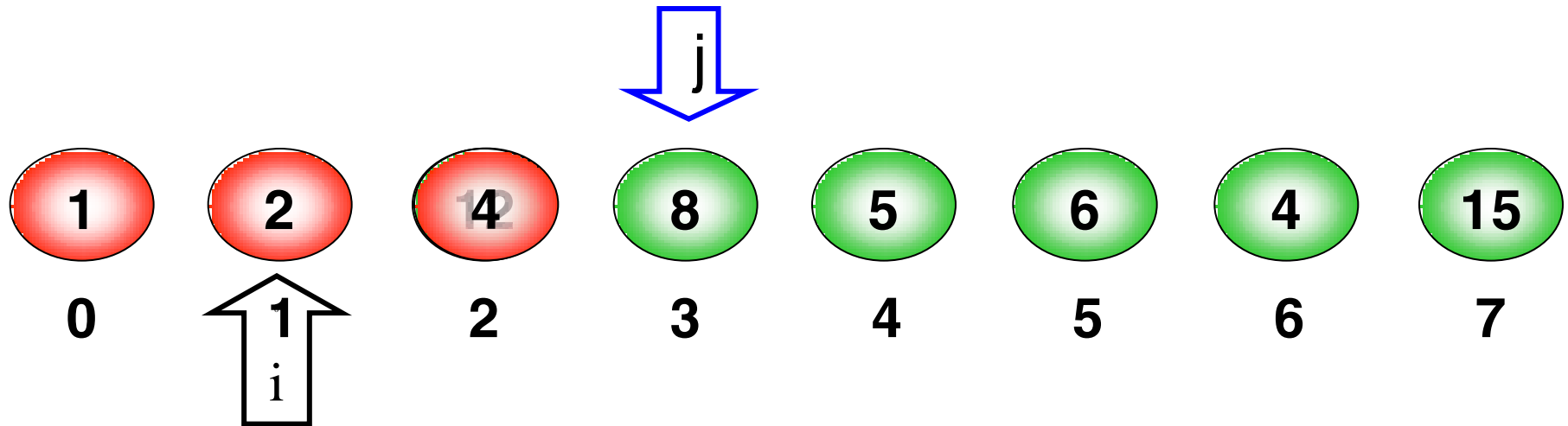
# Cài Đặt Đổi Chỗ Trực Tiếp

```
void InterchangeSort(int a[], int N )
{
    int    i, j;
    for (i = 0 ; i<N-1 ; i++)
        for (j =i+1; j < N ; j++)
            if(a[j ]< a[i]) // Thỏa 1 cặp nghịch thế
                Swap(a[i], a[j]);
}
```

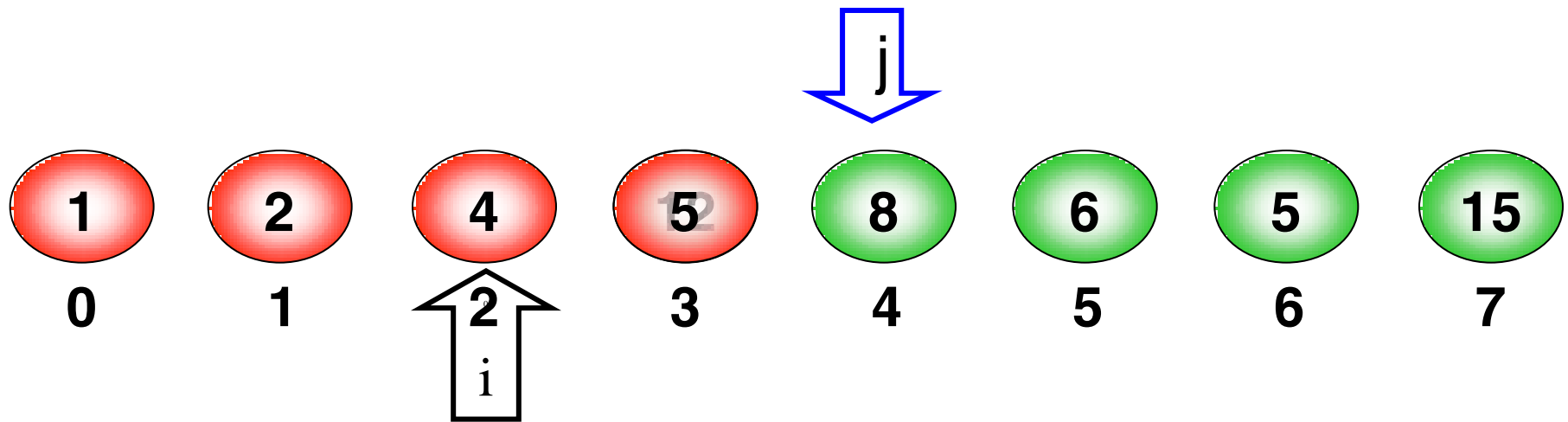


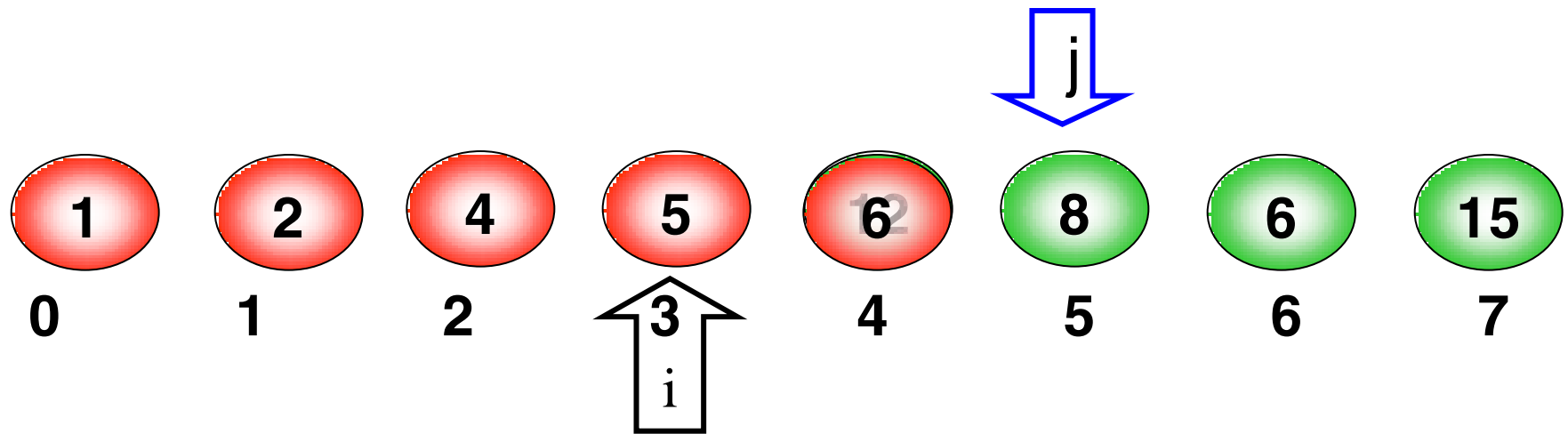


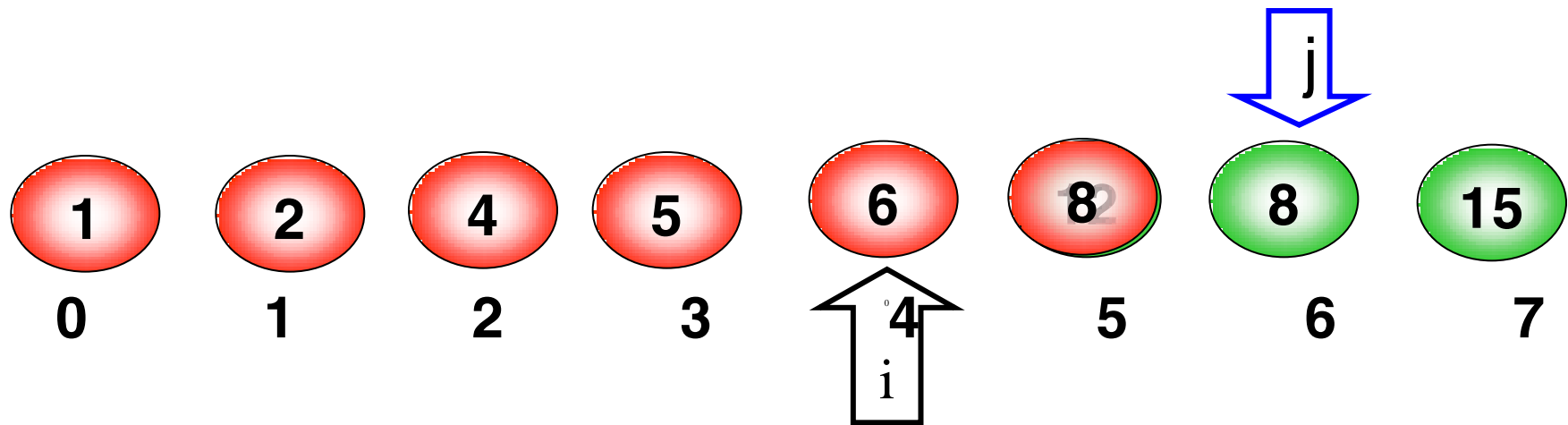


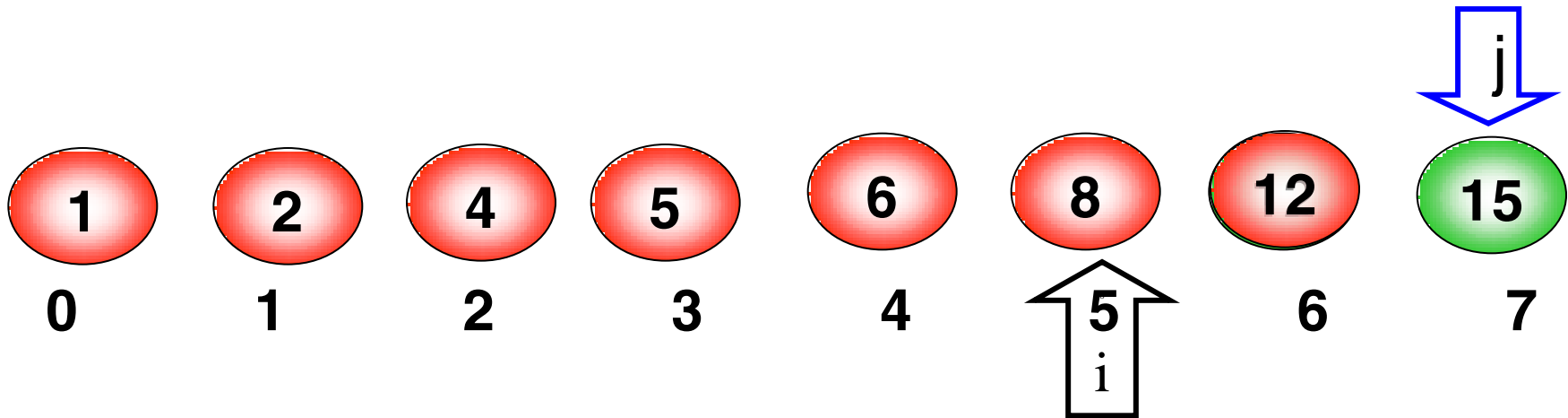


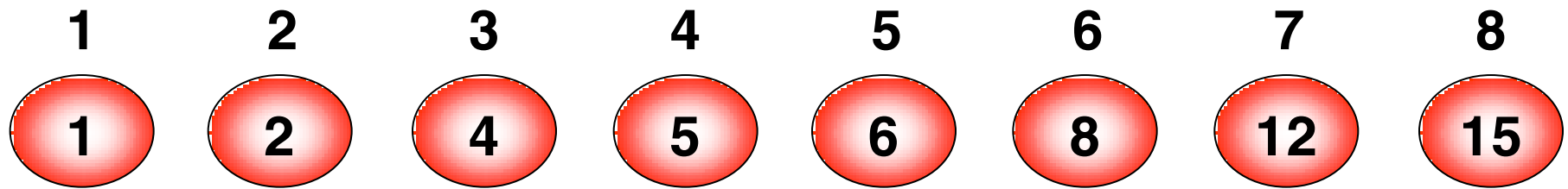












# Độ Phức Tạp Của Thuật Toán

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



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
- 2. Chọn trực tiếp – Selection 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. Radix Sort



# Chọn Trực Tiếp – 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 hiện hành 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





# Các Bước Của Thuật Toán Chọn Trực Tiếp

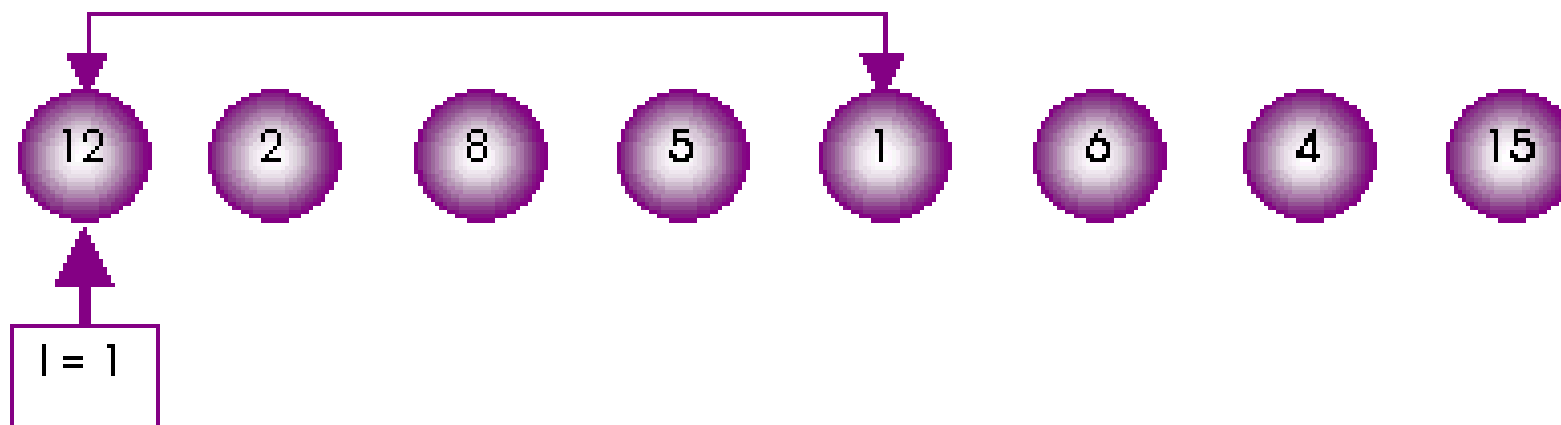
- Bước 1:  $i = 0$ ;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ  $a[i]$  đến  $a[N]$
- Bước 3: Đổi chỗ  $a[\text{min}]$  và  $a[i]$
- Bước 4: Nếu  $i < N-1$  thì  
 $i = i+1$ ; Lặp lại Bước 2;  
Ngược lại: Dừng.



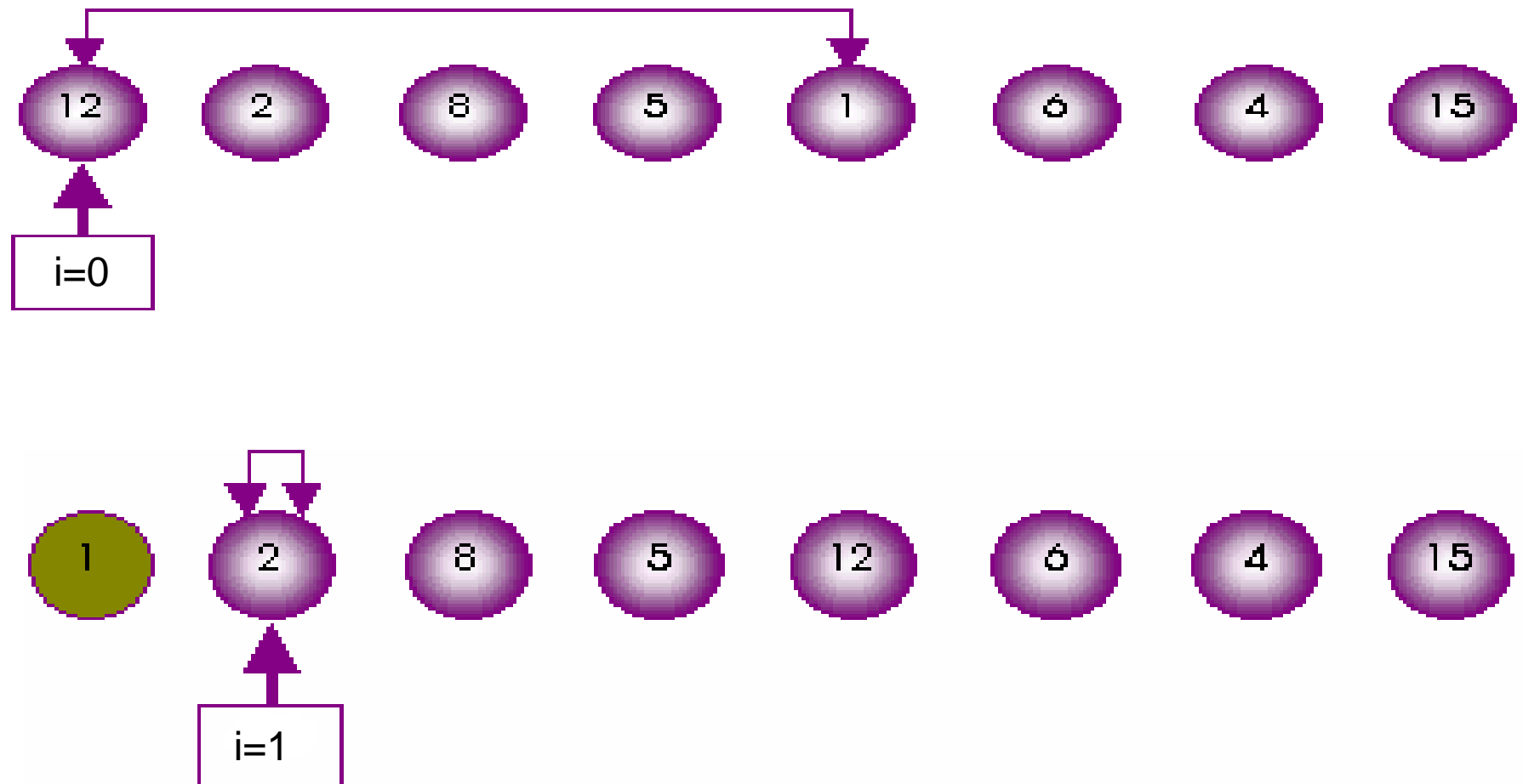
# Chọn Trục Tiếp – Selection Sort

➤ Cho dãy số a:

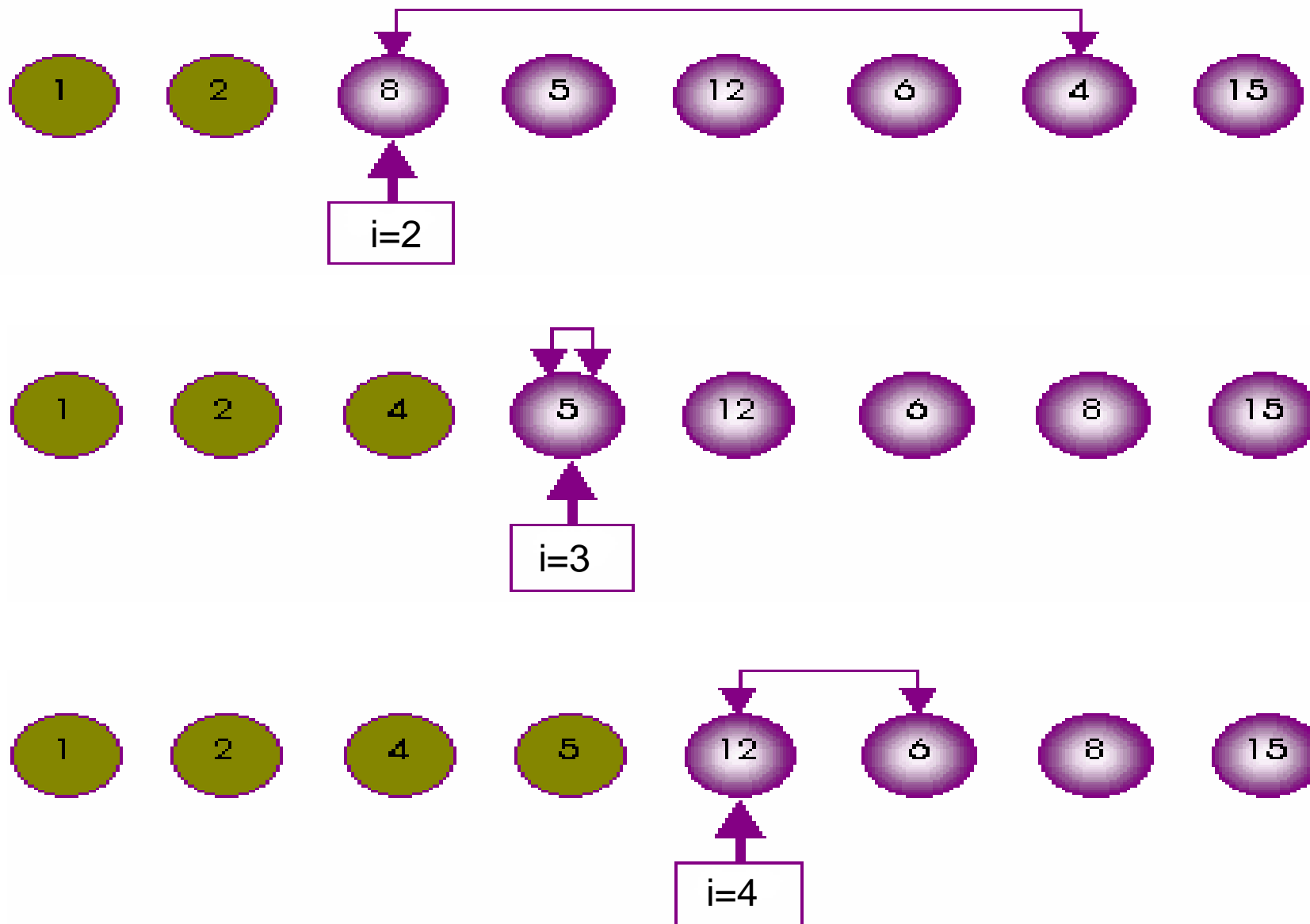
12      2      8      5      1      6      4      15



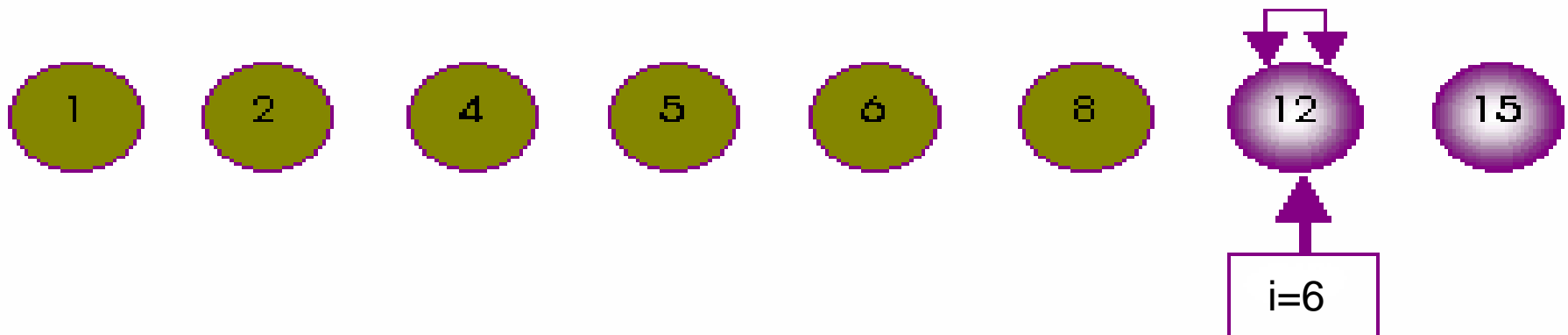
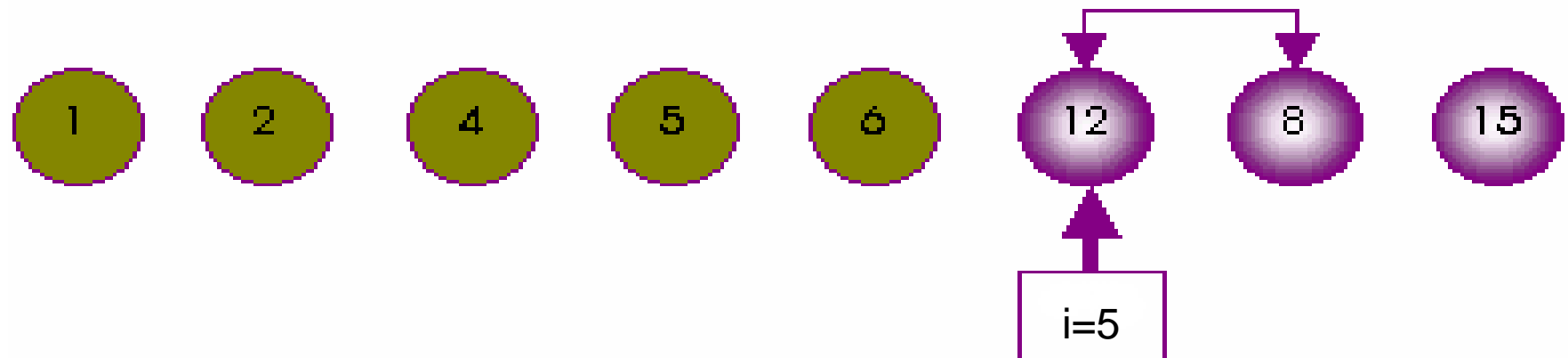
# Chọn Trực Tiếp – Selection Sort



# Chọn Trực Tiếp – Selection Sort



# Chọn Trực Tiếp – Selection Sort



# Cài Đặt Thuật Toán Chọn Trực Tiếp

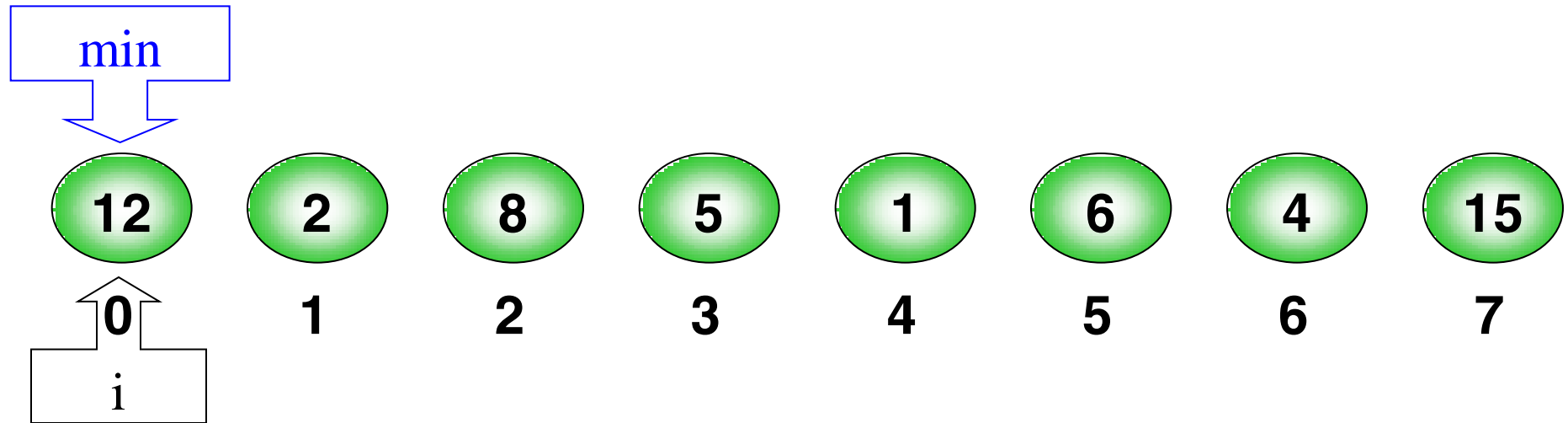
```
void SelectionSort(int a[],int n )
{
    int    min,i,j; // chỉ số phần tử nhỏ nhất trong dãy hiện hành
    for (i=0; i<n-1 ; i++) //chỉ số đầu tiên của dãy hiện hành
    {
        min = i;
        for(j = i+1; j <N ; j++)
            if (a[j ] < a[min])
                min = j; // lưu vtrí phần tử hiện nhỏ nhất
        Swap(a[min],a[i]);
    }
}
```



# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(0,7)

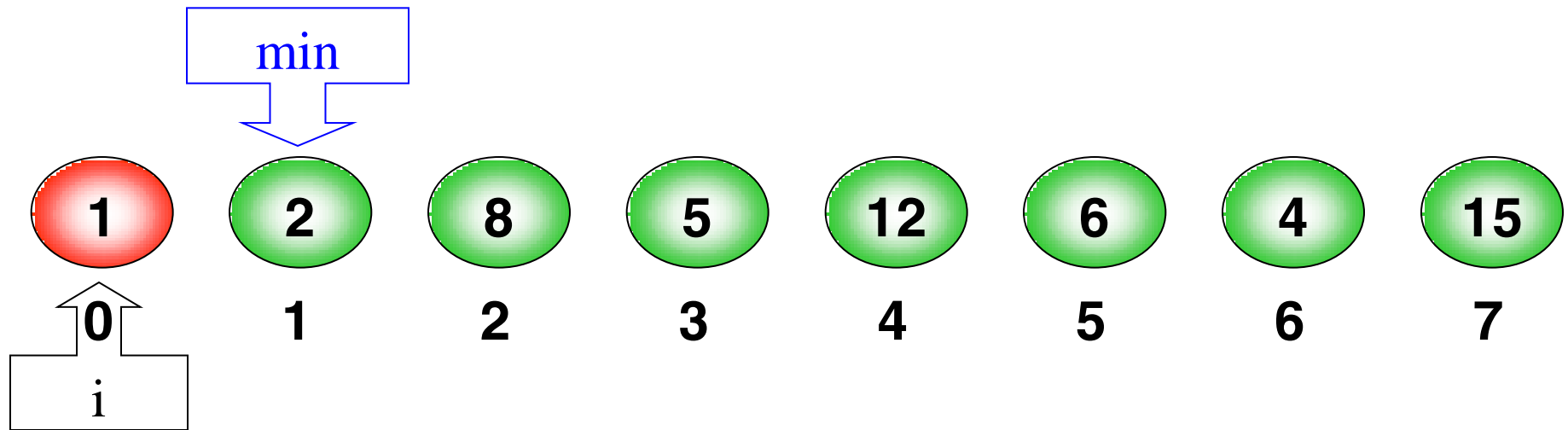
Swap(a[0], a[4])



# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(1,7)

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

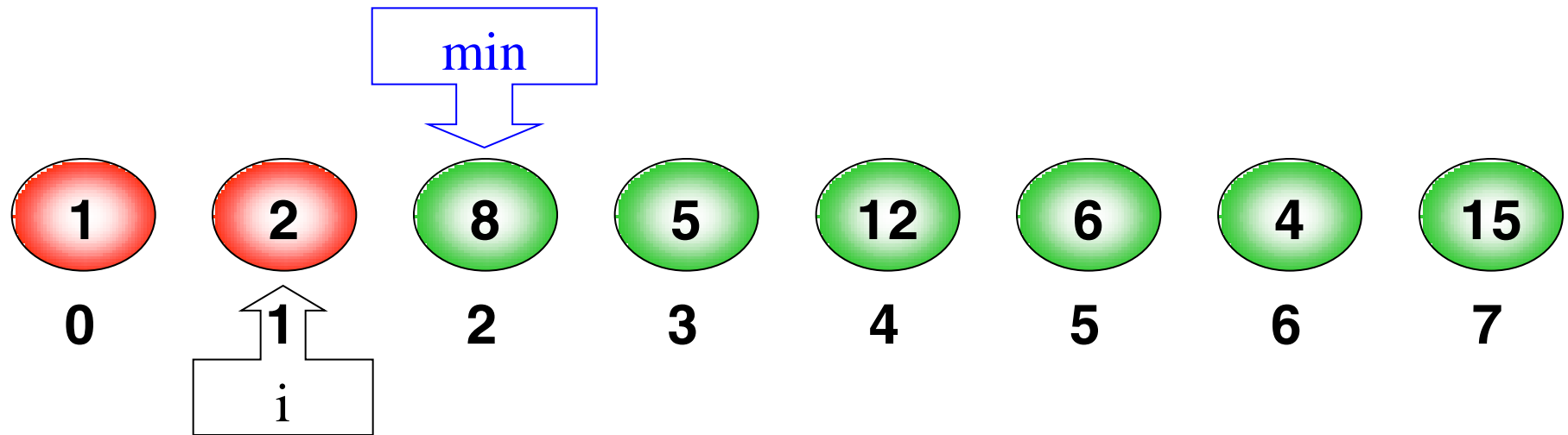




# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(2,7)

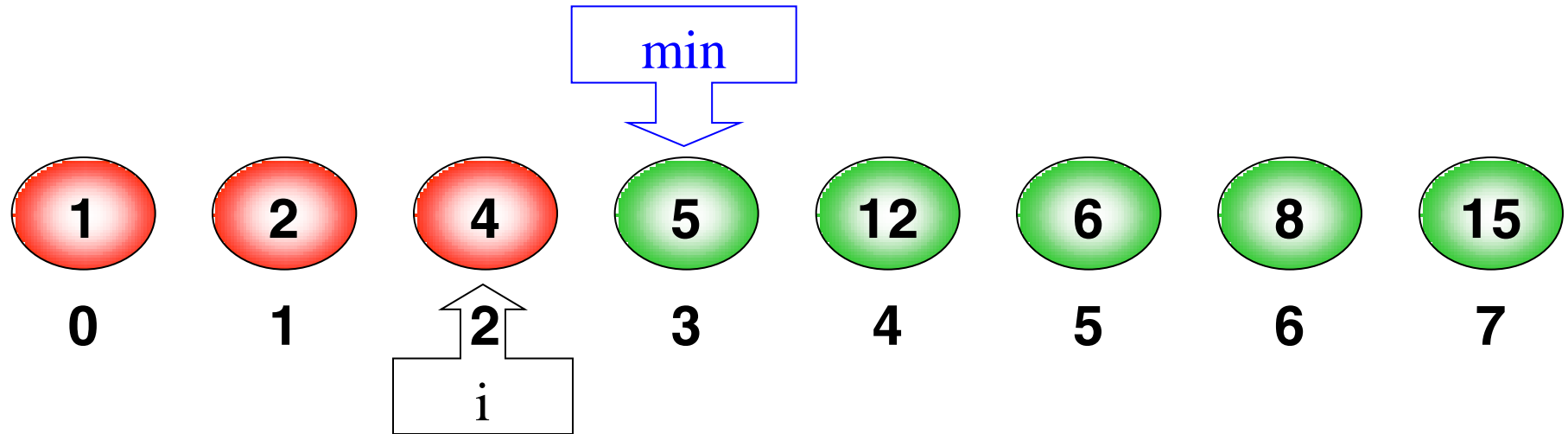
Swap(a[2], a[6])



# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(3, 7)

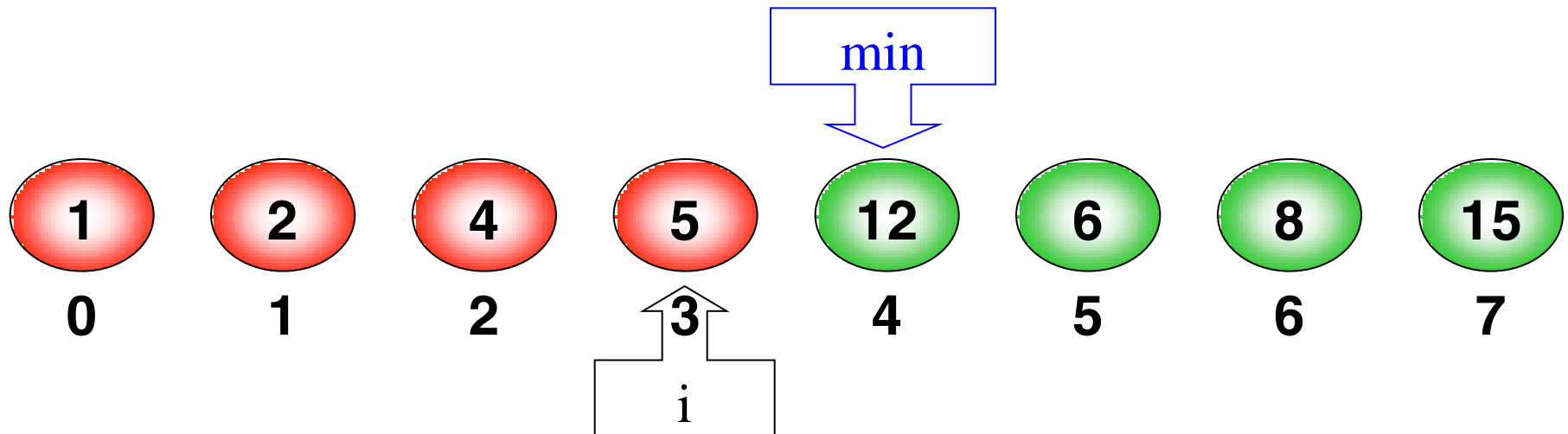
Swap(a[3], a[3])



# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(4, 7)

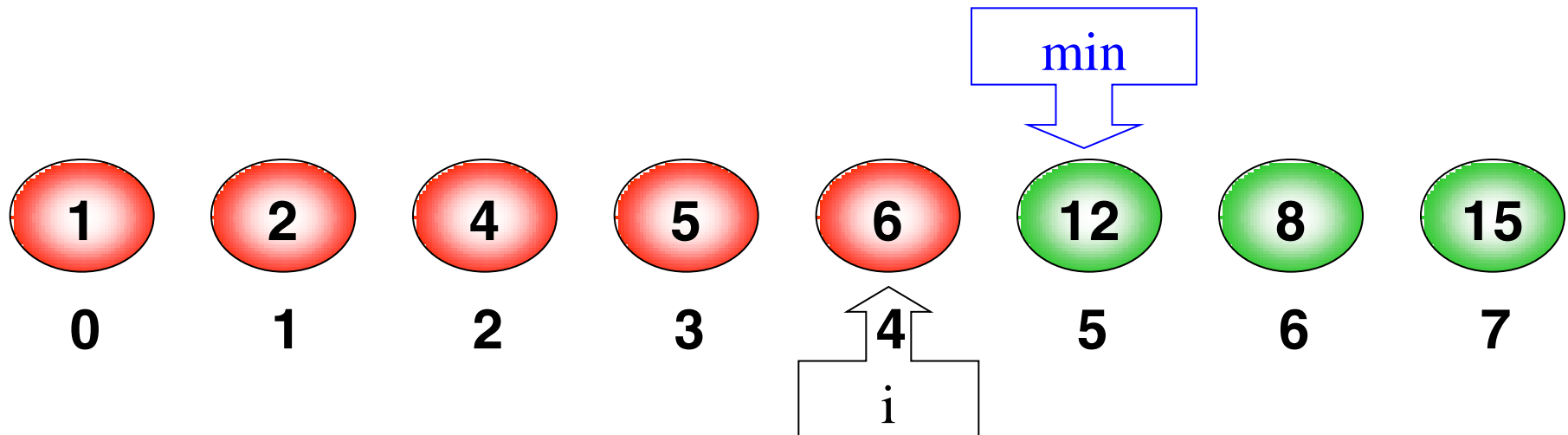
Swap(a[4], a[5])



# Minh Họa Thuật Toán Chọn Trực Tiếp

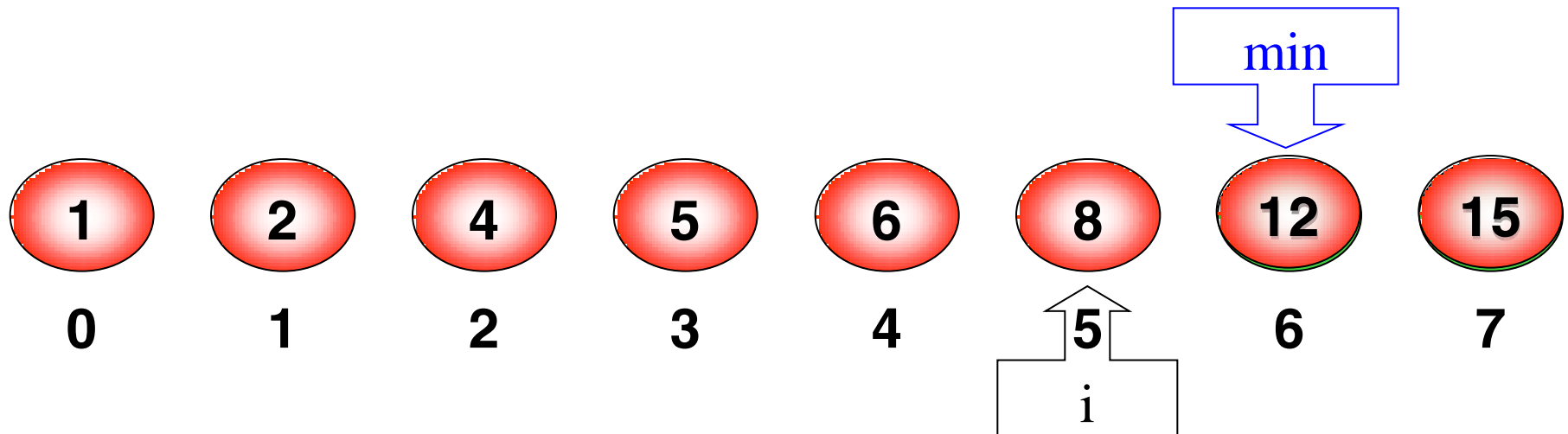
Vị trí nhỏ nhất(5,7)

Swap(a[5], a[6])



# Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(6, 7)



# Độ Phức Tạo Của Thuật Toán

## ➤ Đánh giá giải thuật

$$\text{số lần so sánh} = \sum_{i=1}^{n-1} (n-i) = \frac{n(n-1)}{2}$$

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$n(n-1)/2$	0
Xấu nhất	$n(n-1)/2$	$3n(n-1)/2$

# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Nổi Bọt – 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.





# Nổi Bọt – Bubble Sort

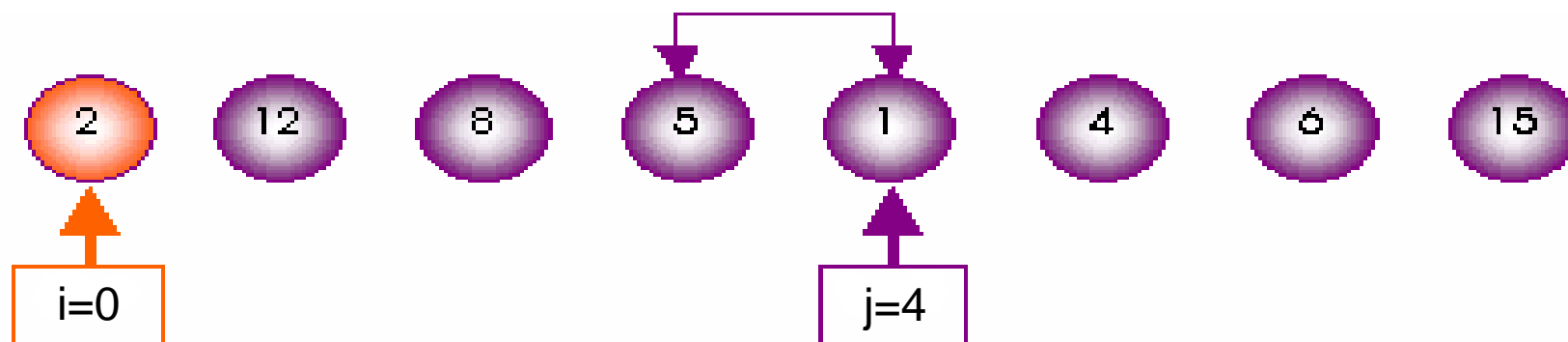
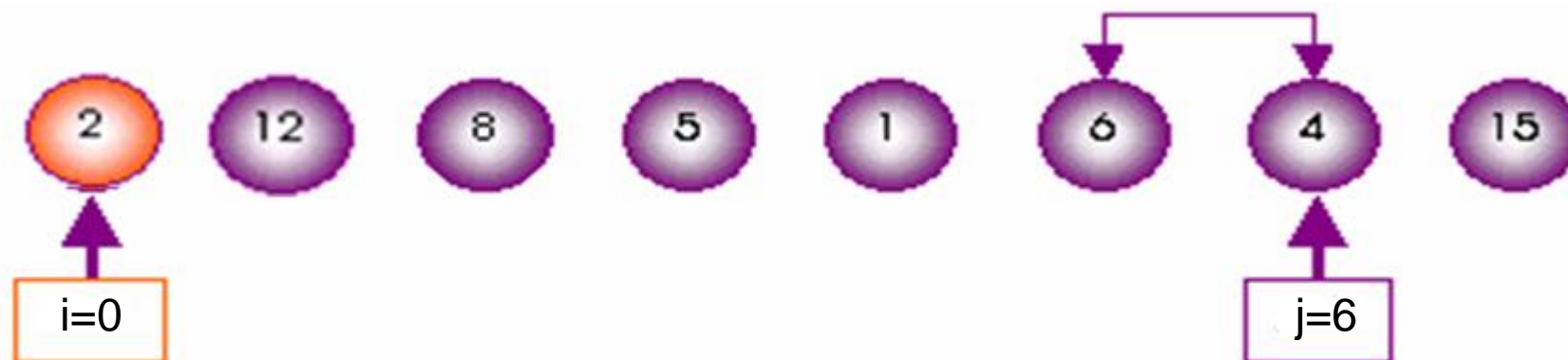
- Bước 1 :  $i = 0$ ; // lần xử lý đầu tiên
- Bước 2 :  $j = N-1$ ; //Duyệt từ cuối dãy ngược về vị trí  $i$   
Trong khi ( $j > i$ ) thực hiện:  
    Nếu  $a[j] < a[j-1]$   
        Doicho( $a[j], a[j-1]$ );  
         $j = j-1$ ;
- Bước 3 :  $i = i+1$ ; // lần xử lý kế tiếp  
    Nếu  $i = N$ : Hết dãy. Dừng  
    Ngược lại : Lặp lại Bước 2.



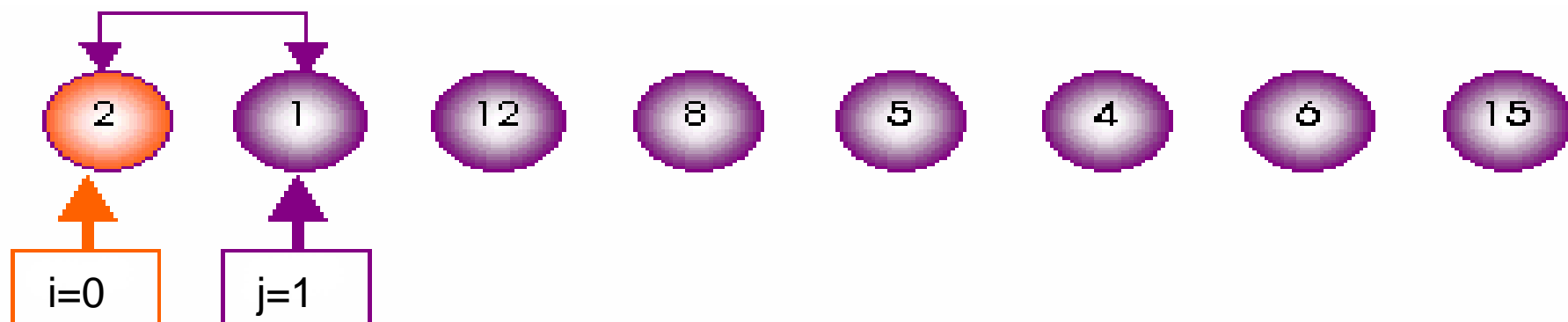
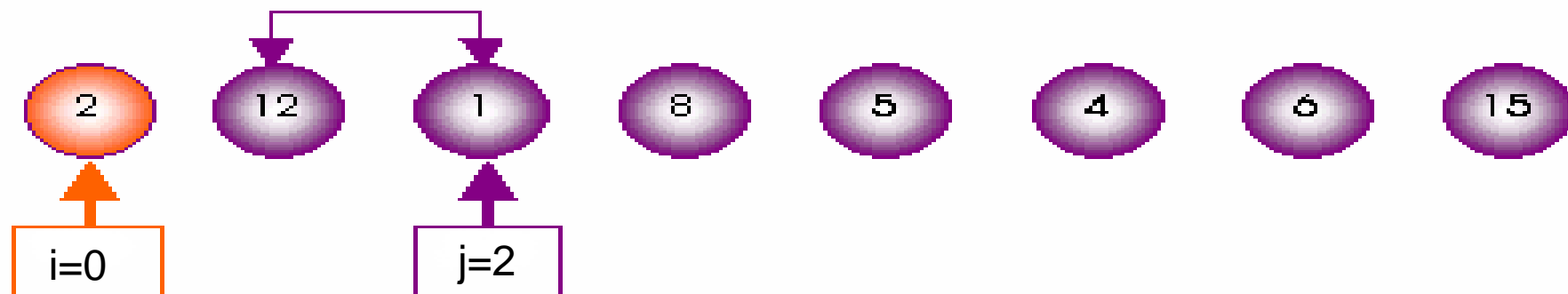
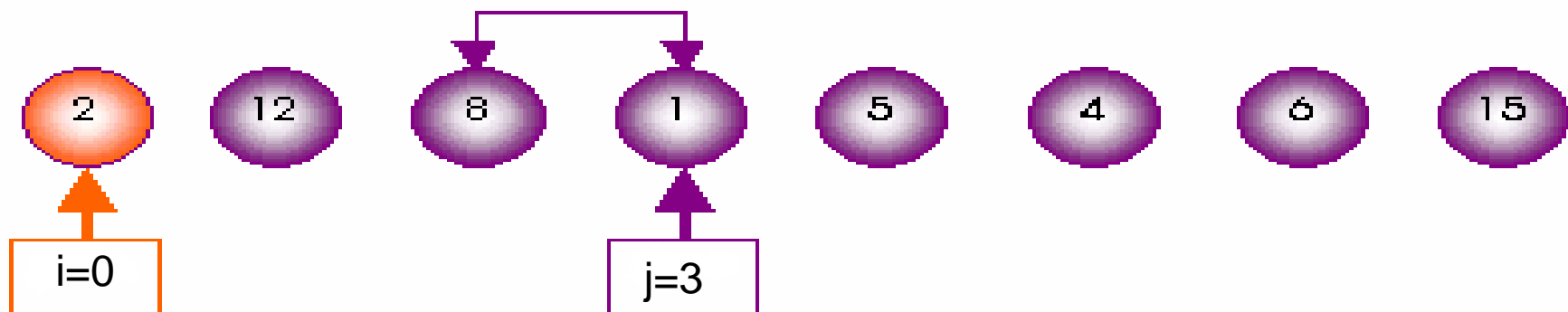
# Nổi Bọt – Bubble Sort

➤ Cho dãy số a:

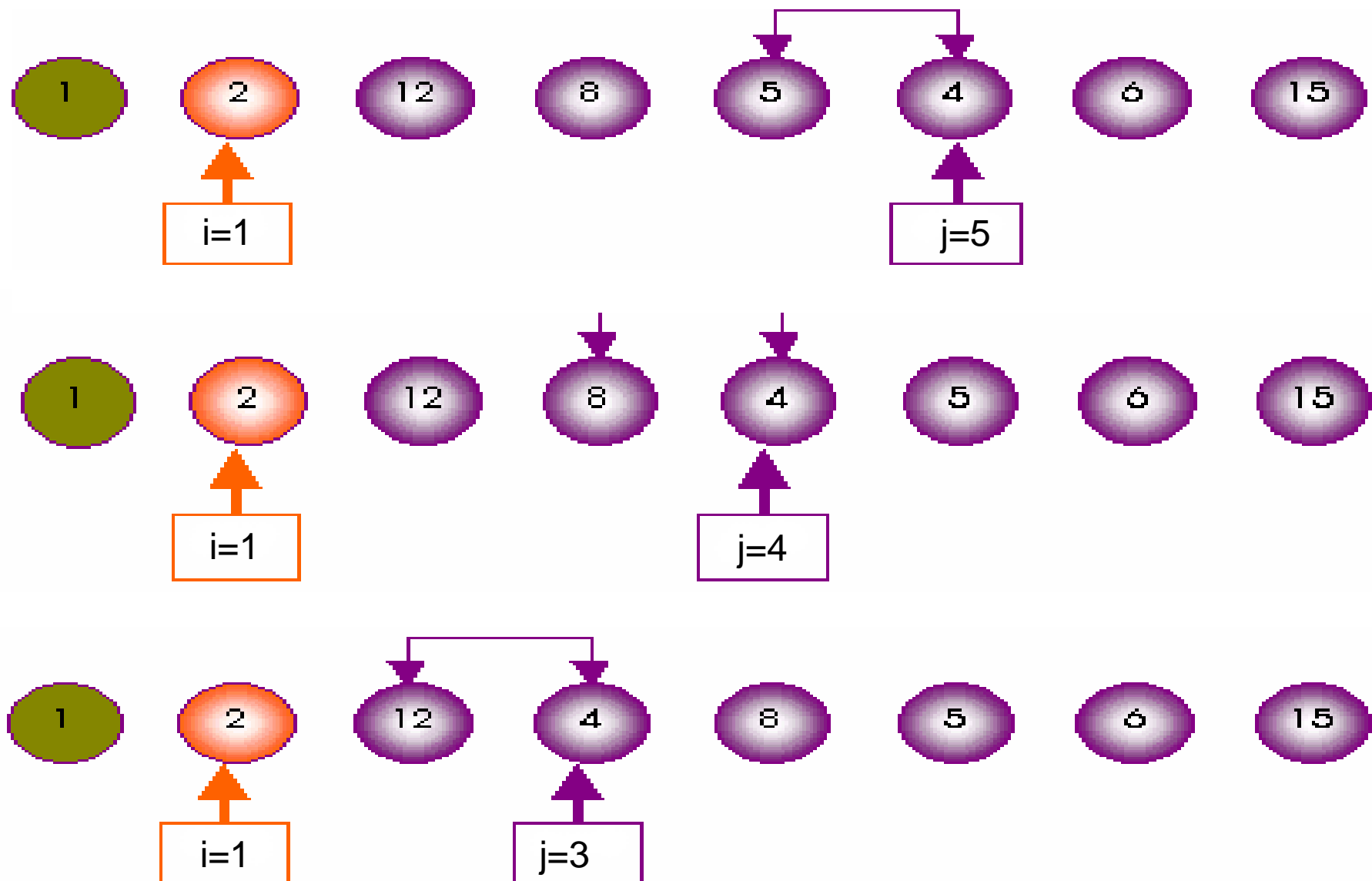
2      12    8      5      1      6      4      15



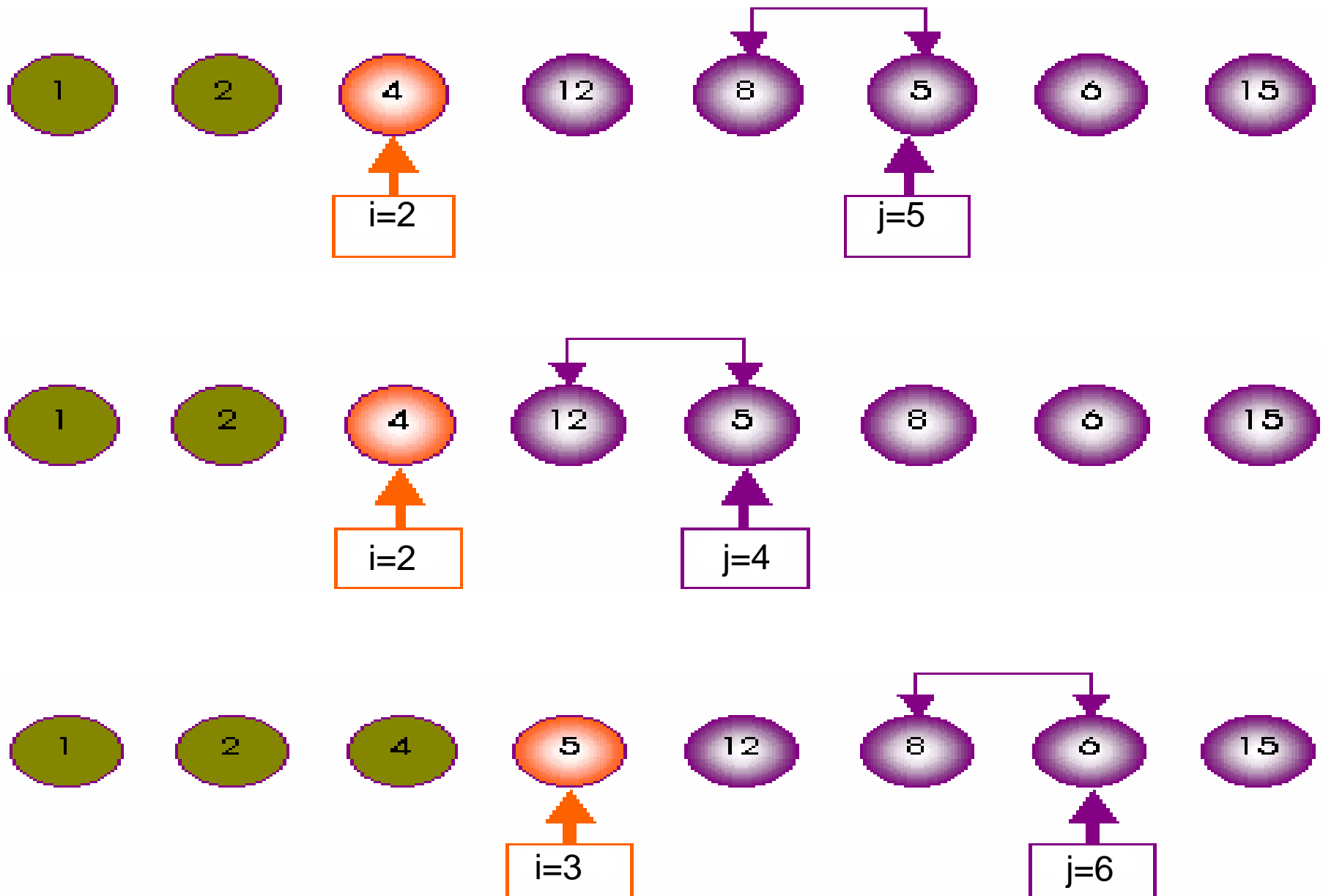
# Nổi Bọt – Bubble Sort



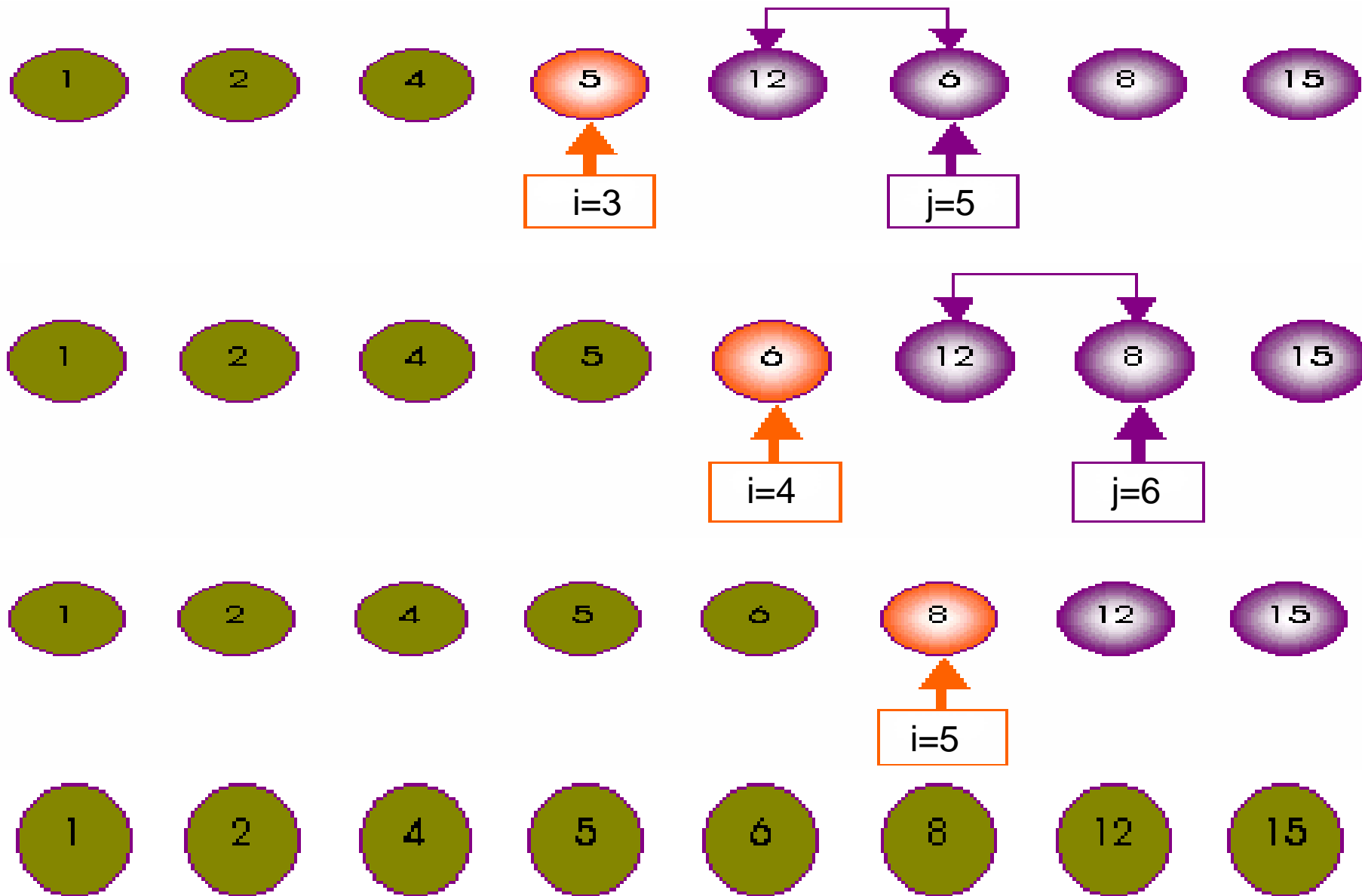
# Nổi Bọt – Bubble Sort



# Nổi Bọt – Bubble Sort



# Nổi Bọt – Bubble Sort

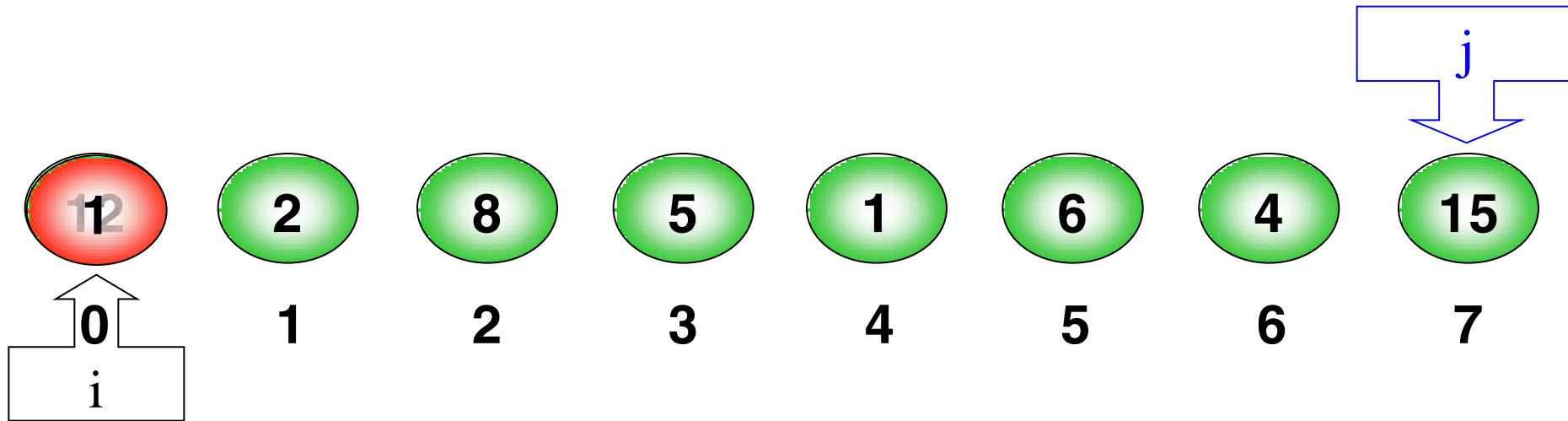


# Cài Đặt Thuật Toán Nổi Bọt

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

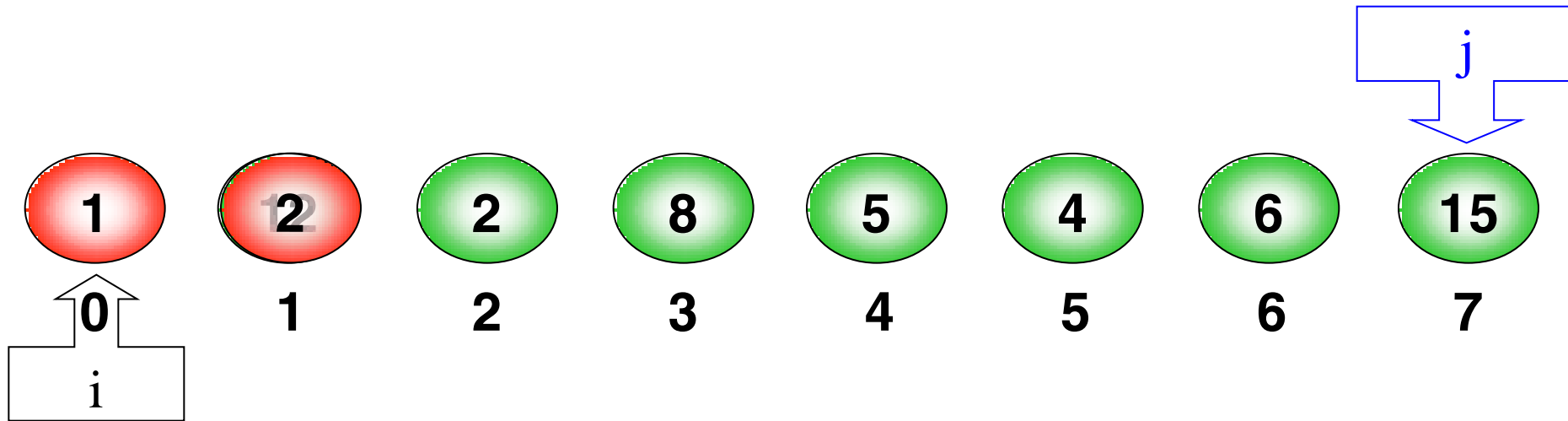


# Minh Họa Thuật Toán

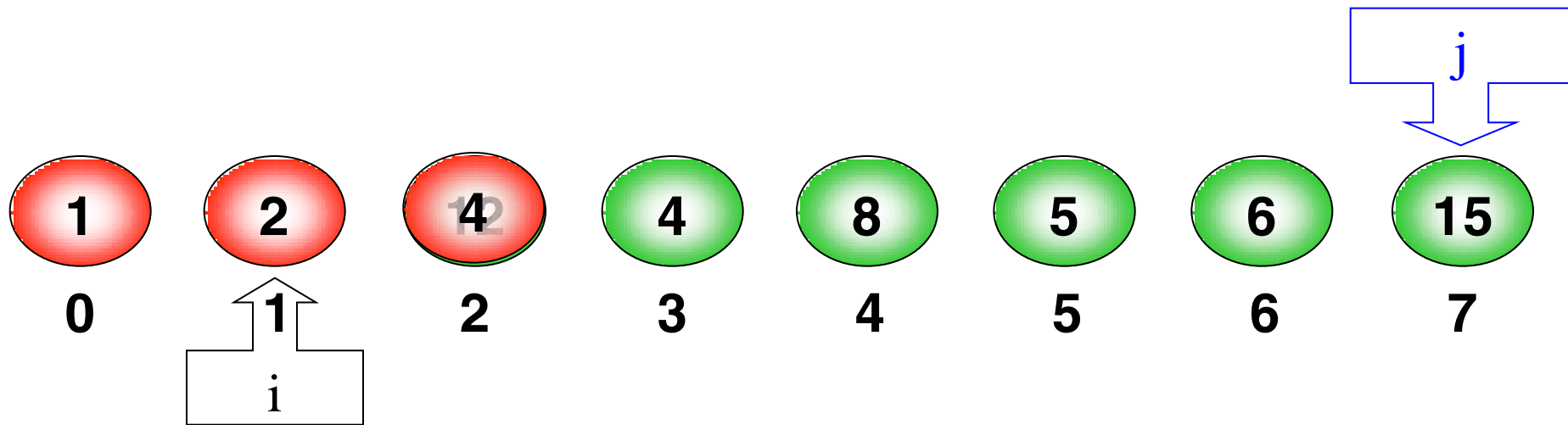




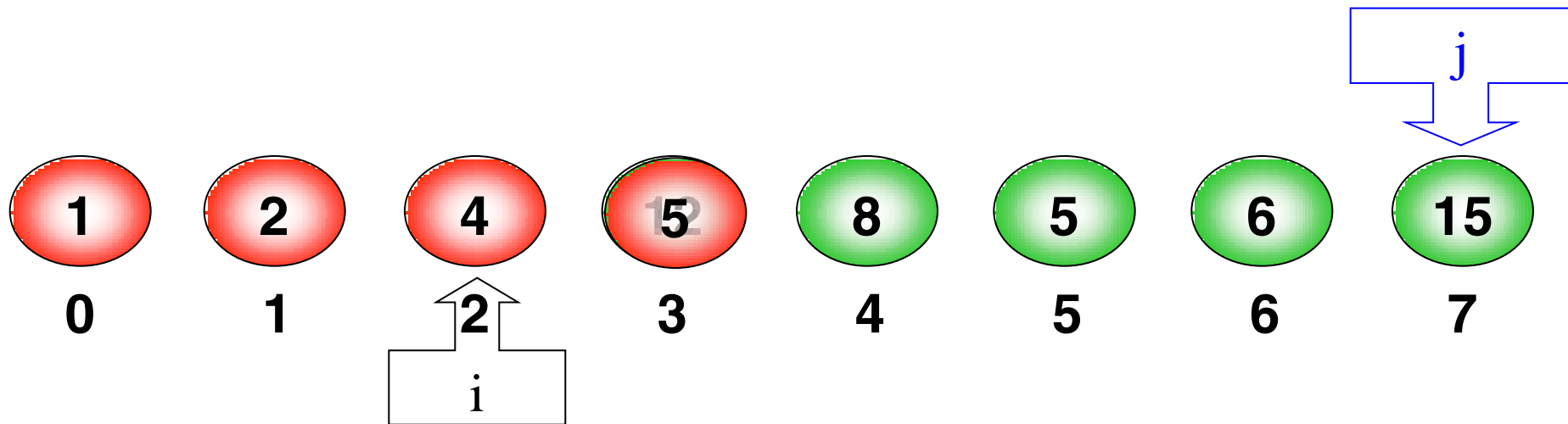
# Minh Họa Thuật Toán



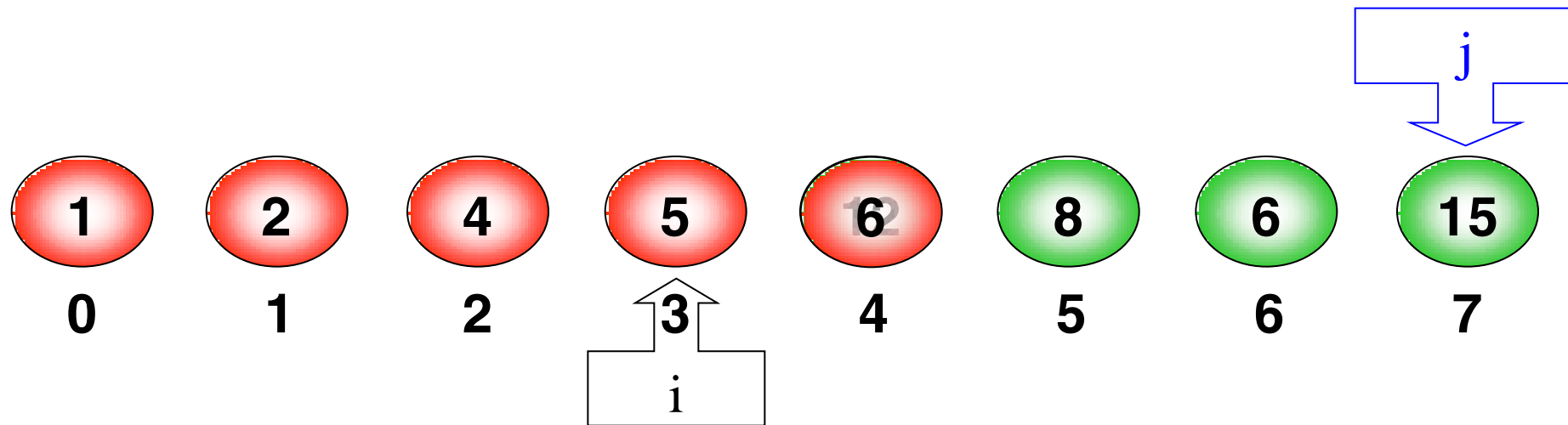
# Minh Họa Thuật Toán



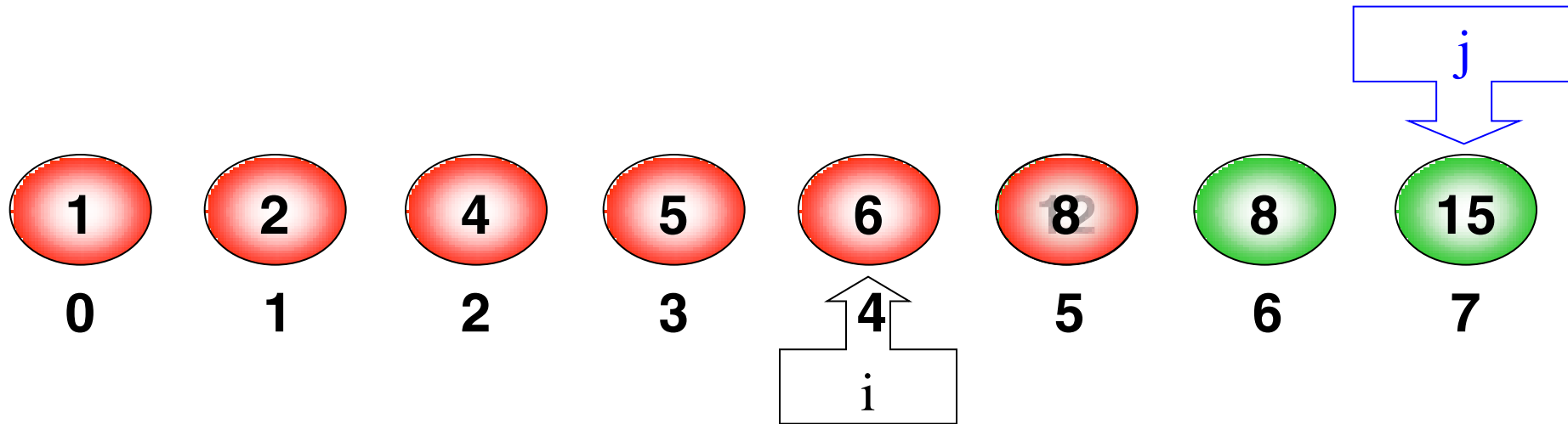
# Minh Họa Thuật Toán



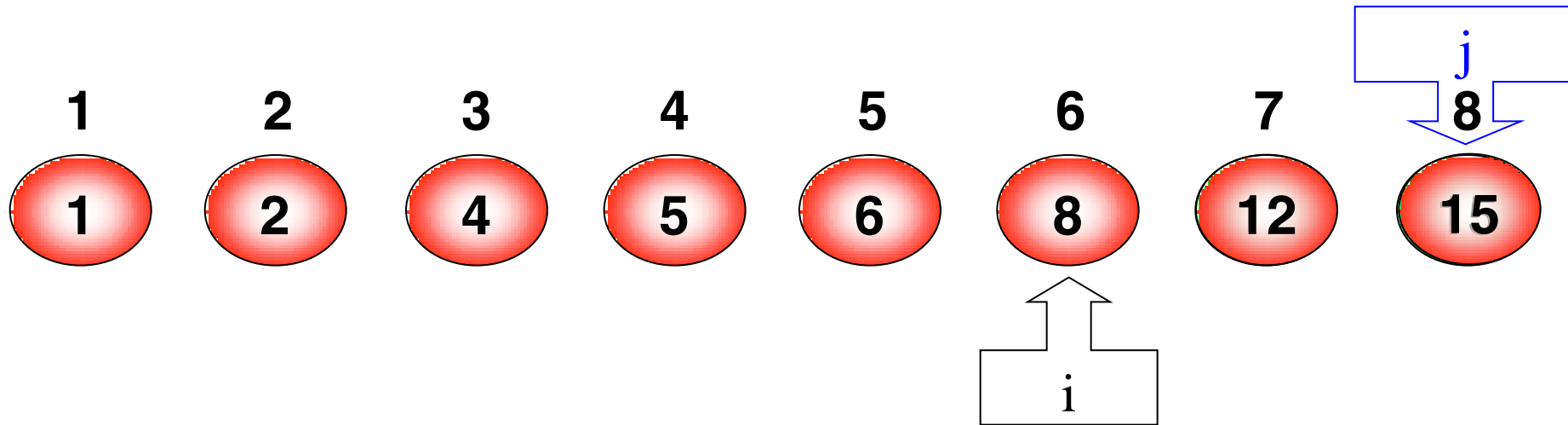
# Minh Họa Thuật Toán



# Minh Họa Thuật Toán



# Minh Họa Thuật Toán



# Độ Phức Tạp Của Thuật Toán Nổi Bọt

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

# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. 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.



# Các Bước Của Thuật Toán

- Bước 1:  $l=0$ ;  $r=n-1$ ; //Đoạn  $l \rightarrow r$  là đoạn cần được sắp xếp  
           $k=n$ ;                   //ghi nhận vị trí  $k$  xảy ra hoán vị sau cùng  
                                  // để làm cơ sở thu hẹp đoạn  $l \rightarrow r$
- 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( $a[j]$ , $a[j-1]$ ):  $k=j$ ;}  
             $j--$ ;  
         $l=k$ ;     //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( $a[j]$ , $a[j+1]$ );  $k=j$ ;}  
             $j++$ ;  
         $r=k$ ;     //loại phần tử đã có thứ tự ở cuối dãy
- Bước 3: Nếu  $l<r$  lặp lại bước 2  
            Ngược lại: dừng



# Cài Đặt Thuật Toán Shaker Sort

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



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Chèn Trực Tiếp – Insertion Sort

- Giả sử có một dãy  $a_0, a_1, \dots, a_{n-1}$  trong đó  $i$  phần tử đầu tiên  $a_0, a_1, \dots, a_{i-1}$  đã có thứ tự.
- Tìm cách chèn phần tử  $a_i$  vào **vị trí thích hợp** của đoạn đã được sắp để có dãy mới  $a_0, a_1, \dots, a_i$  trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử  $a_{k-1}$  và  $a_k$  thỏa  $a_{k-1} < a_i < a_k$  ( $1 \leq k \leq i$ ).



# Chèn Trực Tiếp – Insertion Sort

- Bước 1:  $i = 1$ ; //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 chỗ 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

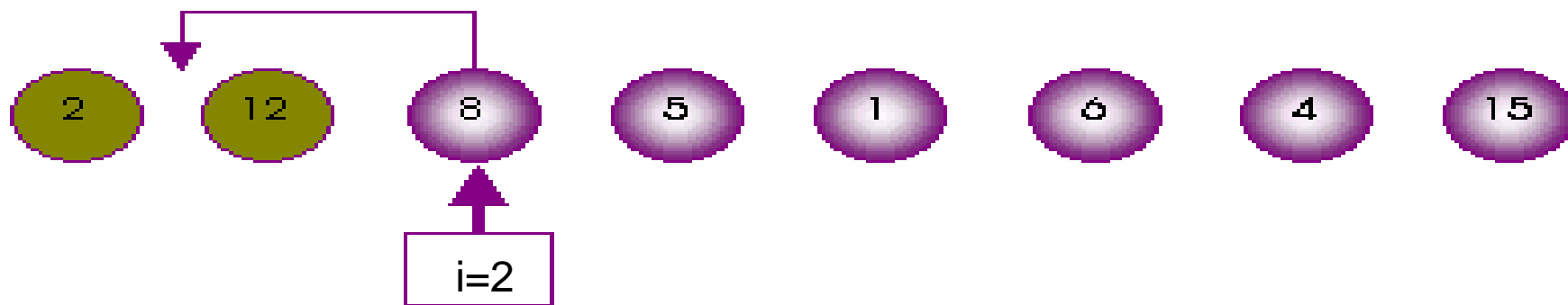
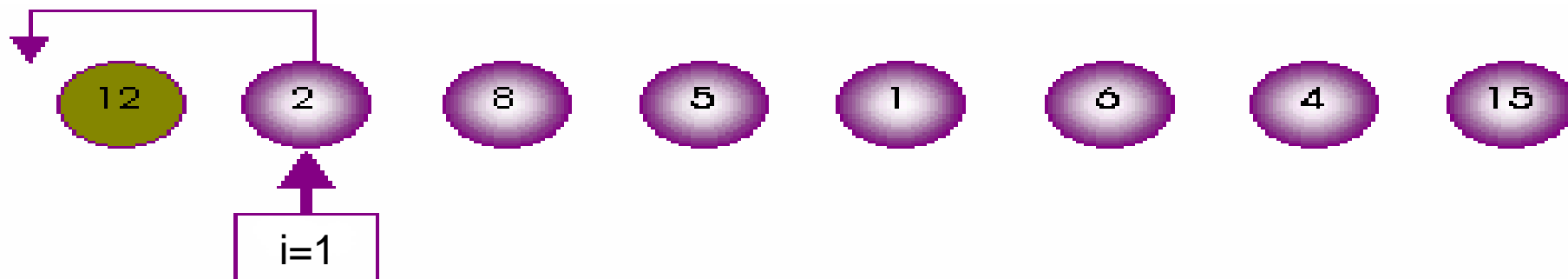
Ngược lại : Dừng



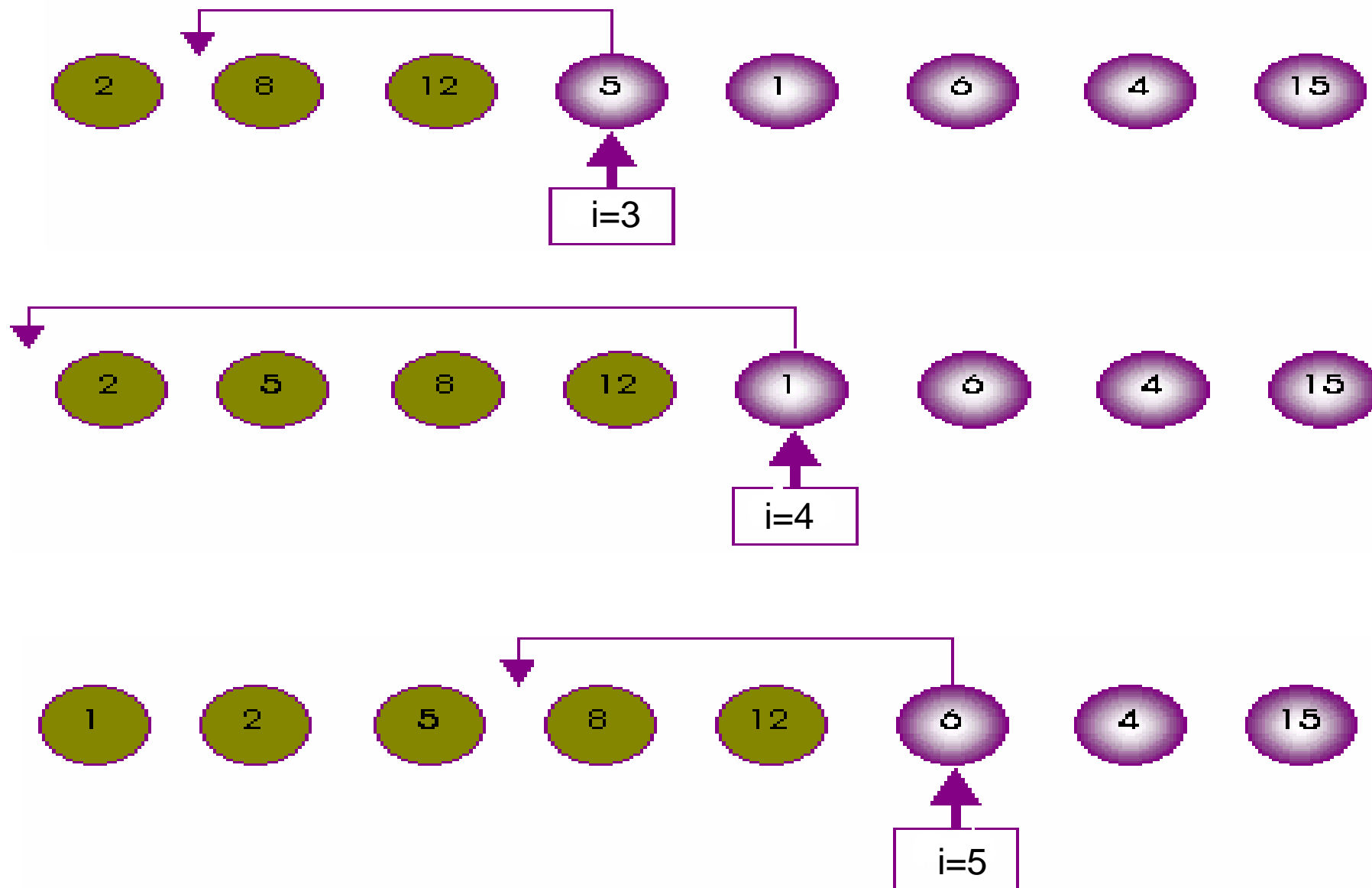
# Chèn Trực Tiếp – Insertion Sort

➤ Cho dãy số :

12      2      8      5      1      6      4      15

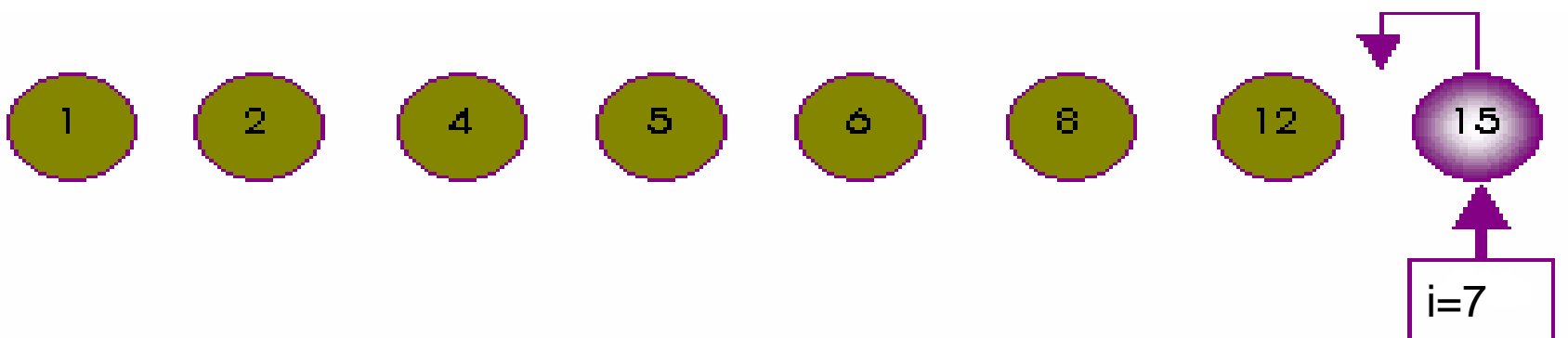
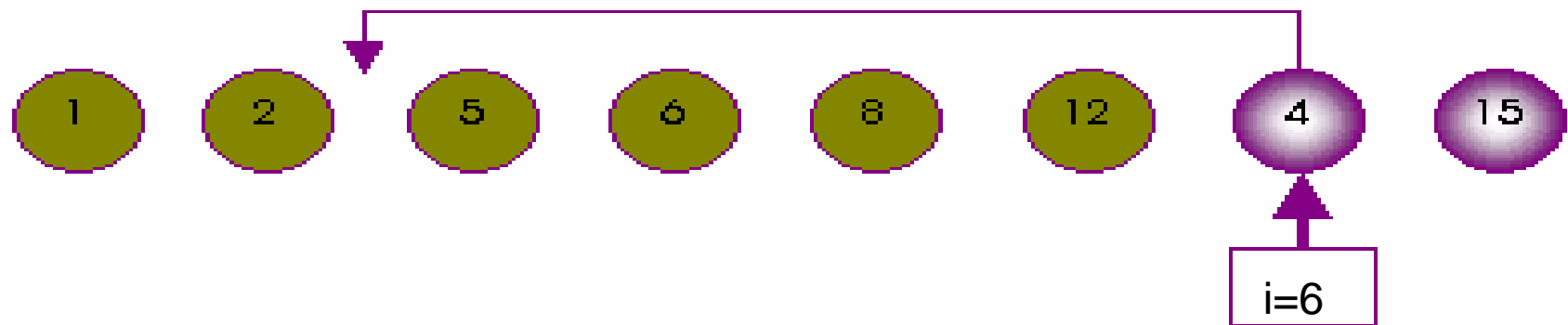


# Chèn Trực Tiếp – Insertion Sort





# Chèn Trực Tiếp – Insertion Sort

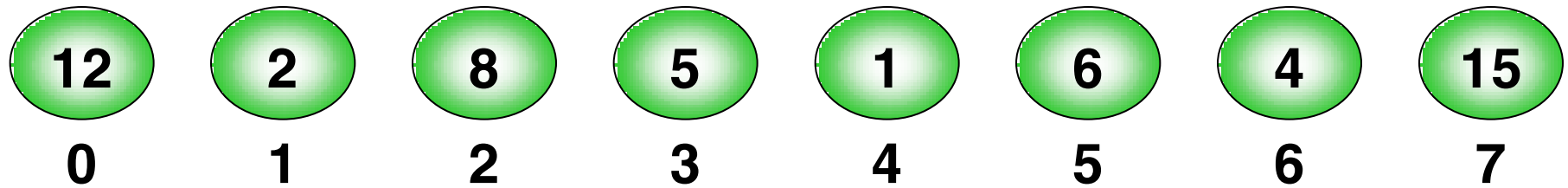


# Cài Đặt Thuật Toán Chèn Trực Tiếp

```
void InsertionSort(int a[], int n )
{   int pos, i;
    int x;    //lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
    for(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
    }
}
```

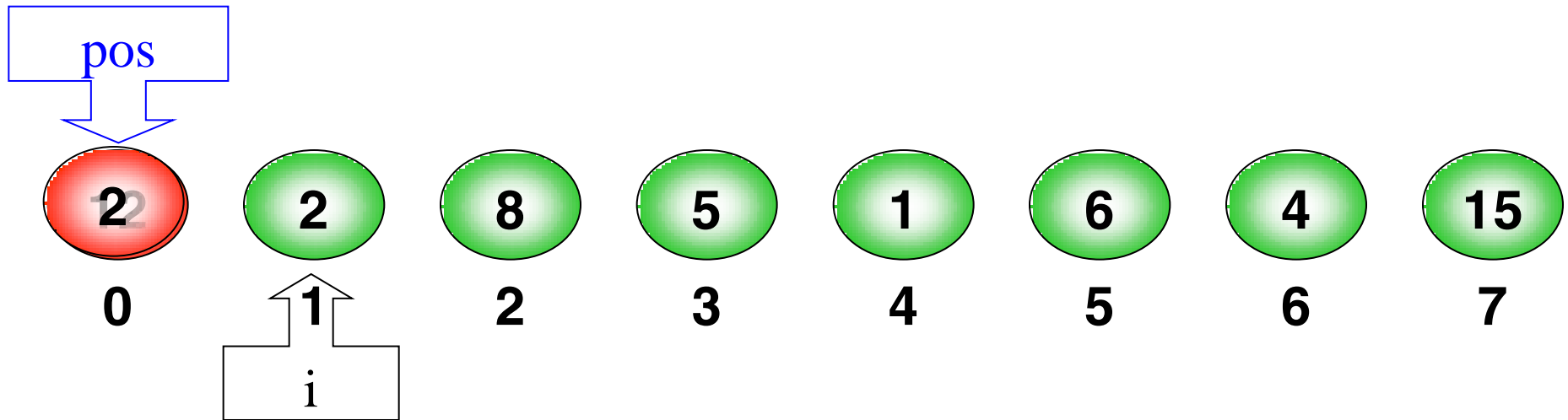


# Minh Họa Thuật Toán Insertion Sort



# Minh Họa Thuật Toán Insertion Sort

Insert  $a[1]$  into  $(0,0)$

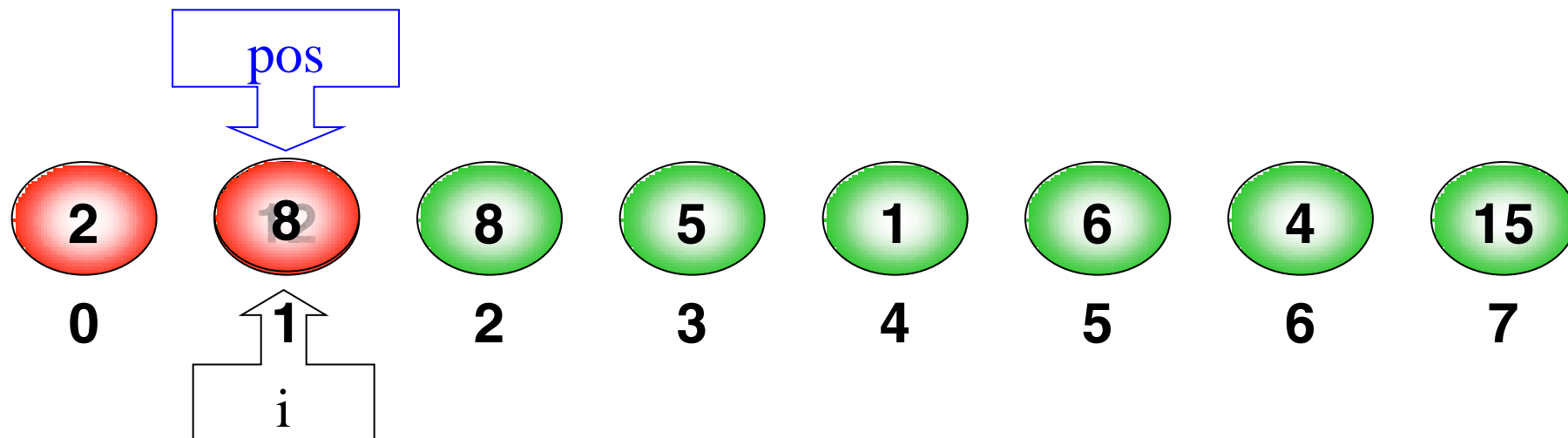


X



# Minh Họa Thuật Toán Insertion Sort

Insert  $a[2]$  into  $(0, 1)$

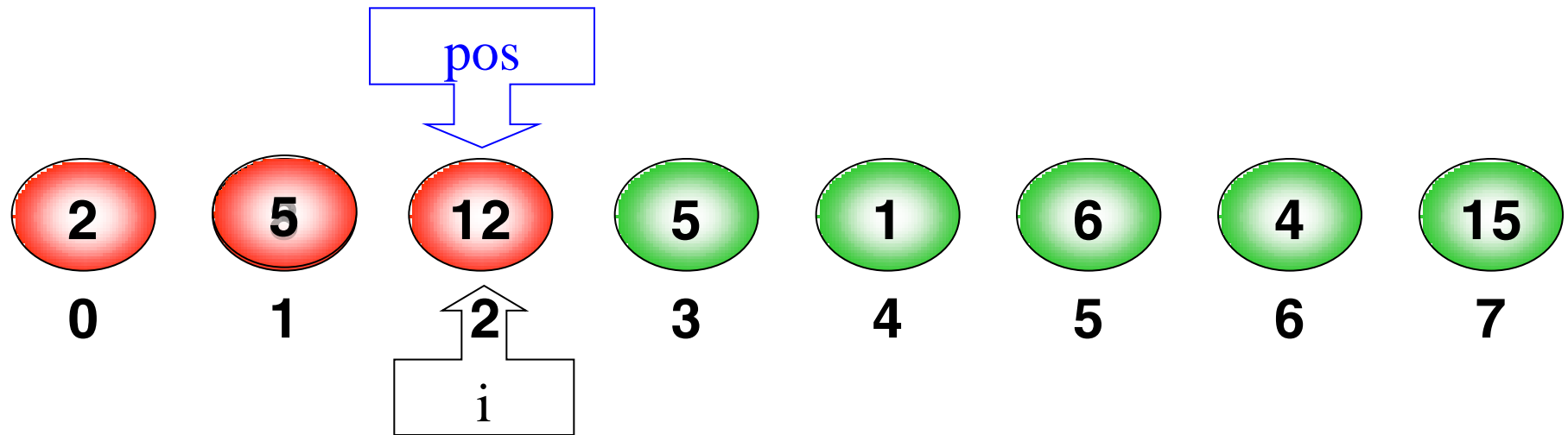


X



# Minh Họa Thuật Toán Insertion Sort

Insert  $a[3]$  into  $(0, 2)$

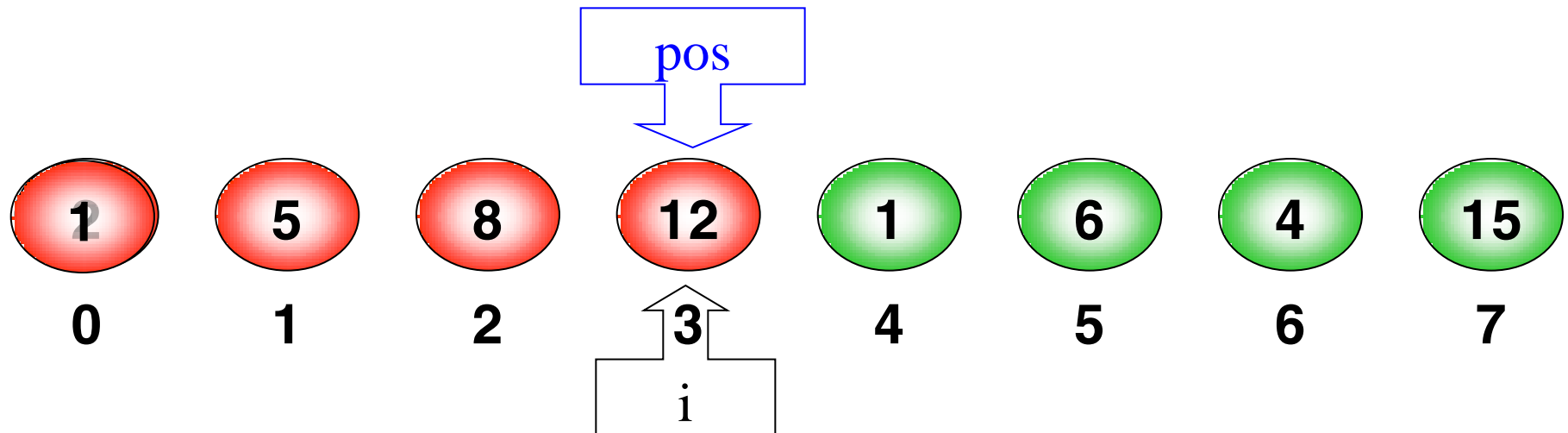


X



# Minh Họa Thuật Toán Insertion Sort

Insert  $a[4]$  into (0, 3)

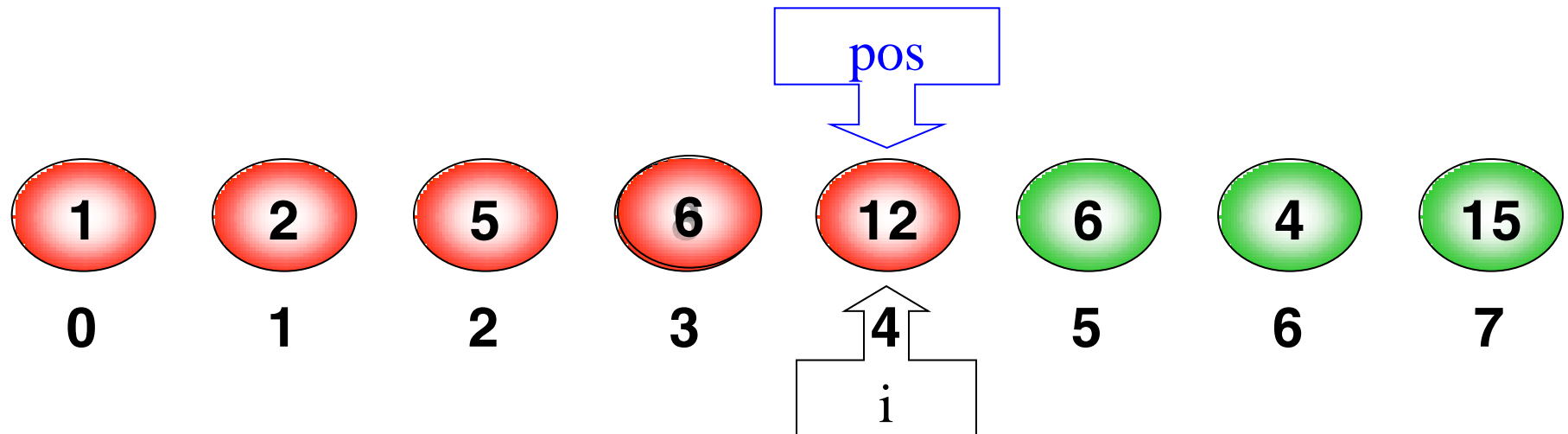


X



# Minh Họa Thuật Toán Insertion Sort

Insert  $a[5]$  into (0, 4)



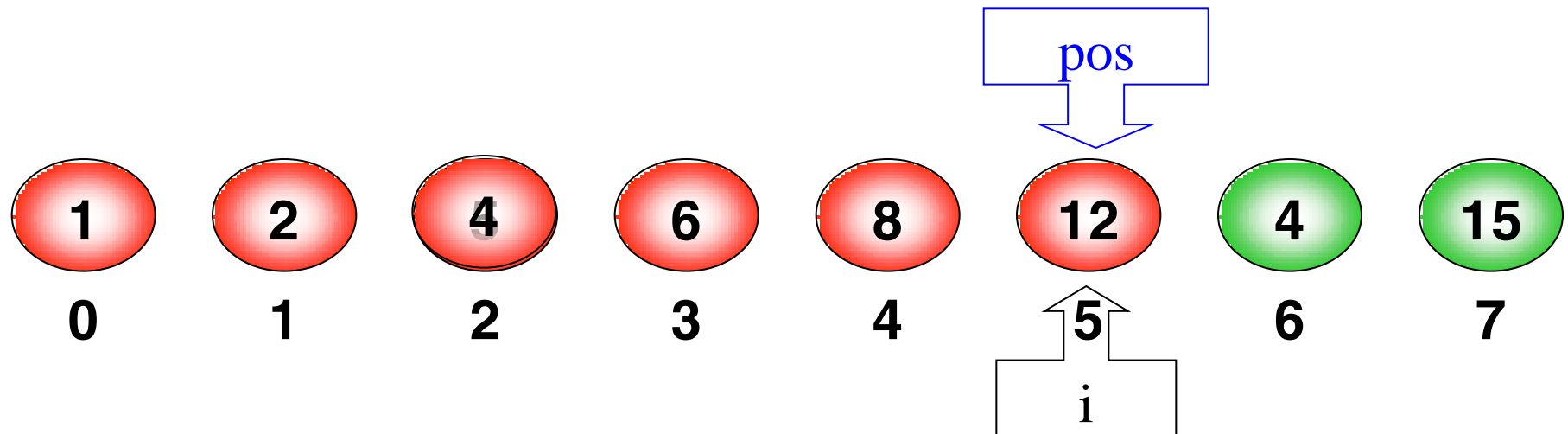
X





# Minh Họa Thuật Toán Insertion Sort

Insert  $a[6]$  into  $(0, 5)$

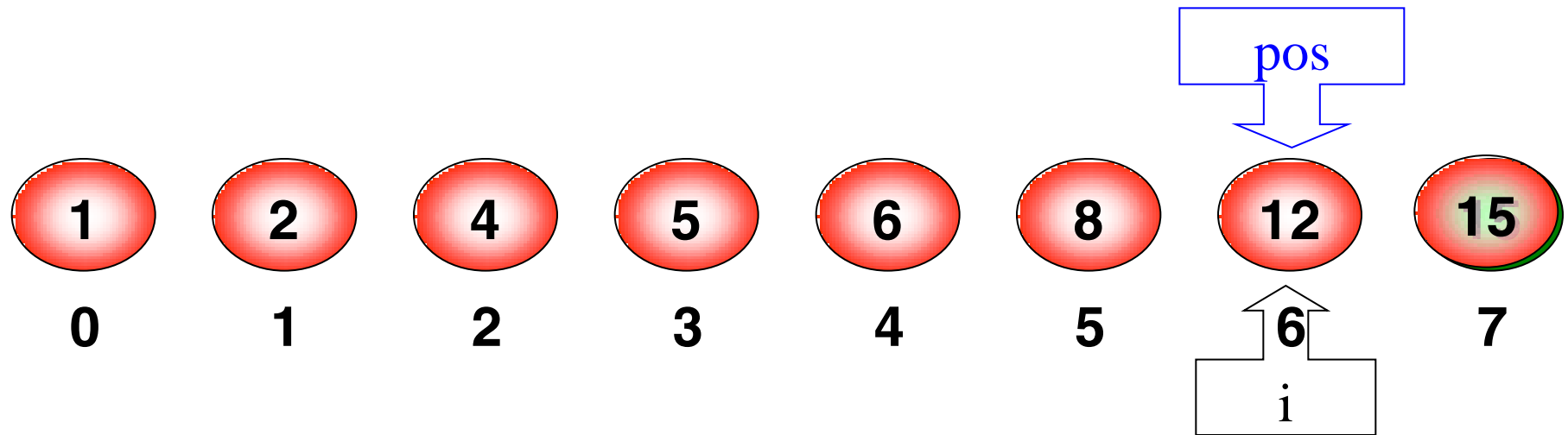


X



# Minh Họa Thuật Toán Insertion Sort

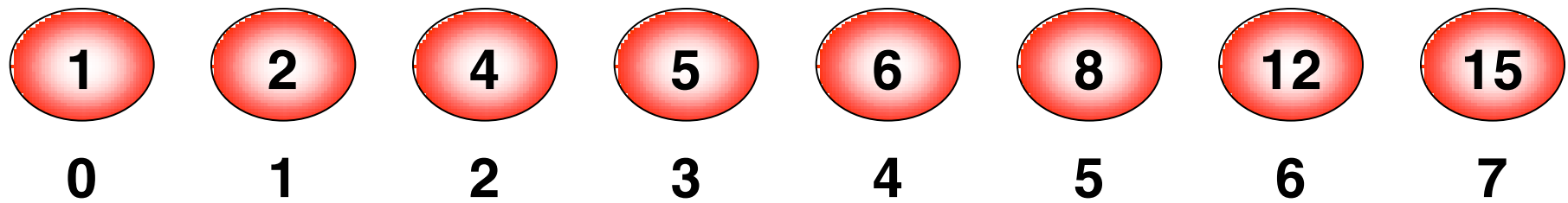
Insert  $a[8]$  into (0, 6)



X



# Minh Họa Thuật Toán Insertion Sort



# Độ Phức Tạp Của Insertion Sort

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

# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Chèn Nhị Phân – Binary Insertion Sort

```
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;      // tìm vị trí thích hợp m
            if(x < a[m]) r = m-1;
            else      l = m+1;
        }
        for(int j = i-1 ; j >=l ; j--)
            a[j+1] = a[j]; // dời các phần tử sẽ đứng sau x
        a[l] = x;           // chèn x vào dãy
    }
}
```



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Shell Sort

- 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 :  $a_1, a_2, \dots, a_n$  được xem như sự xen kẽ của các dãy con sau :
  - Dãy con thứ nhất :  $a_1 a_{h+1} a_{2h+1} \dots$
  - Dãy con thứ hai :  $a_2 a_{h+2} a_{2h+2} \dots$
  - ....
  - Dãy con thứ  $h$  :  $a_h a_{2h} a_{3h} \dots$



# 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 :

$$h_i > h_{i+1} \text{ và } h_k = 1$$

- $h_i = (h_{i-1} - 1)/3$  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 fibonacci: 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

- Bước 1: Chọn **k** khoảng cách  $h[1], h[2], \dots, h[k]$ ;  
 $i = 1$ ;
- 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

- Cho dãy số a:

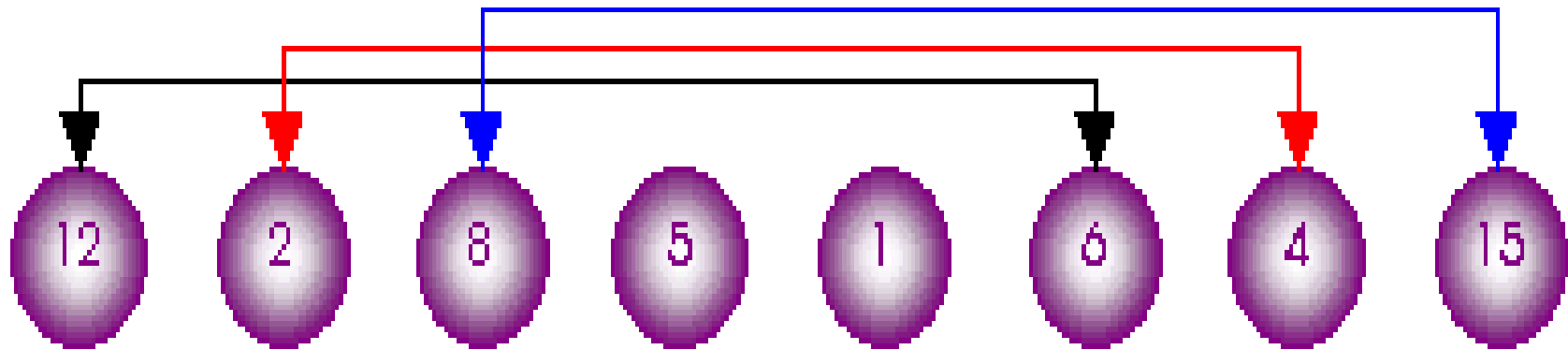
12      2    8      5      1      6      4      15

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



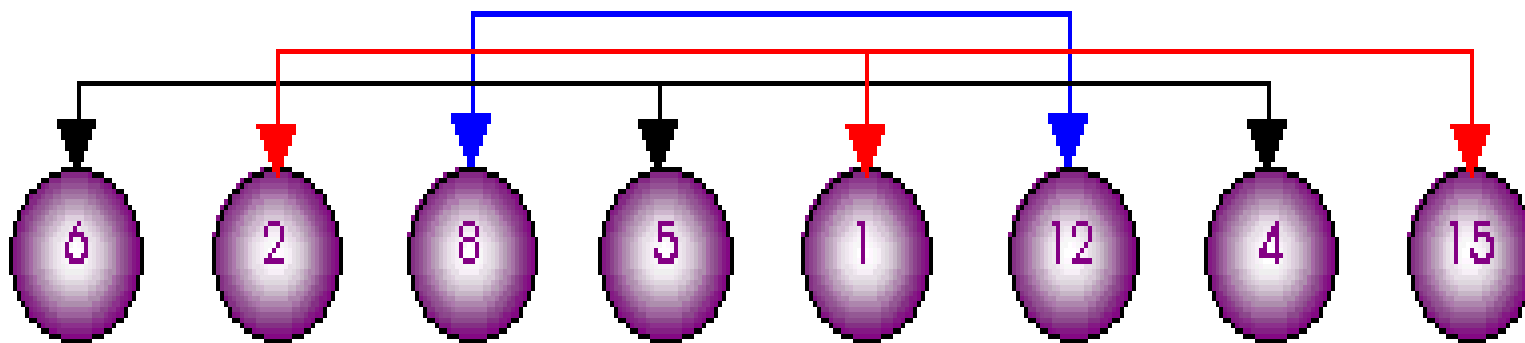
# Shell Sort

- $h = 5$  : xem dãy ban đầu như các dãy con



# Shell Sort

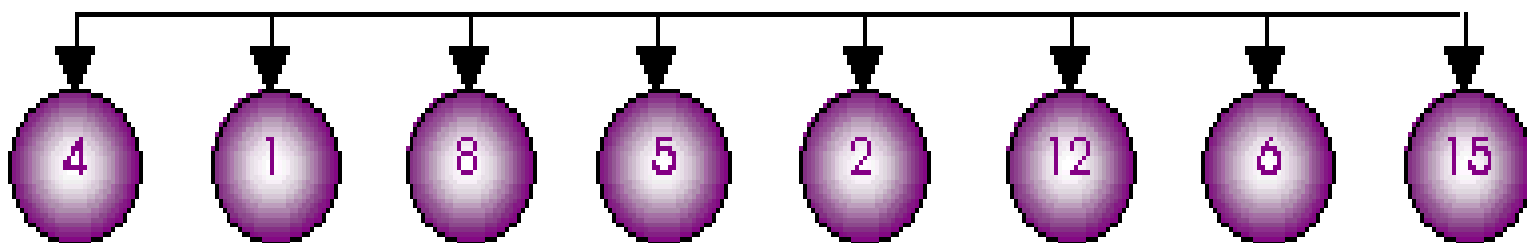
- $h = 3$  : (sau khi đã sắp xếp các dãy con ở bước trước)



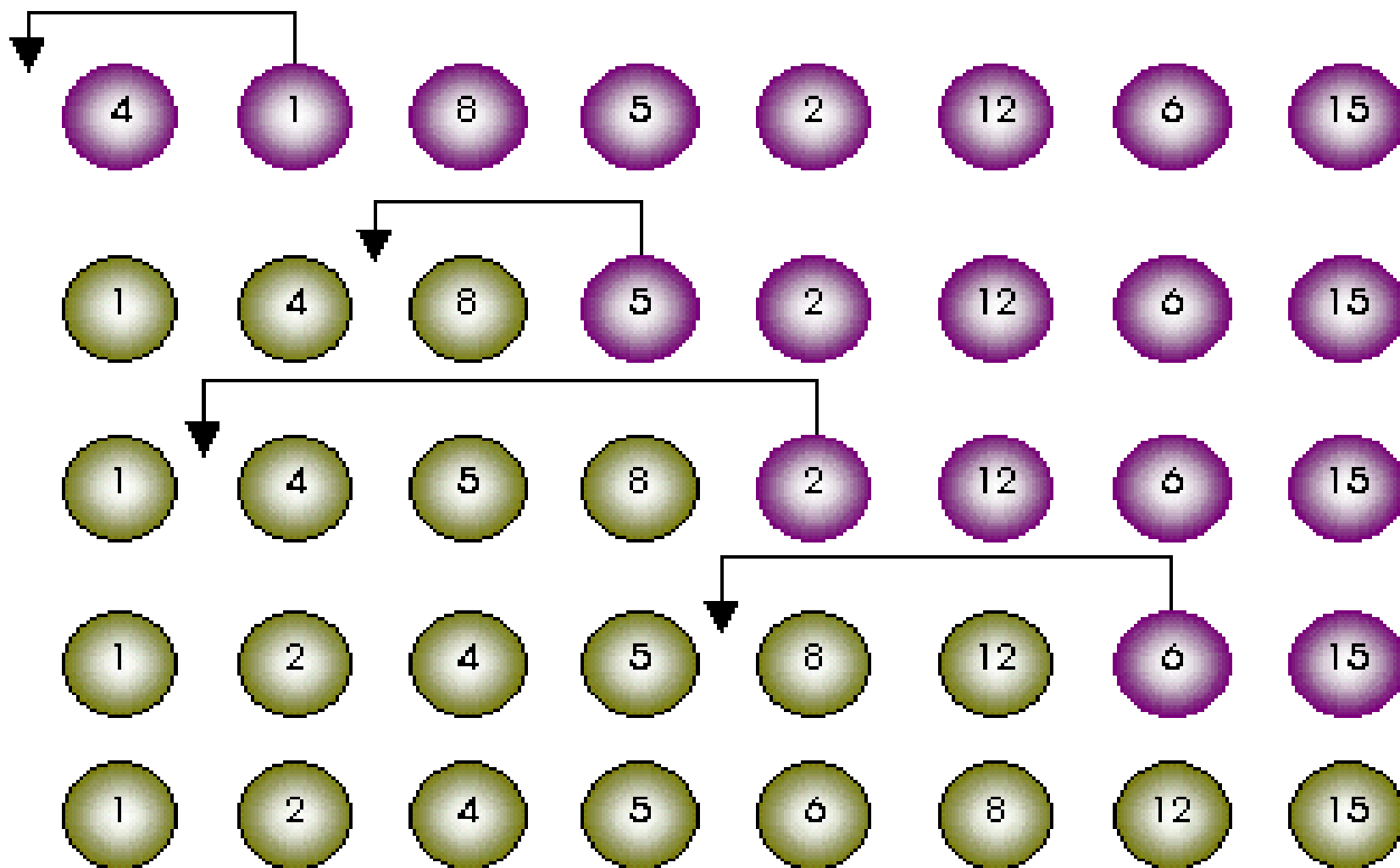


# Shell Sort

- $h = 1$  : (sau khi đã sắp xếp các dãy con ở bước trước)



# Shell Sort



# Shell Sort

```
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 ((x<a[j])&&(j>=0))// 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 – Ví Dụ

len = 5

joint

12  
0

2  
1

8  
2

5  
3

1  
4

6  
5

4  
6

15  
7

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

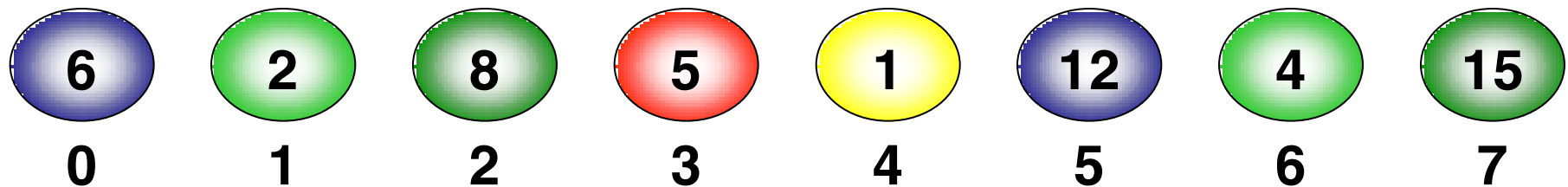
curr



# Shell Sort – Ví Dụ

**len = 5;**

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



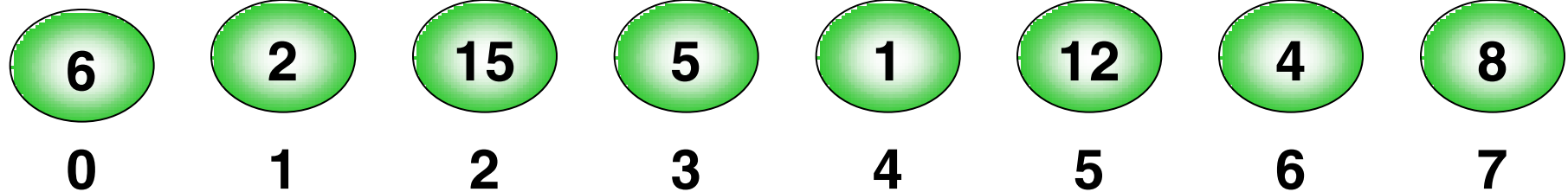
# Shell Sort – Ví Dụ

**len = 3**

joint

curr

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



# Shell Sort – Ví Dụ

**len = 3**

joint



0



1



2



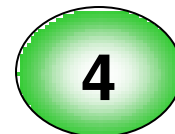
3



4



5



6



7

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

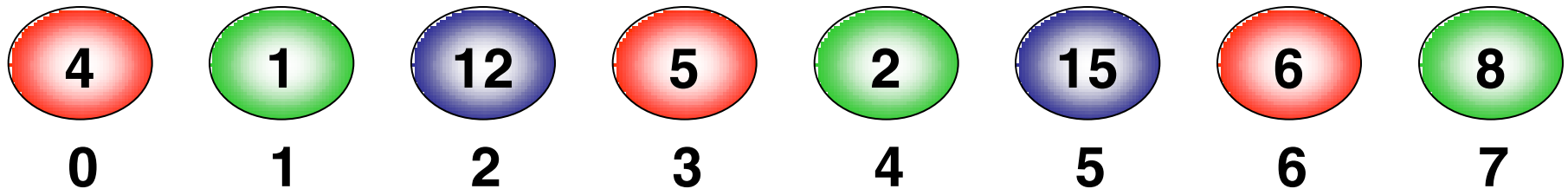
curr



# Shell Sort – Ví Dụ

**len = 3**

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

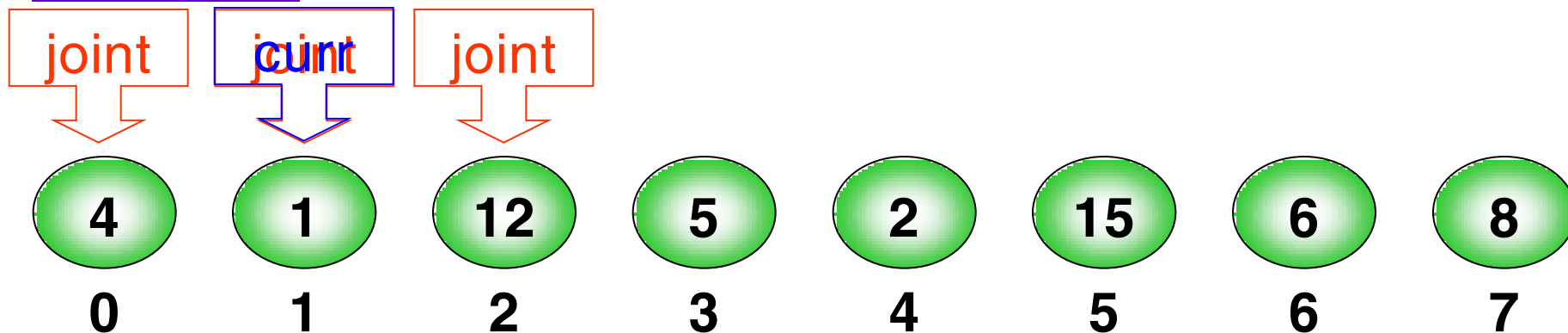




# Shell Sort – Ví Dụ

**len = 1**

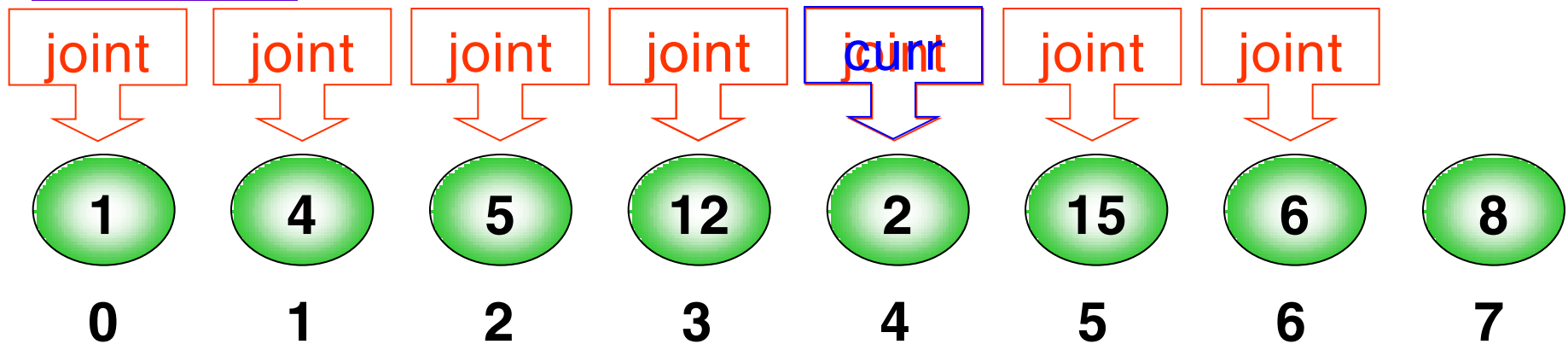
**$h = (5, 3, 1); k = 3$**



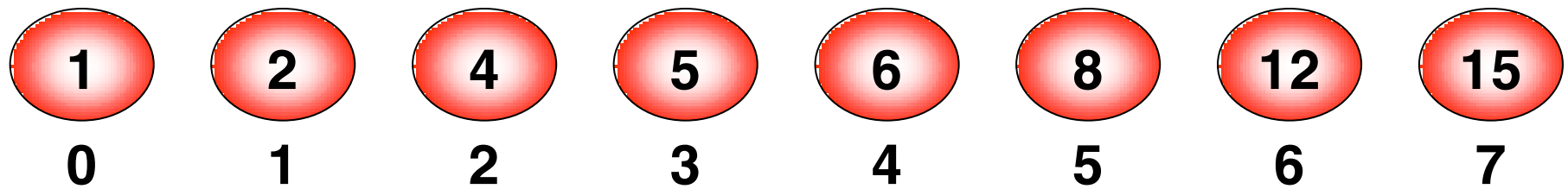
# Shell Sort – Ví Dụ

**len = 1**

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



# Shell Sort – Ví Dụ



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



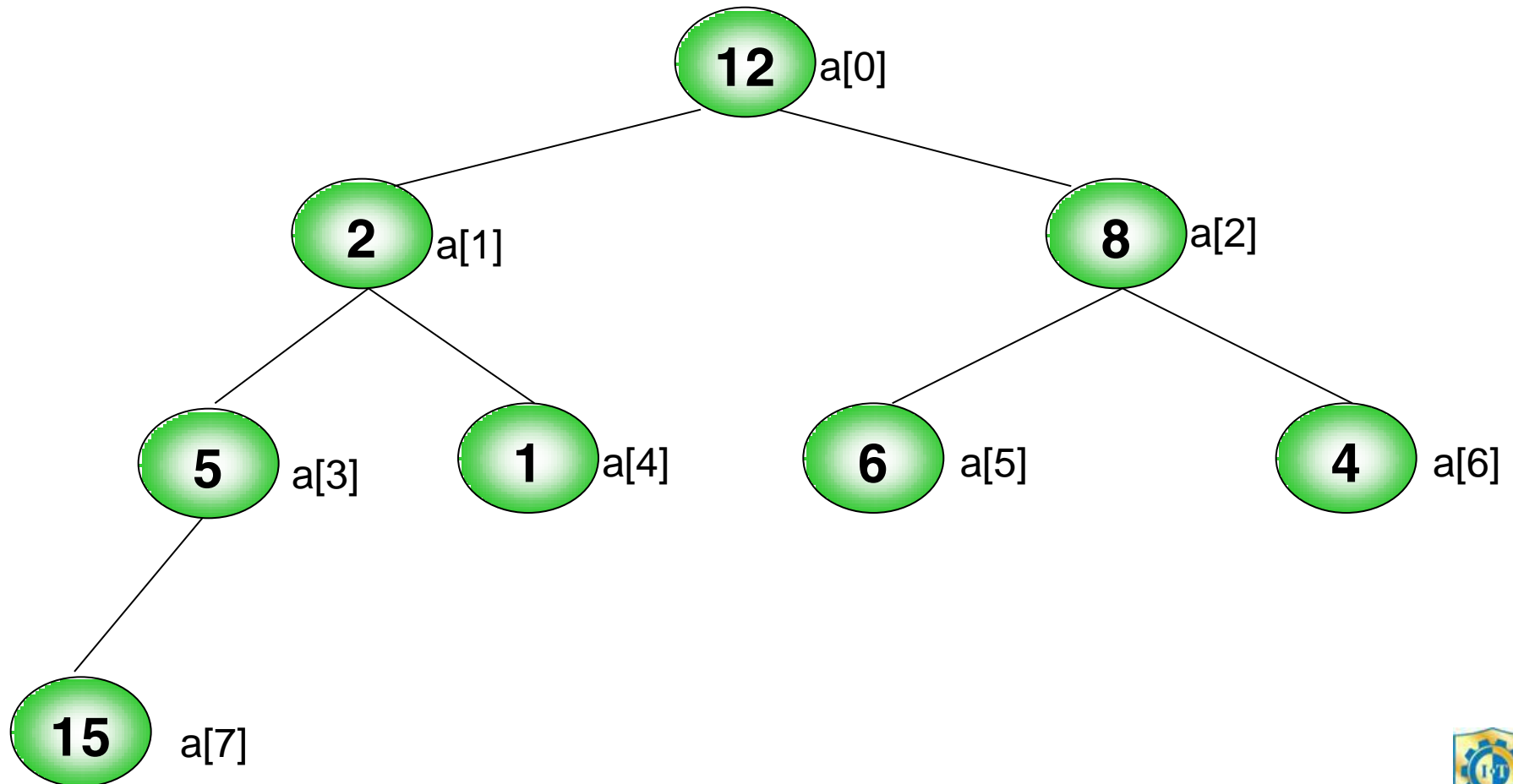
# Thuật Toán Sắp Xếp Heap Sort

- Heap Sort tận dụng được các phép so sánh ở bước  $i-1$  mà thuật toán sắp xếp chọn trực tiếp không tận dụng được
- Để làm được điều này Heap sort thao tác dựa trên cây.



# Thuật Toán Sắp Xếp Heap Sort

➤ Cho dãy số : 12 2 8 5 1 6 4 15  
0 1 2 3 4 5 6 7



# Thuật toán sắp xếp Heap Sort

- Ở cây trên, phần tử ở mức  $i$  chính là phần tử lớn trong cặp phần tử ở mức  $i + 1$ , do đó phần tử ở nút gốc là phần tử lớn nhất.
- Nếu loại bỏ gốc ra khỏi cây, thì việc cập nhật cây chỉ xảy ra trên những nhánh liên quan đến phần tử mới loại bỏ, còn các nhánh khác thì bảo toàn.
- Bước kế tiếp có thể sử dụng lại kết quả so sánh của bước hiện tại.
- Vì thế độ phức tạp của thuật toán  $O(n \log_2 n)$



# Các Bước Thuật Toán

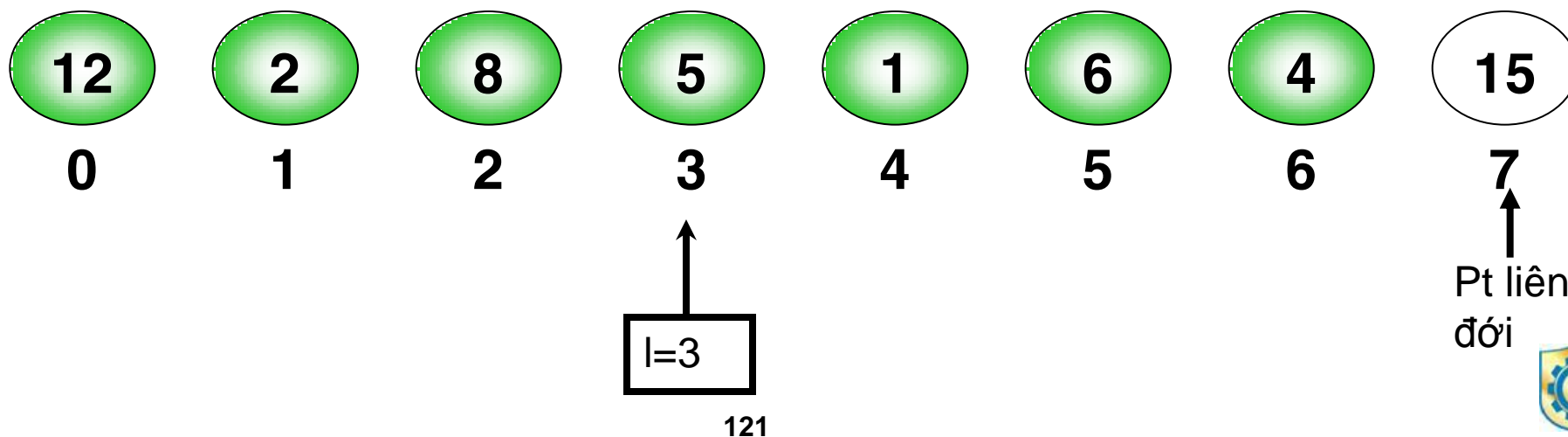
- Giai đoạn 1 : Hiệu chỉnh dãy số ban đầu thành heap
- Giai đoạn 2: Sắp xếp dãy số dựa trên heap:
  - ↪ Bước 1: Đưa phần tử lớn nhất về vị trí đúng ở cuối dãy:  
$$r = n-1; \text{ Swap } (a_1, a_r);$$
  - ↪ Bước 2: Loại bỏ phần tử lớn nhất ra khỏi heap:  $r = r-1;$   
Hiệu chỉnh phần còn lại của dãy từ  $a_1, a_2 \dots a_r$  thành một heap.
  - ↪ Bước 3:  
Nếu  $r > 1$  (heap còn phần tử ): Lặp lại Bước 2  
Ngược lại : Dừng



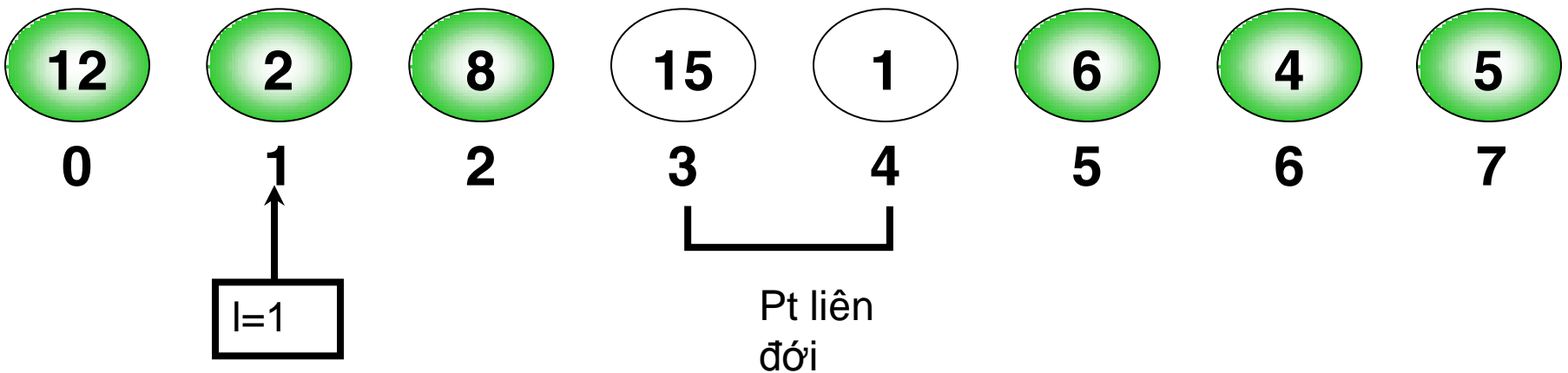
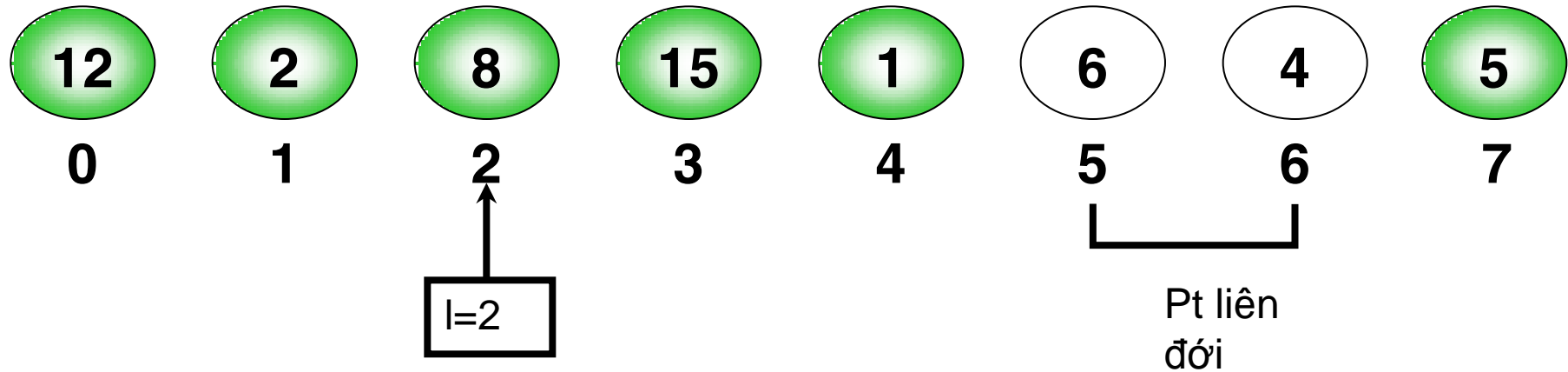


# Minh Họa Thuật Toán

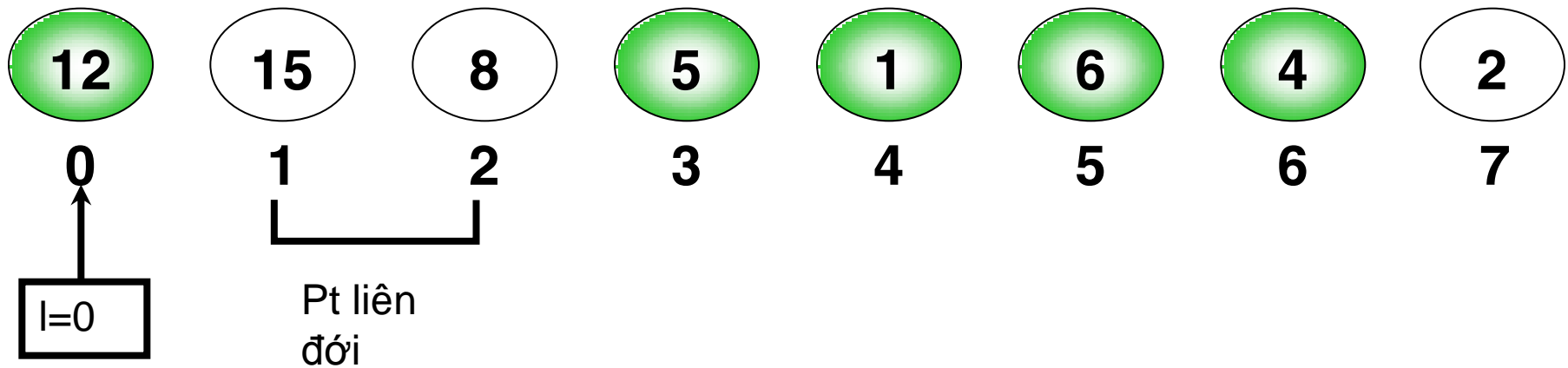
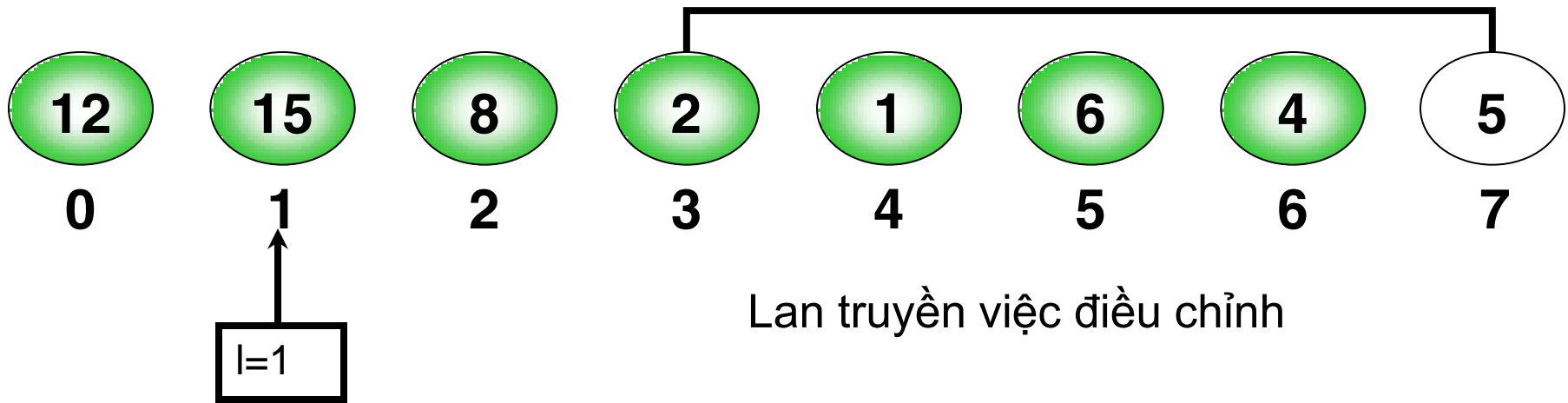
- **Heap:** Là một dãy các phần tử  $a_l, a_{l+1}, \dots, a_r$  thoả các quan hệ với mọi  $i \in [l, r]$ :
  - ↪  $a_i \geq a_{2i+1}$
  - ↪  $a_i \geq a_{2i+2} // (a_i, a_{2i+1}), (a_i, a_{2i+2})$  là các cặp phần tử liên đới
- Cho dãy số : 12 2 8 5 1 6 4 15
- Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành Heap



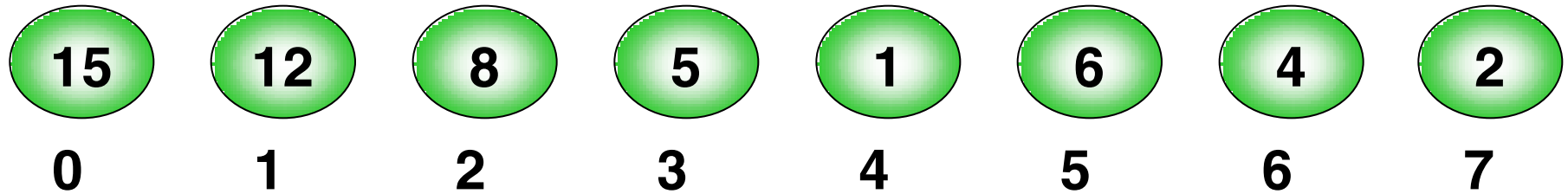
# Minh Họa Thuật Toán



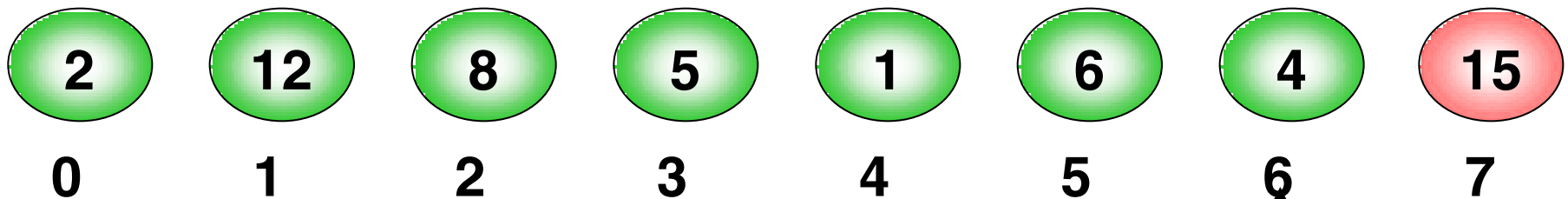
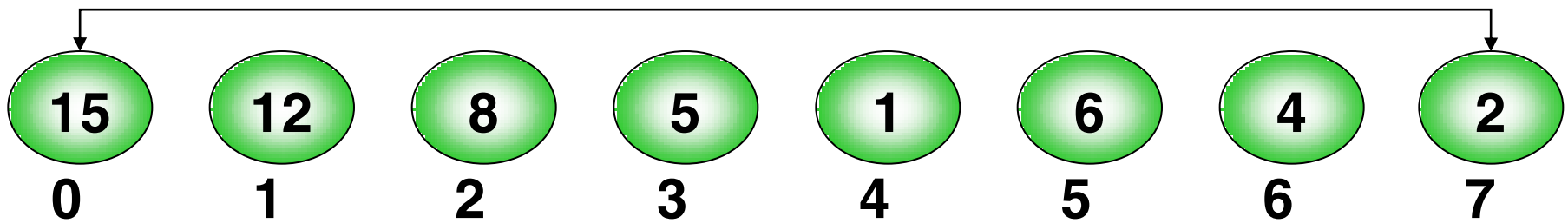
# Minh Họa Thuật Toán



# Minh Họa Thuật Toán



➤ Giai đoạn 2: Sắp xếp dãy số dựa trên Heap

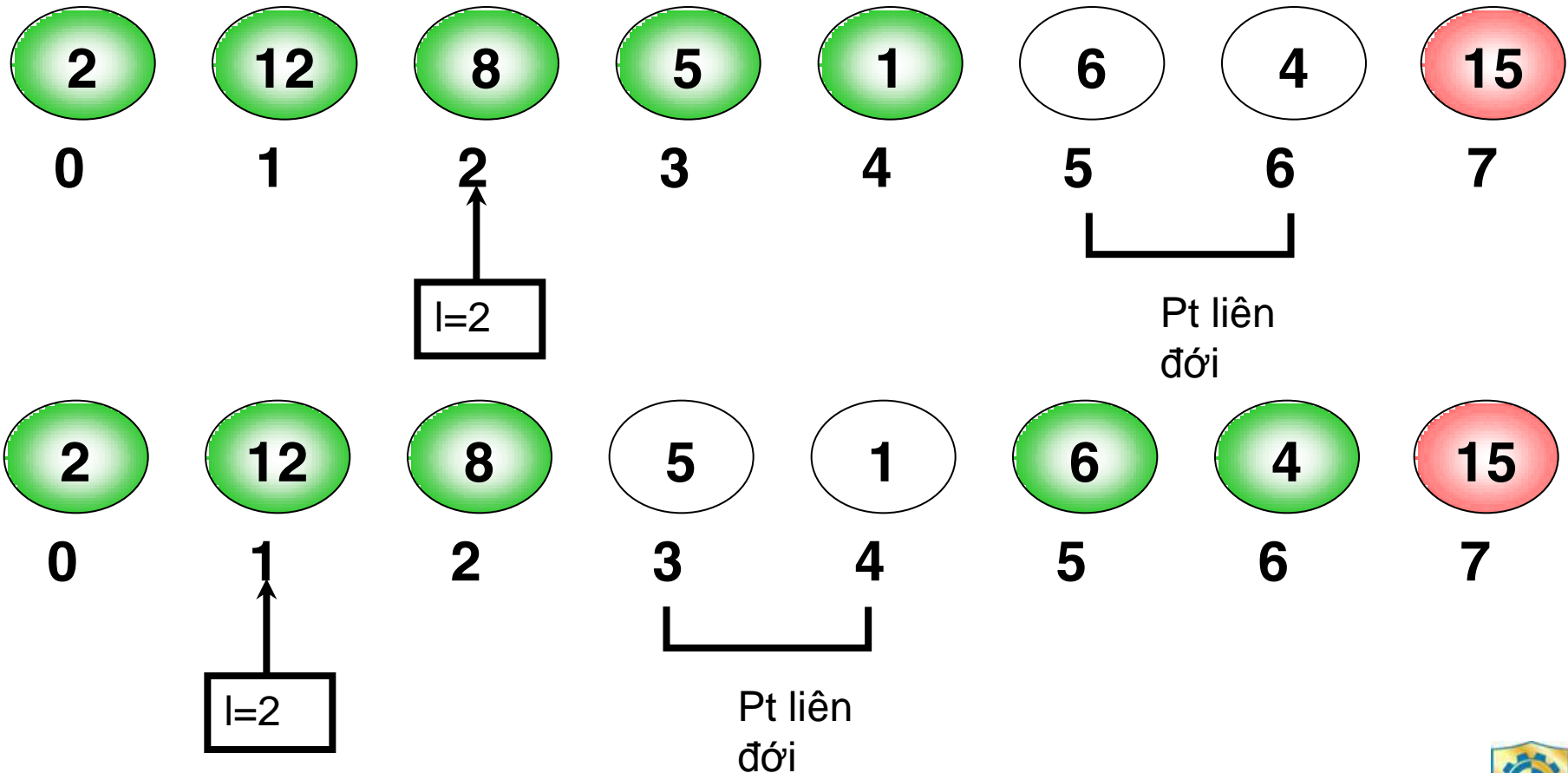


$r=6$

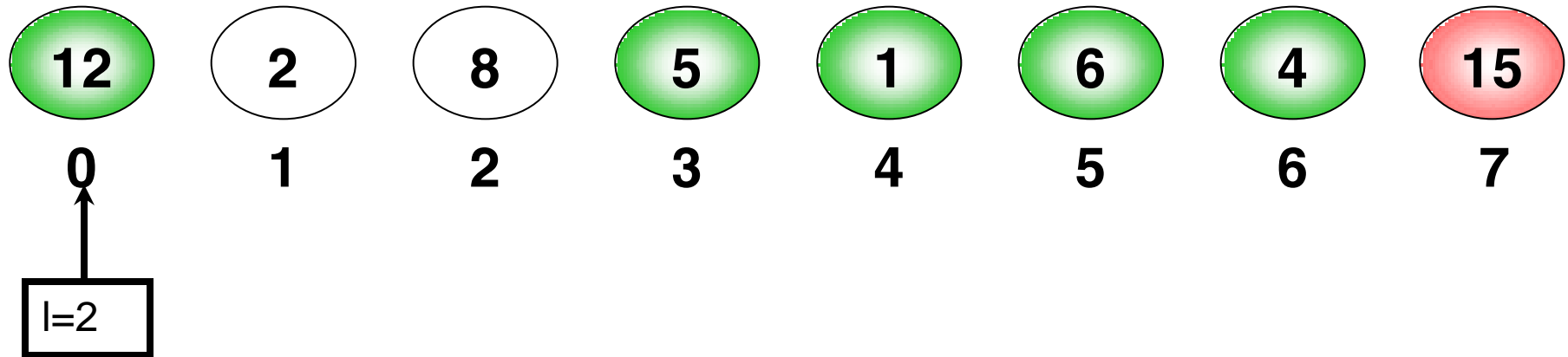
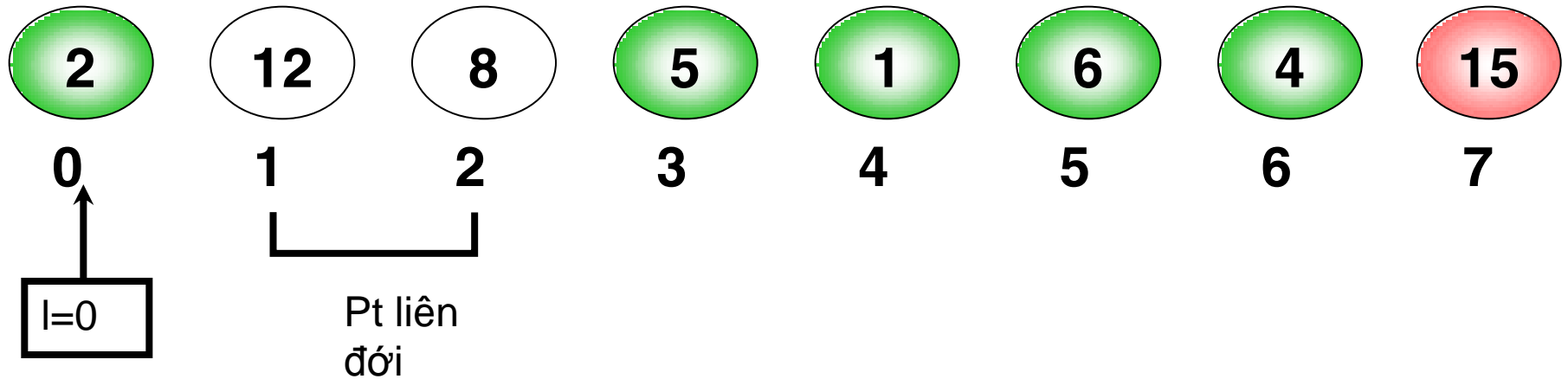


# Minh Họa Thuật Toán

## ➤ Hiệu chỉnh Heap

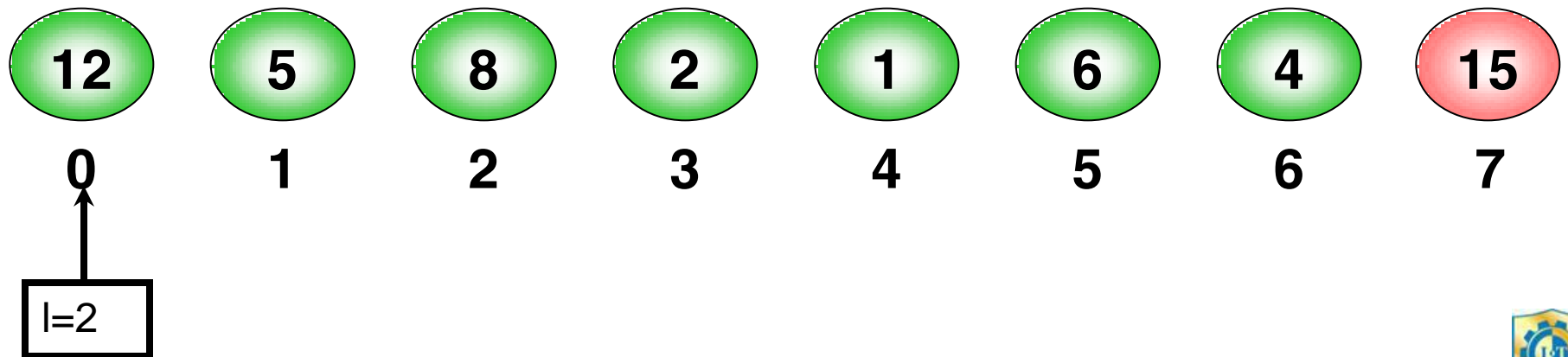
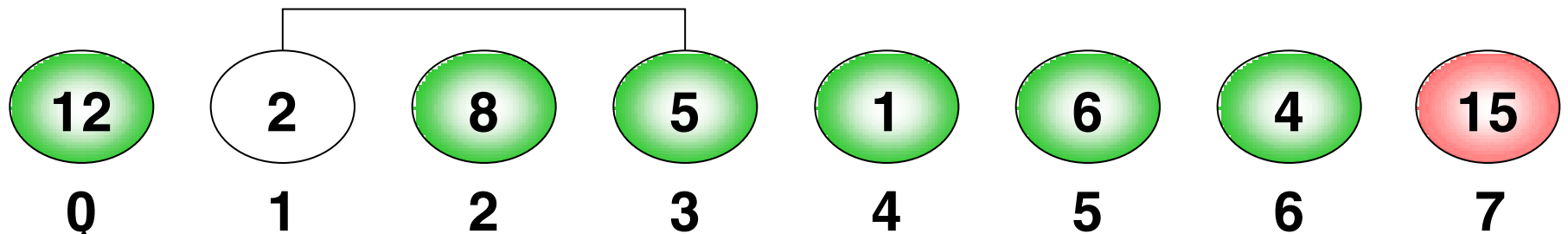


# Minh Họa Thuật Toán

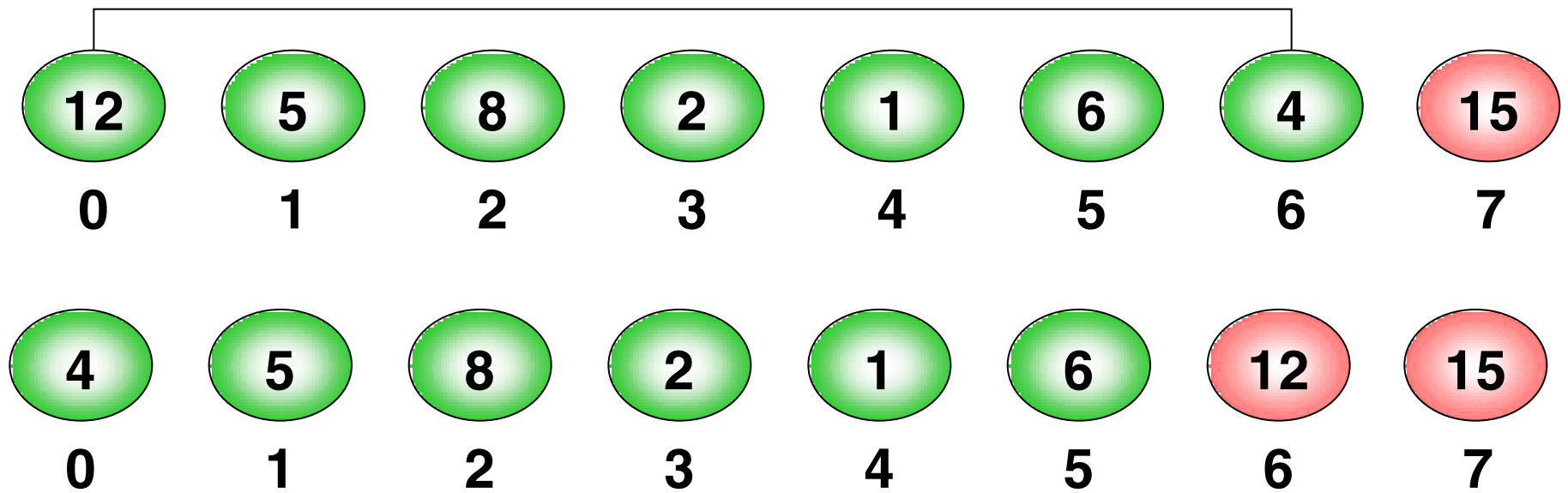


# Minh Họa Thuật Toán

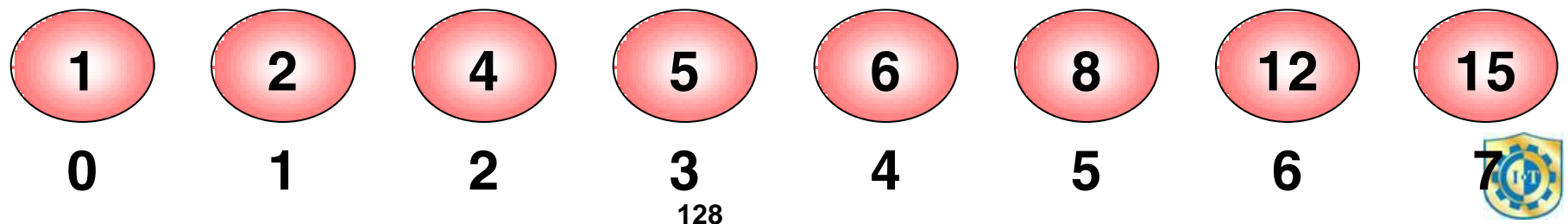
Lan truyền việc điều chỉnh



# Minh Họa Thuật Toán



➤ Thực hiện với  $r = 5, 4, 3, 2$  ta được





# Cài Đặt Thuật Toán

- Hiệu chỉnh  $a_l, a_{l+1}, \dots, a_r$  thành Heap

```
void shift(int a[], int l, int r)
```

```
{
```

```
    int x, i, j;
```

```
    i = l;
```

```
    j = 2 * i + 1;
```

```
    x = a[i];
```

```
    while(j <= r)
```

```
    {        if(j < r)
```

```
                if(a[j] < a[j+1]) // tìm phần tử lớn nhất a[j] và a[j+1]
```



# Cài Đặt Thuật Toán

`j++;` //lưu chỉ số của phần tử nhỏ nhất trong hai phần tử

`if(a[j]<=x) return;`

`else`

`{ a[i]=a[j];`

`a[j]=x;`

`i=j;`

`j=2*i+1;`

`x=a[i];`

`}`

`}`

`}`



# Cài Đặt Thuật Toán

- Hiệu chỉnh  $a_0, \dots, a_{n-1}$  Thành Heap

```
void CreateHeap(int a[],int n)
{   int l;
    l=n/2-1;
    while(l>=0)
    {
        shift(a,l,n-1);
        l=l-1;
    }
}
```



# Cài Đặt Thuật Toán

## ➤ Hàm HeapSort

```
void HeapSort(int a[],int n)
{   int r;
    CreateHeap(a,n);
    r=n-1;
    while(r>0)
    {
        Swap(a[0],a[r]); //a[0] là nút gốc
        r--;
        if(r>0)
            shift(a,0,r);
    }
}
```



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Quick Sort

## ➤ Ý tưởng:

- Giải thuật QuickSort sắp xếp dãy  $a_1, a_2, \dots, a_N$  dựa trên việc phân hoạch dãy ban đầu thành 3 phần :
  - Phần 1: Gồm các phần tử có giá trị bé hơn  $x$
  - Phần 2: Gồm các phần tử có giá trị bằng  $x$
  - Phần 3: Gồm các phần tử có giá trị lớn hơn  $x$

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



# 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  $k = i..N$

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------



# Quick Sort – Ý Tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- 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.





# Quick Sort – Ý Tưởng

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy ban đầu chỉ có thứ tự khi các đoạn 1, 3 được sắp.
- Để sắp xếp các đoạn 1 và 3, ta lần lượt tiến hành việc phân hoạch từng dãy con theo cùng phương pháp phân hoạch dãy ban đầu vừa trình bày ...



# Giải Thuật Quick Sort

- Bước 1: Nếu  $\text{left} \geq \text{right}$  //dãy có ít hơn 2 phần tử

Kết thúc; //dãy đã được sắp xếp

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

*Đoạn 1  $\leq x$*

*Đoạn 2:  $a_{j+1} \dots a_{i-1} = x$*

*Đoạn 3:  $a_i \dots a_{\text{right}} \geq x$*

- Bước 3: **Sắp xếp đoạn 1**:  $a_{\text{left}} \dots a_j$

- Bước 4: **Sắp xếp đoạn 3**:  $a_i \dots a_{\text{right}}$



# Giải Thuật Quick Sort

- Bước 1 : Chọn tùy ý một phần tử  $a[k]$  trong dãy là giá trị mốc ( $l \leq k \leq r$ ):

$$x = a[k]; \quad i = l; \quad j = r;$$

- Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ :

- Bước 2a : Trong khi  $(a[i] < x)$   $i++$ ;
- Bước 2b : Trong khi  $(a[j] > x)$   $j--$ ;
- Bước 2c : Nếu  $i < j$  Swap( $a[i], a[j]$ );

- Bước 3 : Nếu  $i < j$ : Lặp lại Bước 2.  
Ngược lại: Dừng



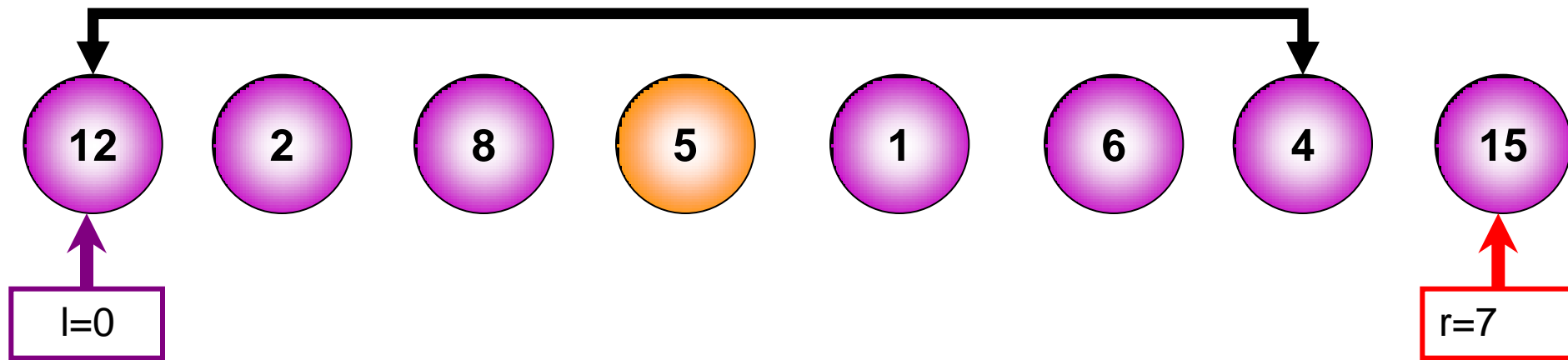
# Quick Sort – Ví Dụ

➤ Cho dãy số a:

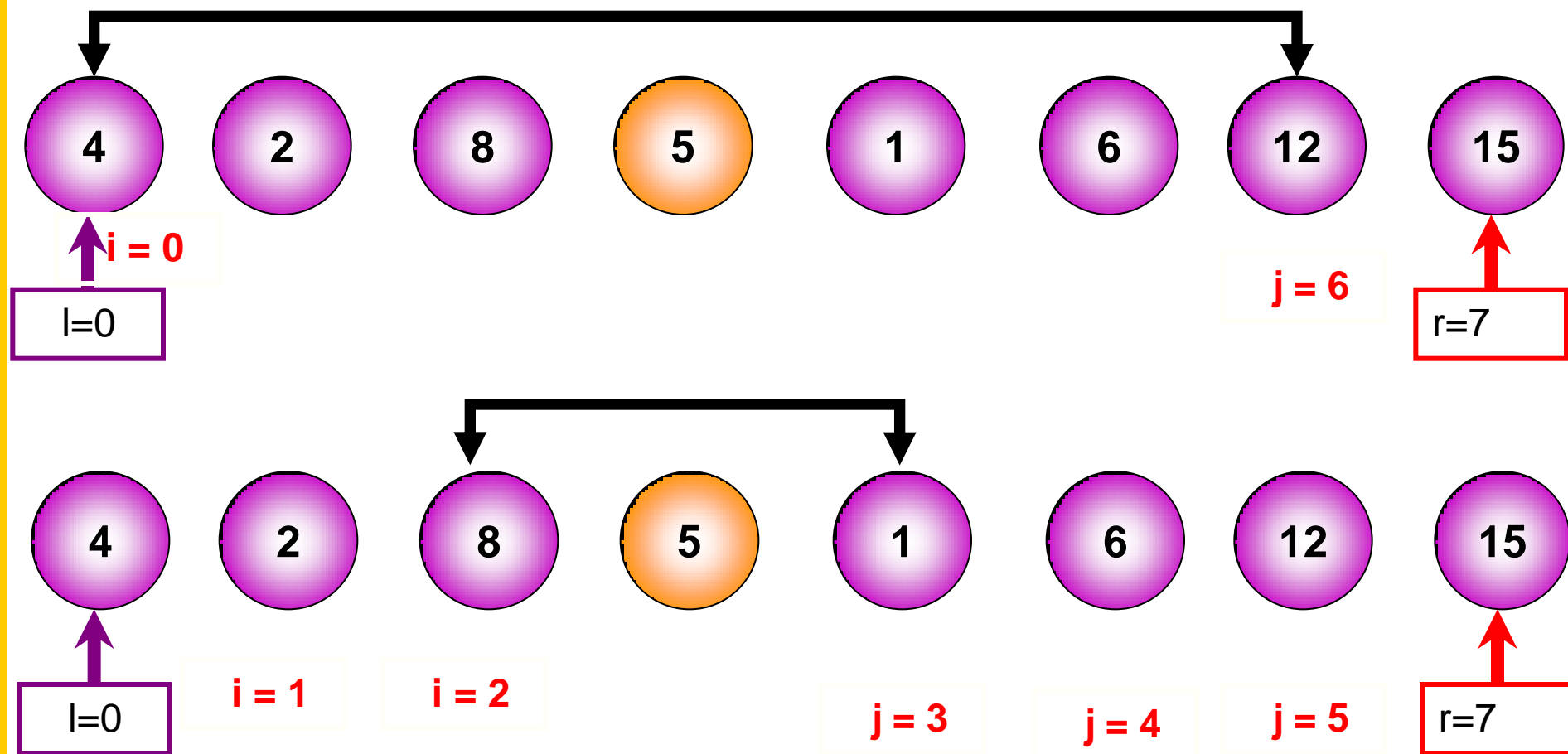
12      2      8      5      1      6      4      15

Phân hoạch đoạn  $l = 0, r = 7$ :

$x = a[3] = 5$

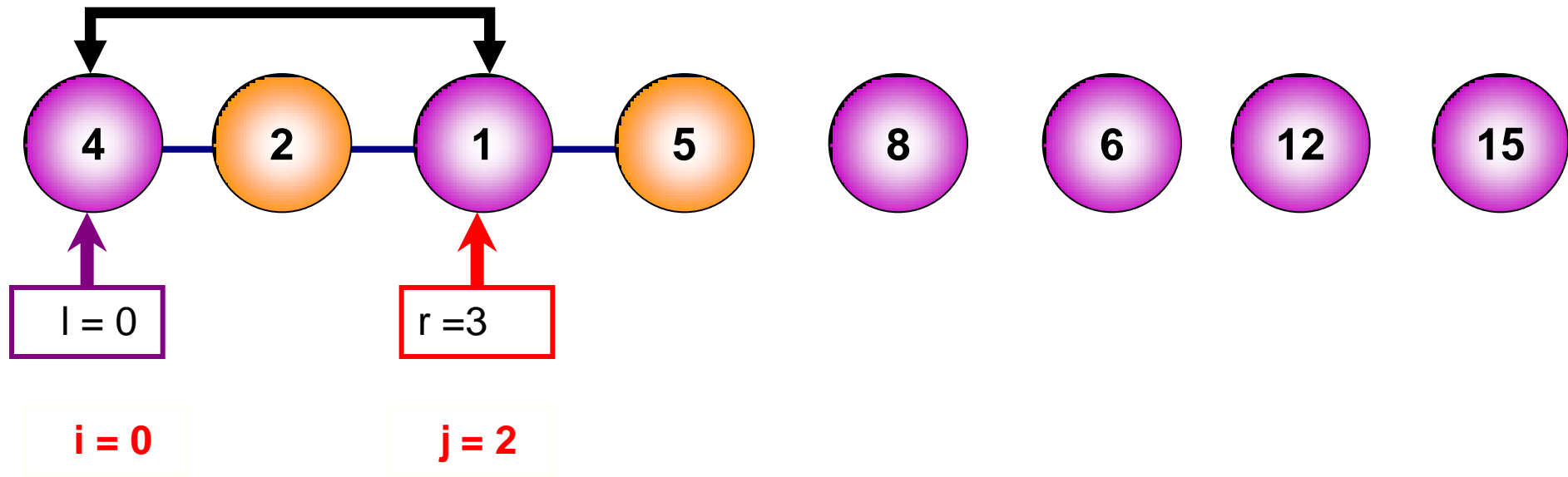


# Quick Sort – Ví Dụ



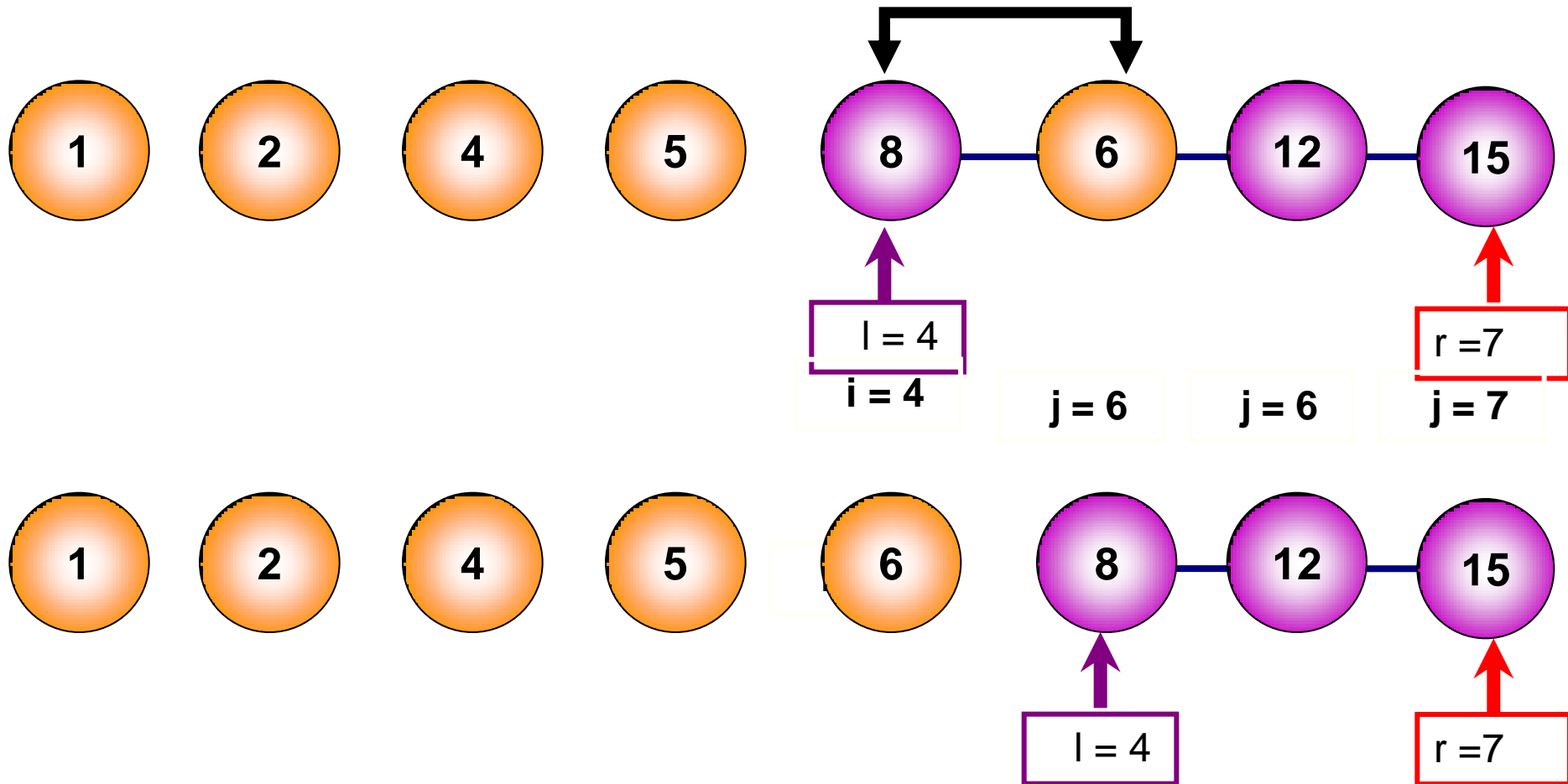
# Quick Sort – Ví Dụ

- Phân hoạch đoạn  $l = 0, r = 2$ :



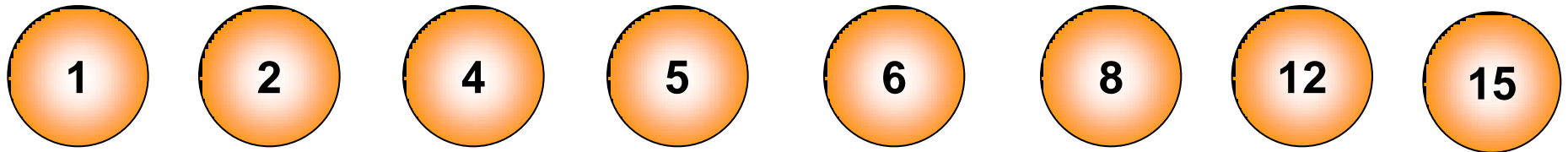
# Quick Sort – Ví Dụ

➤ Phân hoạch đoạn  $l = 4, r = 7$ :



# Quick Sort – Ví Dụ

- Phân hoạch đoạn  $l = 6, r = 7$ :





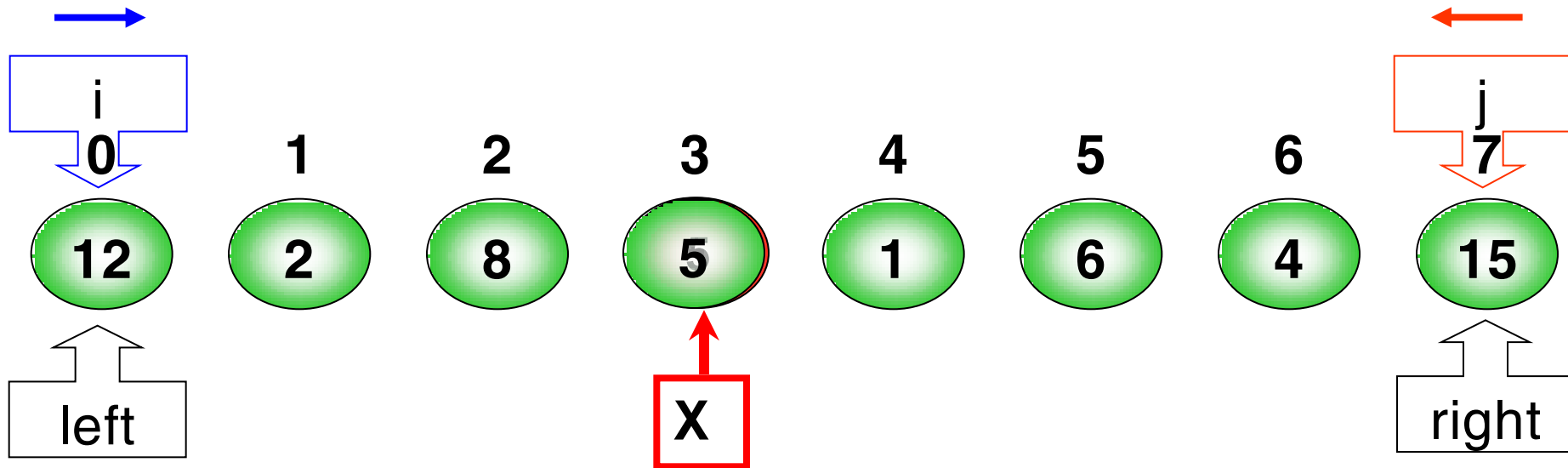
# Quick Sort

```
void QuickSort(int a[], int left, int right)
{
    int i, j, x;
    x = a[(left+right)/2];
    i = left; j = right;
    do
    {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
        if(i <= j)
        {
            Swap(a[i],a[j]);
            i++; j--;
        }
    } while(i <= j);
    if(left < j)
        QuickSort(a, left, j);
    if(i < right)
        QuickSort(a, i, right);
}
```



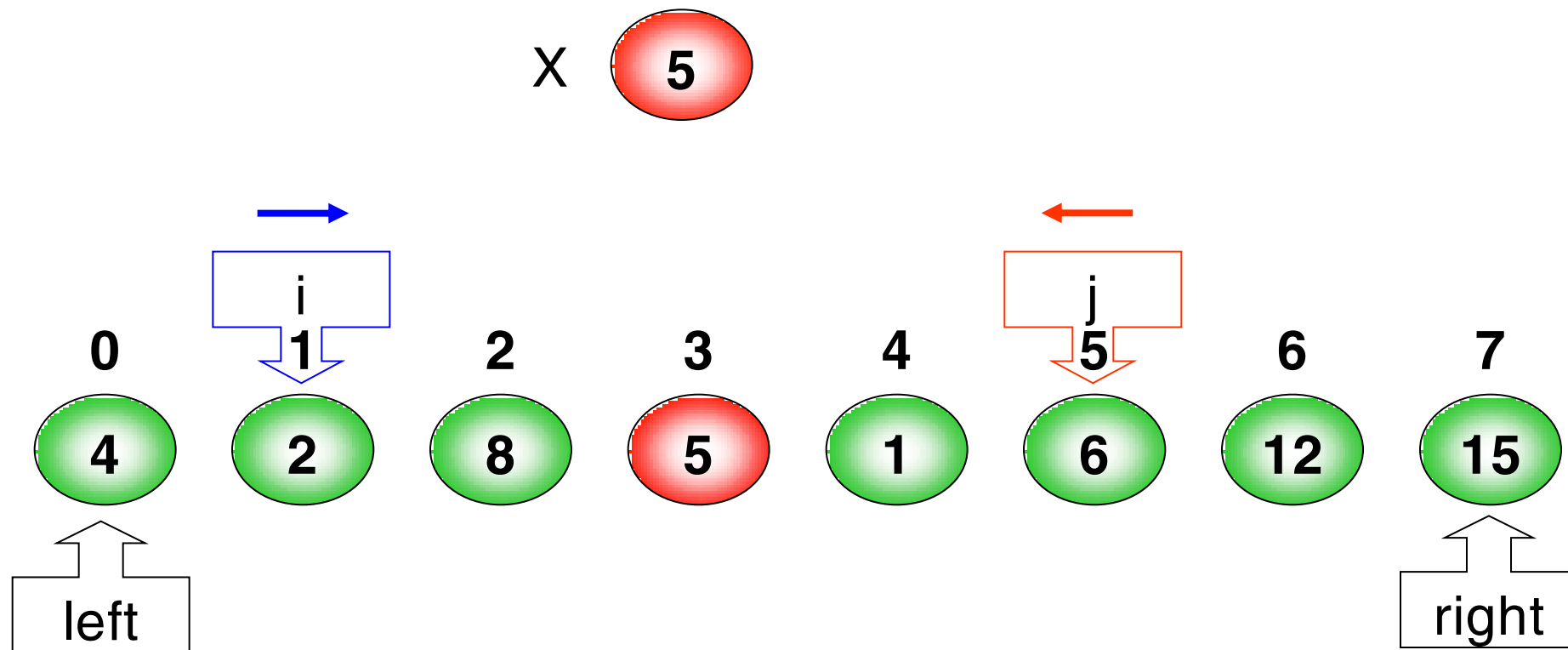
# Quick Sort – Ví Dụ

➤ Phân hoạch đoạn  $[0,7]$



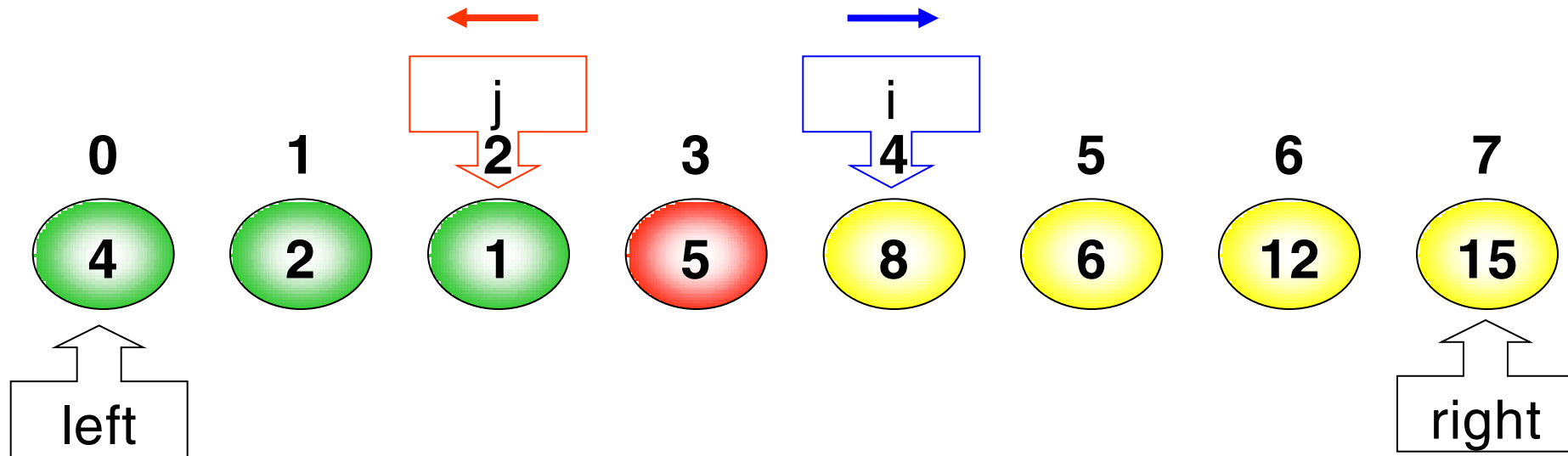
# Quick Sort – Ví Dụ

## ➤ Phân hoạch đoạn $[0,7]$

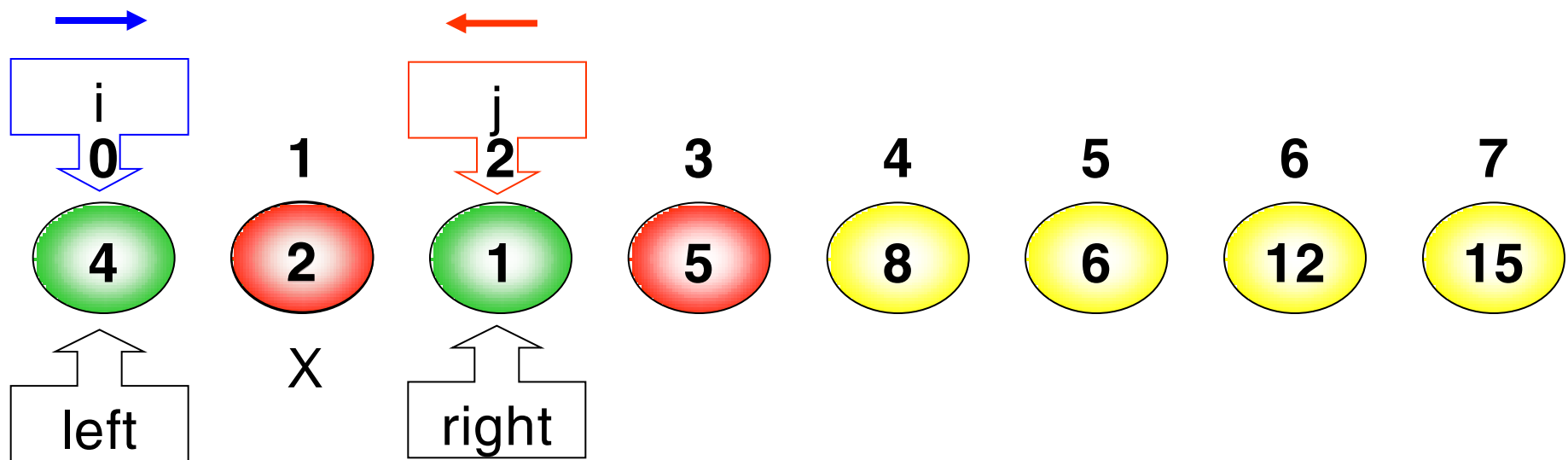


# Quick Sort – Ví Dụ

## ➤ Phân hoạch đoạn $[0,2]$

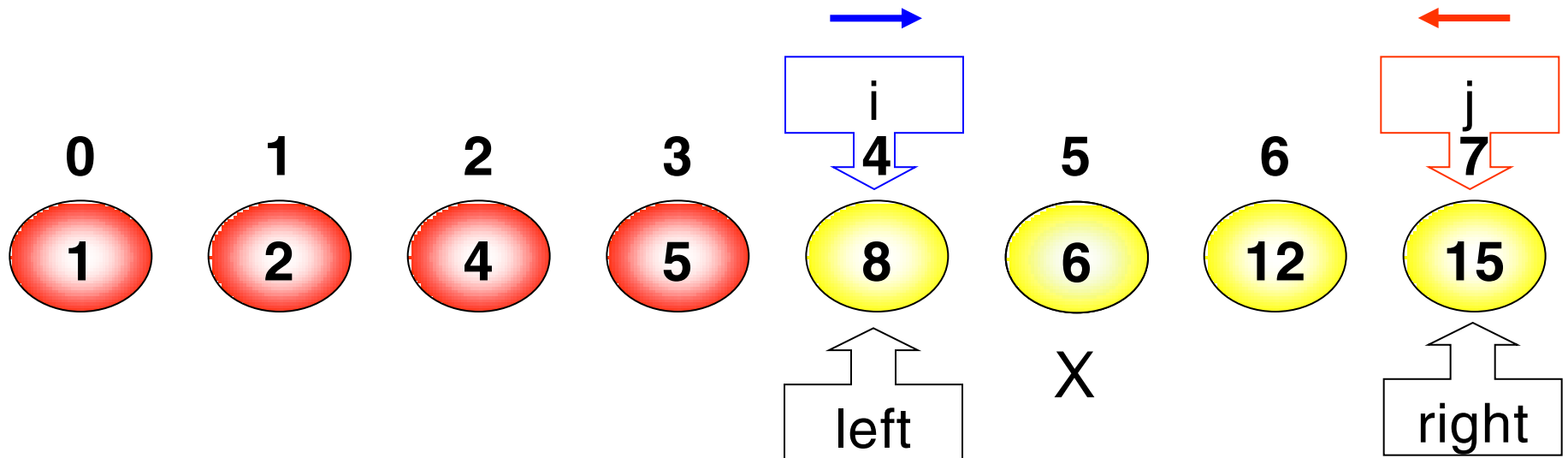


## ➤ Phân hoạch đoạn $[0,2]$



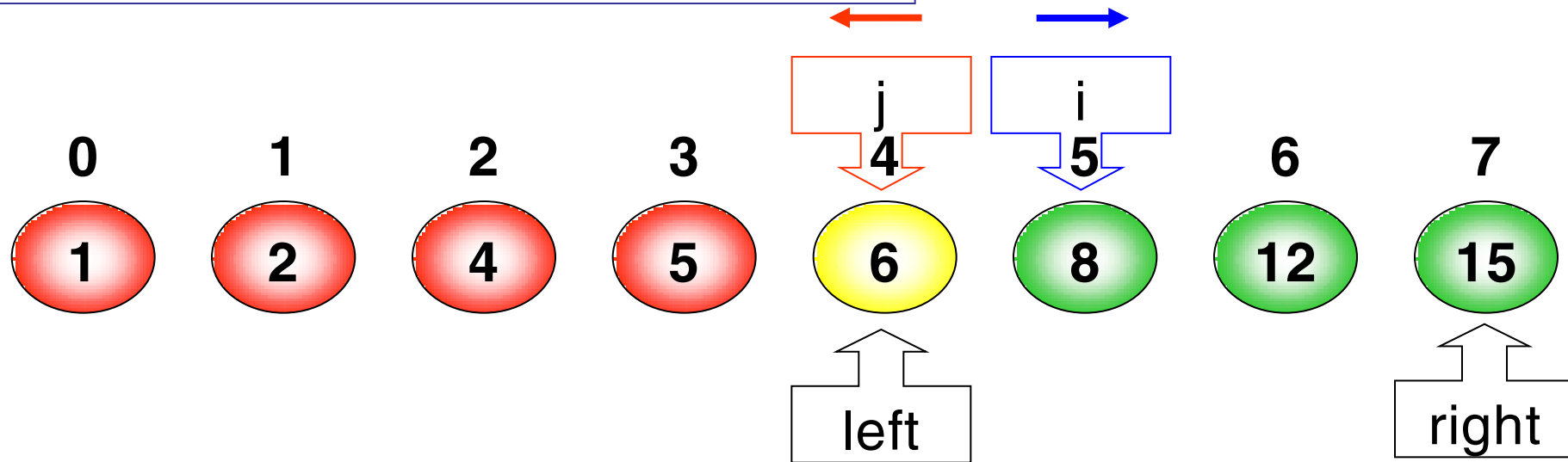
# Quick Sort – Ví Dụ

## ➤ Phân hoạch đoạn [4,7]



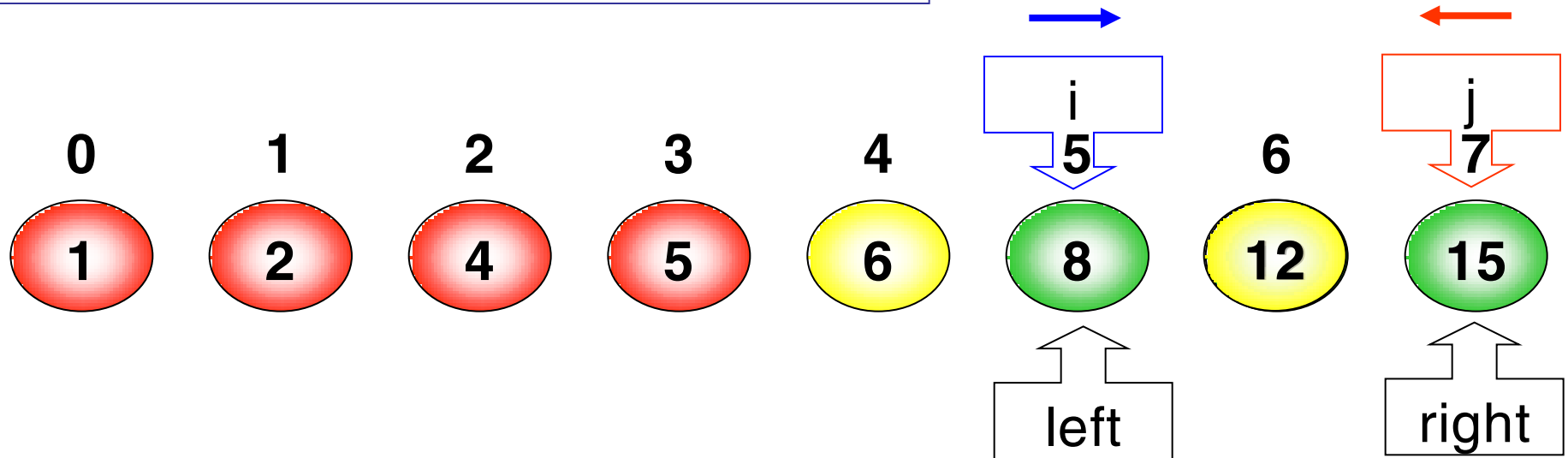
# Quick Sort – Ví Dụ

➤ Phân hoạch đoạn [5,7]



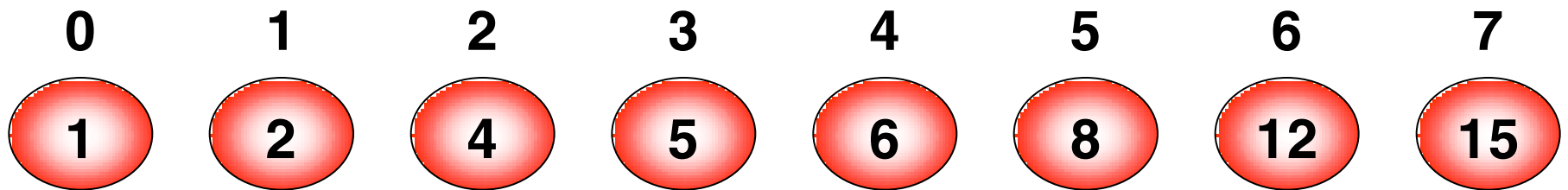
# Quick Sort – Ví Dụ

## ➤ Phân hoạch đoạn [5,7]





# Quick Sort – Ví Dụ



# Độ Phức Tạp Của Quick Sort

Trường hợp	Độ phức tạp
Tốt nhất	$n \cdot \log(n)$
Trung bình	$n \cdot \log(n)$
Xấu nhất	$n^2$



# Các Thuật Toán Sắp Xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection 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. Radix Sort



# Merge Sort – Ý Tưởng

- Giải thuật Merge sort sắp xếp dãy  $a_1, a_2, \dots, a_n$  dựa trên nhận xét sau:
  - ↪ Mỗi dãy  $a_1, a_2, \dots, a_n$  bất kỳ là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự.
    - Ví dụ: dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy con không giảm (12); (2, 8); (5); (1, 6); (4, 15).
  - ↪ Dãy đã có thứ tự coi như có 1 dãy con.
- ➔ Hướng tiếp cận: tìm cách làm giảm số dãy con không giảm của dãy ban đầu.



# Merge Sort – thuật toán

Bước 1 : // Chuẩn bị

$k = 1$ ; //  $k$  là chiều dài của dãy con trong bước hiện hành

Bước 2 :

Tách dãy  $a_0, a_1, \dots, a_{n-1}$  thành 2 dãy  $b, c$  theo nguyên tắc luân phiên từng nhóm  $k$  phần tử:

$b = a_0, \dots, a_k, a_{2k}, \dots, a_{3k}, \dots$

$c = a_{k+1}, \dots, a_{2k+1}, a_{3k+1}, \dots$

Bước 3 :

Trộn từng cặp dãy con gồm  $k$  phần tử của 2 dãy  $b, c$  vào  $a$ .

Bước 4 :

$k = k * 2$ ;

Nếu  $k < n$  thì trở lại bước 2.

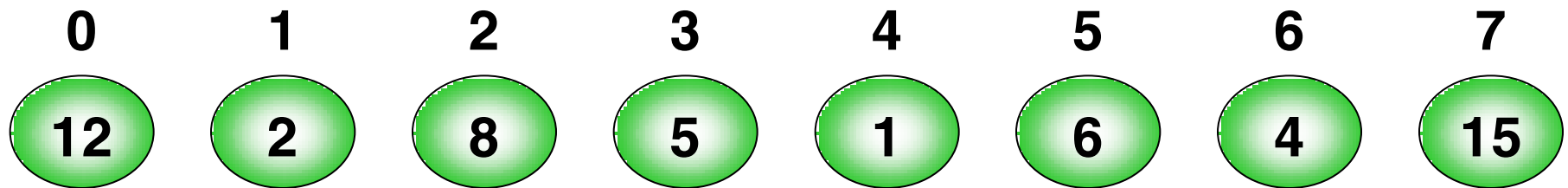
Ngược lại: Dừng



# Merge Sort – Ví Dụ

➤  $k=1$

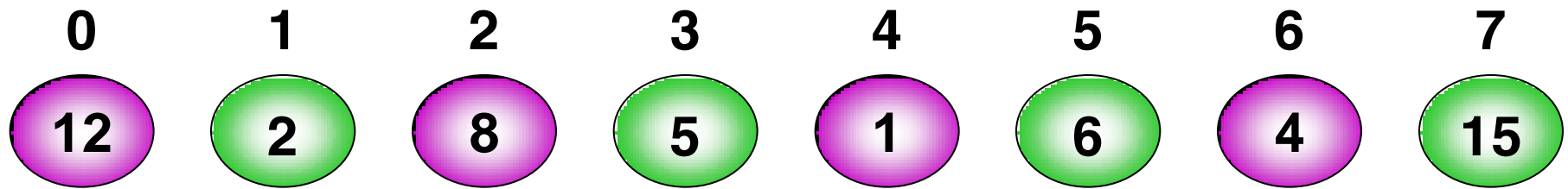
➤ Phân phối luân phiên



# Merge Sort – Ví Dụ

➤  $k = 1$

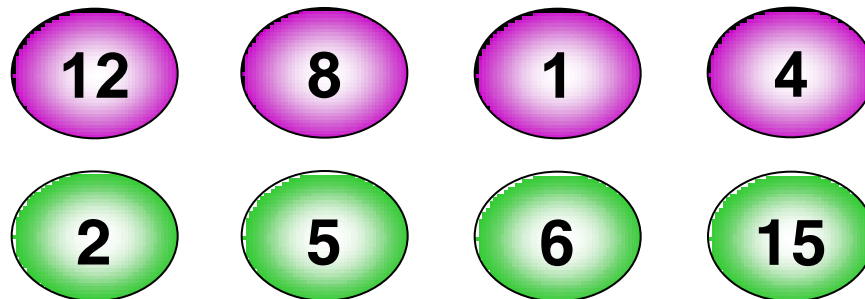
➤ Phân phối luân phiên



# Merge Sort – Ví Dụ

➤ Trộn từng cặp đường chạy

0            1            2            3            4            5            6            7





# Merge Sort – Ví Dụ

➤  $k = 1$

➤ Trộn từng cặp đường chạy

0

1

2

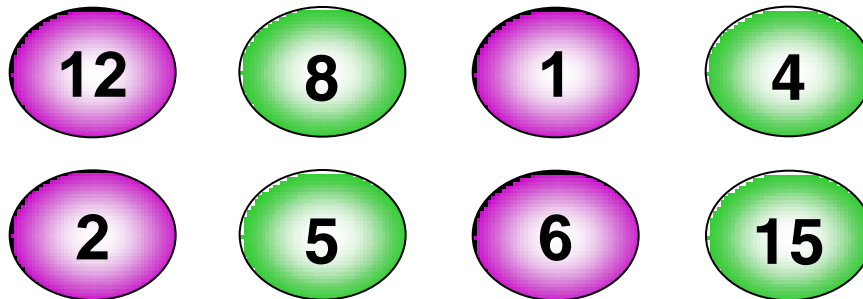
3

4

5

6

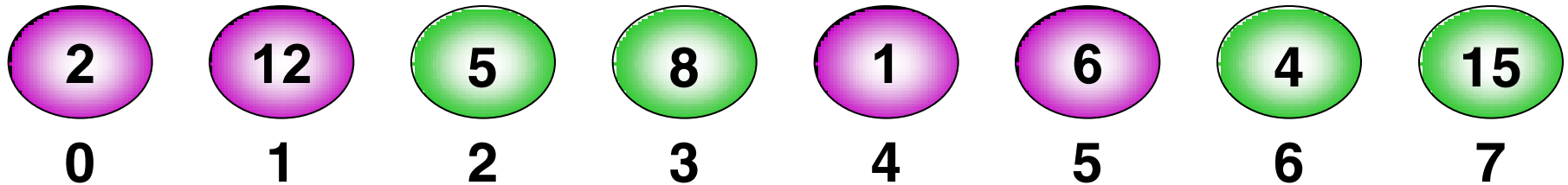
7



# Merge Sort – Ví Dụ

➤  $k = 2$

➤ Phân phối luân phiên



# Merge Sort – Ví Dụ

➤  $k = 2$

➤ Trộn từng cặp đường chạy

0

1

2

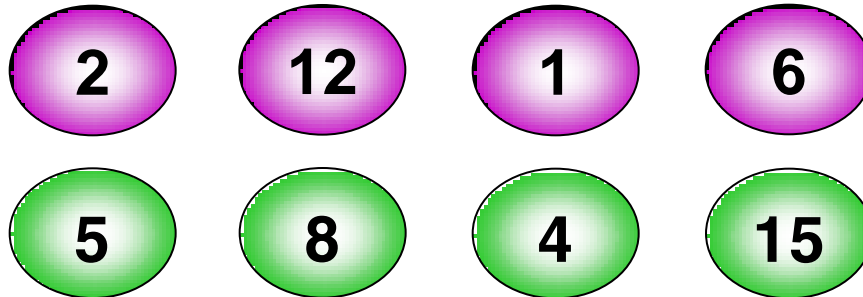
3

4

5

6

7



# Merge Sort – Ví Dụ

➤  $k = 2$

➤ Trộn từng cặp đường chạy

0

1

2

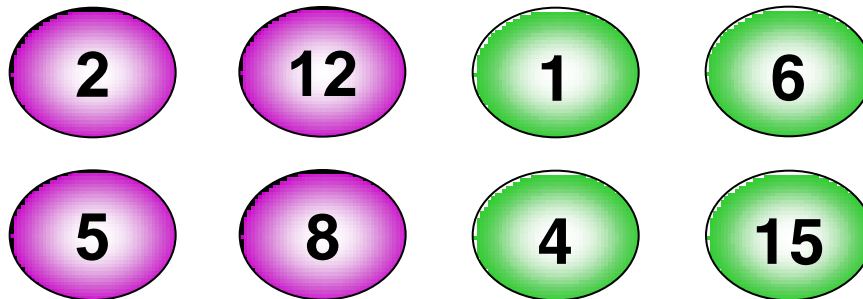
3

4

5

6

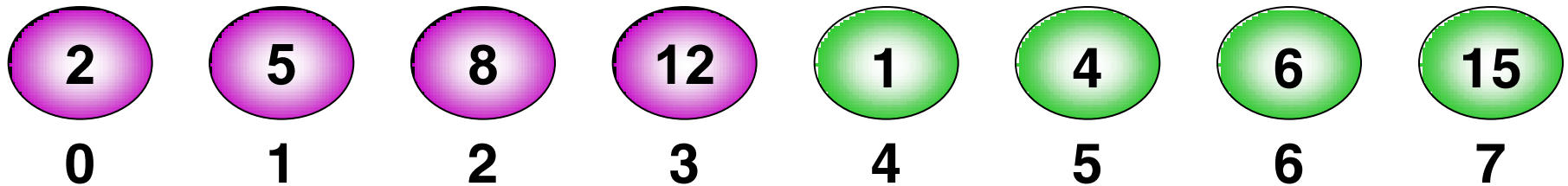
7



# Merge Sort – Ví Dụ

➤  $k = 4$

➤ Phân phối luân phiên



# Merge Sort – Ví Dụ

➤  $k = 4$

➤ Trộn từng cặp đường chạy

0

1

2

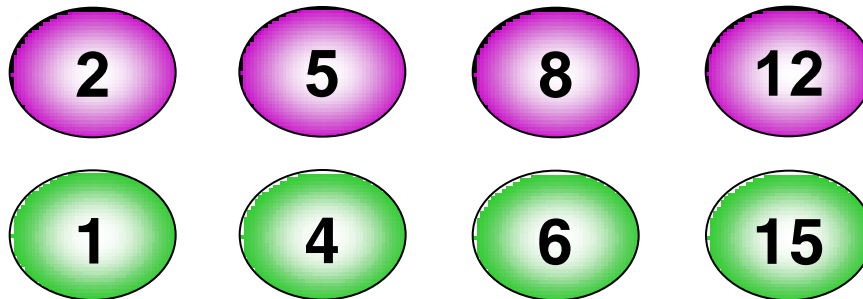
3

4

5

6

7



# Merge Sort – Ví Dụ

➤  $k = 4$

➤ Trộn từng cặp đường chạy

0

1

2

3

4

5

6

7

2

5

8

12

1

4

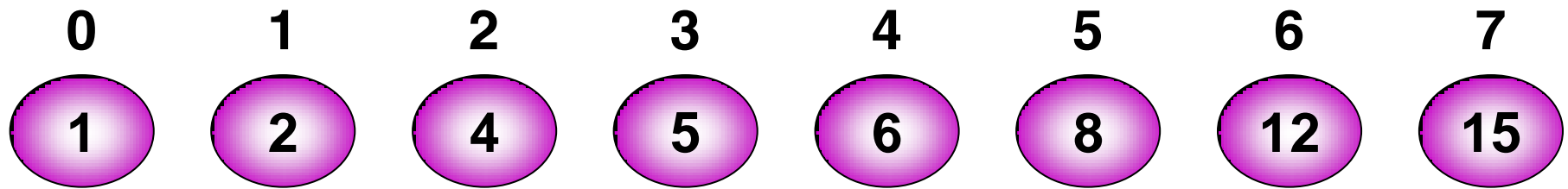
6

15



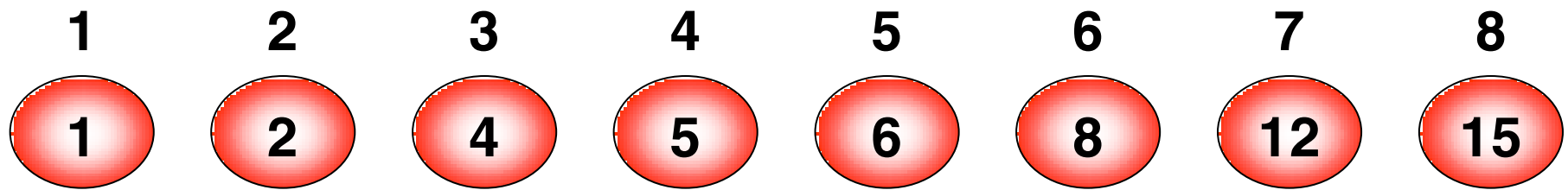
# Merge Sort – Ví Dụ

➤  $k = 8$





# Merge Sort – Ví Dụ



# Merge Sort – Cài Đặt

- Dữ liệu hỗ trợ: 2 mảng b, c:

**int** b[MAX], c[MAX], nb, nc;

- Các hàm cần cài đặt:

➡ **void MergeSort(int a[], int N);** : Sắp xếp mảng (a, N) tăng dần

➡ **void Distribute(int a[], int N, int &nb, int &nc, int k);**  
Phân phối đều luân phiên các dãy con độ dài k từ mảng a vào hai mảng con b và c

➡ **void Merge(int a[], int nb, int nc, int k);** : Trộn mảng b và mảng c vào mảng a

➡ **void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k);** : Trộn một cặp dãy con từ b và c vào a



# Merge Sort – Cài Đặt

```
int b[MAX], c[MAX], nb, nc;

void MergeSort(int a[], int N)
{
    int k;
    for (k = 1; k < N; k *= 2)
    {
        Distribute(a, N, nb, nc, k);
        Merge(a, nb, nc, k);
    }
}
```



# Merge Sort – Cài Đặt

```
void Distribute(int a[], int N, int &nb, int &nc, int k)
{
    int i, pa, pb, pc;
    pa = pb = pc = 0;
    while (pa < N)
    {
        for (i=0; (pa<N) && (i<k); i++, pa++, pb++)
            b[pb] = a[pa];
        for (i=0; (pa<N) && (i<k); i++, pa++, pc++)
            c[pc] = a[pa];
    }
    nb = pb;    nc = pc;
}
```



# Merge Sort – Cài Đặt

```
void Merge(int a[],int nb, int nc,int k)
{   int p, pb, pc, ib, ic, kb, kc;
    p=pb=pc=0; ib=ic=0;
    while((nb>0)&&(nc>0))
    {   kb=min(k,nb); kc=min(k,nc);
        if(b[pb+ib]<=c[pc+ic])
        {   a[p++]=b[pb+ib]; ib++;
            if(ib==kb)
            {   for(;ic<kc;ic++    a[p++]=c[pc+ic];
                pb+=kb; pc+=kc; ib = ic=0;
                nb-=kb; nc-=kc;
            }
        }
    }
}
```



# Merge Sort – Cài Đặt

else

```
{    a[p++]=c[pc+ic]; ic++;
```

```
    if(ic==kc)
```

```
    {
```

```
        for(;ib<kb;ib++)    a[p++]=b[pb+ib];
```

```
        pb+=kb; pc+=kc; ib = ic=0;
```

```
        nb-=kb; nc-=kc;
```

```
    }
```

```
}
```

```
}
```

```
}
```



# Merge Sort – Cài Đặt

```
int min(int a,int b)
{
    if(a>b) return b;
    else    return a;
}
```



# Độ phức tạp của Merge Sort

- Số lần lặp của Bước 2, 3 là  $\log_2 n$  do sau mỗi lần lặp giá trị  $k$  tăng gấp đôi. Chi phí thực hiện ở bước 2 và 3 tỉ lệ thuận với  $n$ . Do đó chi phí của dãy thuật MergeSort là  $O(n \log_2 n)$





# Các Thuật Toán Sắp Xếp

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



# Sắp Xếp Theo Phương Pháp Cơ Sở Radix Sort

- Radix Sort là một thuật toán tiếp cận theo một hướng hoàn toàn khác.
- Nếu như trong các thuật toán khác, cơ sở để sắp xếp luôn là việc so sánh giá trị của 2 phần tử thì Radix Sort lại dựa trên nguyên tắc phân loại thư của bưu điện. Vì lý do đó Radix Sort còn có tên là Postman's Sort.
- Radix Sort không hề quan tâm đến việc so sánh giá trị của phần tử mà bản thân việc phân loại và trình tự phân loại sẽ tạo ra thứ tự cho các phần tử.



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Mô phỏng lại qui trình trên, để sắp xếp dãy  $a_1, a_2, \dots, a_n$ , giải thuật Radix Sort thực hiện như sau:
  - ↪ Trước tiên, ta có thể giả sử mỗi phần tử  $a_i$  trong dãy  $a_1, a_2, \dots, a_n$  là một số nguyên có tối đa  $m$  chữ số.
  - ↪ Ta phân loại các phần tử lần lượt theo các chữ số hàng đơn vị, hàng chục, hàng trăm, ... tương tự việc phân loại thư theo tỉnh thành, quận huyện, phường xã, ....



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Bước 1 :// k cho biết chữ số dùng để phân loại hiện hành
  - ↪  $k = 0$ ; //  $k = 0$ : hàng đơn vị;  $k = 1$ : hàng chục; ...
- Bước 2 : //Tạo các lô chứa các loại phần tử khác nhau
  - ↪ Khởi tạo 10 lô  $B_0, B_1, \dots, B_9$  rỗng;



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Bước 3 :
  - ↳ For  $i = 1 \dots n$  do
    - Đặt  $a_i$  vào lô  $B_t$  với  $t$ : chữ số thứ  $k$  của  $a_i$ ;
- Bước 4 :
  - ↳ Nối  $B_0, B_1, \dots, B_9$  lại (theo đúng trình tự) thành  $a$ .
- Bước 5 :
  - ↳  $k = k+1$ ; Nếu  $k < m$  thì trở lại bước 2. Ngược lại: Dừng



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	070 <u>1</u>										
11	172 <u>5</u>										
10	099 <u>9</u>										
9	917 <u>0</u>										
8	325 <u>2</u>										
7	451 <u>8</u>										
6	700 <u>9</u>										
5	142 <u>4</u>										
4	042 <u>8</u>										
3	123 <u>9</u>										099 <u>9</u>
2	842 <u>5</u>						172 <u>5</u>			451 <u>8</u>	700 <u>9</u>
1	701 <u>3</u>	917 <u>0</u>	070 <u>1</u>	325 <u>2</u>	701 <u>3</u>	142 <u>4</u>	842 <u>5</u>			042 <u>8</u>	123 <u>9</u>
CS	A	0	1	2	3	4	5	6	7	8	9



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	09 <u>9</u> 9										
11	70 <u>0</u> 9										
10	12 <u>3</u> 9										
9	45 <u>1</u> 8										
8	04 <u>2</u> 8										
7	17 <u>2</u> 5										
6	84 <u>2</u> 5										
5	14 <u>2</u> 4										
4	70 <u>1</u> 3			04 <u>2</u> 8							
3	32 <u>5</u> 2			17 <u>2</u> 5							
2	07 <u>0</u> 1	70 <u>0</u> 9	45 <u>1</u> 8	84 <u>2</u> 5							
1	91 <u>7</u> 0	07 <u>0</u> 1	70 <u>1</u> 3	14 <u>2</u> 4	12 <u>3</u> 9		32 <u>5</u> 2		91 <u>7</u> 0		09 <u>9</u> 9
CS	A	0	1	2	3	4	5	6	7	8	9



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	0 <u>9</u> 99										
11	9 <u>1</u> 70										
10	3 <u>2</u> 52										
9	1 <u>2</u> 39										
8	0 <u>4</u> 28										
7	1 <u>7</u> 25										
6	8 <u>4</u> 25										
5	1 <u>4</u> 24										
4	4 <u>5</u> 18										
3	7 <u>0</u> 13					0 <u>4</u> 28					
2	7 <u>0</u> 09	7 <u>0</u> 13		3 <u>2</u> 52		8 <u>4</u> 25			1 <u>7</u> 25		
1	0 <u>7</u> 01	7 <u>0</u> 09	9 <u>1</u> 70	1 <u>2</u> 39		1 <u>4</u> 24	4 <u>5</u> 18		0 <u>7</u> 01		0 <u>9</u> 99
CS	A	0	1	2	3	4	5	6	7	8	9





# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	<u>0</u> 999										
11	<u>1</u> 725										
10	<u>0</u> 701										
9	<u>4</u> 518										
8	<u>0</u> 428										
7	<u>8</u> 425										
6	<u>1</u> 424										
5	<u>3</u> 252										
4	<u>1</u> 239										
3	<u>9</u> 170	<u>0</u> 999	<u>1</u> 725								
2	<u>7</u> 013	<u>0</u> 701	<u>1</u> 424						<u>7</u> 013		
1	<u>7</u> 009	<u>0</u> 428	<u>1</u> 239		<u>3</u> 252	<u>4</u> 518			<u>7</u> 009	<u>8</u> 425	<u>9</u> 170
CS	A	0	1	2	3	4	5	6	7	8	9



# Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	<u>9</u> 170										
11	<u>8</u> 425										
10	<u>7</u> 013										
9	<u>7</u> 009										
8	<u>4</u> 518										
7	<u>3</u> 252										
6	<u>1</u> 725										
5	<u>1</u> 424										
4	<u>1</u> 239										
3	<u>0</u> 999										
2	<u>0</u> 701										
1	<u>0</u> 428										
CS	A	0	1	2	3	4	5	6	7	8	9



- Nhập một dãy số nguyên  $n$  phần tử.
- Sắp xếp lại dãy sao cho:
  - số nguyên dương đầu ở đầu dãy và theo thứ tự giảm.
  - số nguyên âm tăng ở cuối dãy và theo thứ tự tăng.
  - số 0 ở giữa.
- *Lưu ý: Không dùng đổi chỗ trực tiếp.*

