CHƯƠNG 2

TÌM KIẾM VÀ SẮP XẾP NỘI

Nội Dung

- 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

Nội Dung (Tt)

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…, 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;

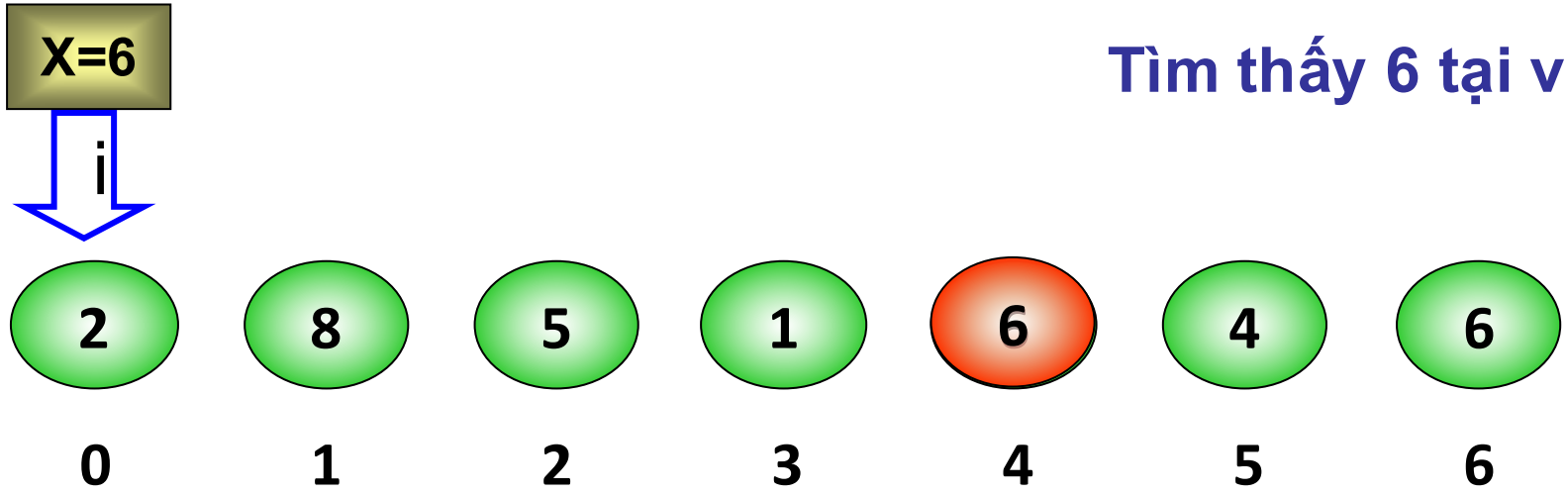
    Ngược lại: Lặp lại bước 2;

Thuật Toán Tìm Kiếm Tuyến Tính

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

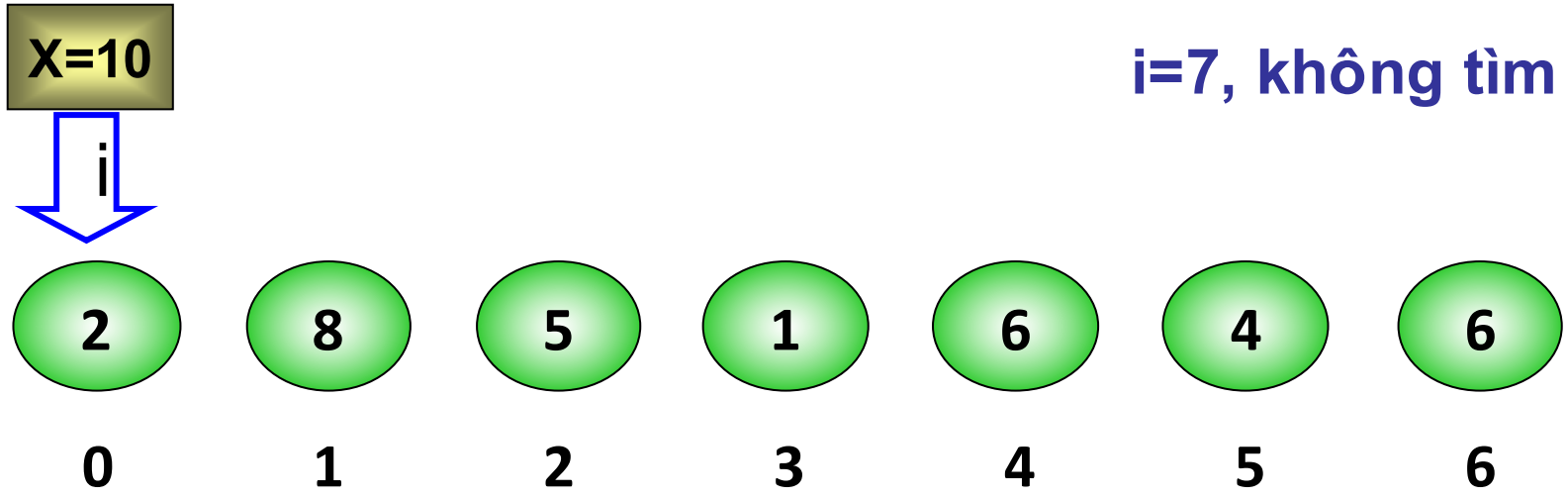
```
int LinearSearch(int a[],int n, int x)
{
        int i=0;
        while((i<n)&&(a[i]!=x))
                i++;
        if(i==n)
                return 0; //Tìm không thấy x
        else
                return 1; //Tìm thấy
}
```

Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính



Tìm thấy 6 tại vị trí 4

Minh Họa Thuật Toán Tìm Kiếm Tuyến Tính (tt)



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*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 xử 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 nữa dưới hay nữa 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_{left}, a_{right}, các bước của giải thuật như sau:
- Bước 1: left=0; right=N-1;
- Bước 2:
  - mid=(left+right)/2; *//chỉ số phần tử giữa dãy hiện hành*
  - So sánh a[mid] với x. Có 3 khả năng
    - a[mid]= x: tìm thấy. Dừng
    - a[mid]>x : Right= mid-1;
    - a[mid]<x : Left= mid+1;
- Bước 3: Nếu Left <=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;
            while(left<=right)
            {
                    mid=(left+right)/2;
                    if(a[mid]==x) return 1;
                    else    if(a[mid]<x)  left=mid+1;
                            else  right=mid-1;
            }
     return 0;
}
```

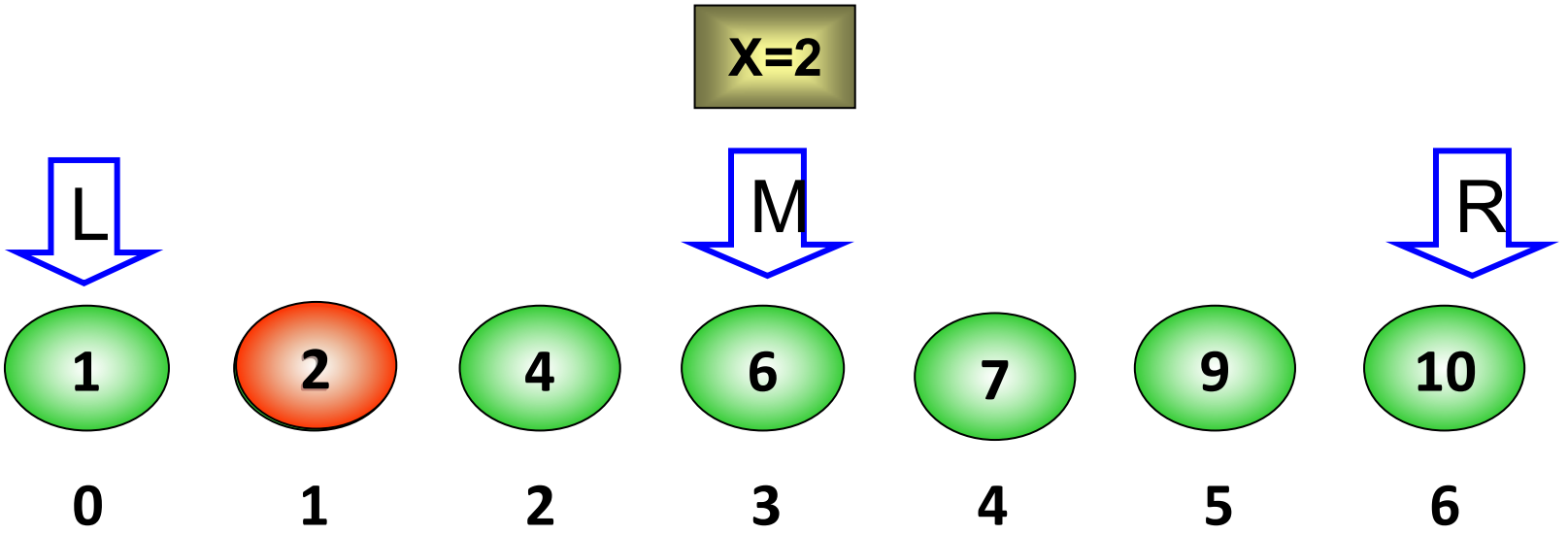

Đánh Giá Thuật Toán Tìm Tuyến Tính

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

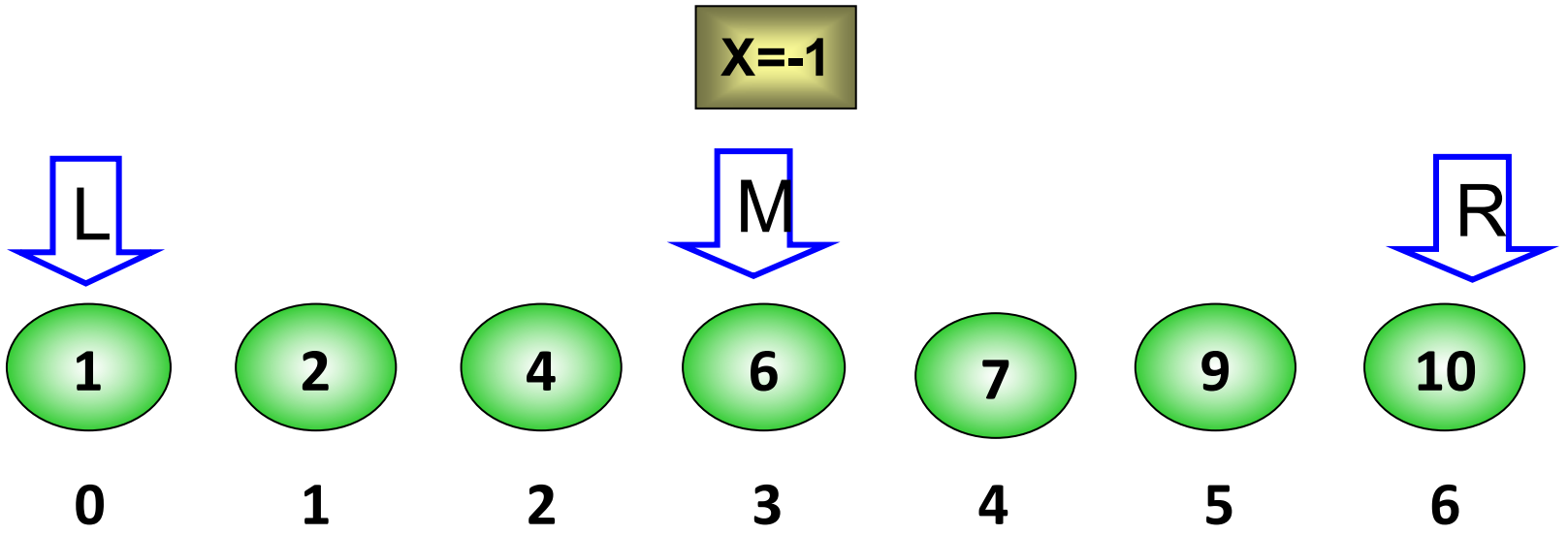
- Độ phức tạp $O(\log_2 N)$

Minh Họa Thuật Toán Tìm Nhị Phân

Tìm thấy 2 tại vị trí 1



Minh Họa Thuật Toán Tìm Nhị Phân (tt)



L=0

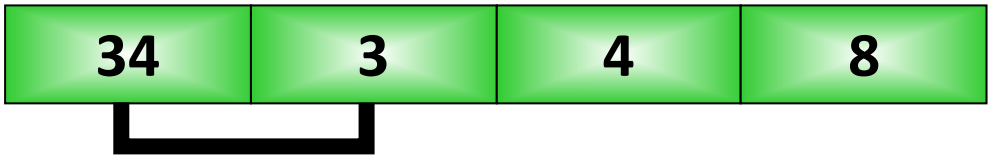
R=-1 => không tìm thấy X=-1

Bài Toán Sắp Xếp

- Cho danh sách có n phần tử a_0, a_1, a_2…, 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

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 Trực Tiếp – 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.

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;
```

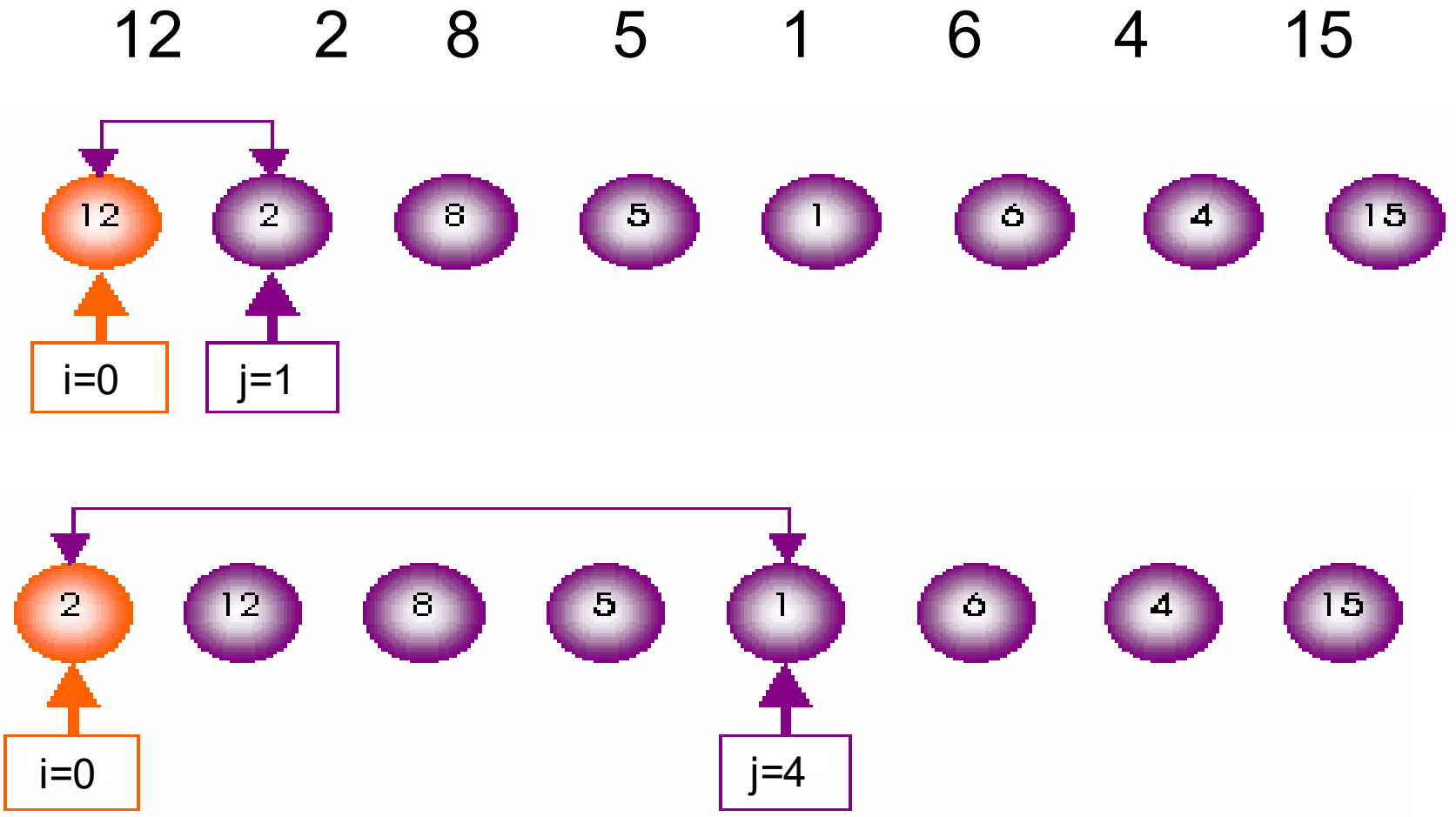
- Bước 4: i = i+1;

  Nếu i < N-1: Lặp lại Bước 2.

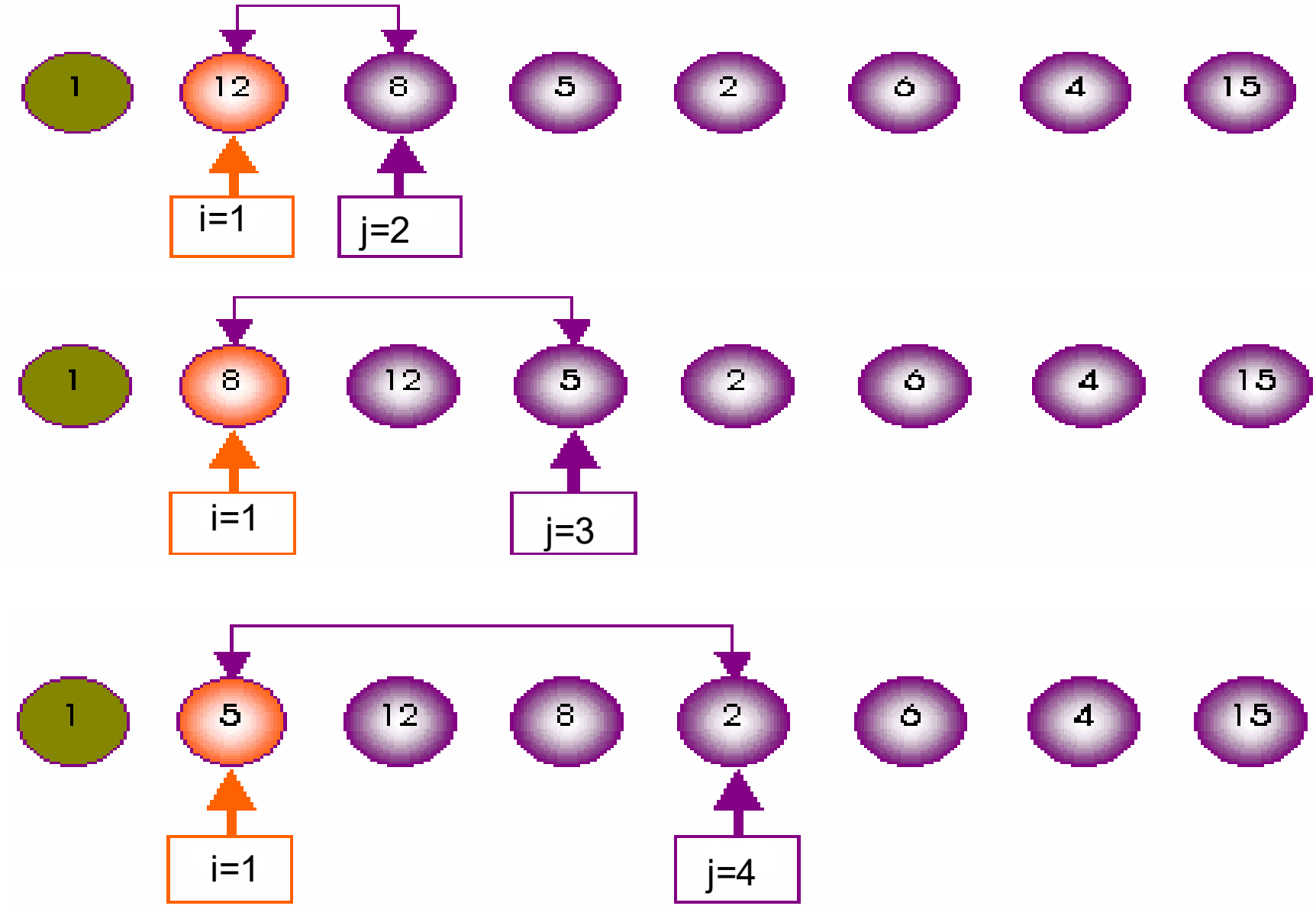
  Ngược lại: Dừng.

Đổi Chỗ Trực Tiếp – Interchange Sort

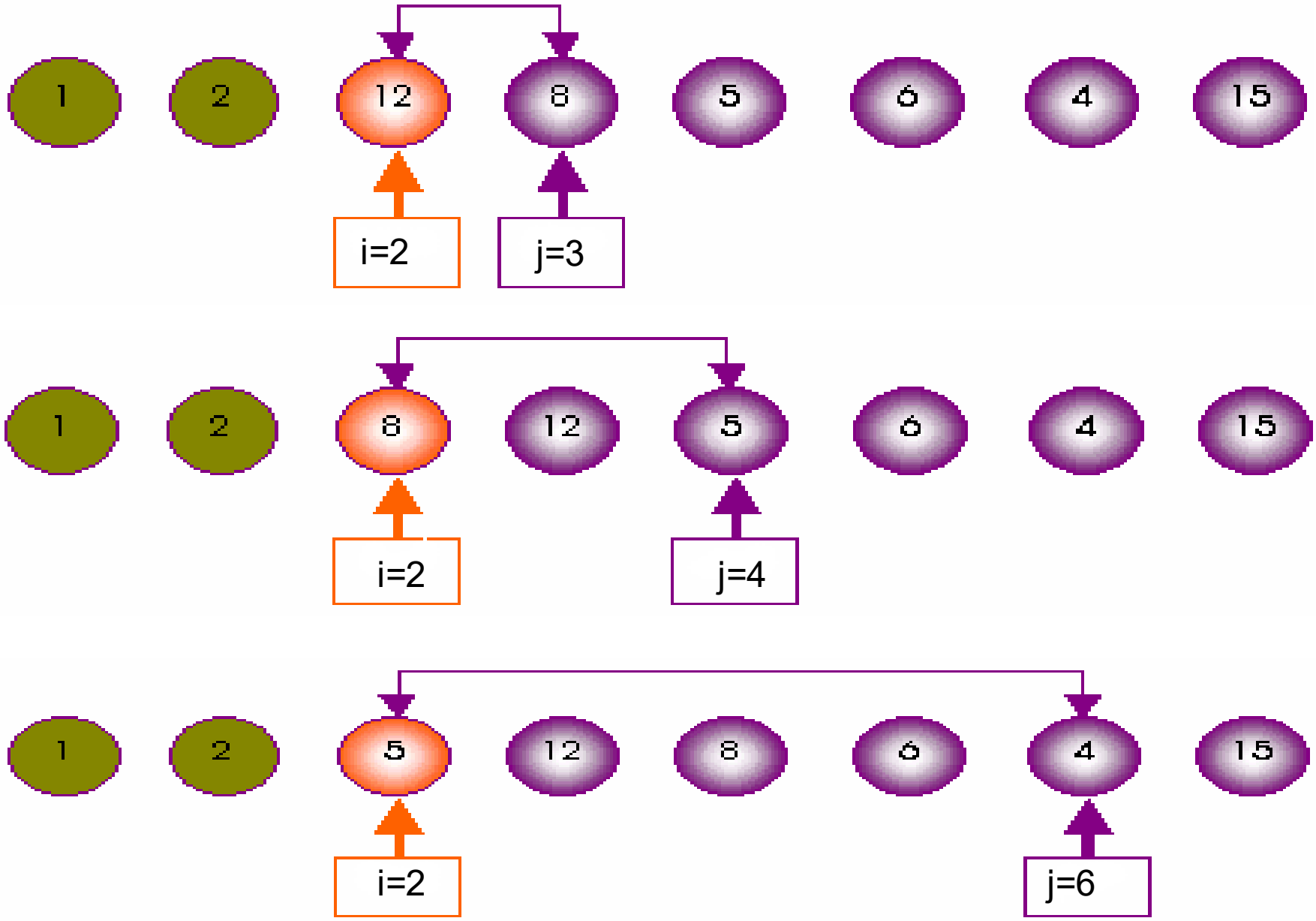
- Cho dãy số a:



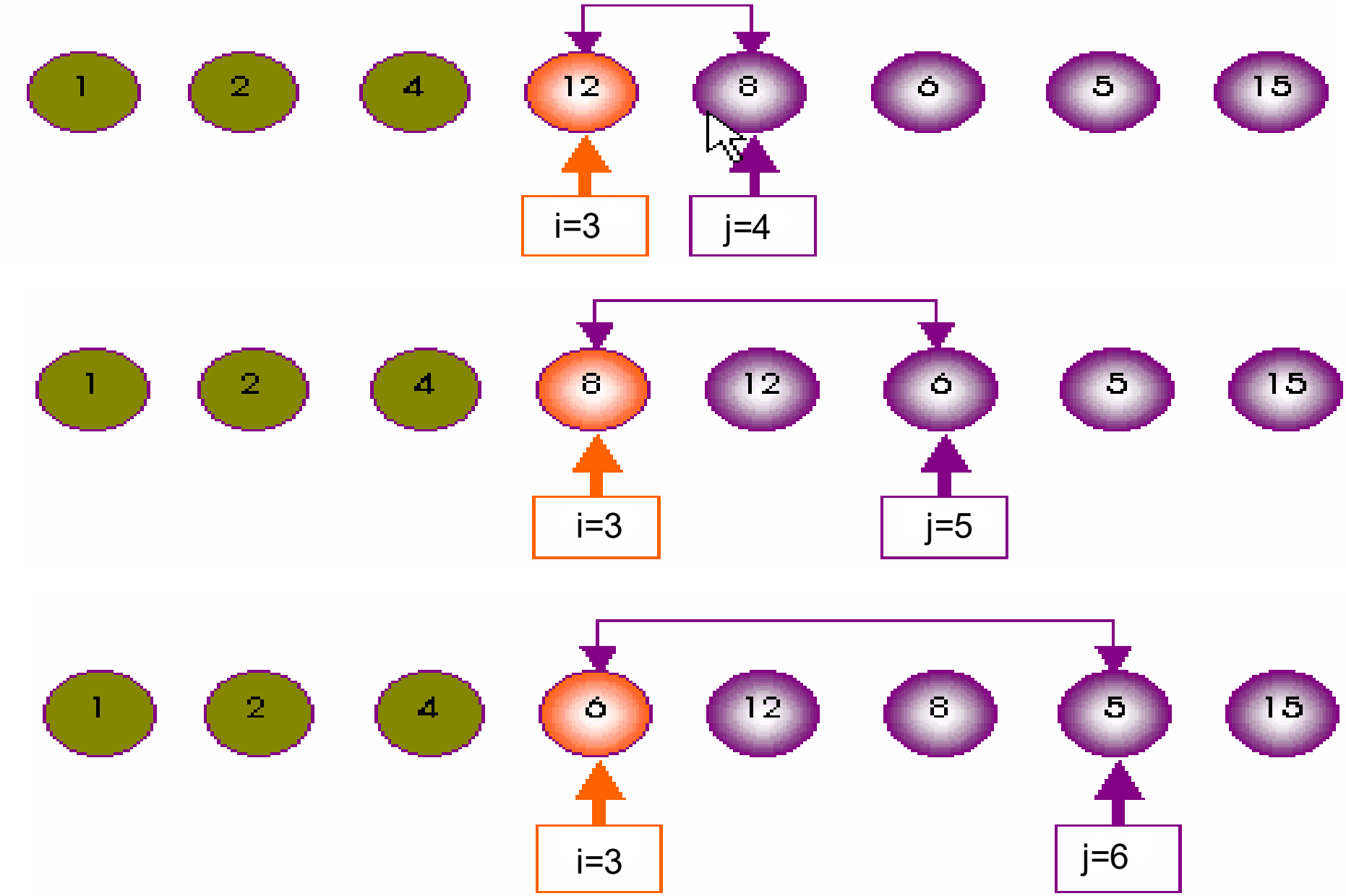
Đổi Chỗ Trực Tiếp – Interchange Sort



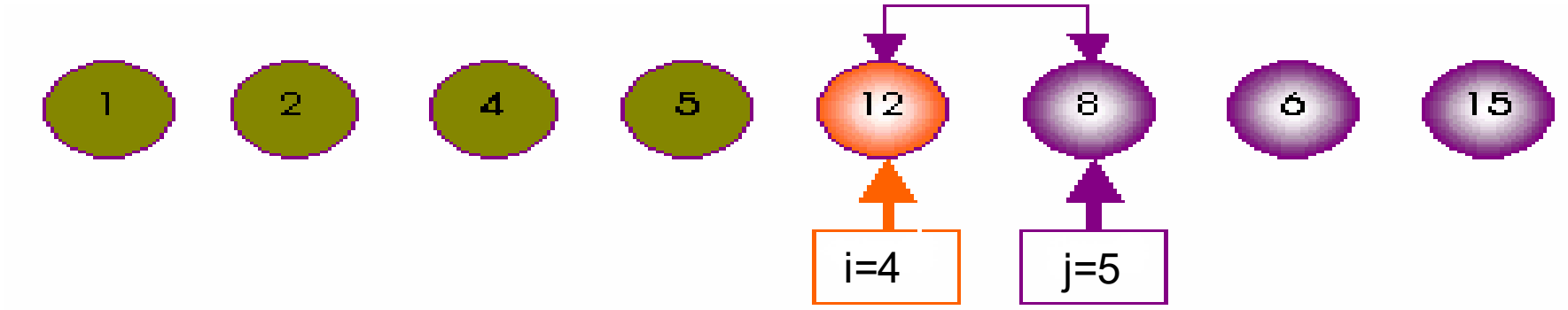
Đổi Chỗ Trực Tiếp – Interchange Sort

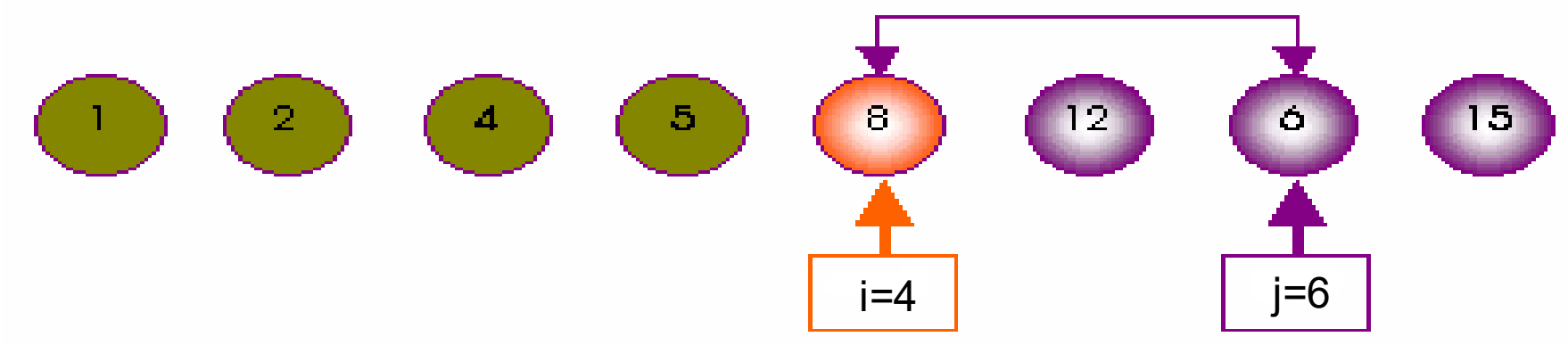


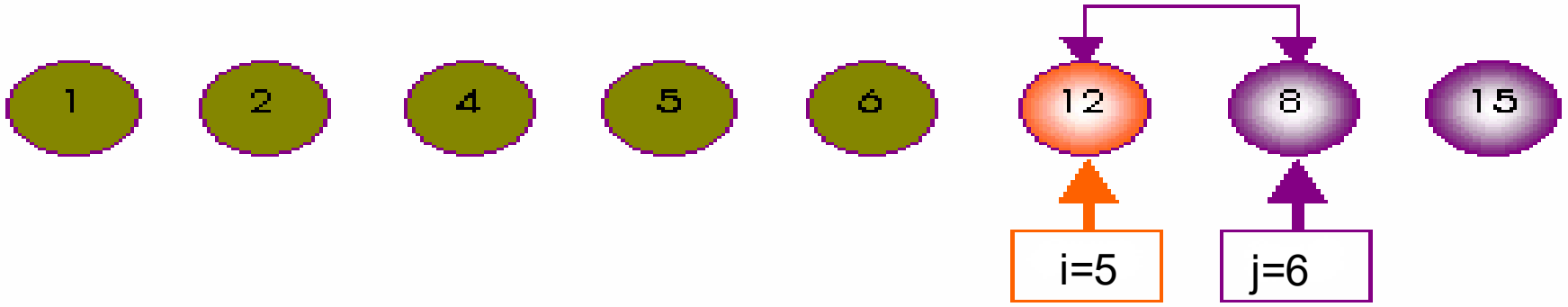
Đổi Chỗ Trực Tiếp – Interchange Sort



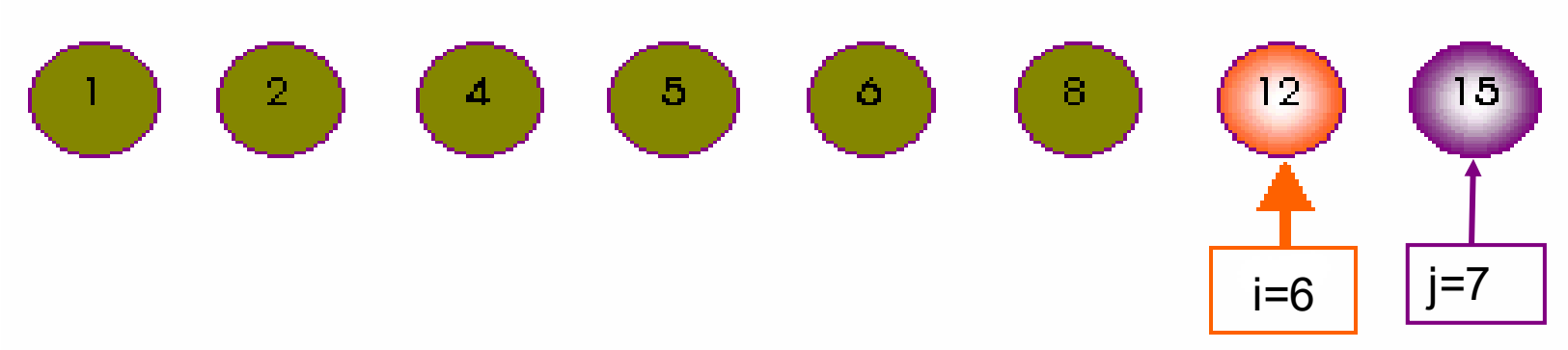
Đổi Chỗ Trực Tiếp – Interchange Sort

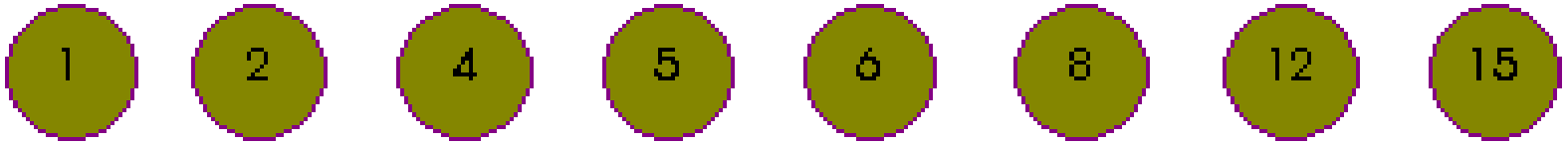






Đổi Chỗ Trực Tiếp – Interchange Sort

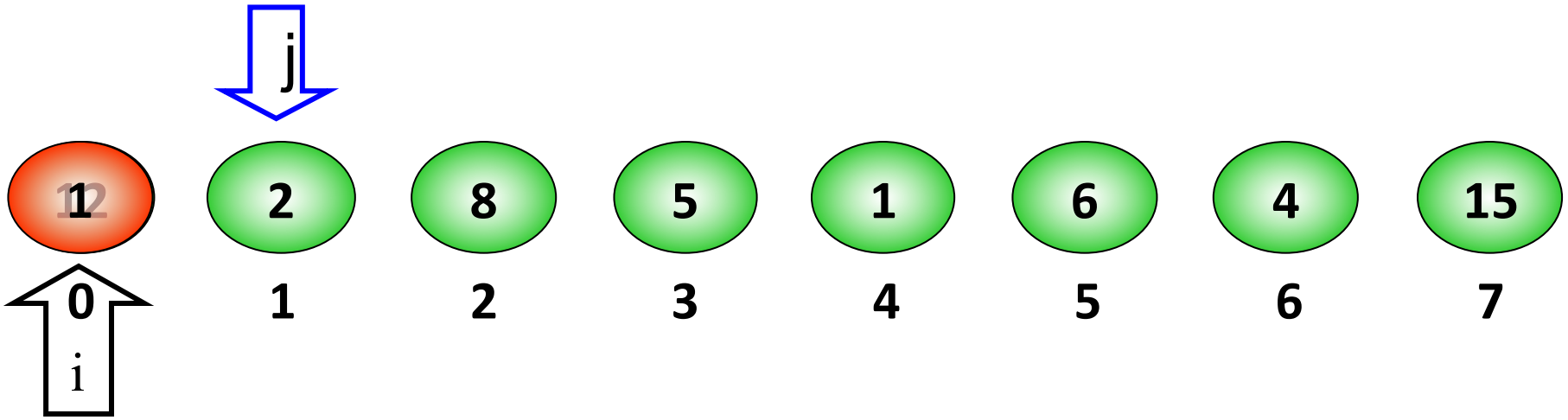




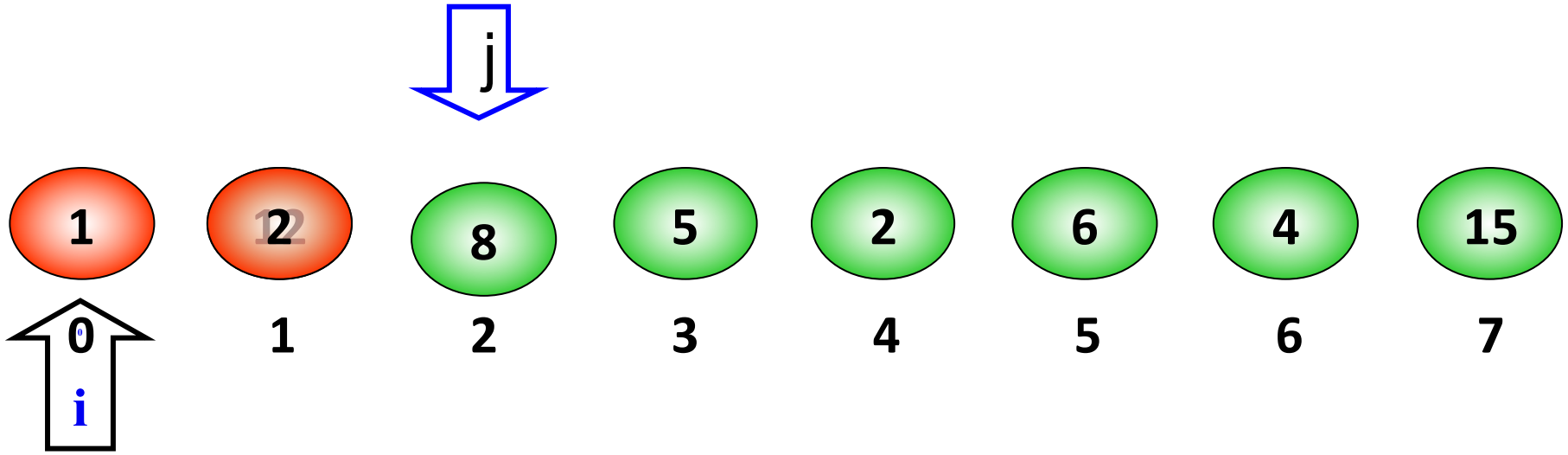
Cài Đặt Đổi Chỗ Trực Tiếp

```
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]);
}
```

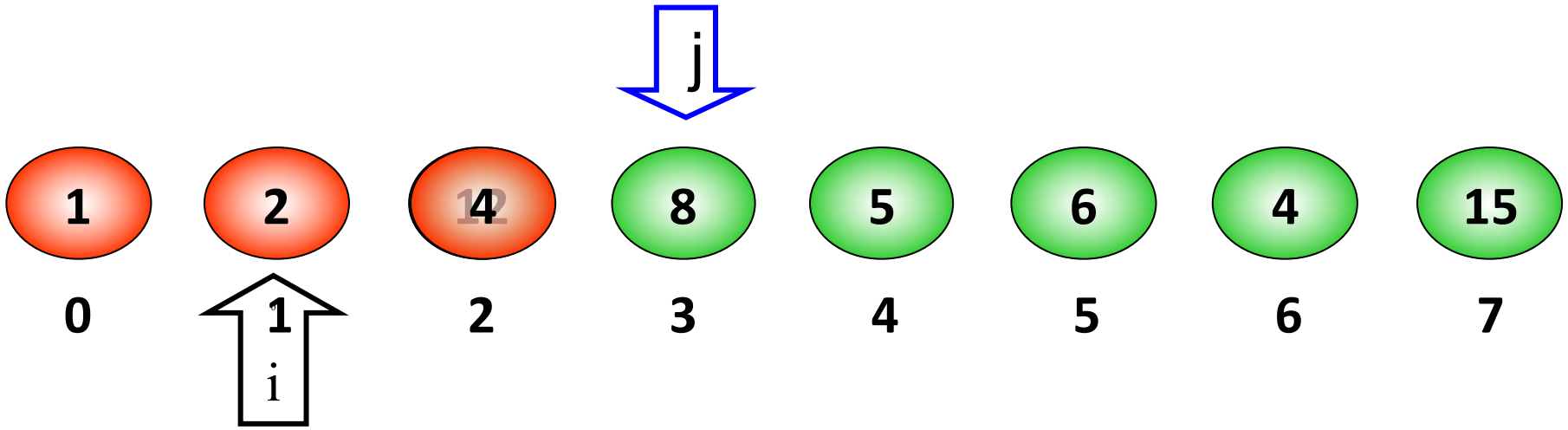

Minh Họa Thuật Toán



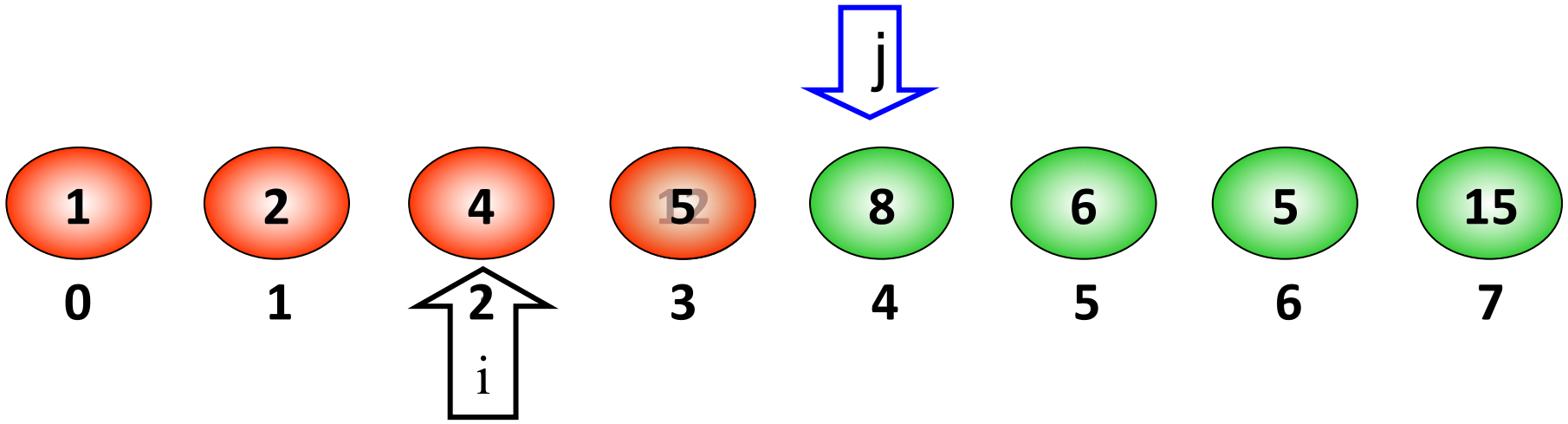
Minh Họa Thuật Toán



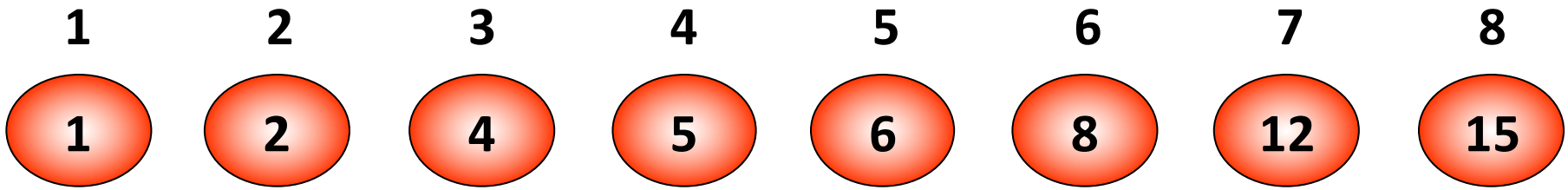
Minh Họa Thuật Toán



Minh Họa Thuật Toán



Minh Họa Thuật Toán



Độ Phức Tạp Của Thuật Toán

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[min] và a[i]
- Bước 4 : Nếu i < N-1 thì

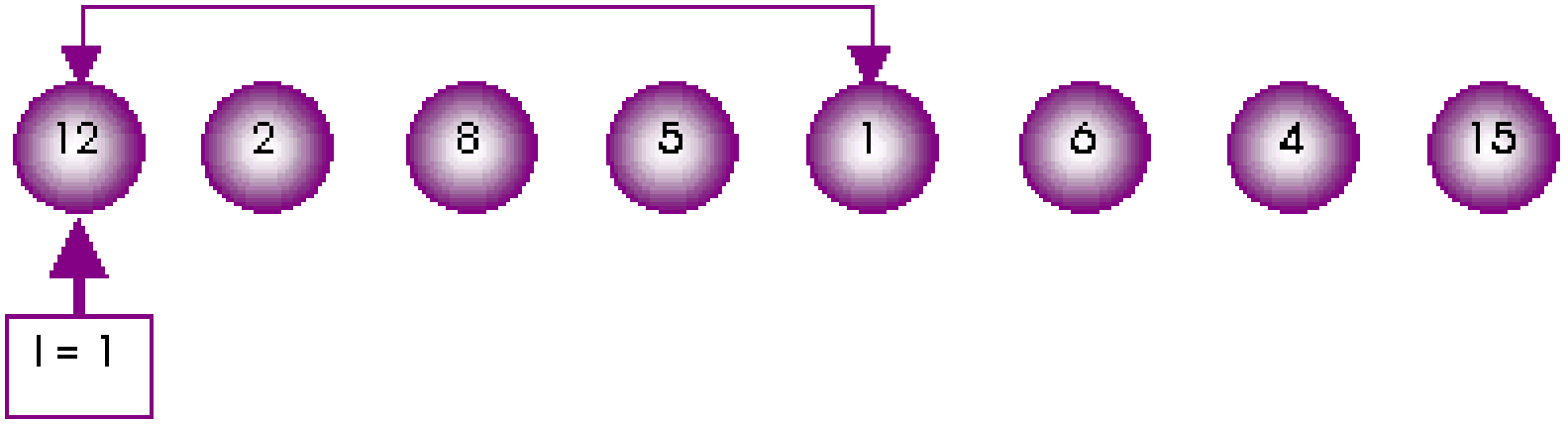
  i = i+1; Lặp lại Bước 2;

  Ngược lại: Dừng.

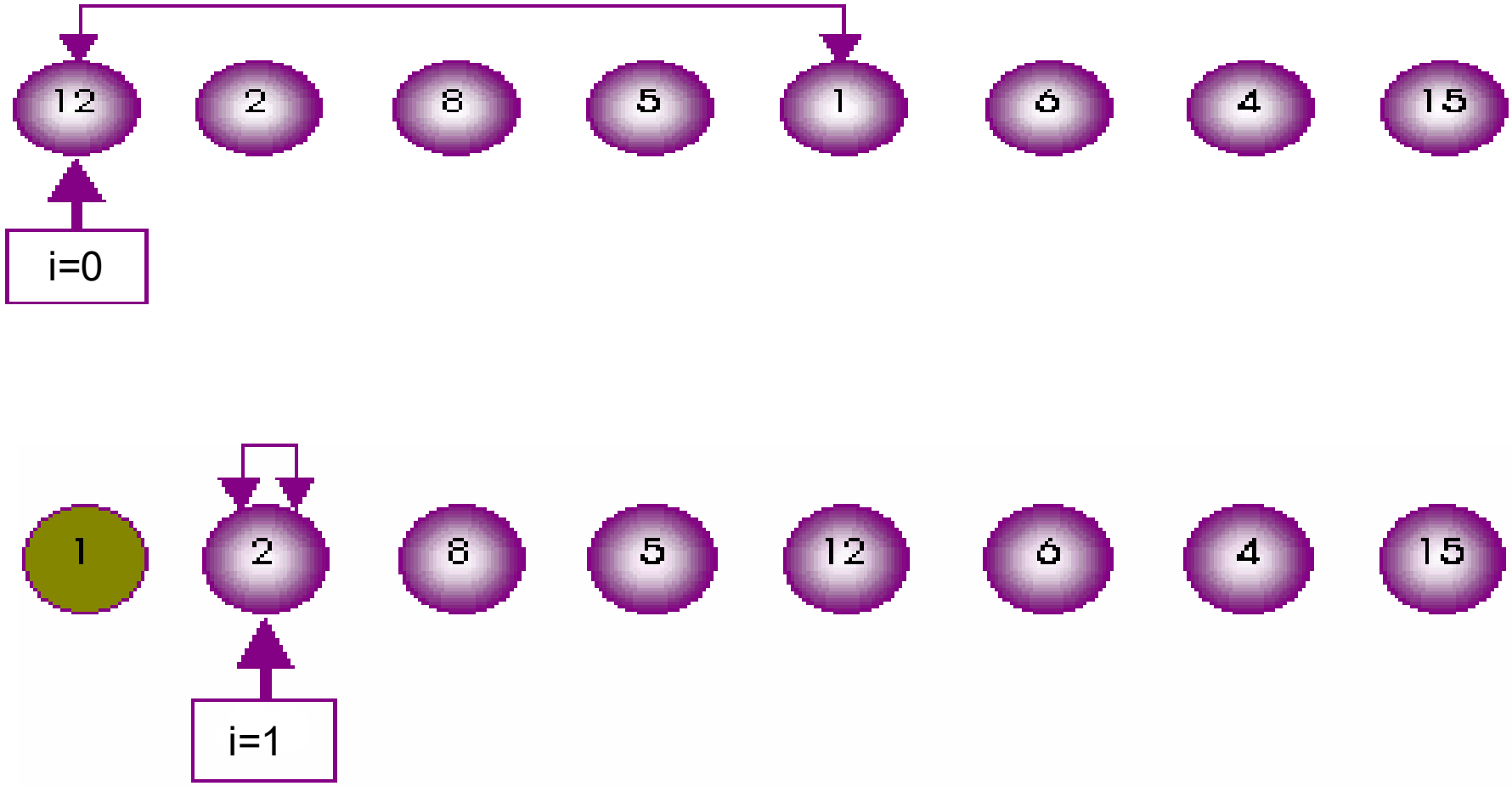
Chọn Trực Tiếp – Selection Sort

- Cho dãy số a:

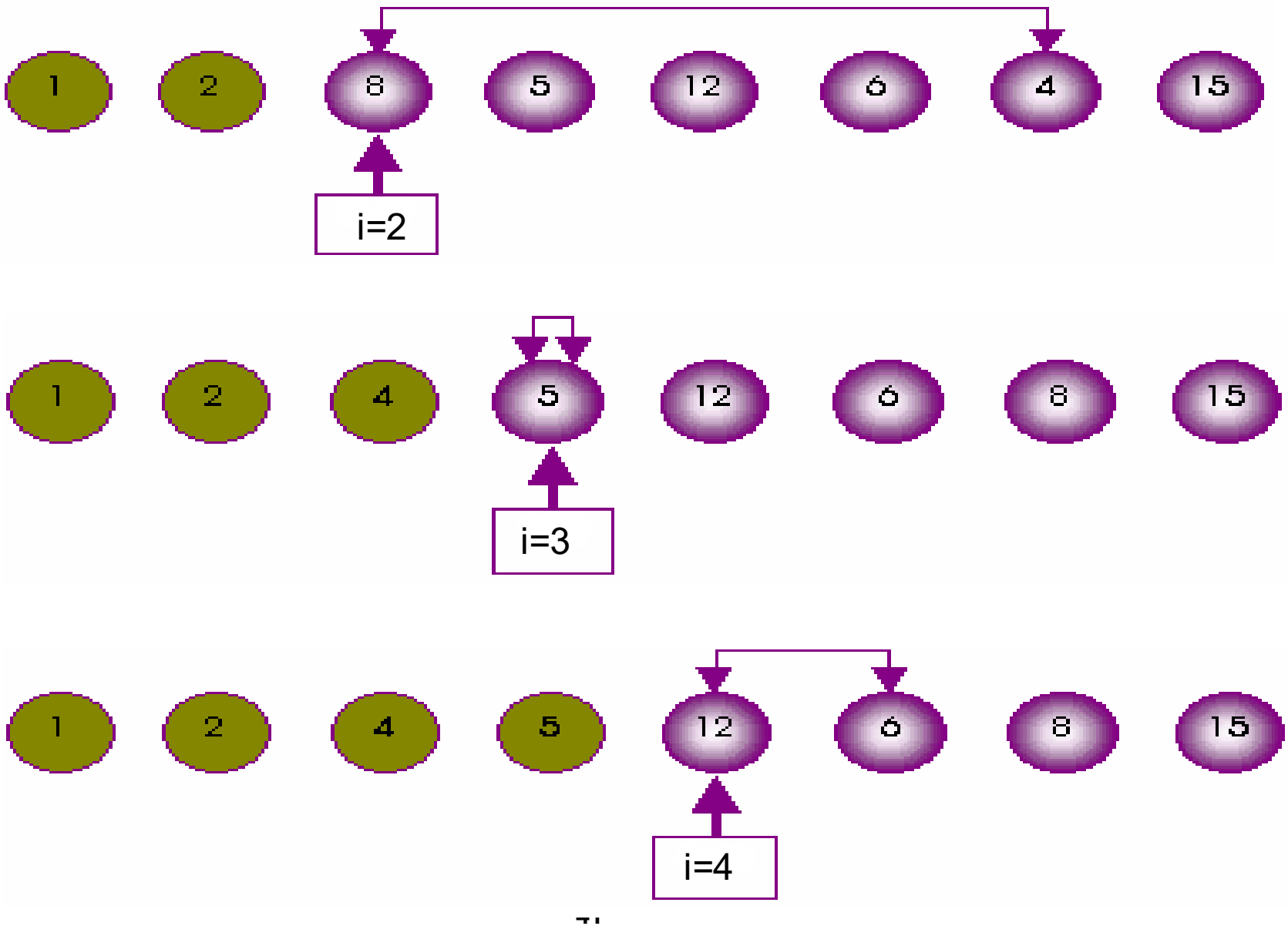
12 2 8 5 1 6 4 15



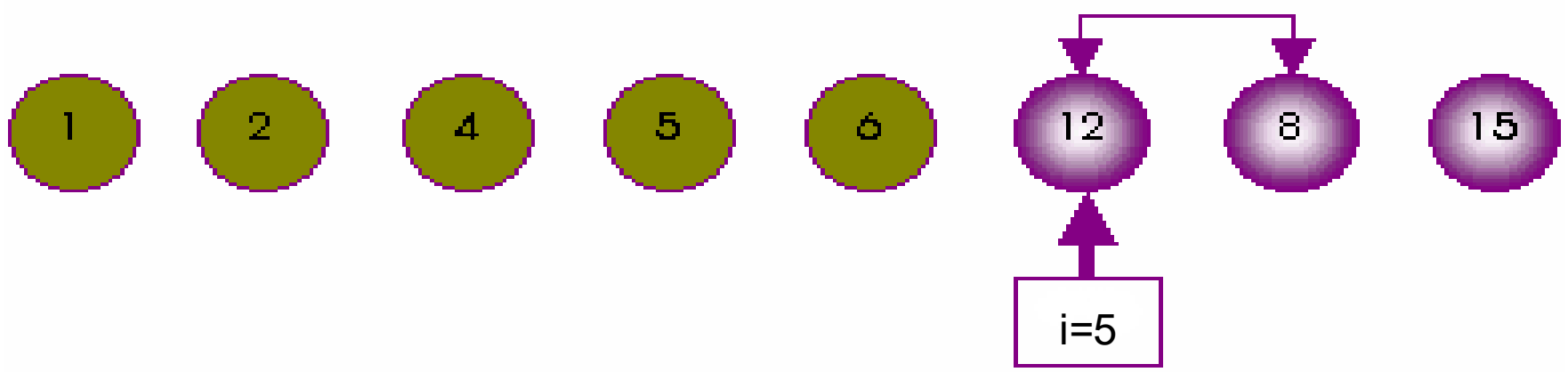
Chọn Trực Tiếp – Selection Sort

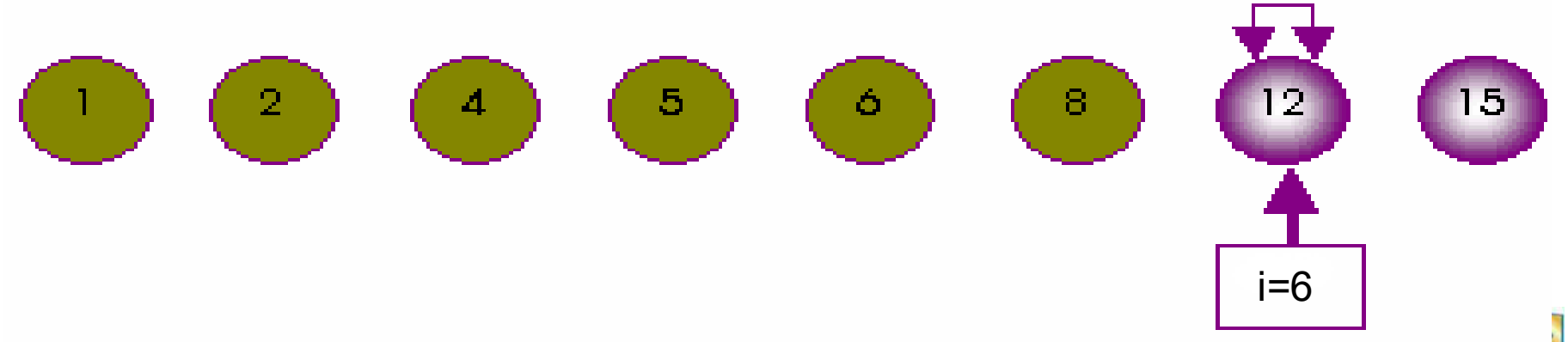


Chọn Trực Tiếp – Selection Sort



Chọn Trực Tiếp – Selection Sort





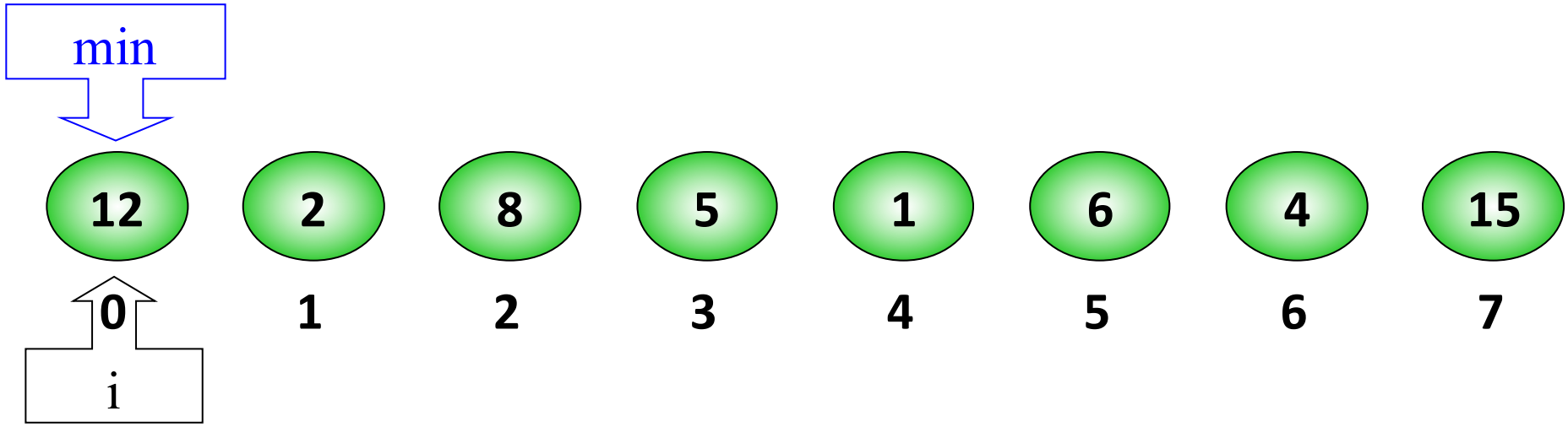
Cài Đặt Thuật Toán Chọn Trực Tiếp

```
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)

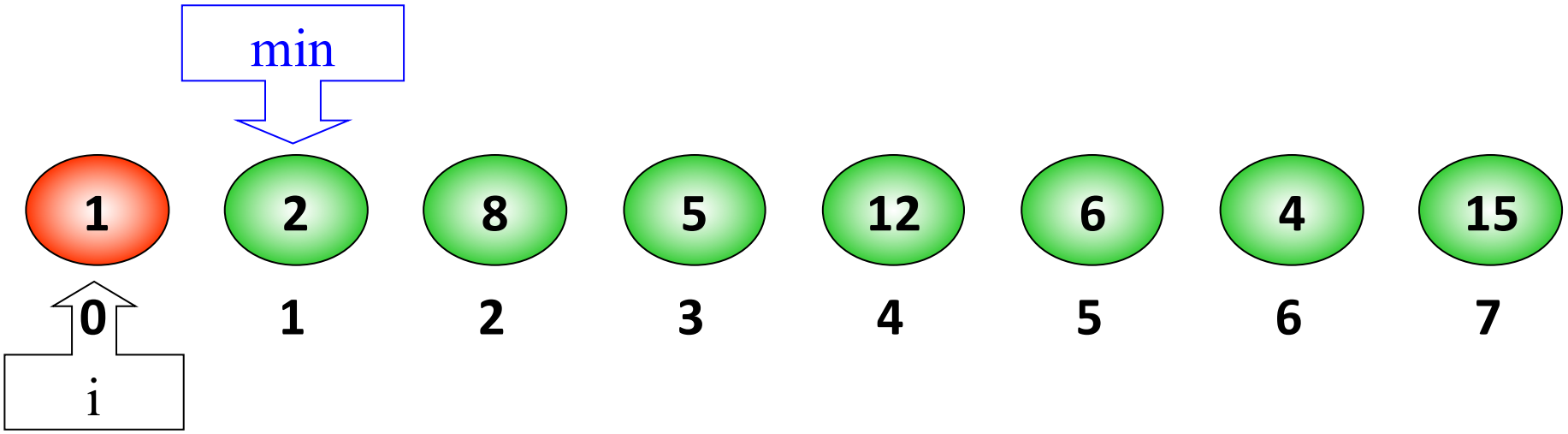
Swap(a[0], a[4])



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(1,7)

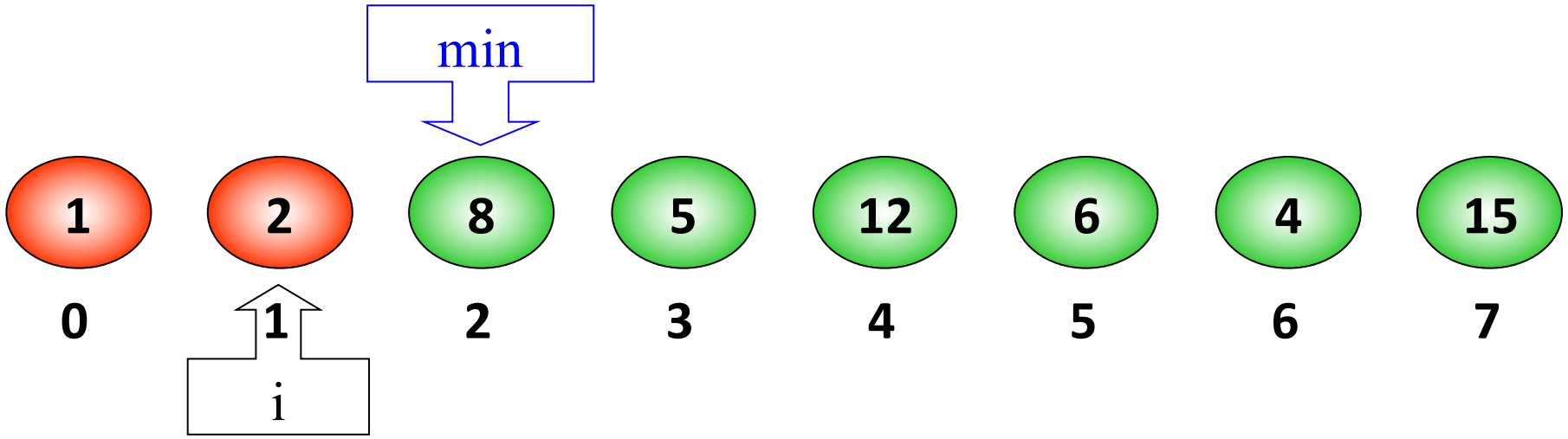
Swap(a[1], a[1])



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(2,7)

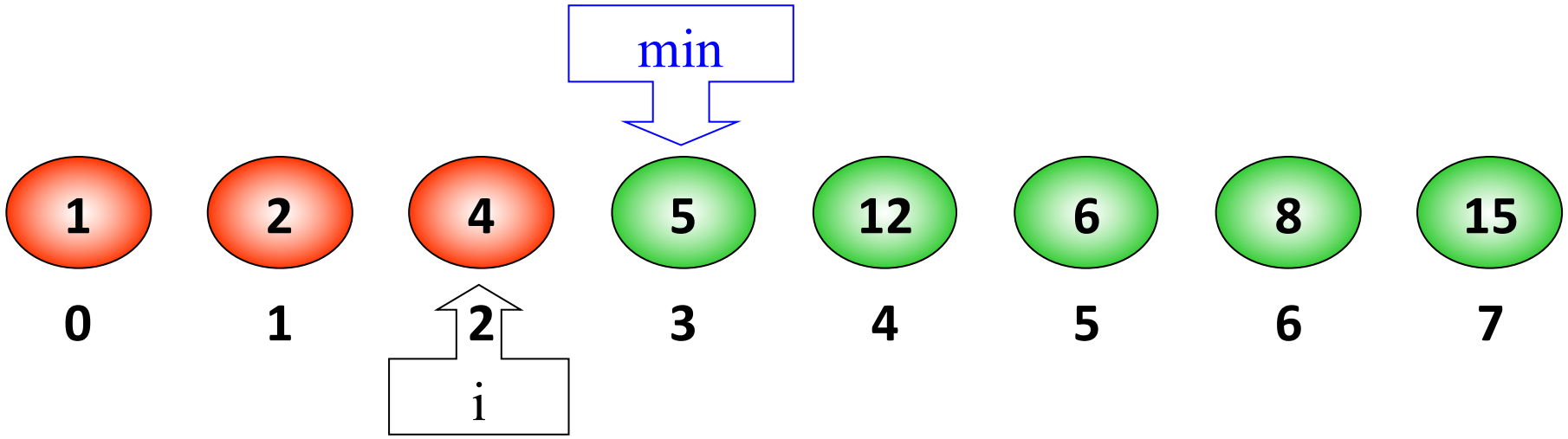
Swap(a[2], a[6])



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(3, 7)

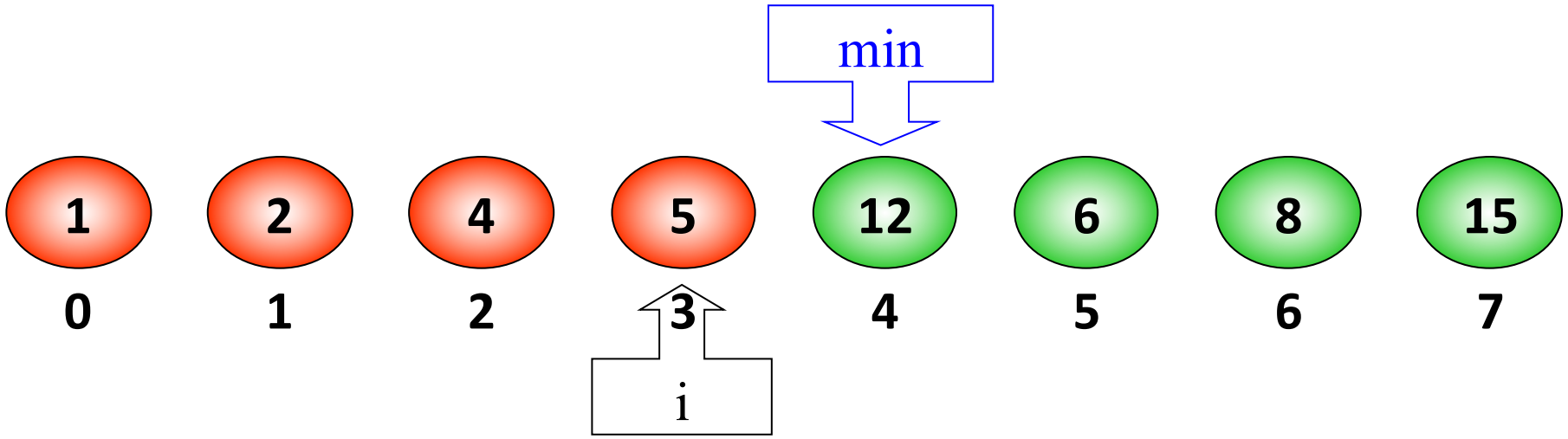
Swap(a[3], a[3])



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(4, 7)

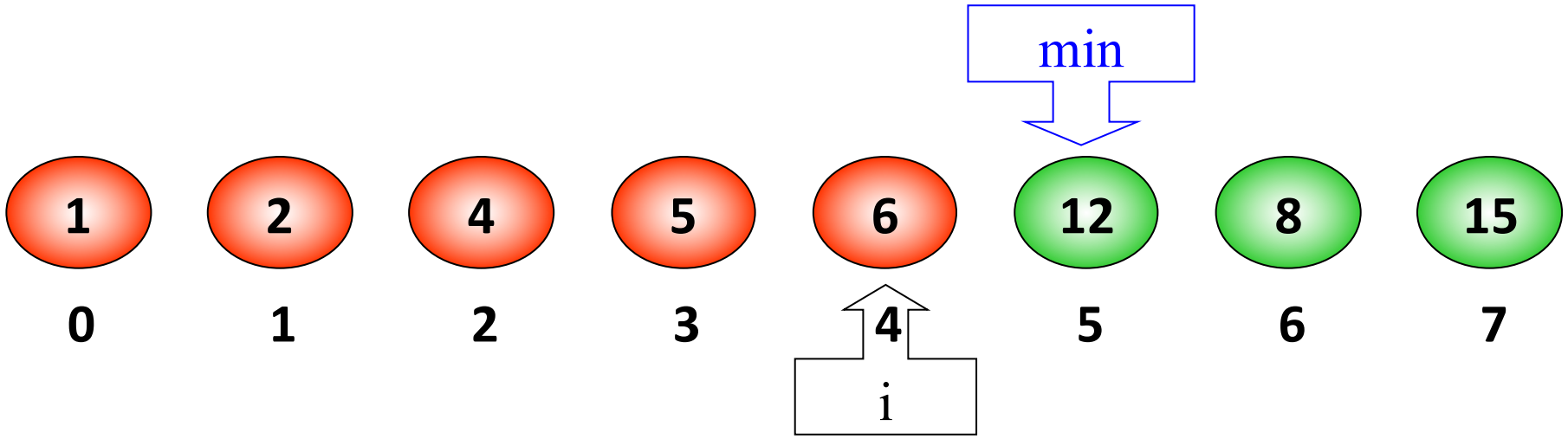
Swap(a[4], a[5])



Minh Họa Thuật Toán Chọn Trực Tiếp

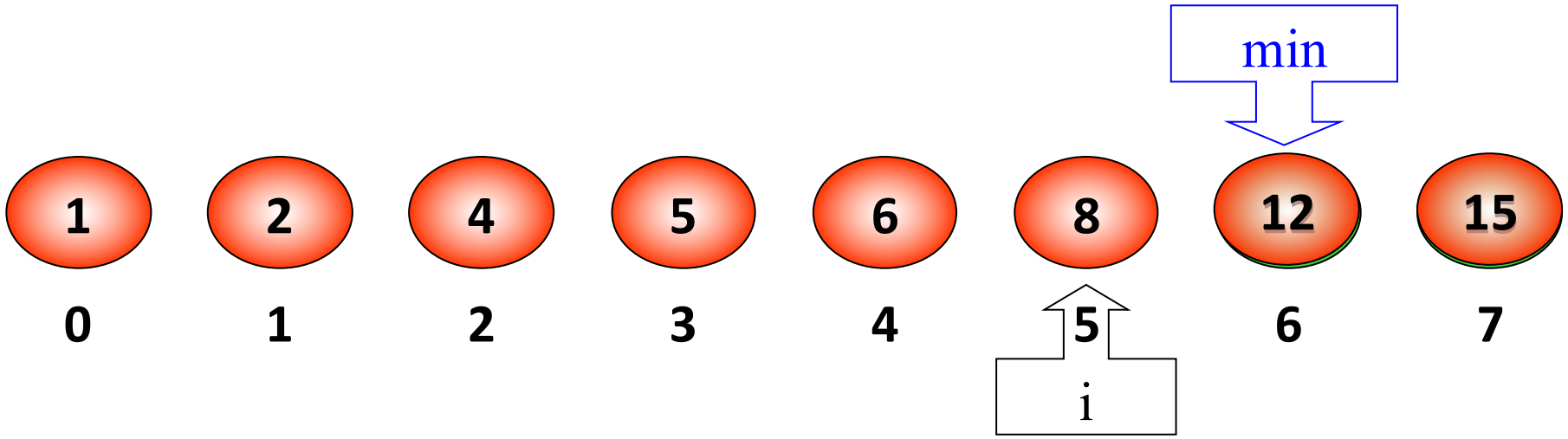
Vị trí nhỏ nhất(5,7)

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



Minh Họa Thuật Toán Chọn Trực Tiếp

Vị trí nhỏ nhất(6, 7)



Độ Phức Tạp 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

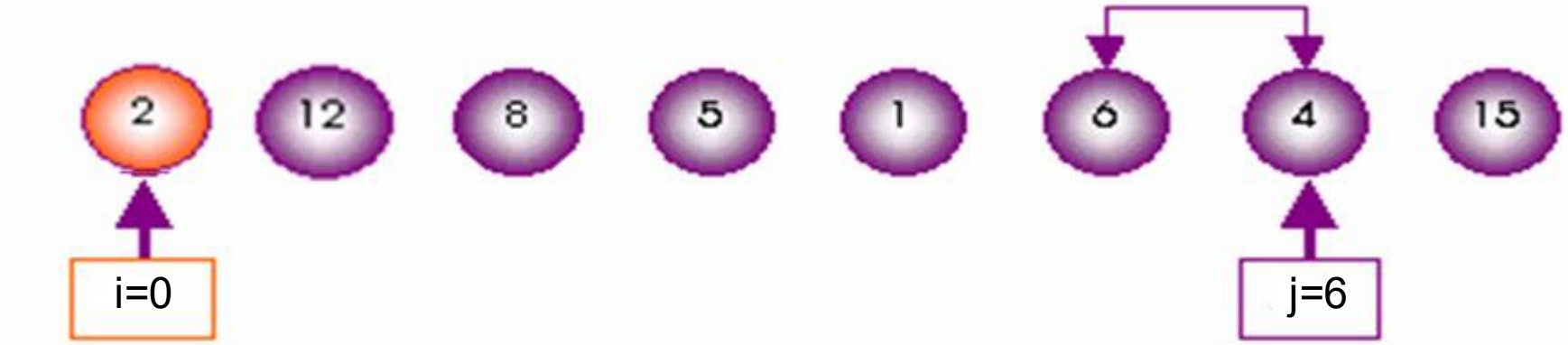
  Nếu i >=N-1: Hết dãy. Dừng

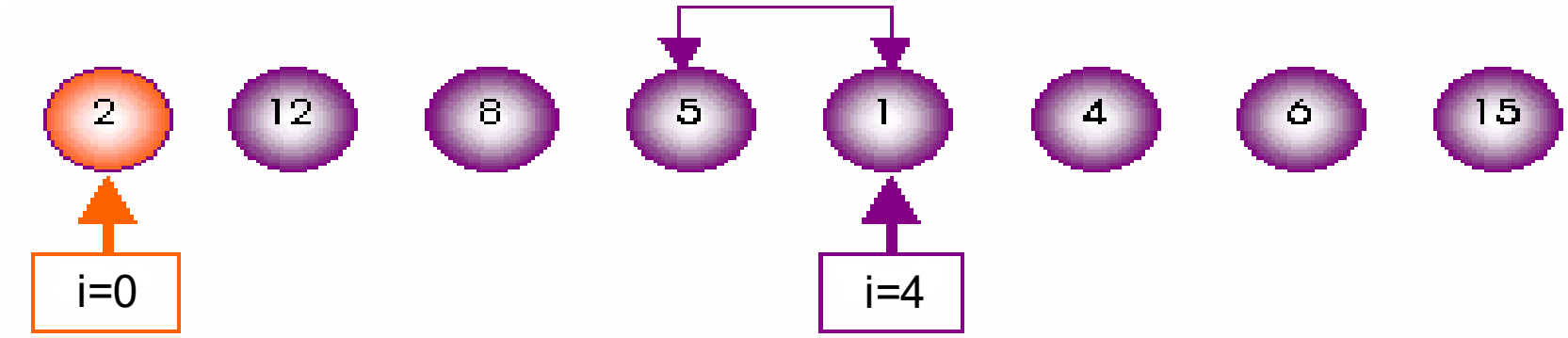
  Ngược lại : Lặp lại Bước 2.

Nổi Bọt – Bubble Sort

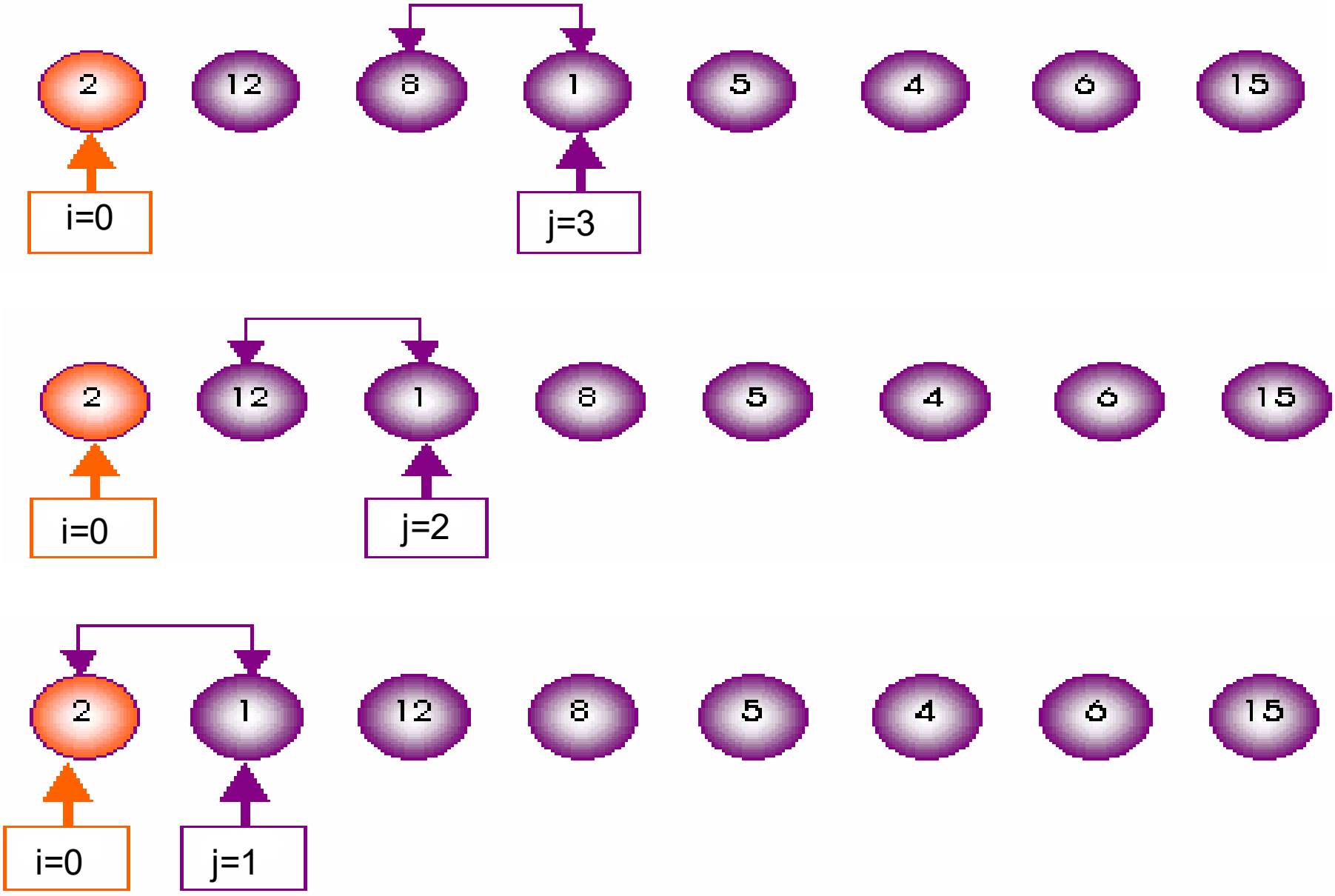
- Cho dãy số a:

2 12 8 5 1 6 4 15

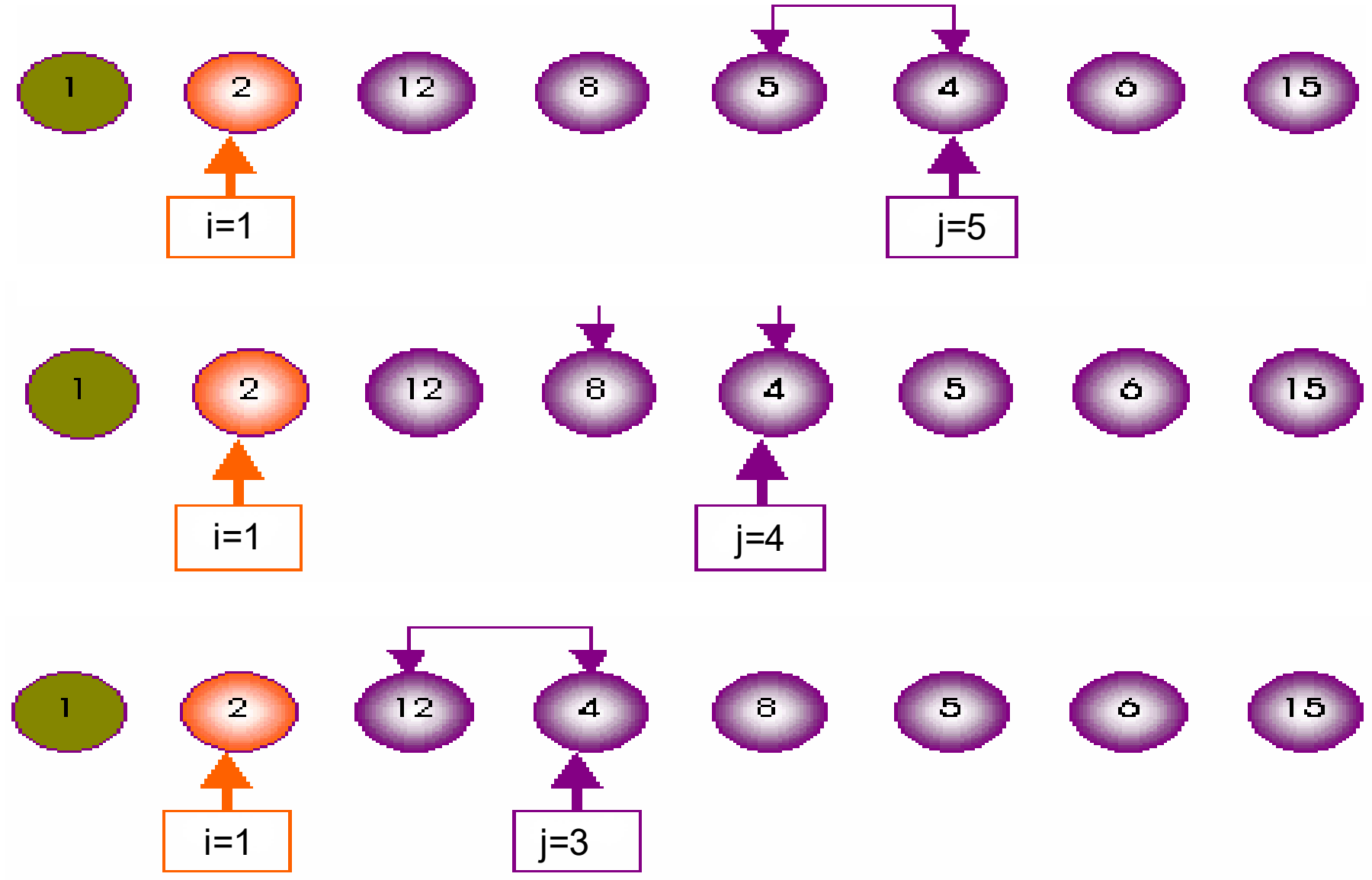




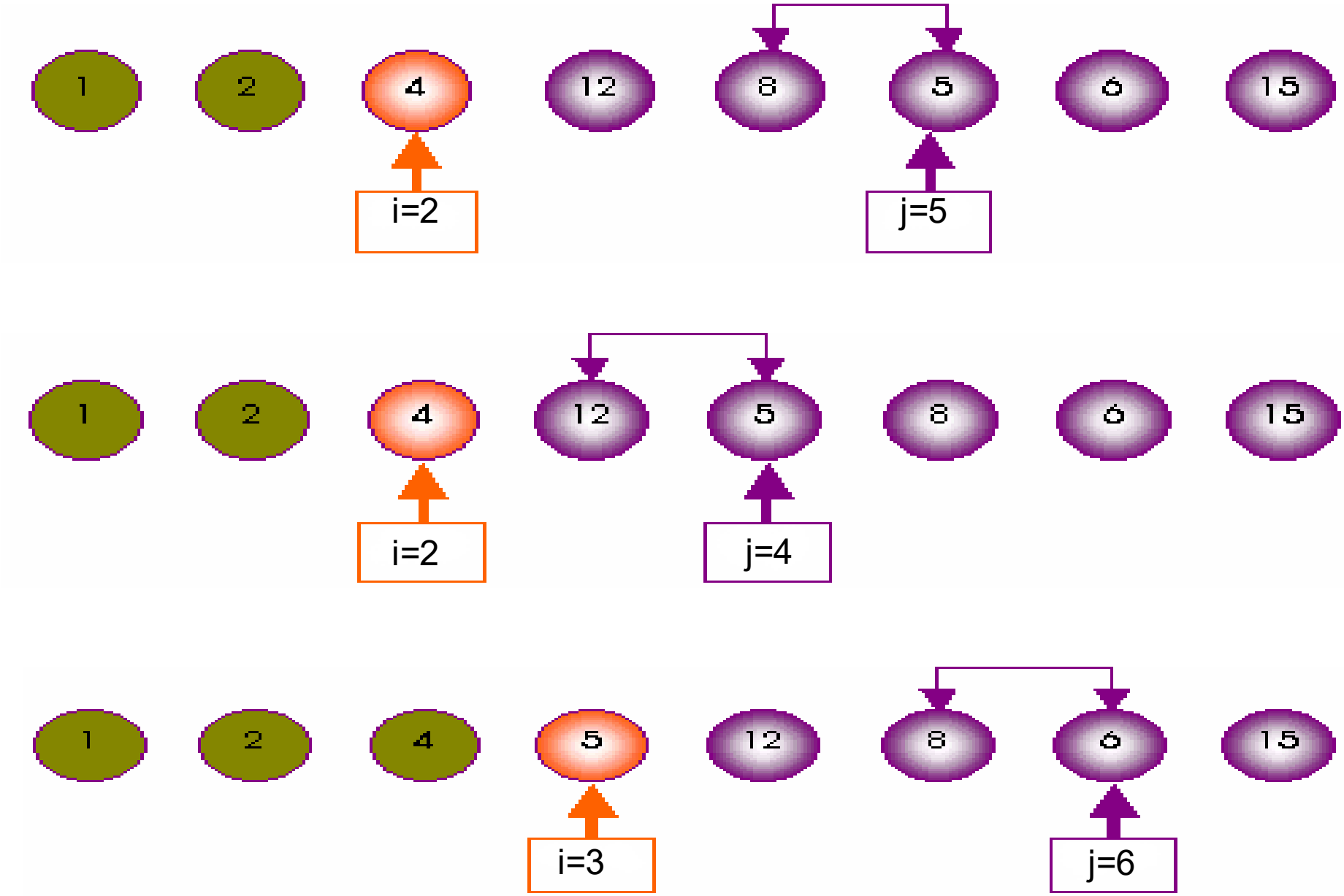
Nổi Bọt – Bubble Sort



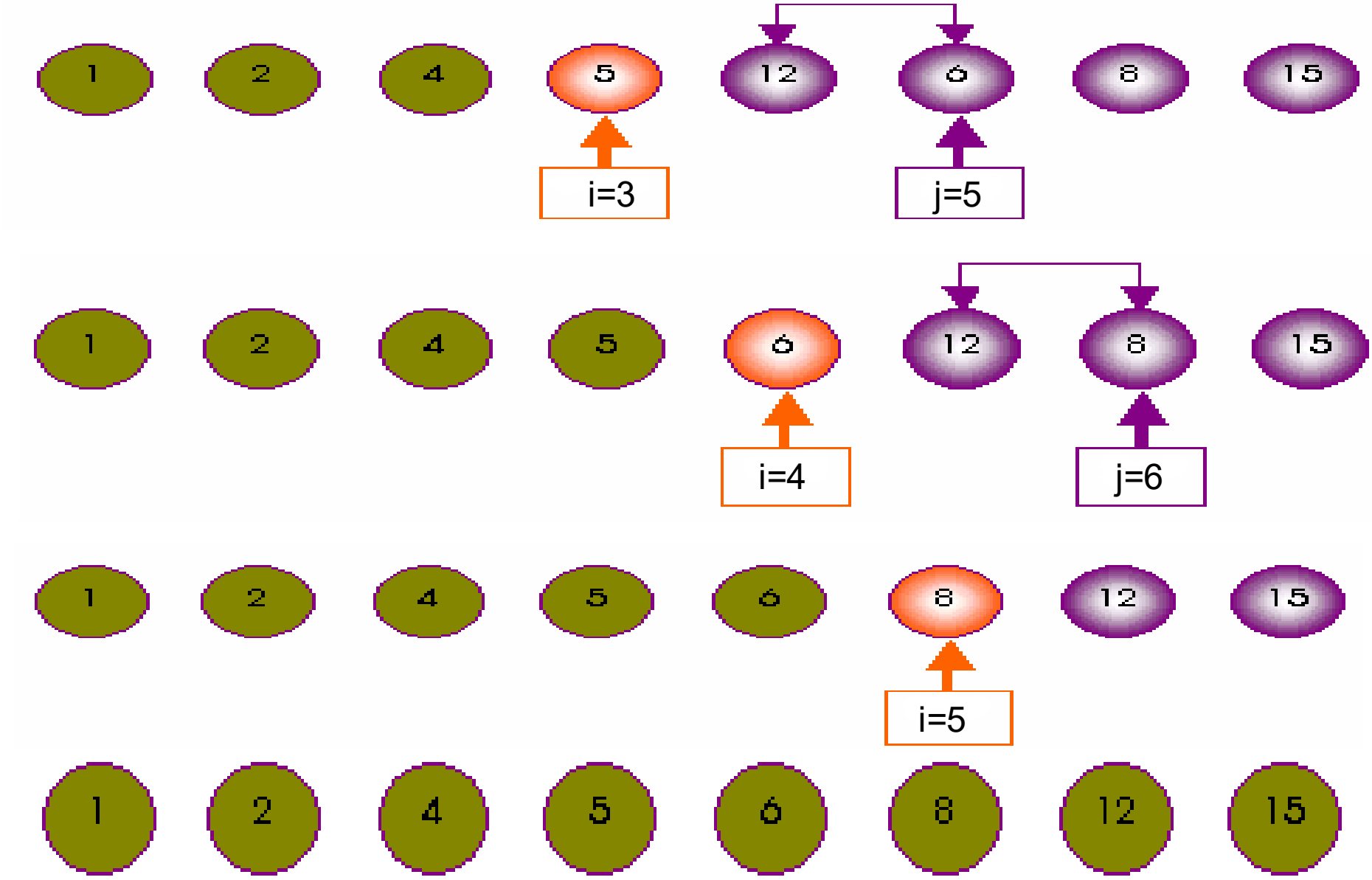
Nổi Bọt – Bubble Sort



Nổi Bọt – Bubble Sort



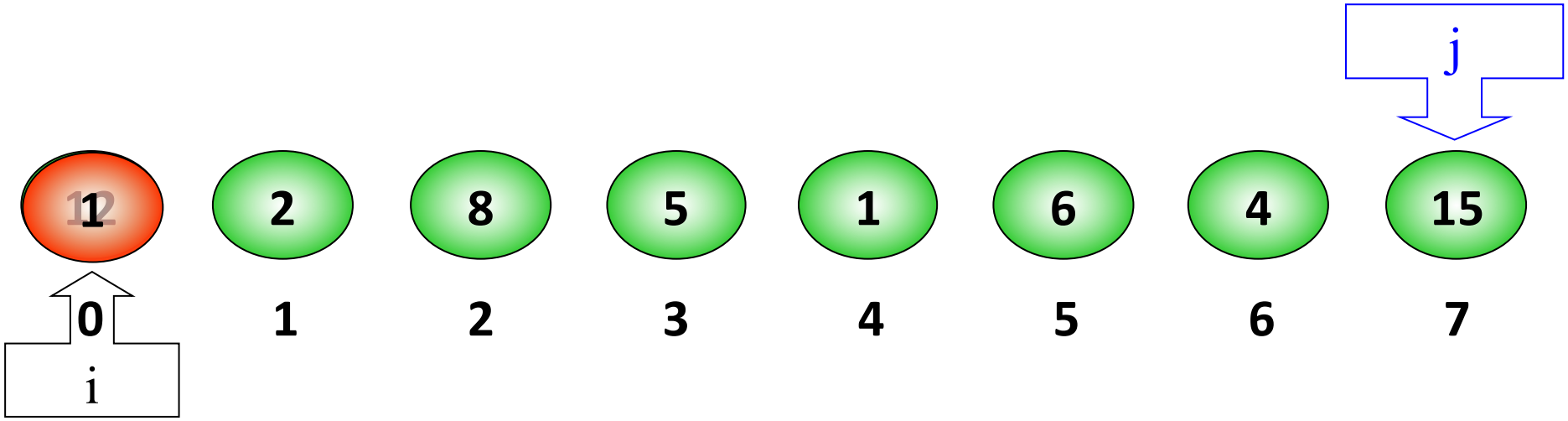
Nổi Bọt – Bubble Sort



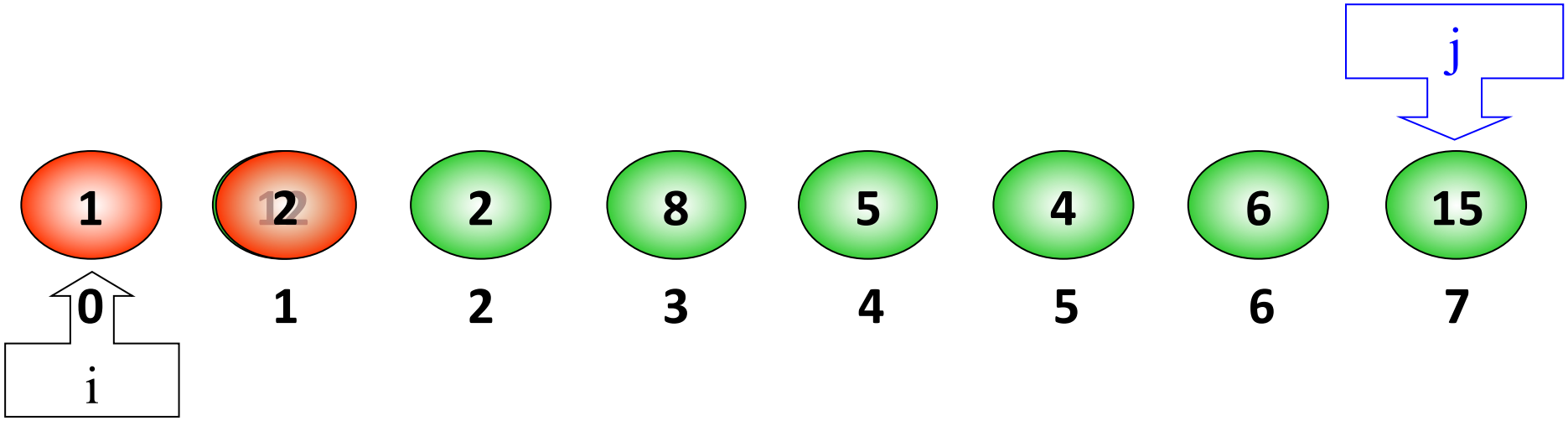
Cài Đặt Thuật Toán Nổi Bọt

```
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]);
}
```

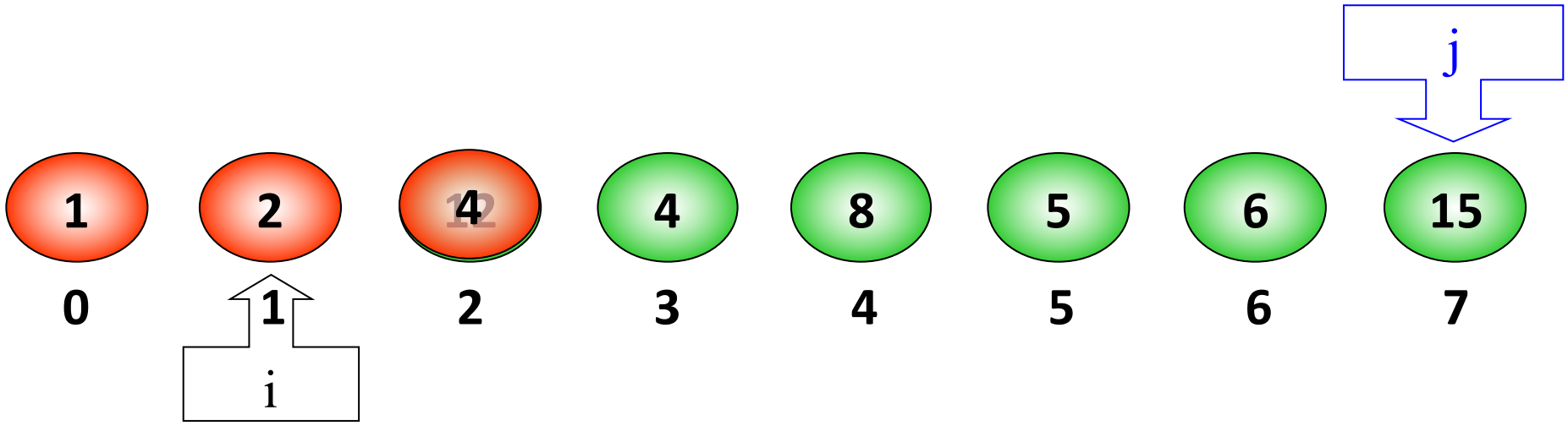

Minh Họa Thuật Toán



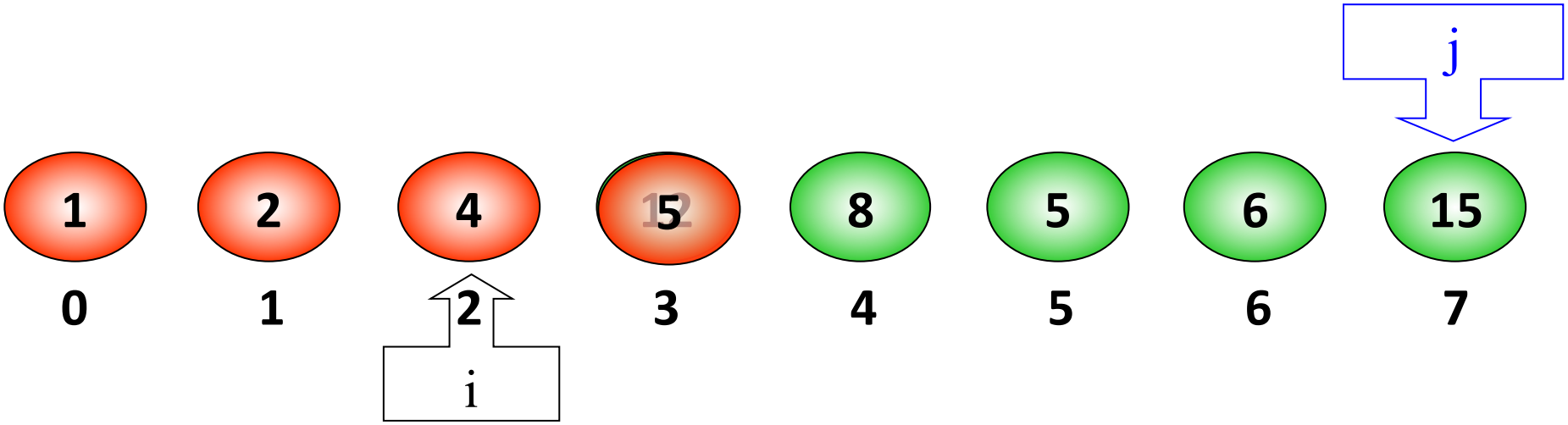
Minh Họa Thuật Toán



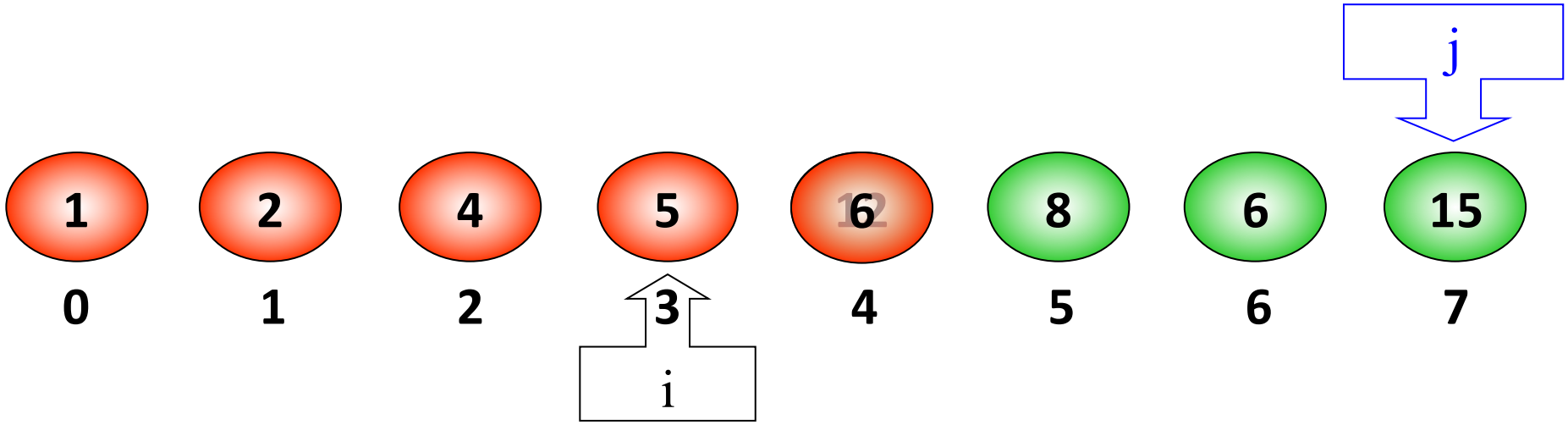
Minh Họa Thuật Toán



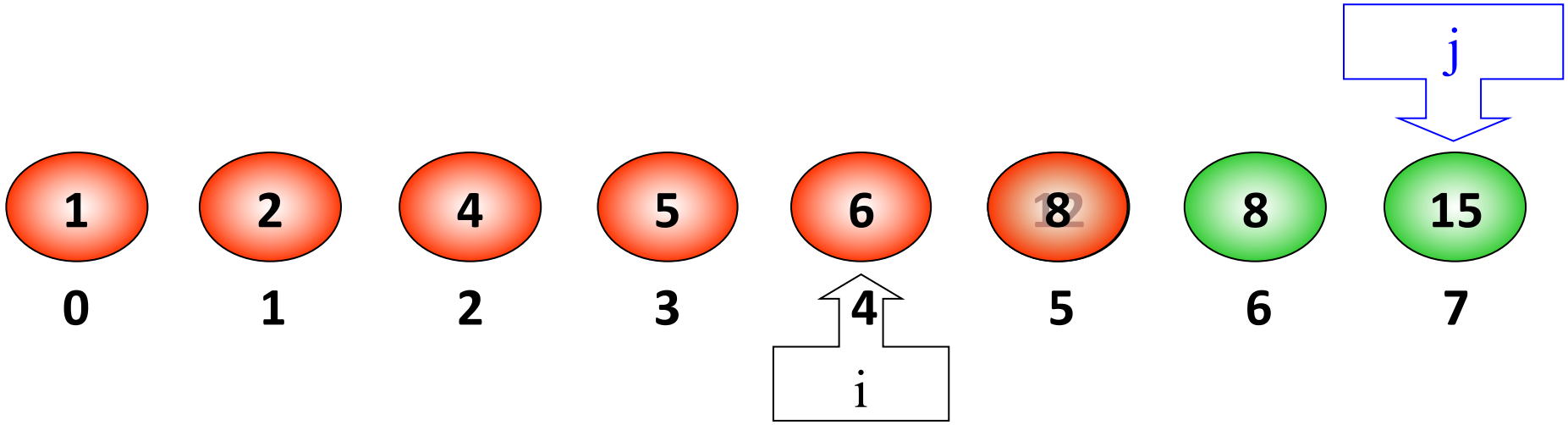
Minh Họa Thuật Toán



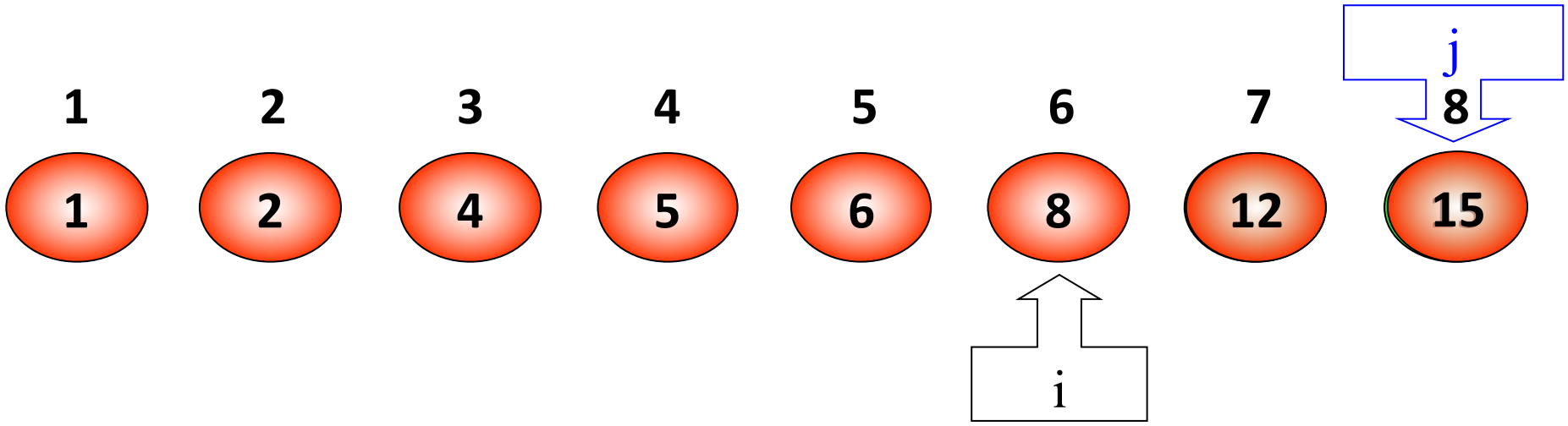
Minh Họa Thuật Toán



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->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

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 $a_0 , a_1 ,... ,a_{n-1}$ trong đó i phần tử đầu tiên $a_0 , a_1 ,... ,a_{i-1}$ 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 $a_0 , a_1,... ,a_i$ trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử a_{k-1} và a_k thỏa $a_{k-1} < a_i < a_k$ $(1 \le k \le i)$.

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 chổ cho a[i]
- Bước 4: a[pos] = x; //có đoạn a[1]..a[i] đã được sắp
- Bước 5: i = i+1;

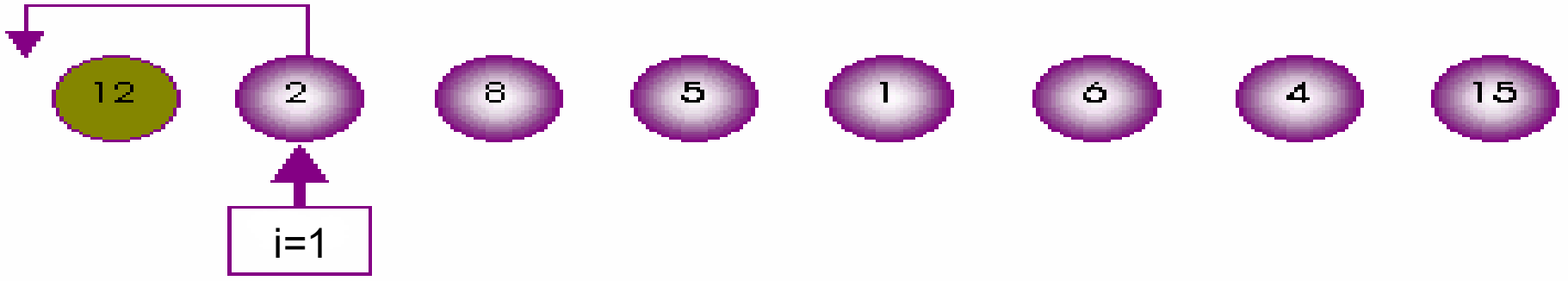
  Nếu i < n : Lặp lại Bước 2

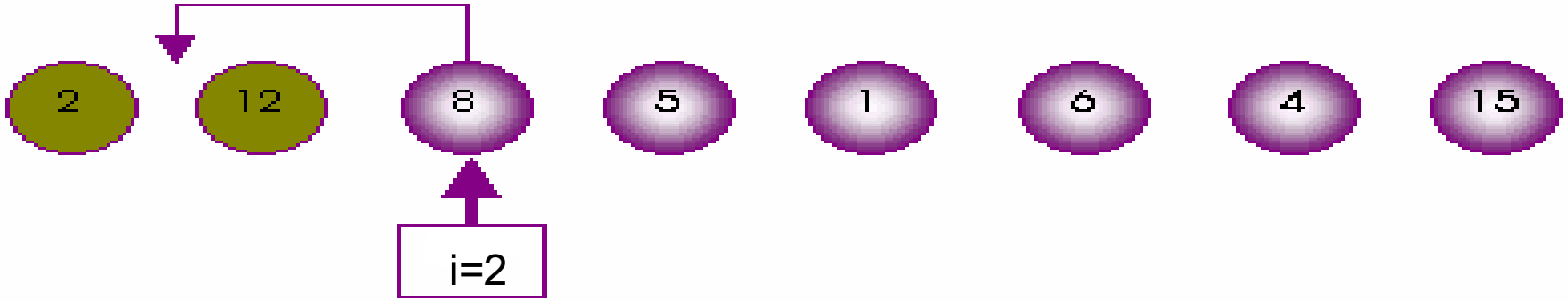
  Ngược lại : Dừng

Chèn Trực Tiếp – Insertion Sort

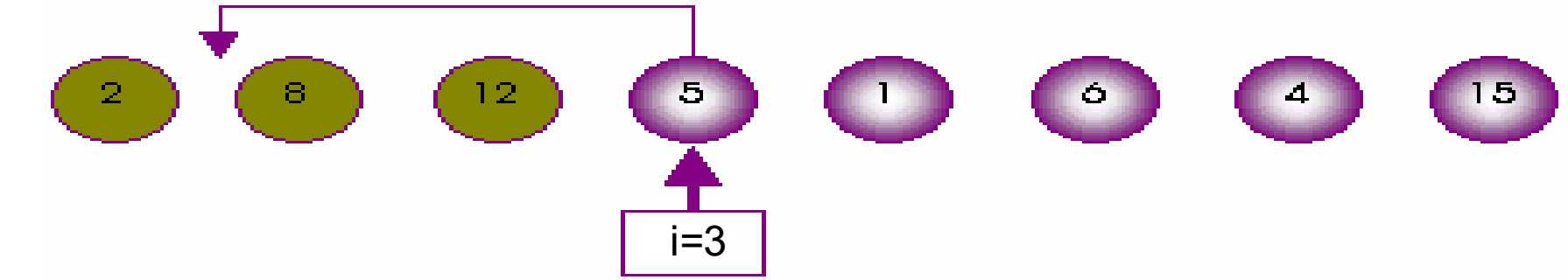
- Cho dãy số :

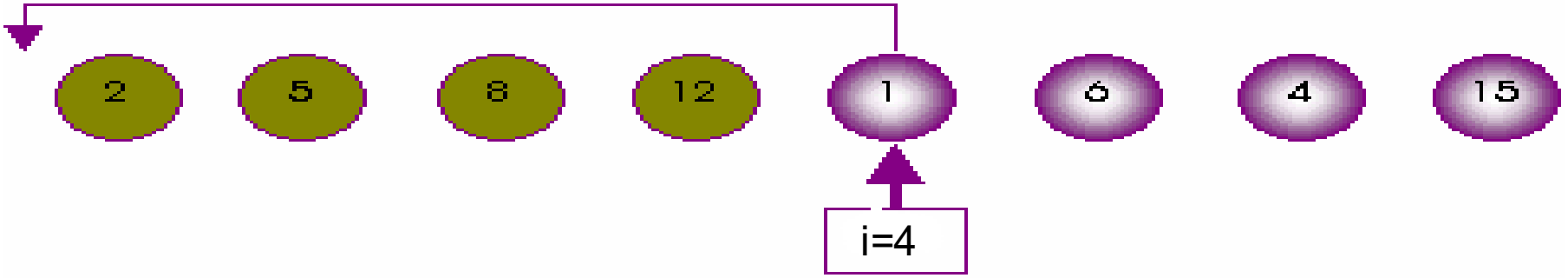
12 2 8 5 1 6 4 15

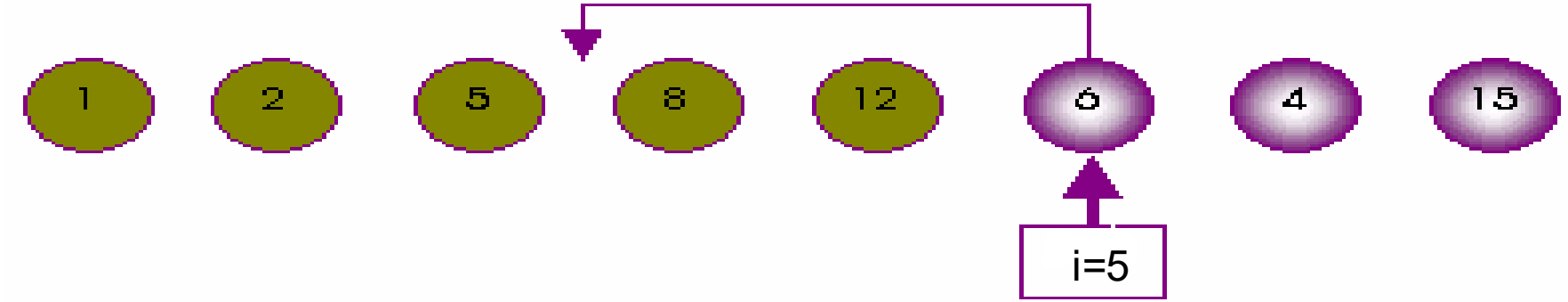




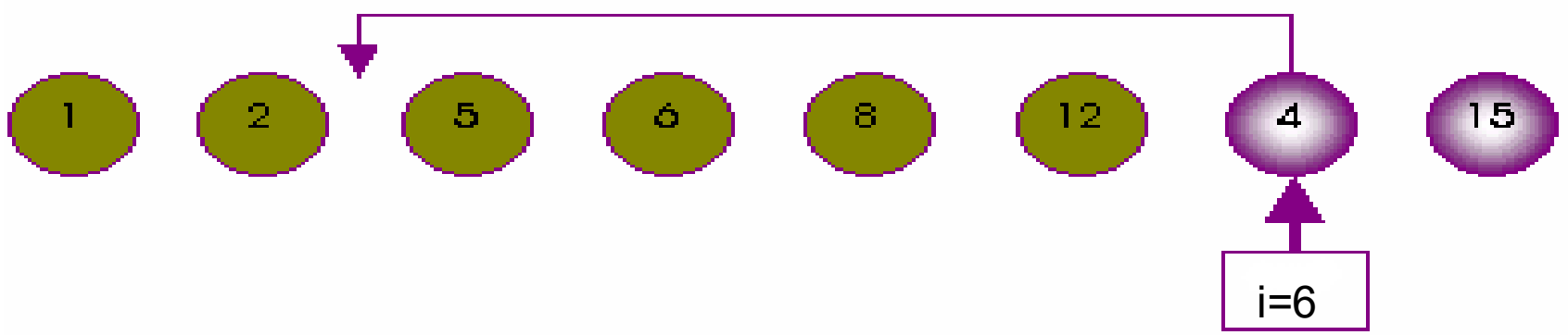
Chèn Trực Tiếp – Insertion Sort

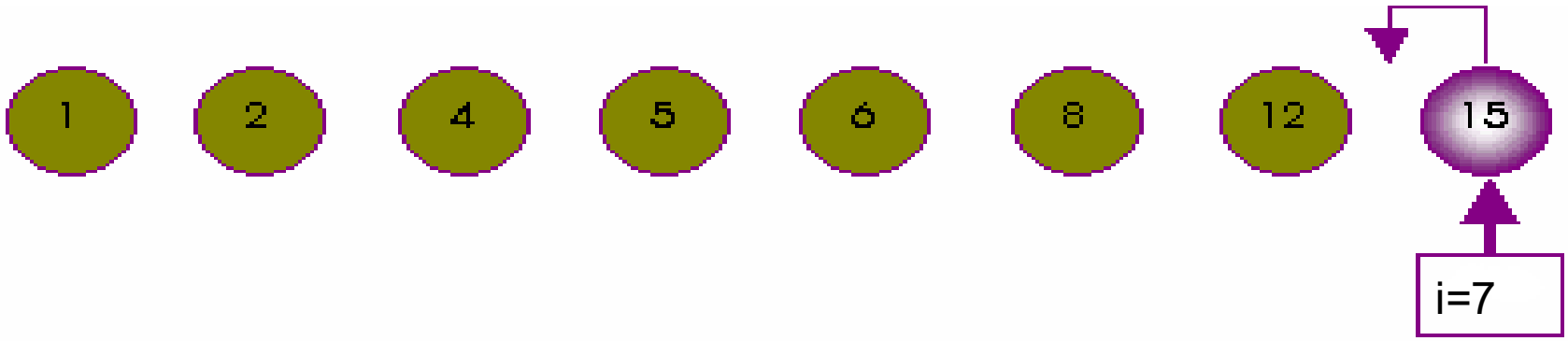






Chèn Trực Tiếp – Insertion Sort

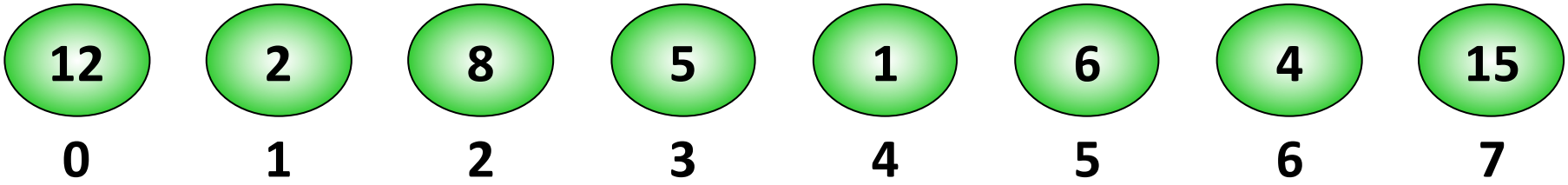




Cài Đặt Thuật Toán Chèn Trực Tiếp

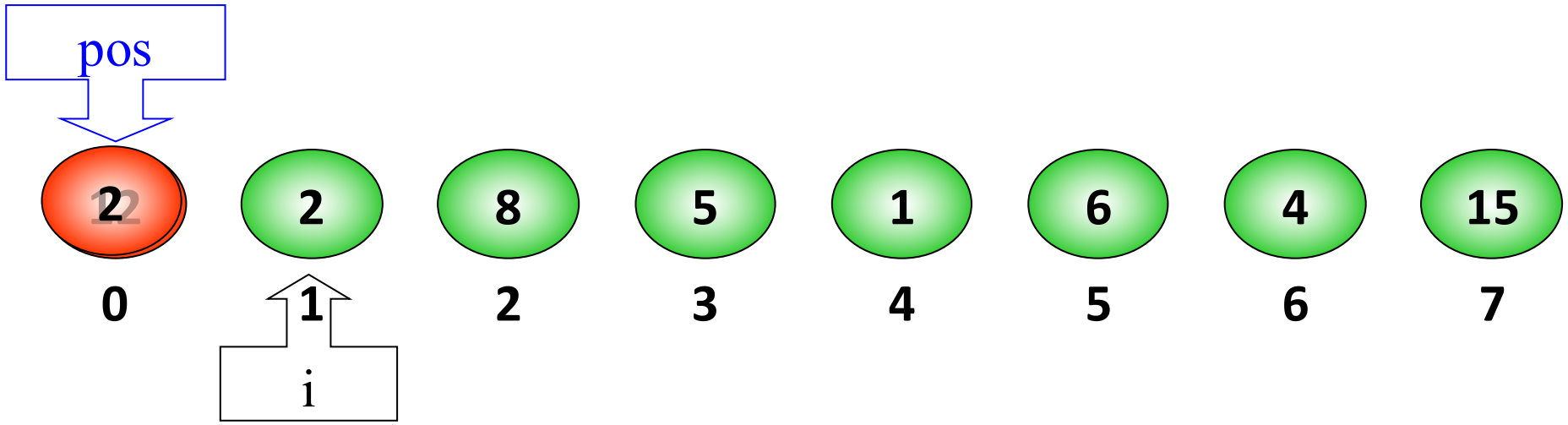
```
void InsertionSort(int d, int n )
    {    int pos, i;
         int x;//lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
         for(i=1 ; i<n ; i++) //đoạn a[0] đã sắp
         {
                  x = a[i]; pos = i-1;
                  // tìm vị trí chèn x
                  while((pos >= 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

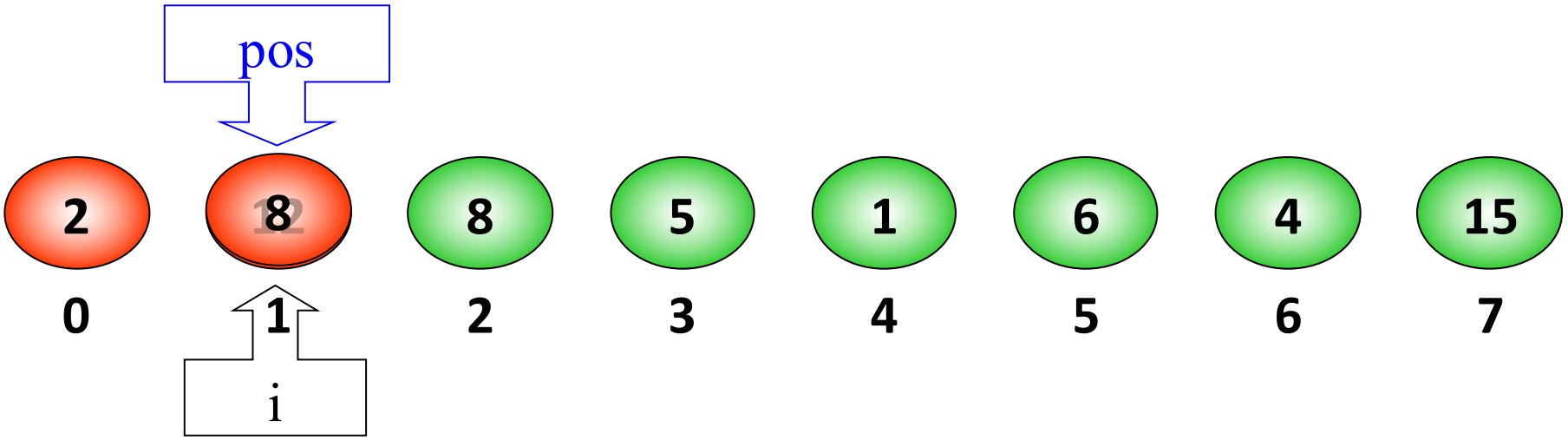
Insert a[1] into (0,0)



X

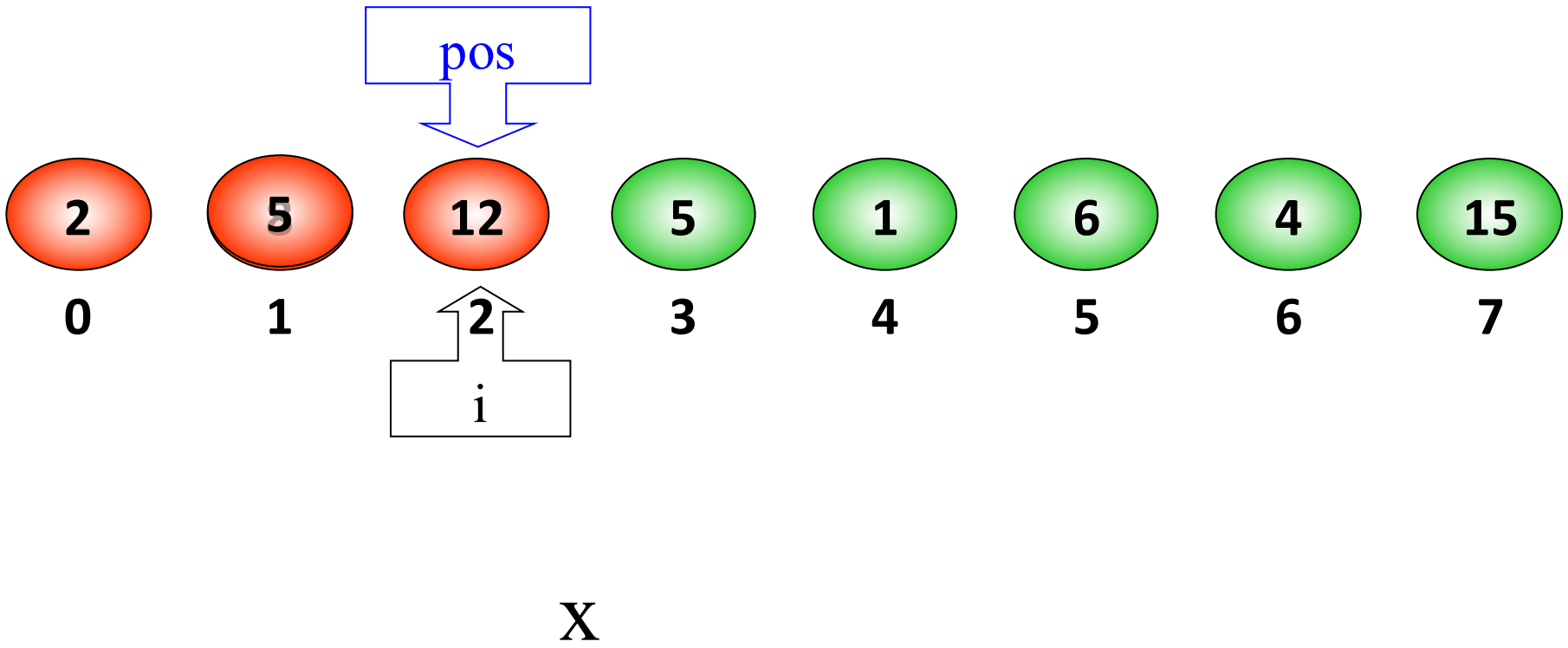
Minh Họa Thuật Toán Insertion Sort

Insert a[2] into (0, 1)

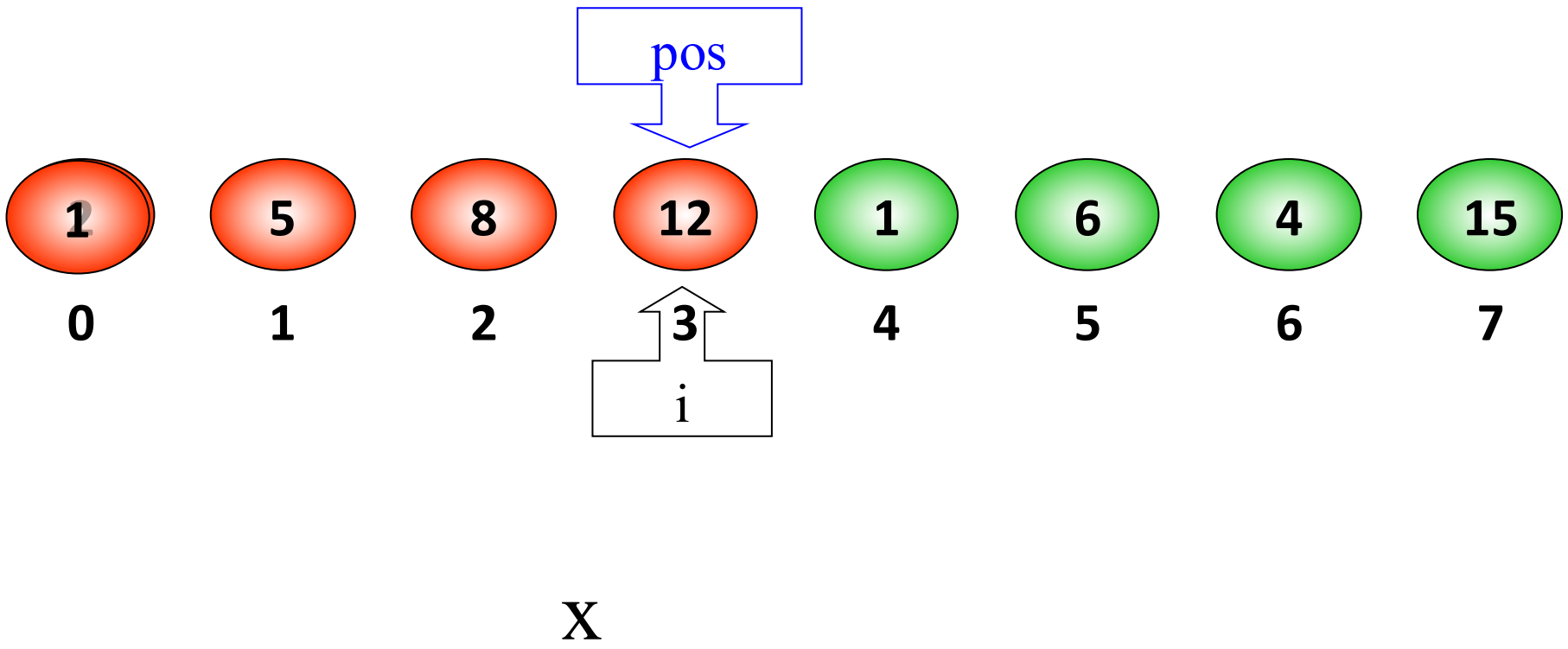


X

Minh Họa Thuật Toán Insertion Sort

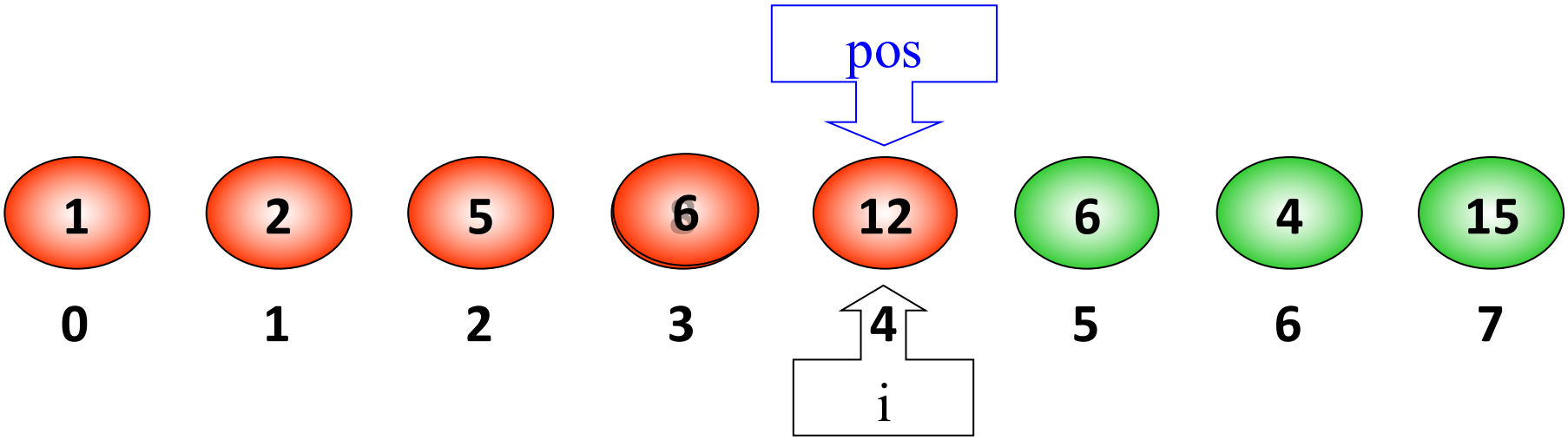


Minh Họa Thuật Toán Insertion Sort



Minh Họa Thuật Toán Insertion Sort

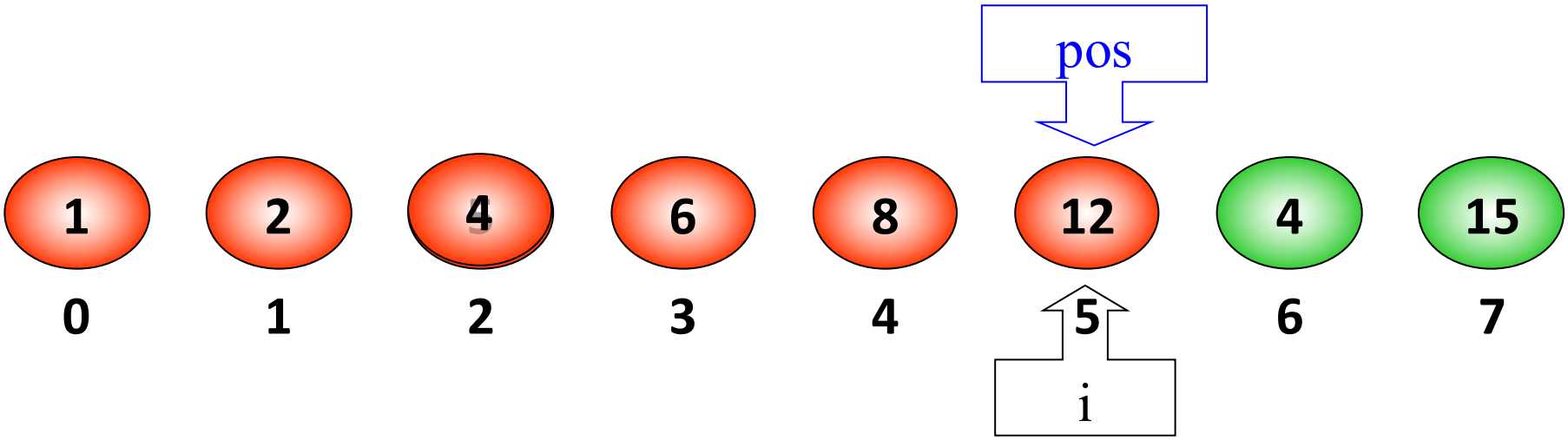
Insert a[5] into (0, 4)



X

Minh Họa Thuật Toán Insertion Sort

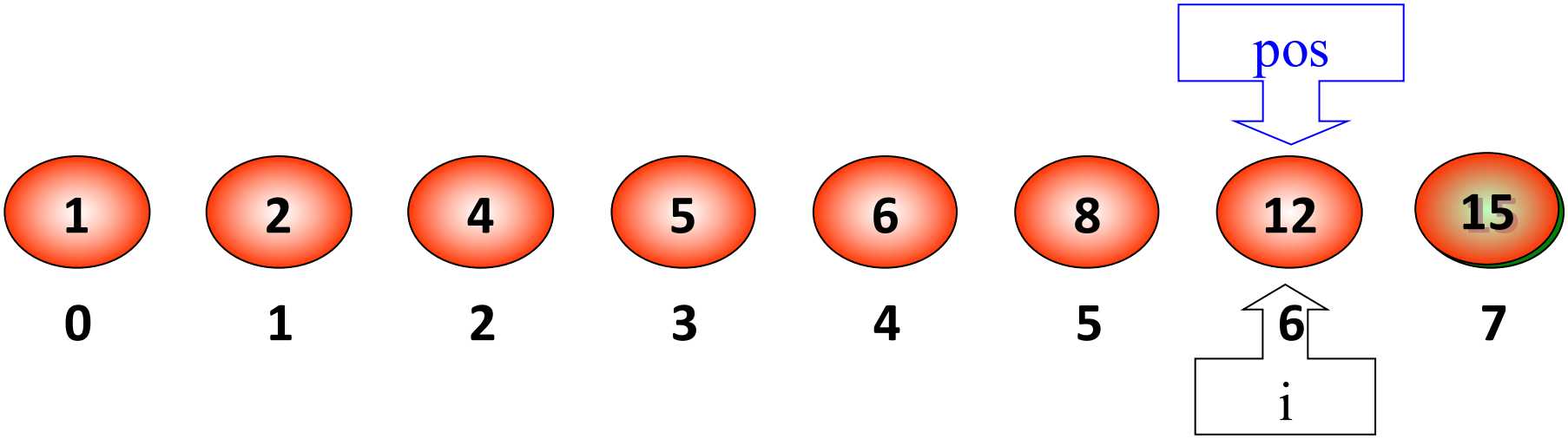
Insert a[6] into (0, 5)



X

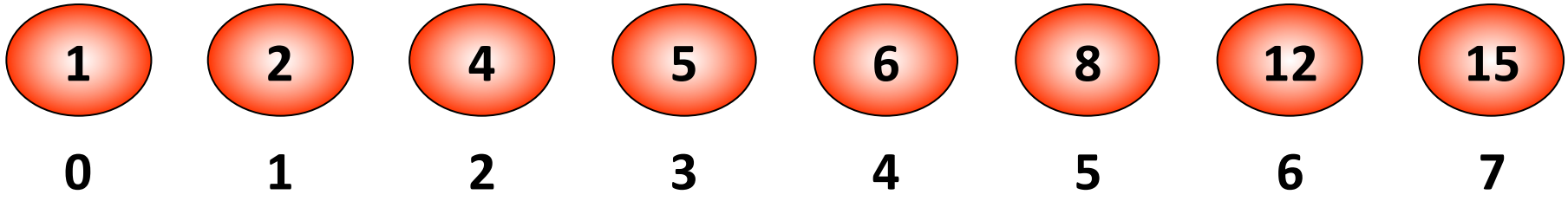
Minh Họa Thuật Toán Insertion Sort

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

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

- Bước 1: Chọn **k** khoảng cách h[1], h[2], ..., 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

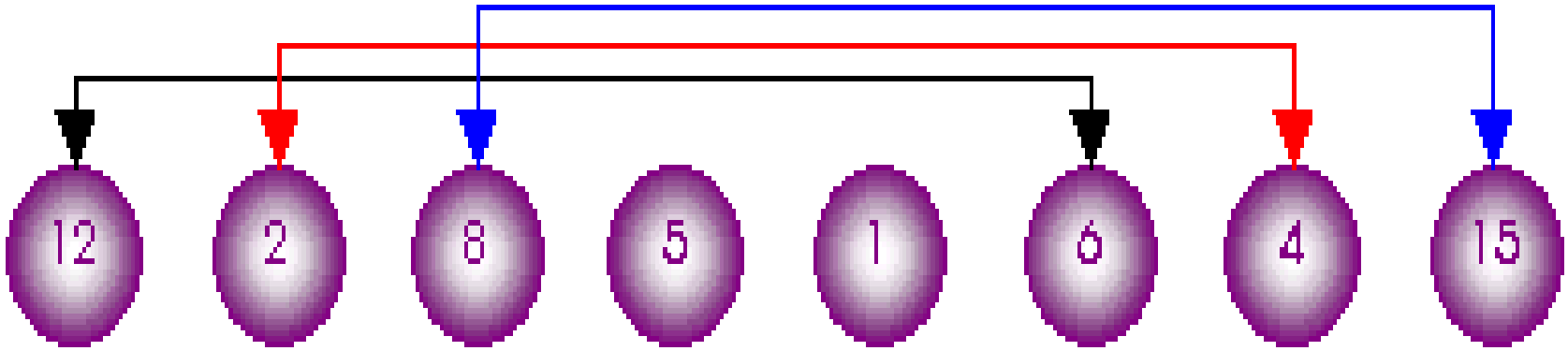
- Cho dãy số a:

  12 2 8 5 1 6 4 15

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

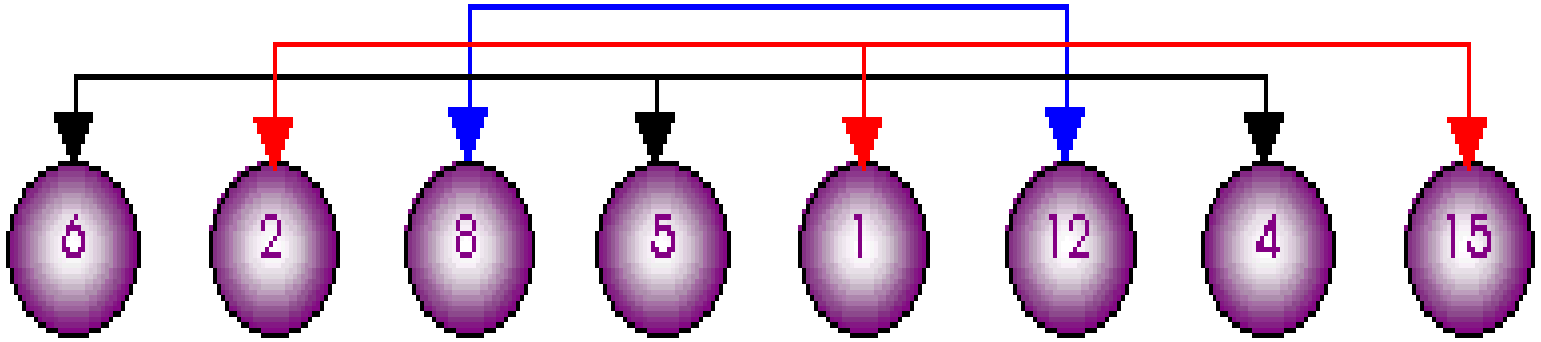
Shell Sort

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



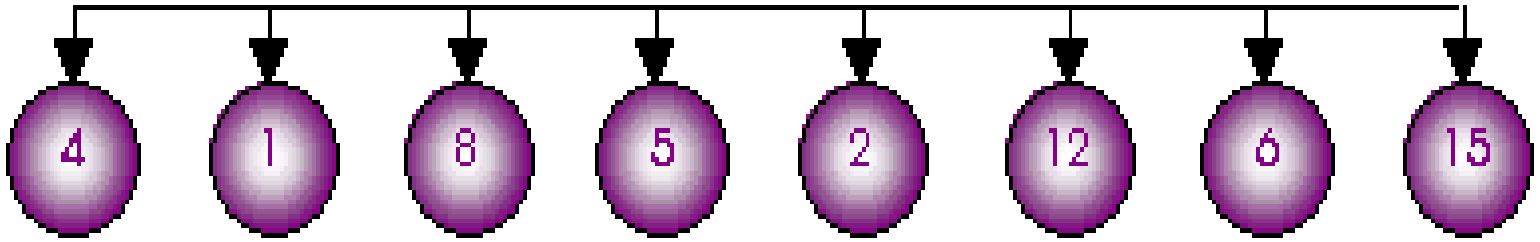
Shell Sort

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

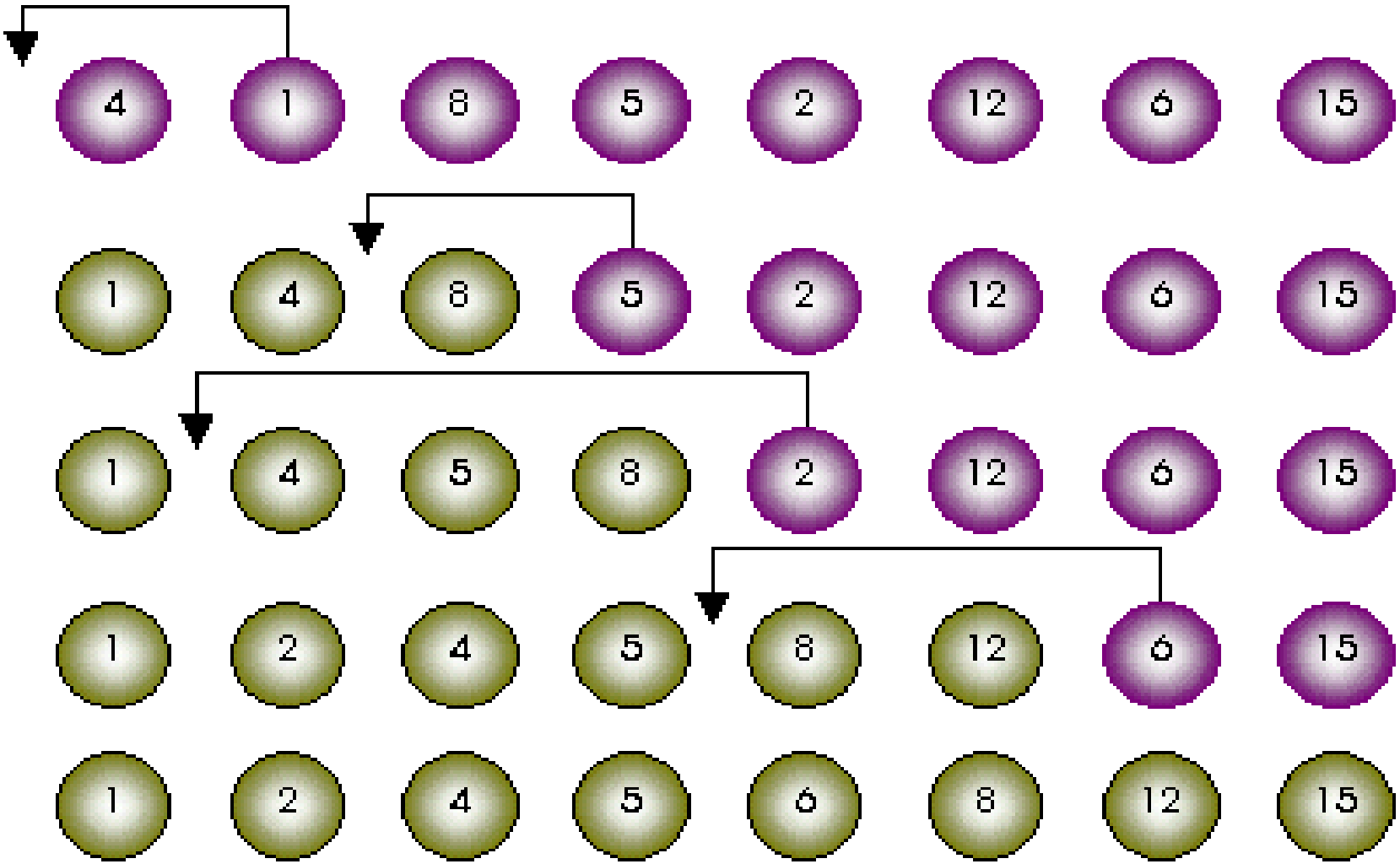


Shell Sort

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



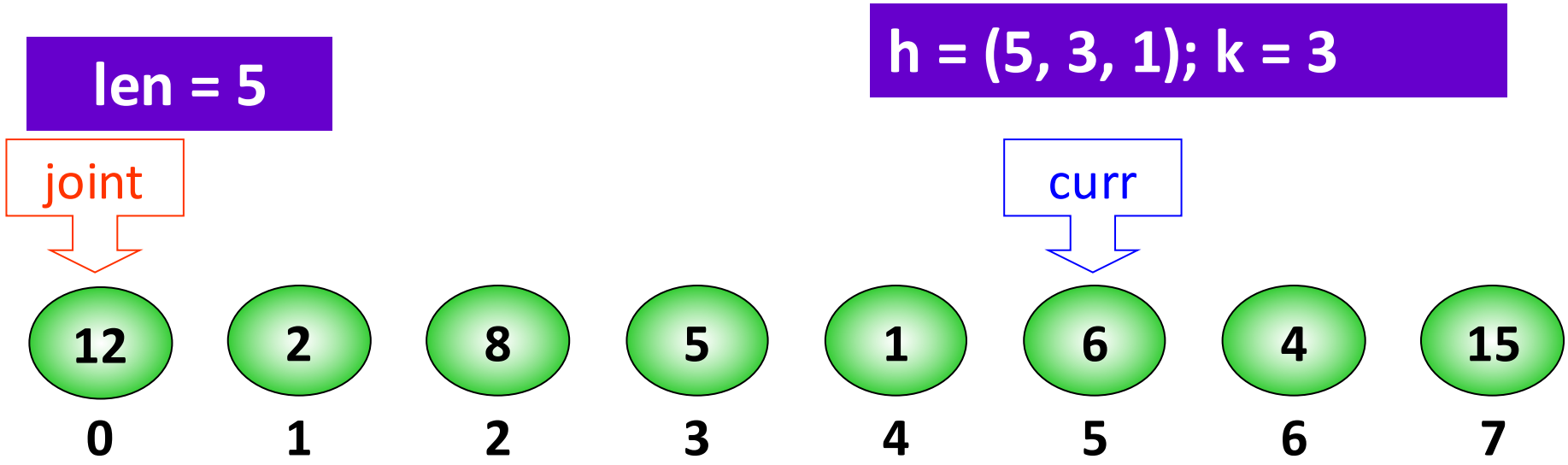
Shell Sort


4 1 8 5 2 12 6 15
1 4 8 5 2 12 6 15
1 4 5 8 2 12 6 15
1 2 4 5 8 12 6 15
1 2 4 5 6 8 12 15

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];
                  pos = i-len; // a[j] đứng kề trước a[i] trong cùng dãy con
                  while ((pos>=0)&&(x<a[pos])// sắp xếp dãy con chứa  x
                  {                    // bằng phương pháp chèn trực tiếp
                        a[pos+len]  = a[pos];
                        pos = pos - len;
                  }
                  a[pos+len] = x;
            }
      }
}
```

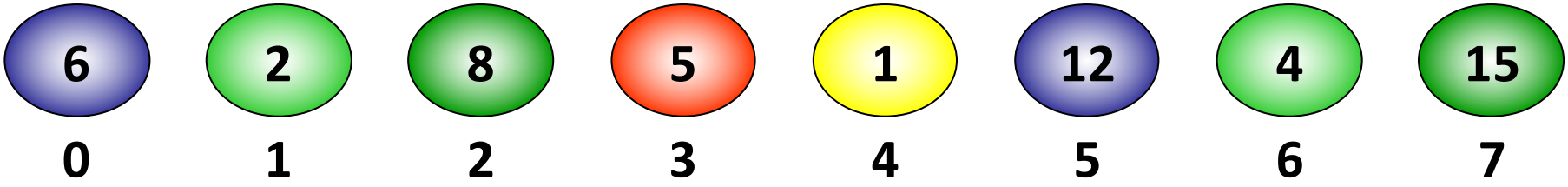

Shell Sort – Ví Dụ



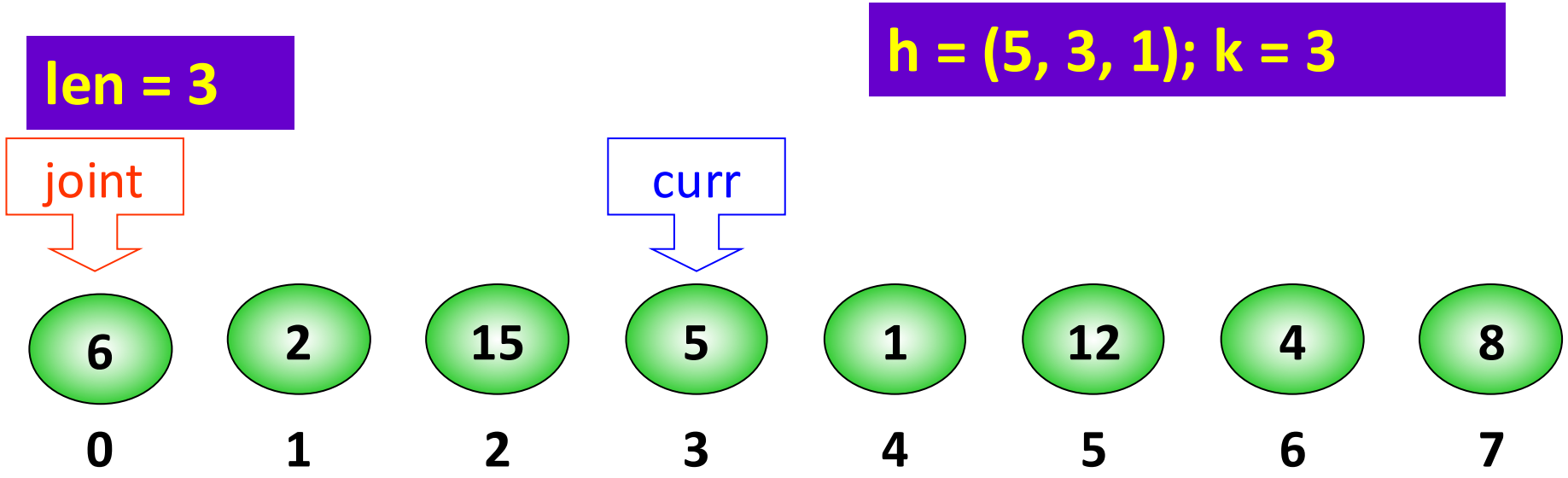
Shell Sort – Ví Dụ

len = 5;

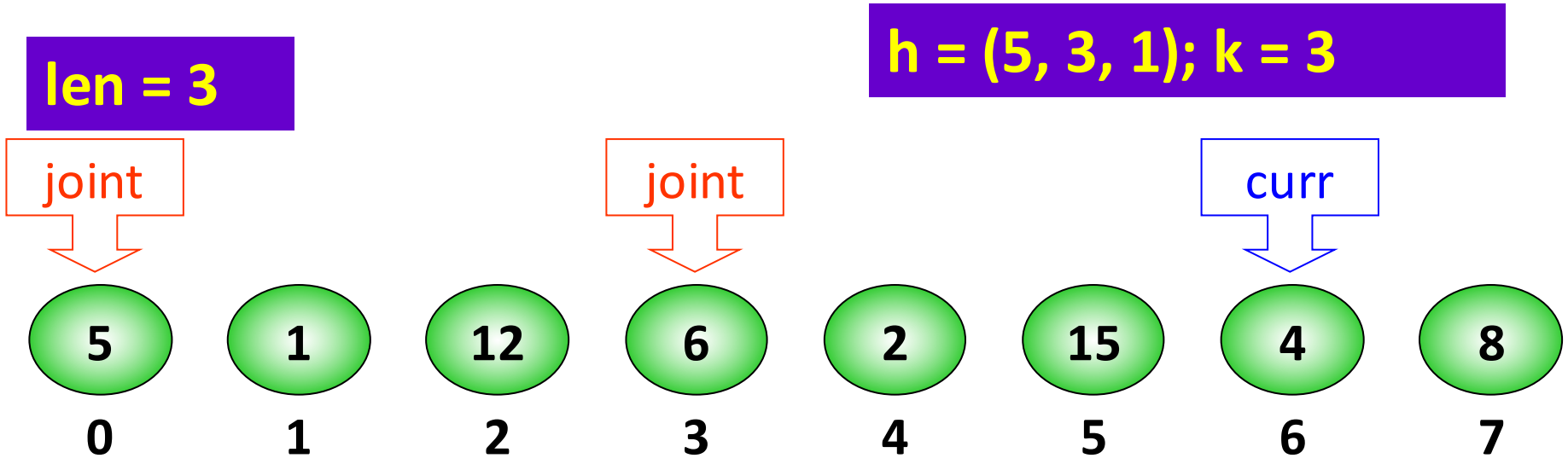
h = (5, 3, 1); k = 3



Shell Sort – Ví Dụ



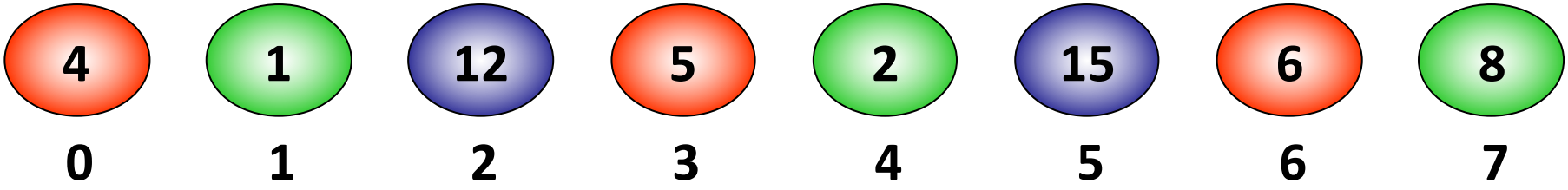
Shell Sort – Ví Dụ



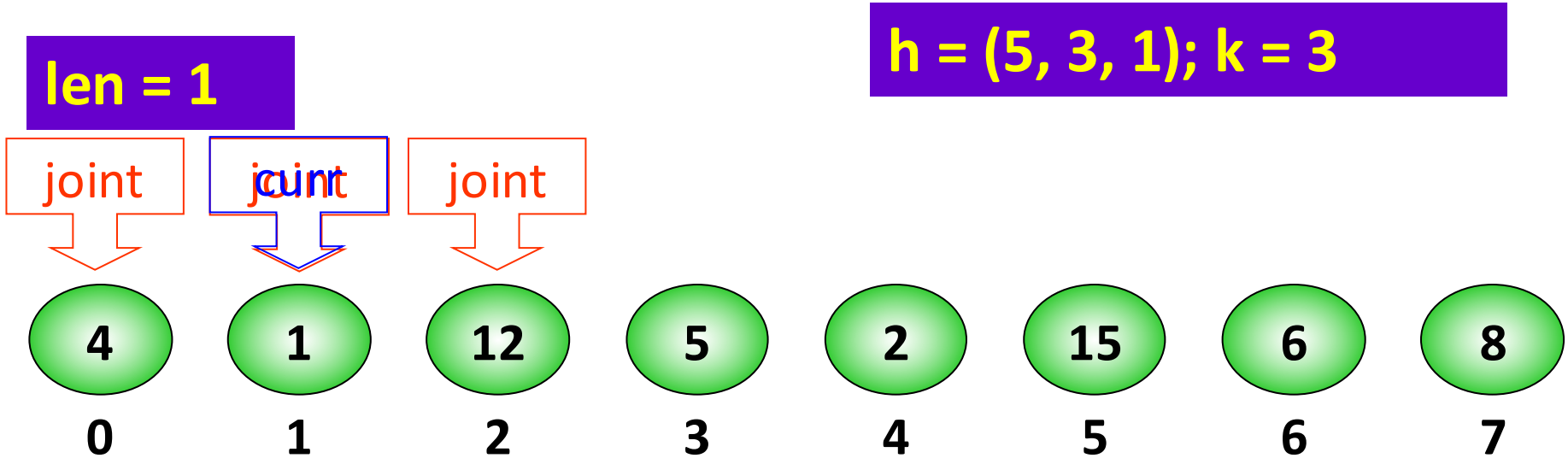
Shell Sort – Ví Dụ

len = 3

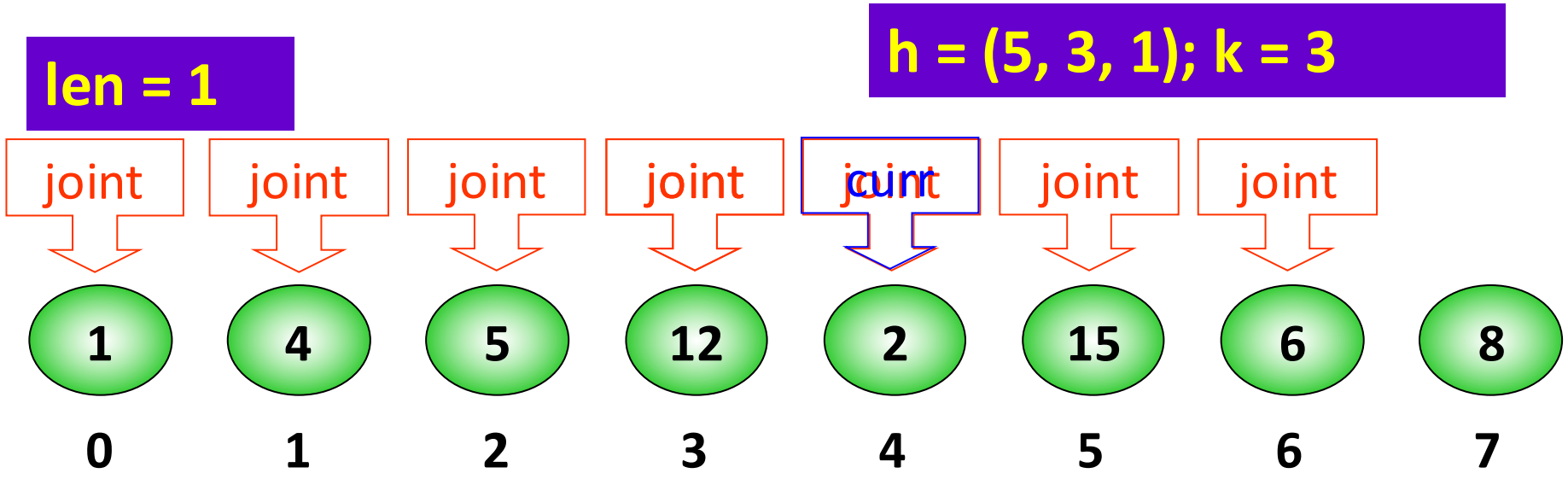
h = (5, 3, 1); k = 3



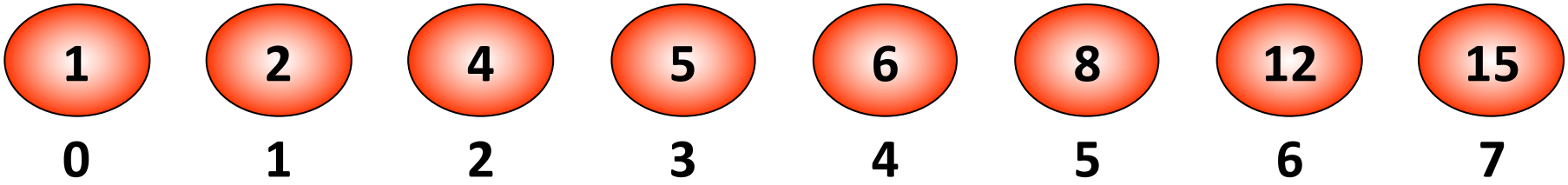
Shell Sort – Ví Dụ



Shell Sort – Ví Dụ



Shell Sort – Ví Dụ



Các Thuật Toán Sắp Xếp

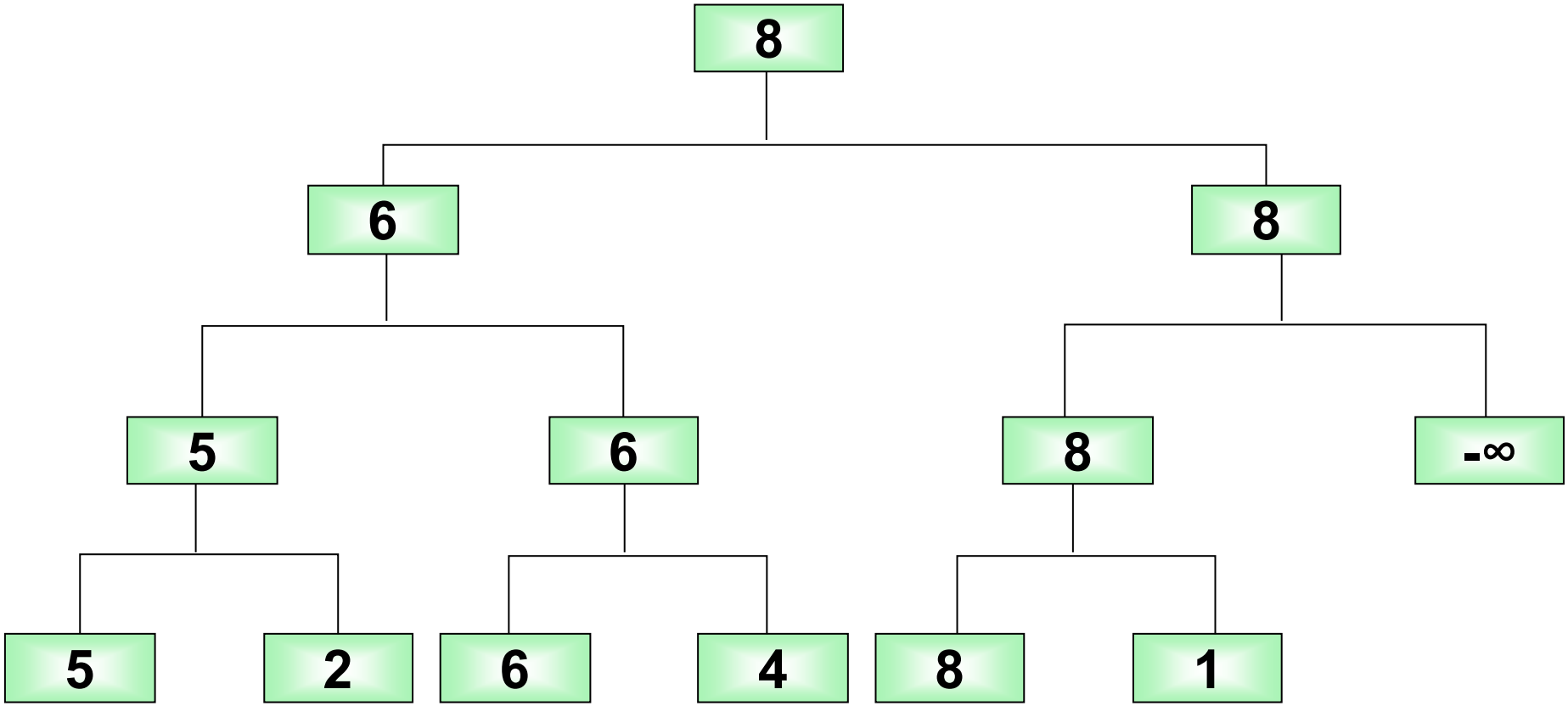
1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Shaker Sort
5. Chèn trực tiếp – Insertion Sort
6. Chèn nhị phân – Binary Insertion Sort
7. Shell Sort
8. **Heap Sort**
9. Quick Sort
10. Merge Sort
11. Radix Sort

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

Xét dãy số: 5 2 6 4 8 1



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 xãy ra trên những nhấn 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; Hoánvị (a_1 , a_r);
  - Bước 2: Loại bỏ phần tử lớn nhất ra khỏi heap: r = r-1;

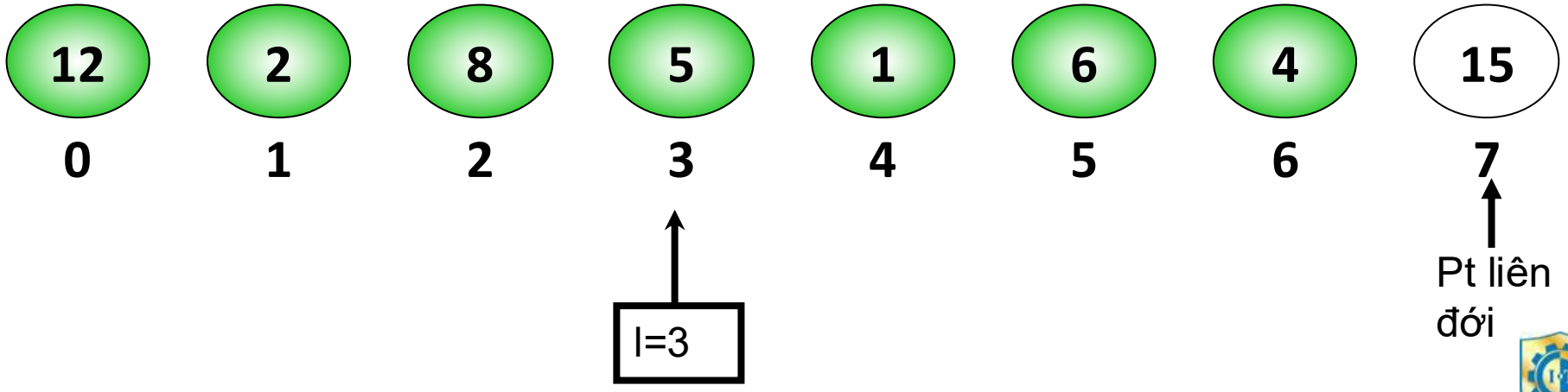
    Hiệu chỉnh phần còn lại của dãy từ a1 , a2 ... ar thành một heap.
  - Bước 3:

    Nếu r>1 (heap còn phần tử): Lặp lại Bước 2

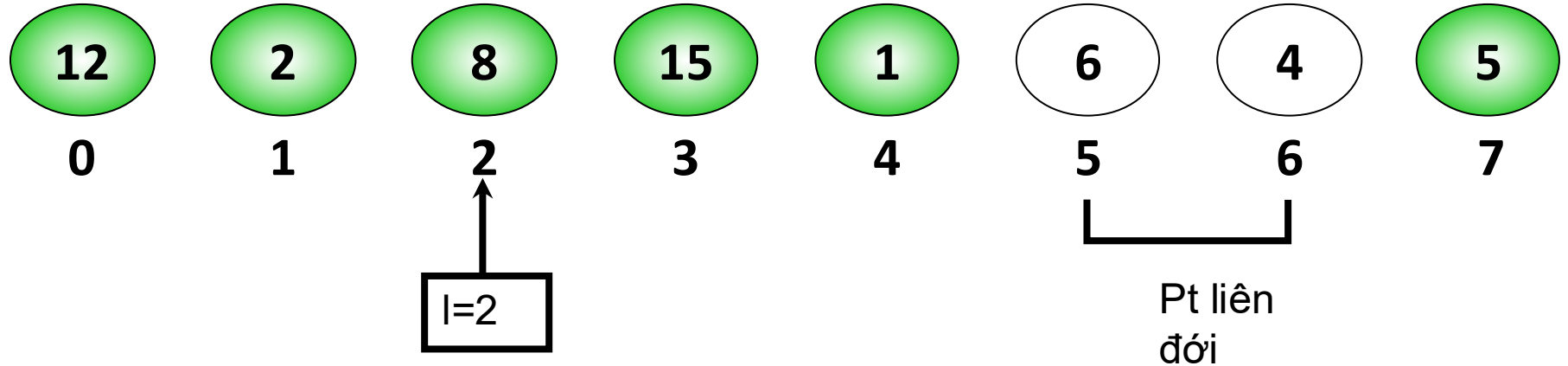
    Ngược lại : Dừng

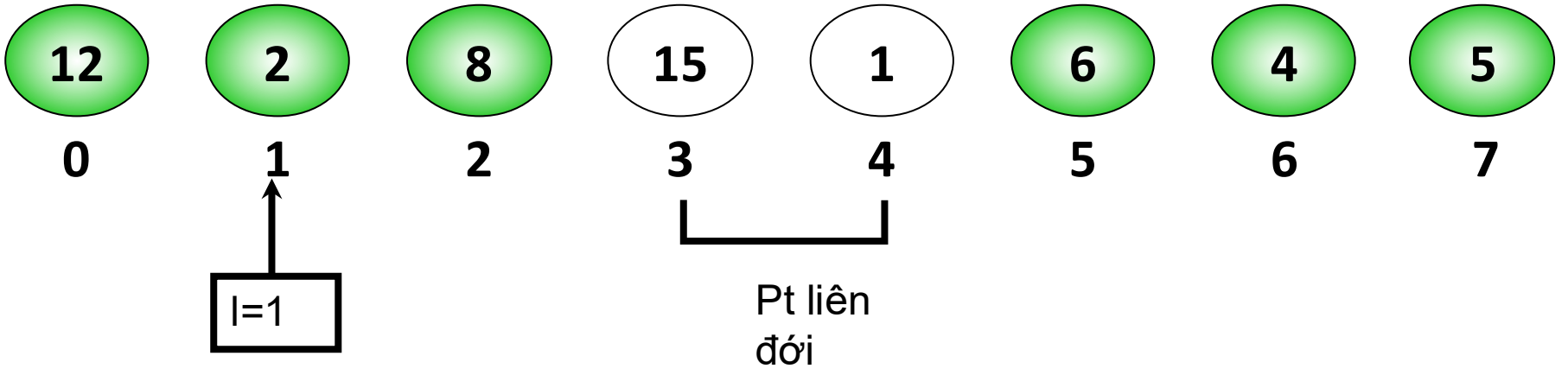
Minh Họa Thuật Toán

- **Heap:** Là một dãy các phần tử al , a2 ,... , ar thoả các quan hệ với mọi i ∈ [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

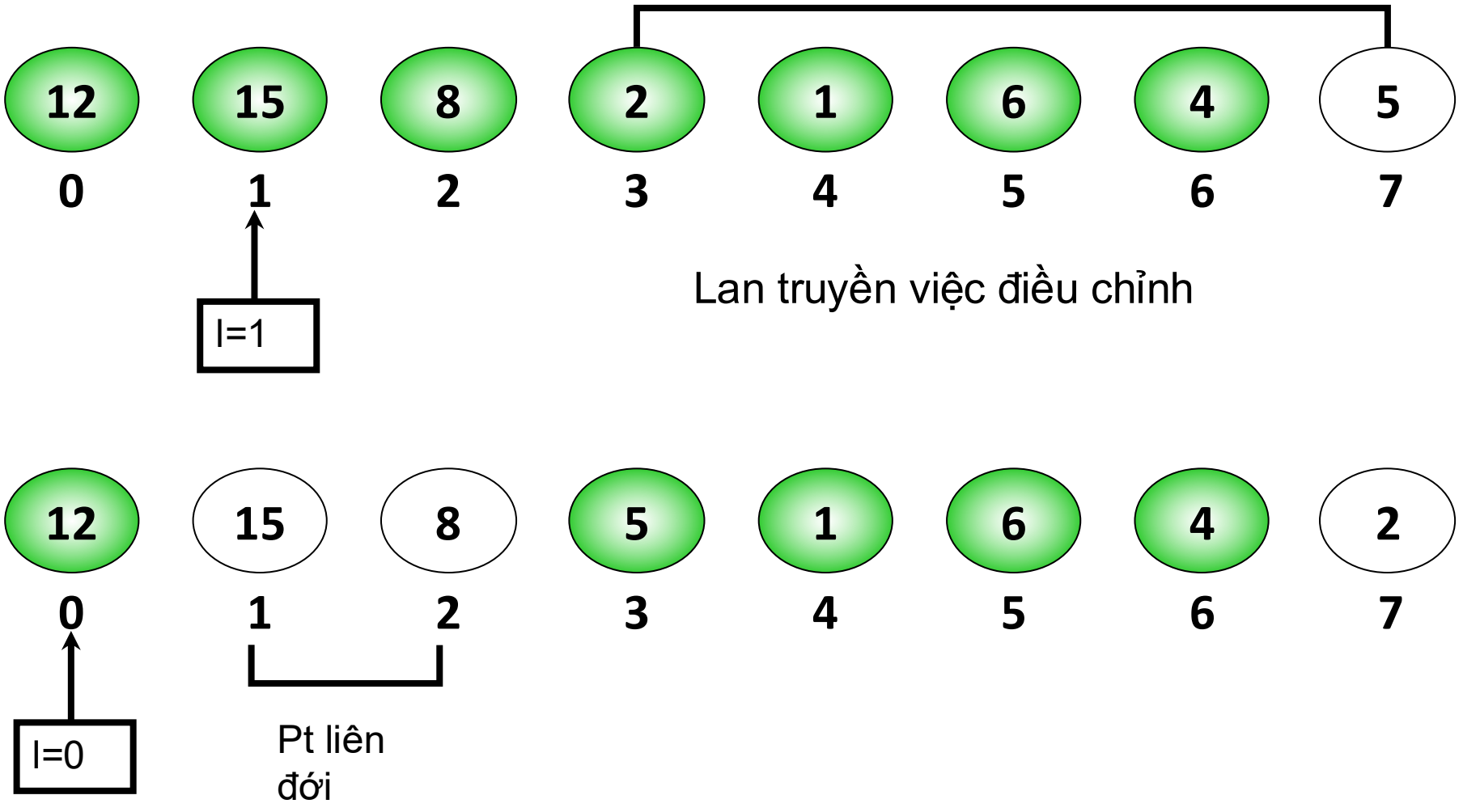


Minh Họa Thuật Toán

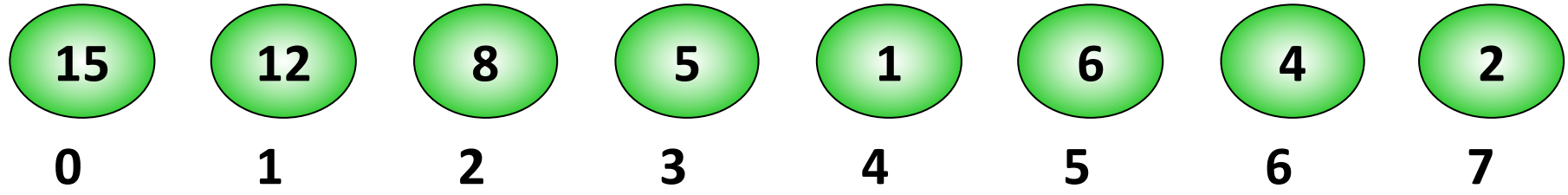




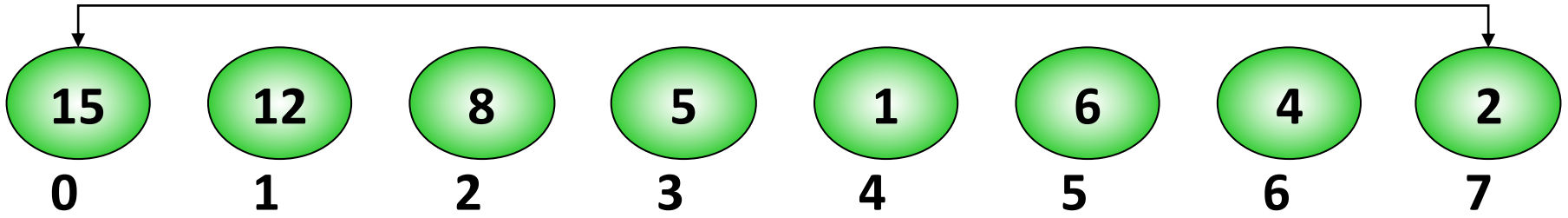
Minh Họa Thuật Toán

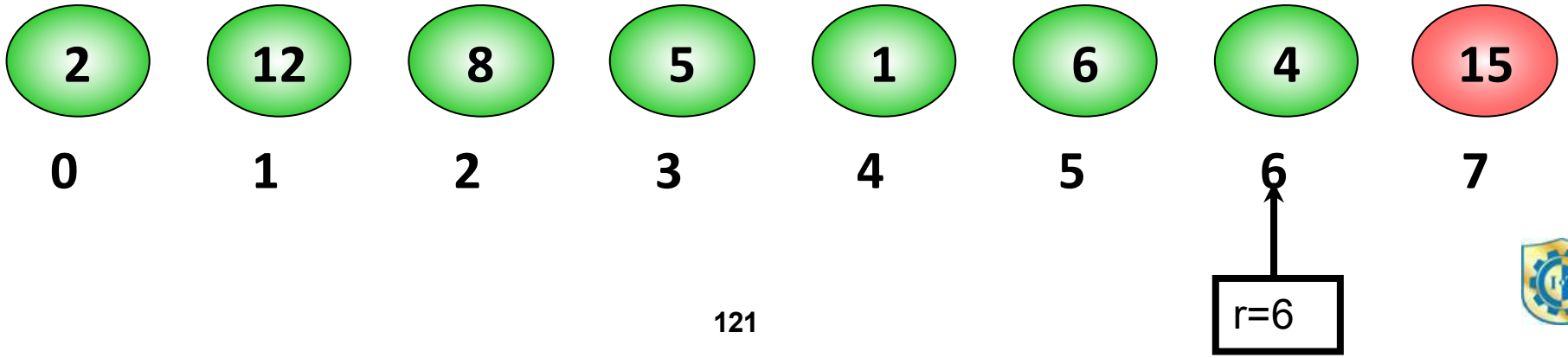


Minh Họa Thuật Toán



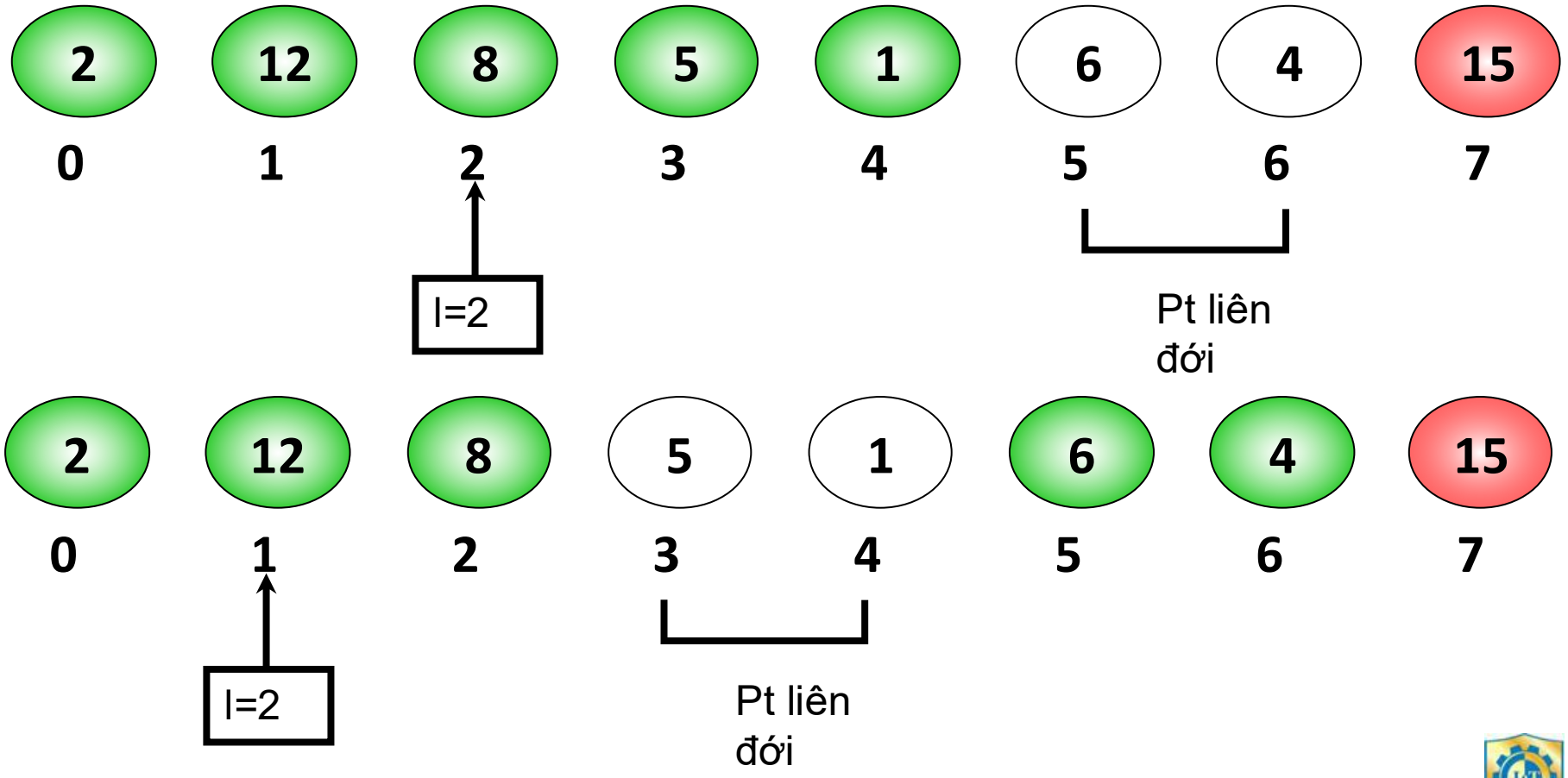
➢ Giai đoạn 2: Sắp xếp dãy số dựa trên Heap



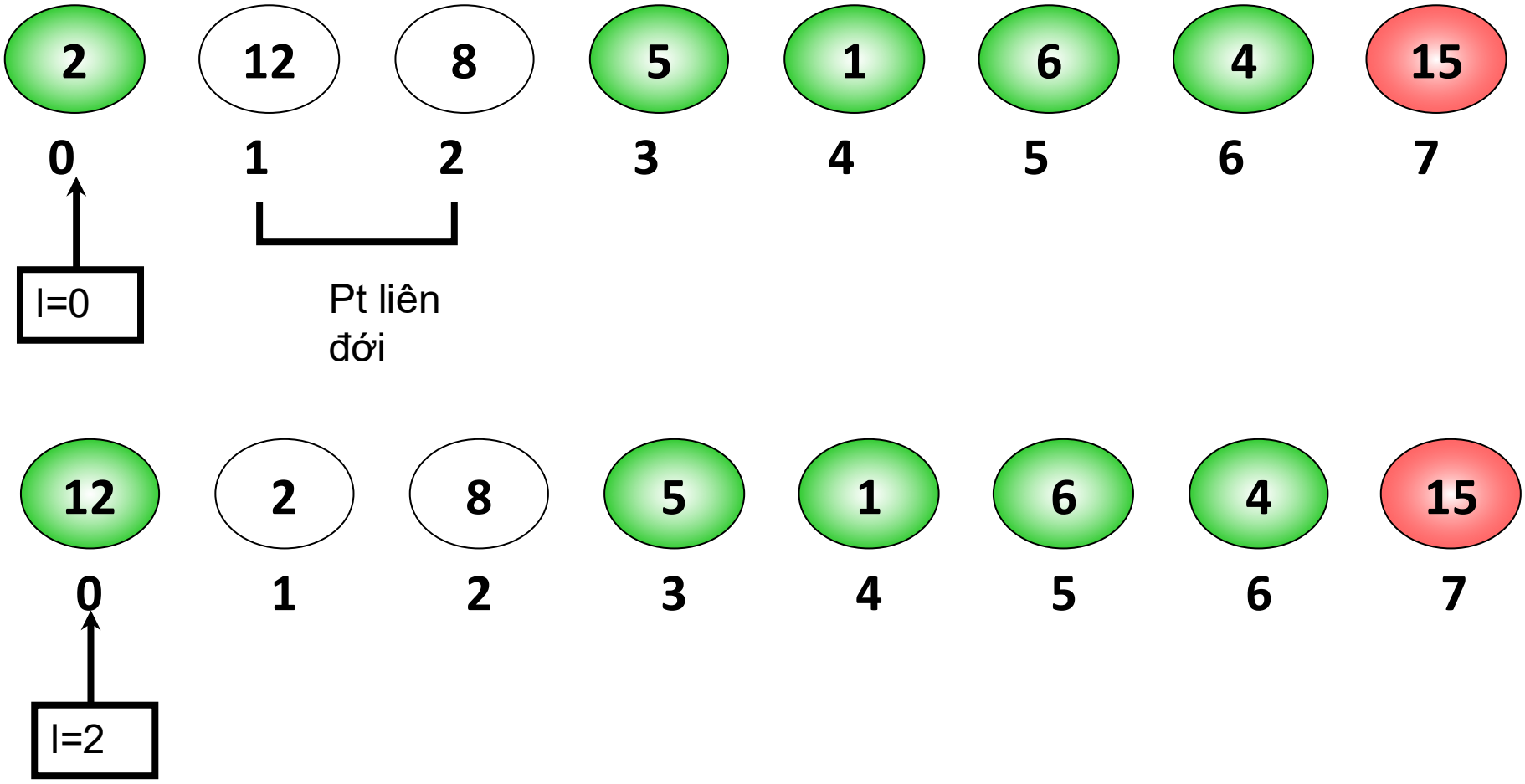


Minh Họa Thuật Toán

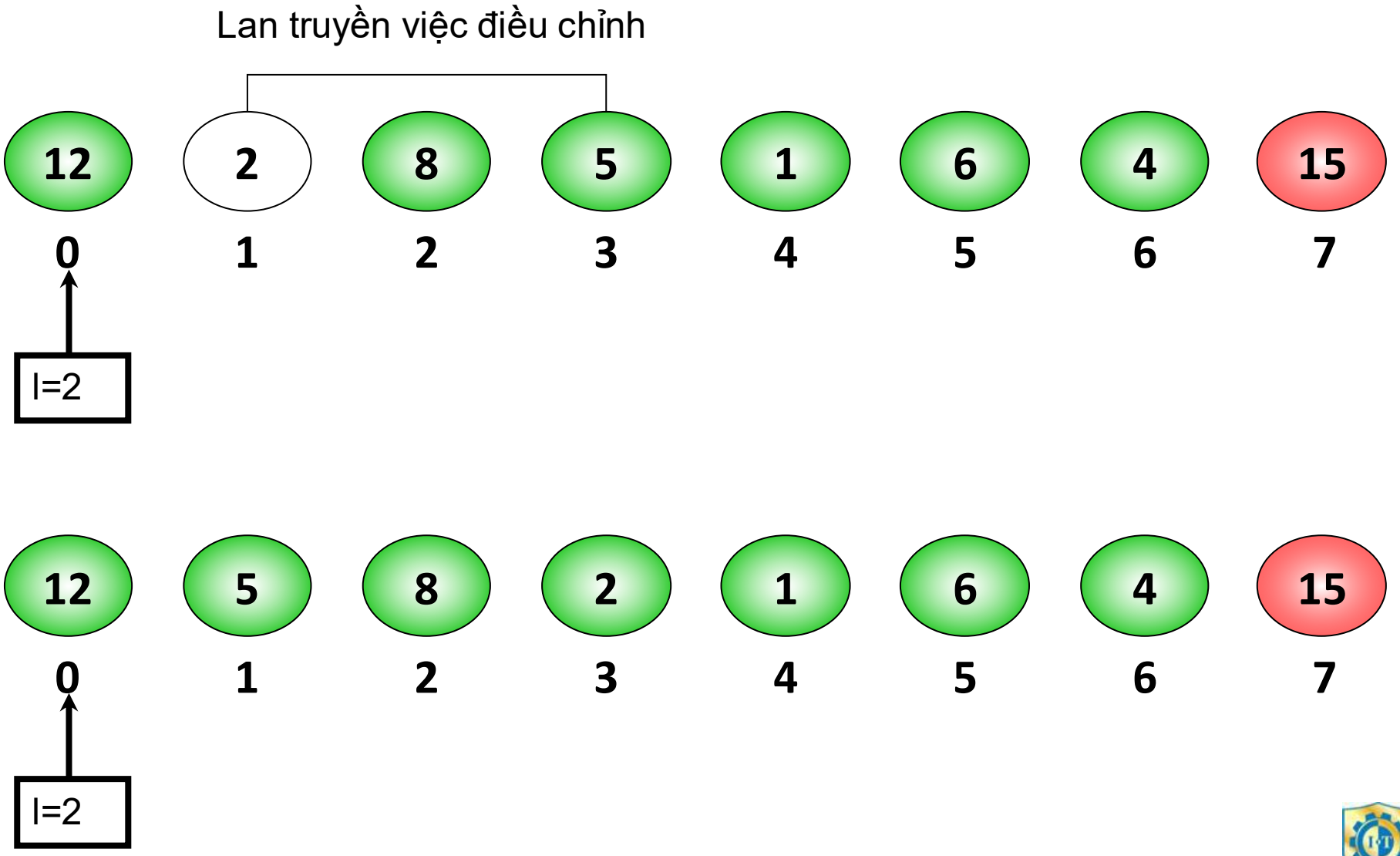
- Hiệu chỉnh Heap



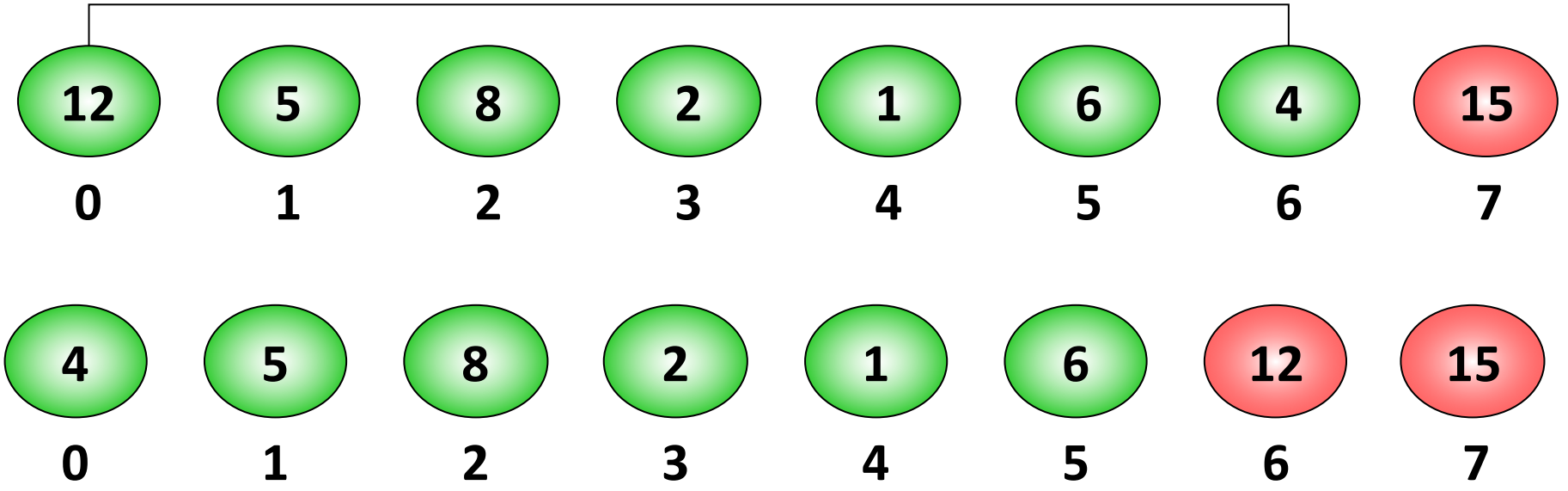
Minh Họa Thuật Toán



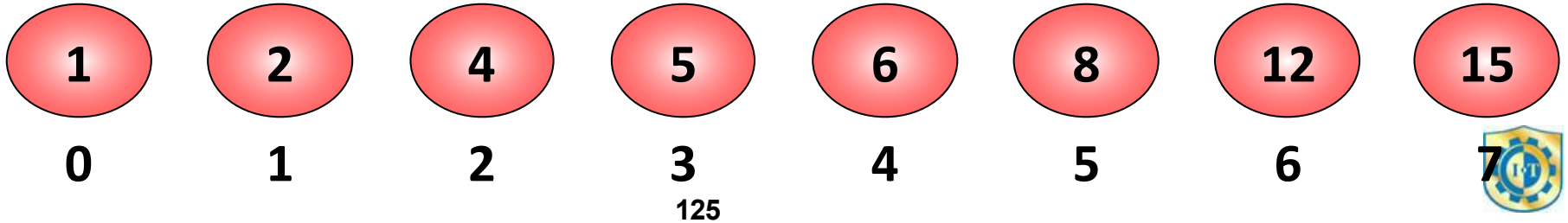
Minh Họa Thuật Toán



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,..,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]) //tim phan tu lon nhat a[j] va a[j+1]
```


Cài Đặt Thuật Toán

```
        j++; //luu chi so cua phan tu nho nhat trong hai phan tu
        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 $a_0,..a_{n-1}$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] la 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

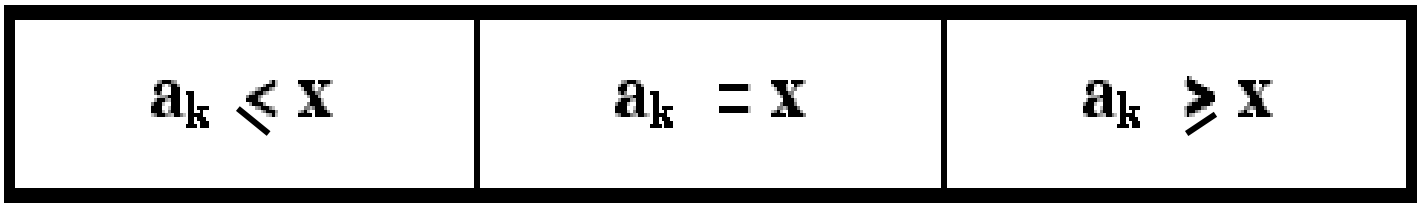
- Ý tưởng:

- Giải thuật QuickSort sắp xếp dãy $a_1, a_2 ..., a_N$ 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**

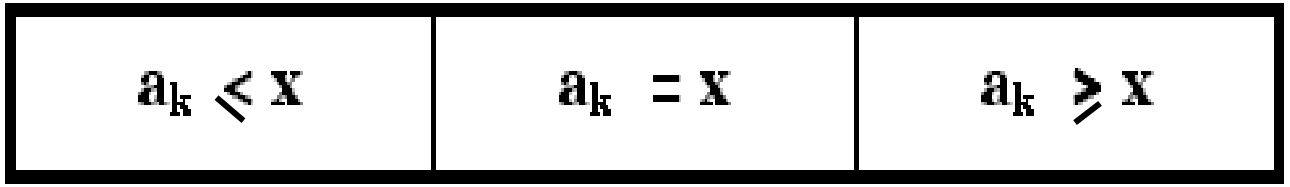
  với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

Quick Sort - Ý Tưởng

- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:
  - 1. ak $\leq$ x , với k = 1 .. j
  - 2. ak = x , với k = j+1 .. i-1
  - 3. ak $\geq$ x , với k = i..N



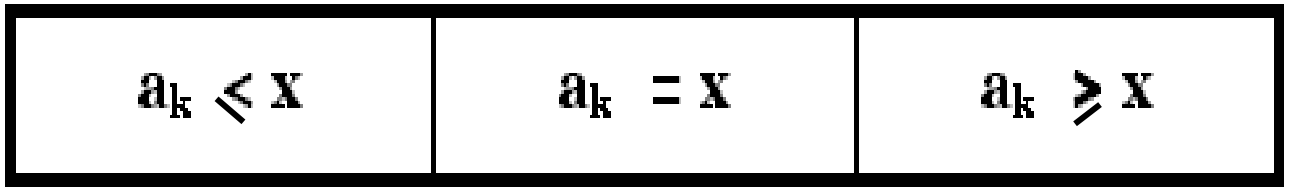
Quick Sort – Ý Tưởng



- 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



- 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 left ≥ right //dãy có ít hơn 2 phần tử

  Kết thúc; //dãy đã được sắp xếp

- Bước 2: Phân hoạch dãy a_{left} … a_{right} thành các đoạn:

  a_{left}.. a_j, a_{j+1}.. a_{i-1}, a_i.. a_{right}

  Đoạn 1 $\leq x$

  Đoạn 2: a_{j+1}.. a_{i-1} $= x$

  Đoạn 3: a_i.. a_{right} $\geq x$

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

Giải Thuật Quick Sort

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

  x = a[k]; i = l; j = r;

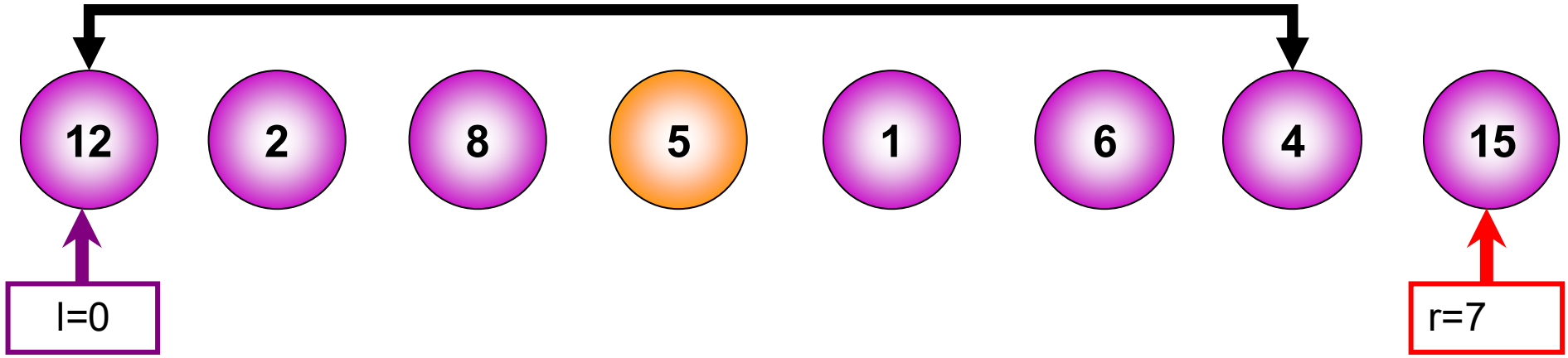
- Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :
  - Bước 2a : Trong khi (a[i]<x) i++;
  - Bước 2b : Trong khi (a[j]>x) j--;
  - Bước 2c : Nếu i< j Đoicho(a[i],a[j]);
- Bước 3 : Nếu i < j: Lặp lại Bước 2.
  Ngược lại: Dừng

Quick Sort – Ví Dụ

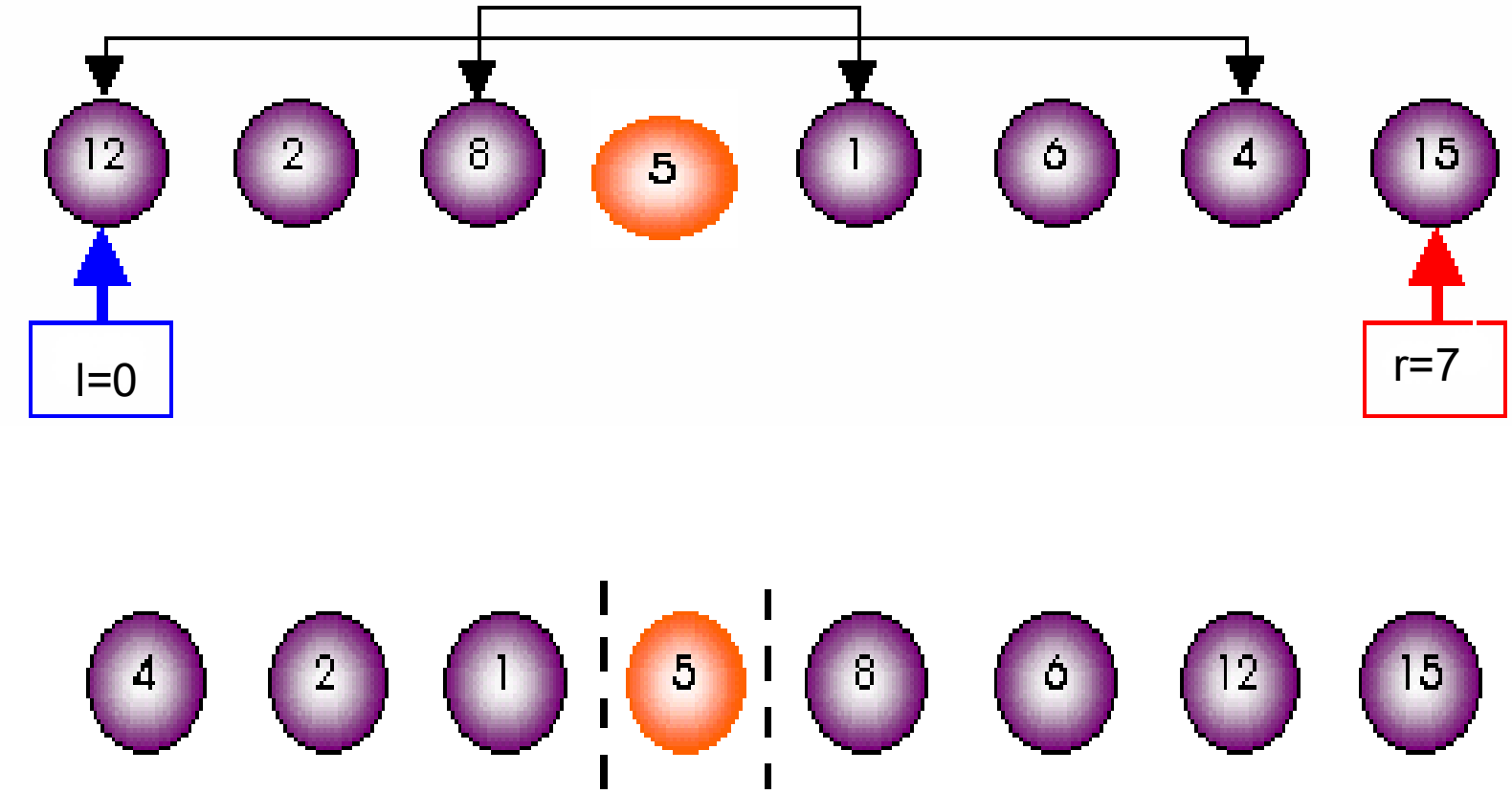
- Cho dãy số a:

12	2	8	5
1	6	4	15

Phân hoạch đoạn l =0, r = 7: x = a[3] = 5



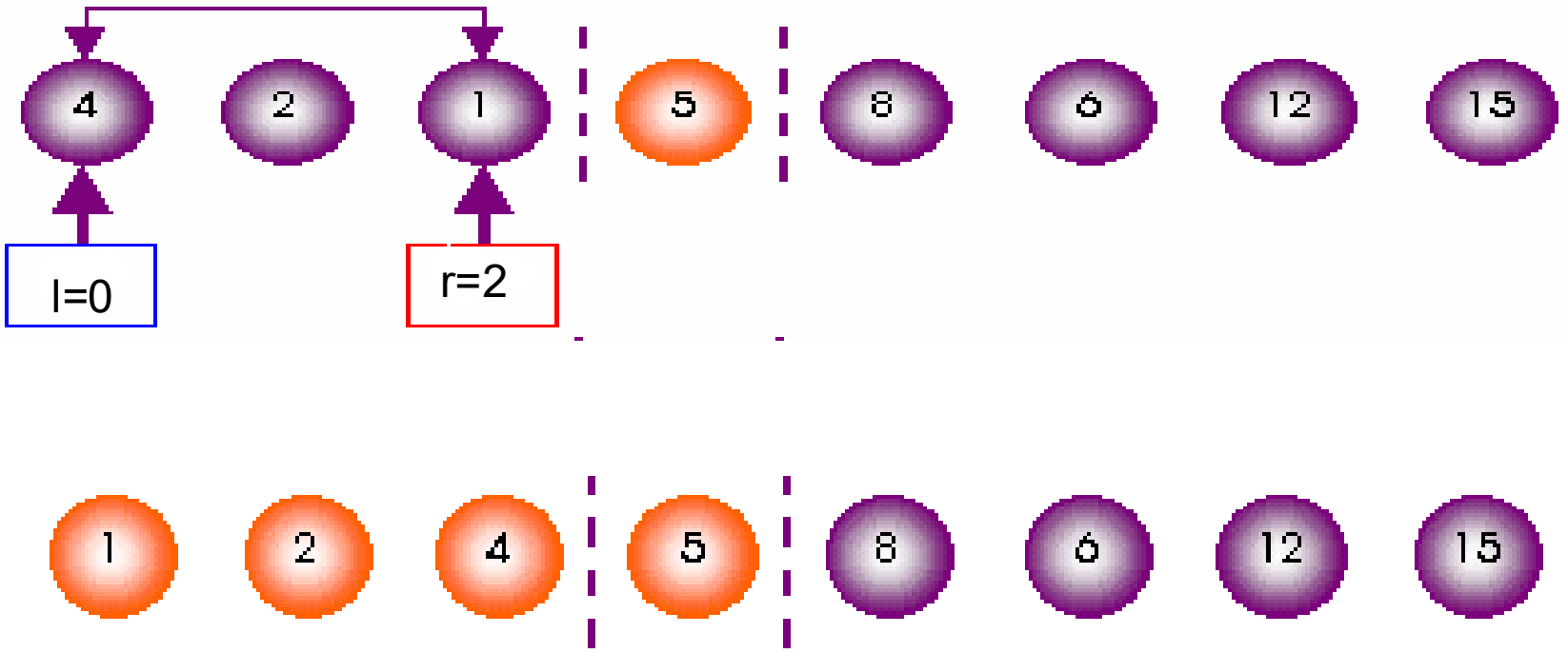
Quick Sort – Ví Dụ



Quick Sort – Ví Dụ

- Phân hoạch đoạn l =0, r = 2:

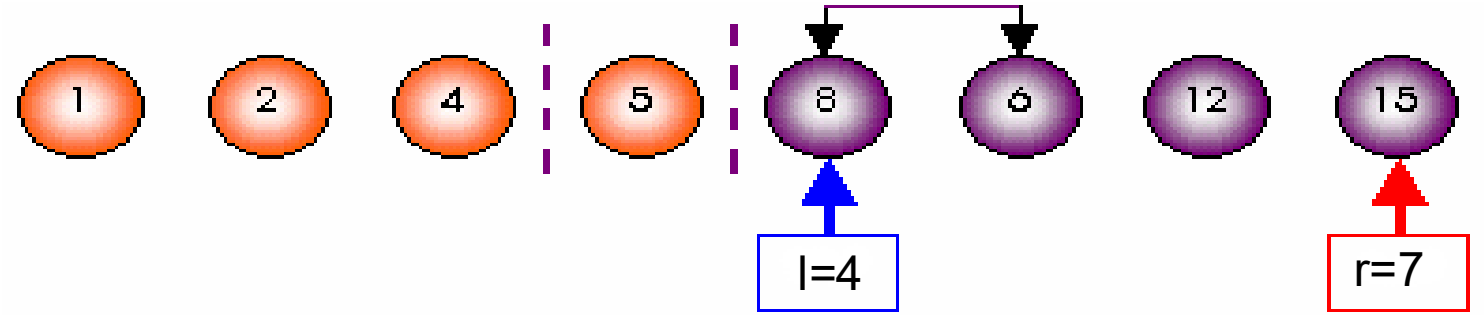
x = a[2] = 2

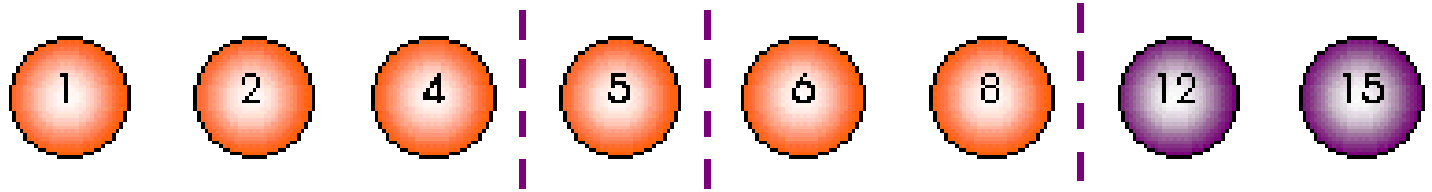


Quick Sort – Ví Dụ

- Phân hoạch đoạn l = 4, r = 7:

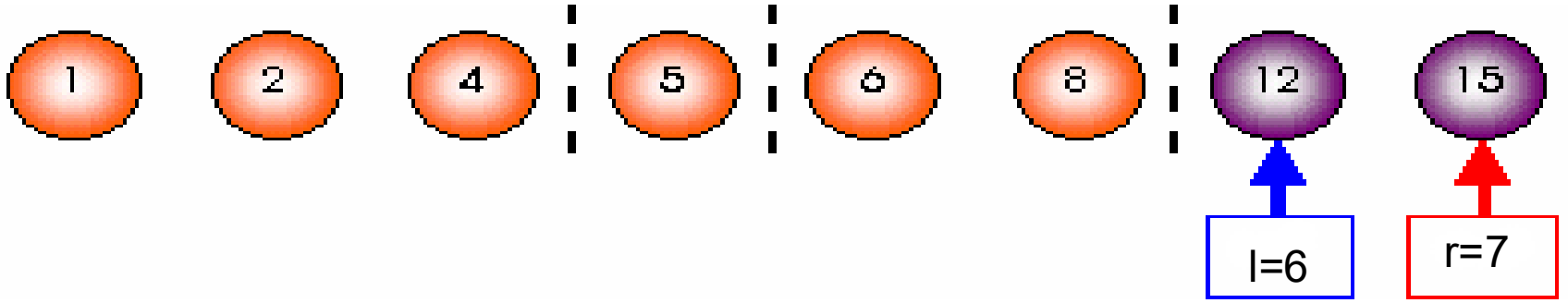
x = a[5] = 6

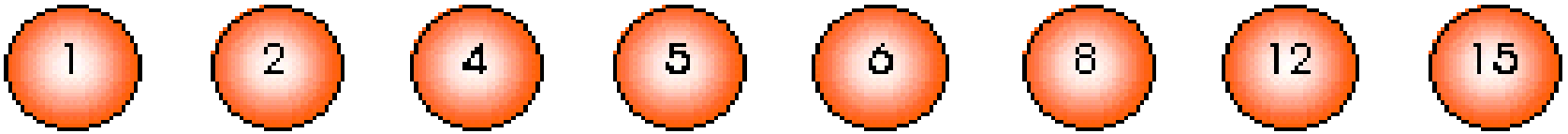




- Phân hoạch đoạn l = 6, r = 7:

x = a[6] = 6



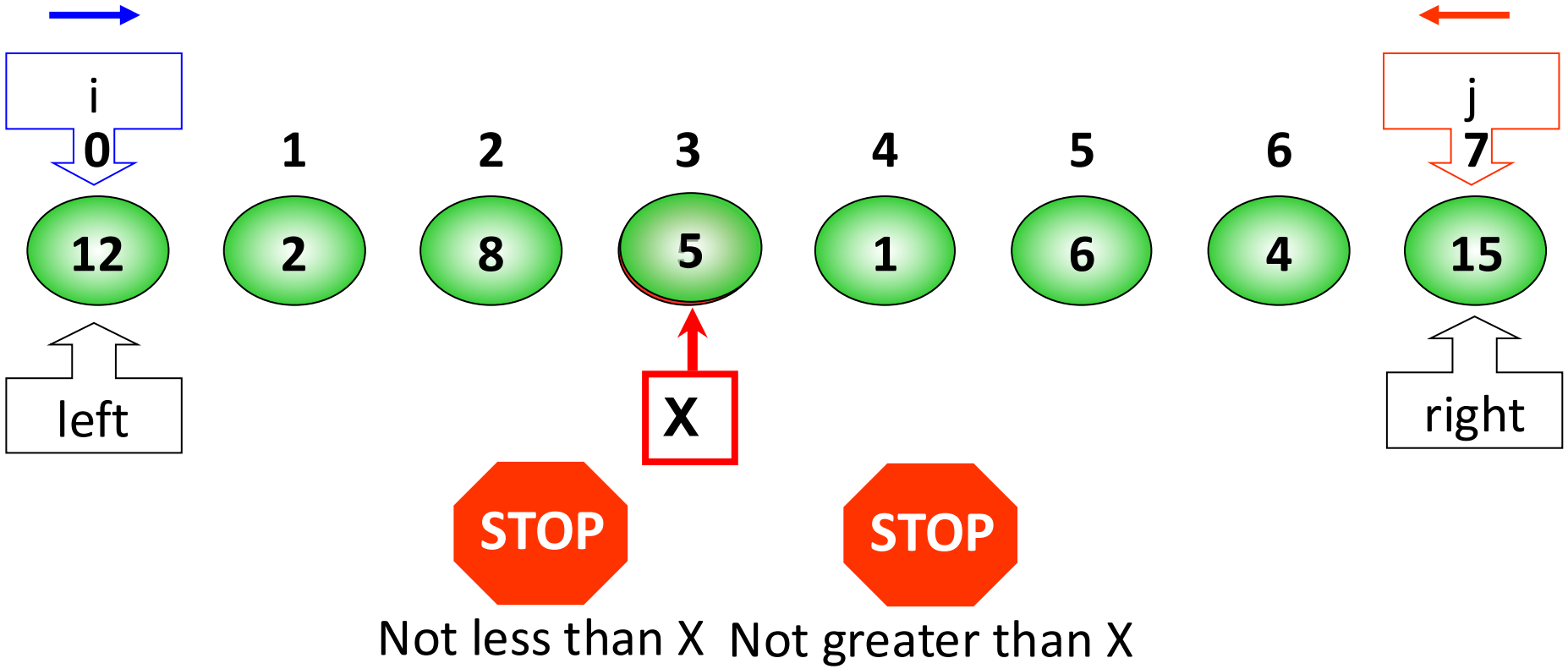


Quick Sort

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

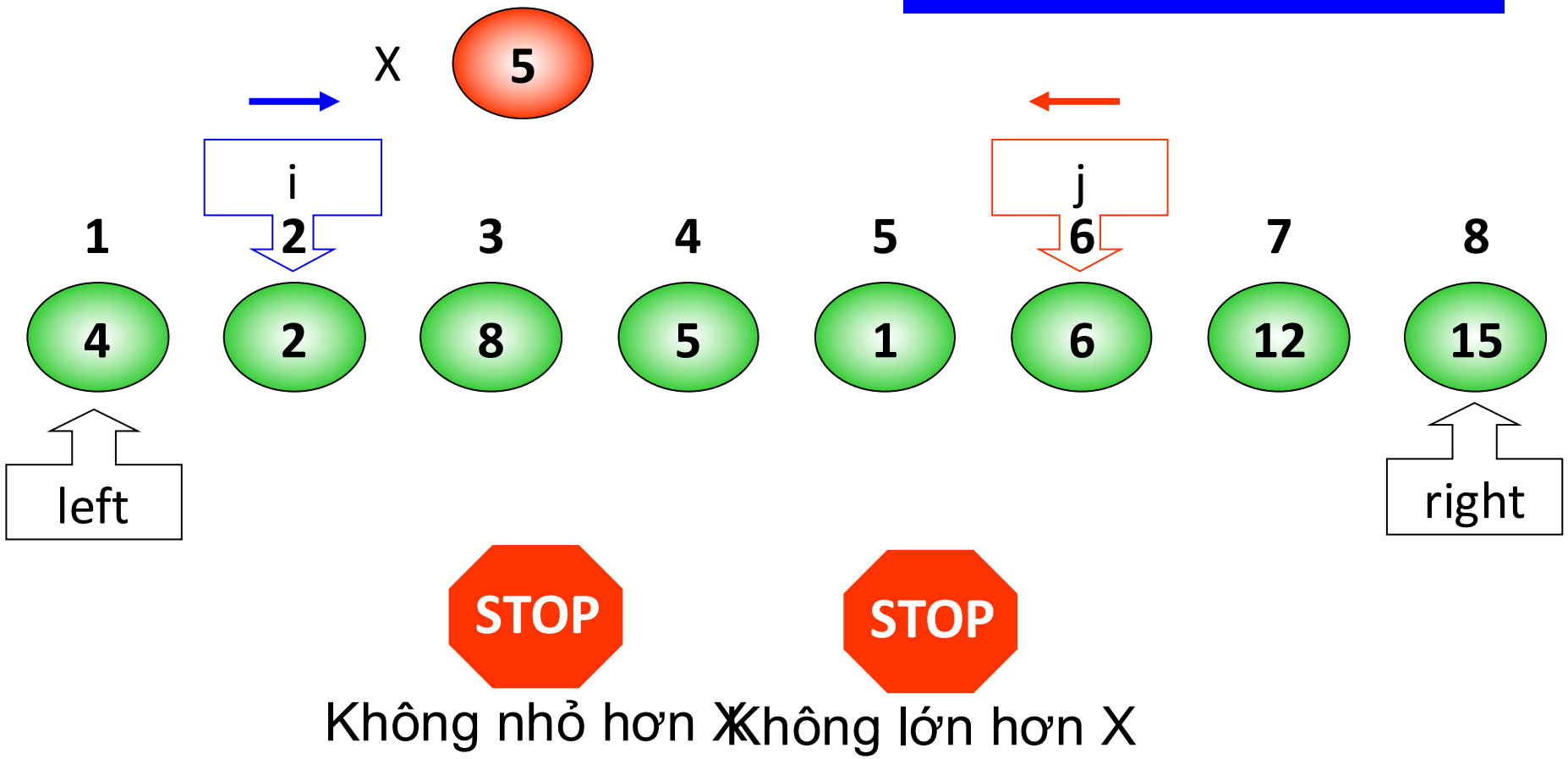

Quick Sort – Ví Dụ

Phaân hoaïch daõy

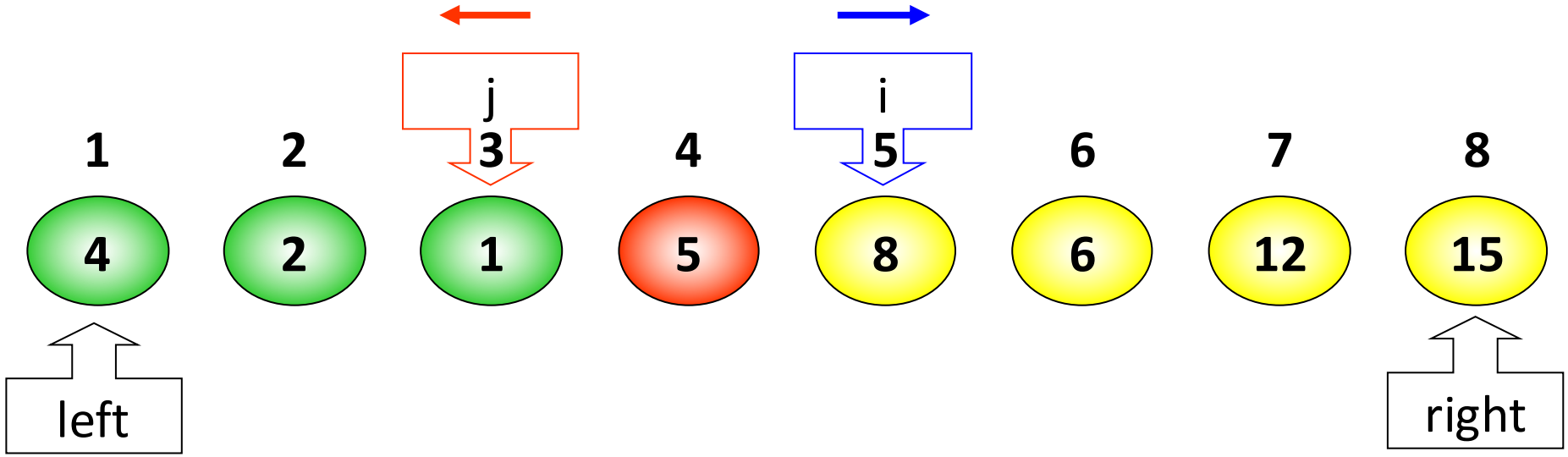


Quick Sort – Ví Dụ

Phaân hoaïch daõy

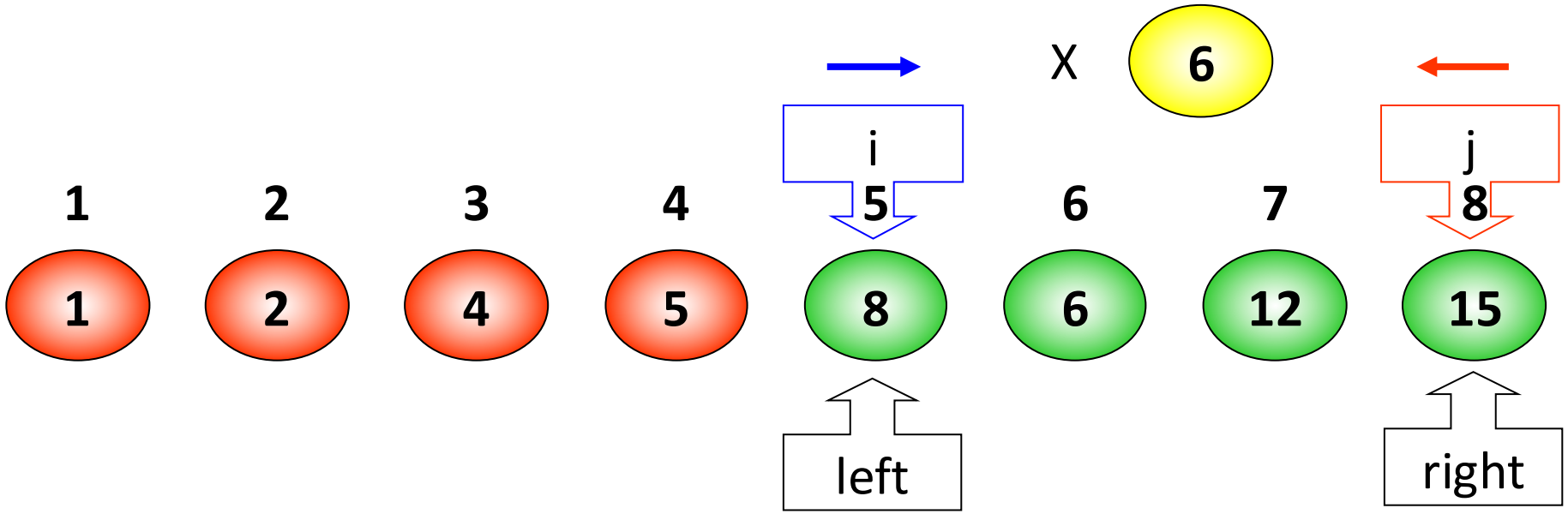


Quick Sort – Ví Dụ



Quick Sort – Ví Dụ

Phaân hoaïch daõy



Saép xeáp ñoaïn 3

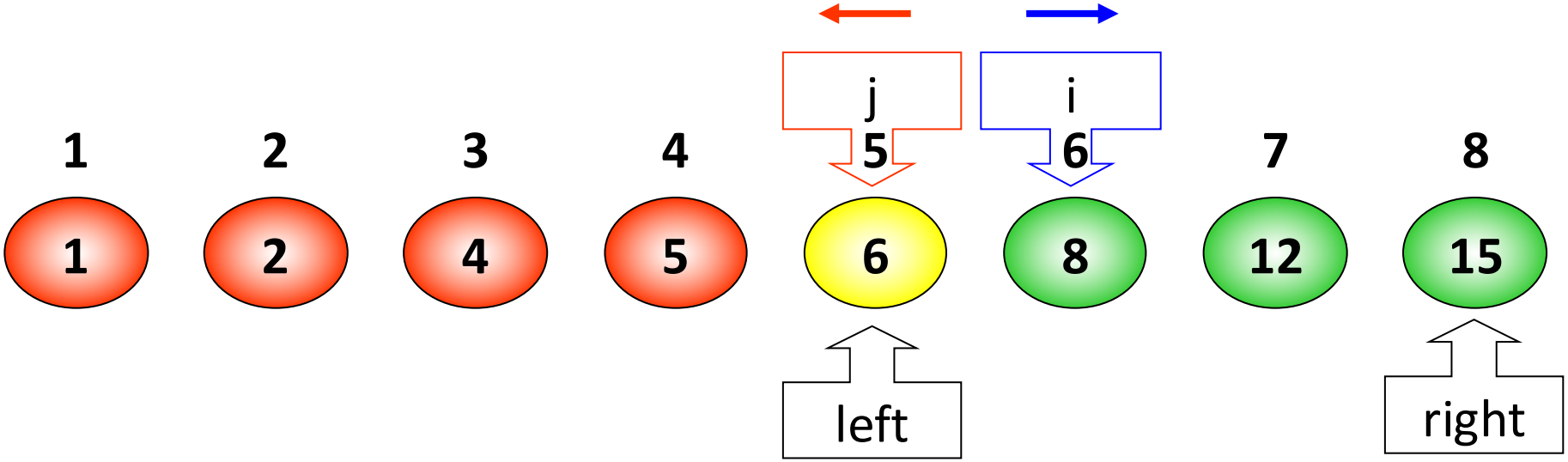
STOP

STOP

Không nhỏ hơn X

Không lớn hơn X

Quick Sort – Ví Dụ



Saép xeáp ñoaïn 3

Độ Phức Tạp Của Quick Sort

Trường hợp	Độ phức tạp
Tốt nhất	$n*log(n)$
Trung bình	$n*log(n)$
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

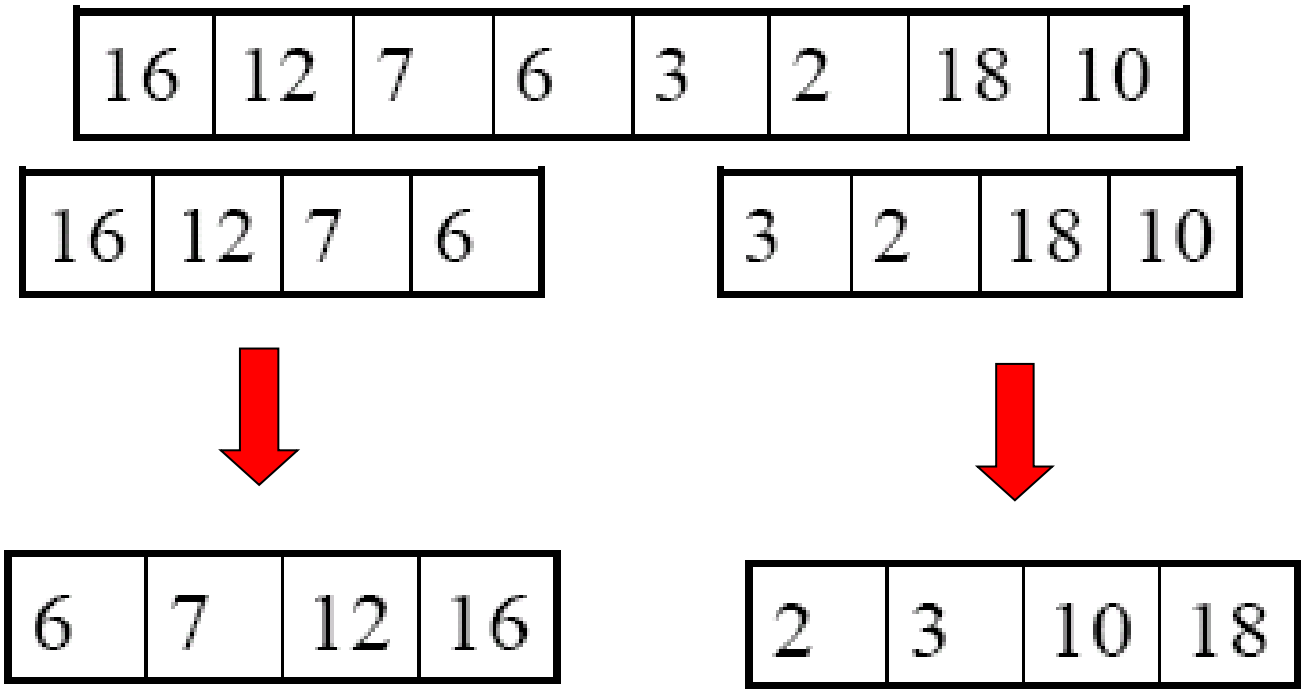
Merge Sort – Ý Tưởng

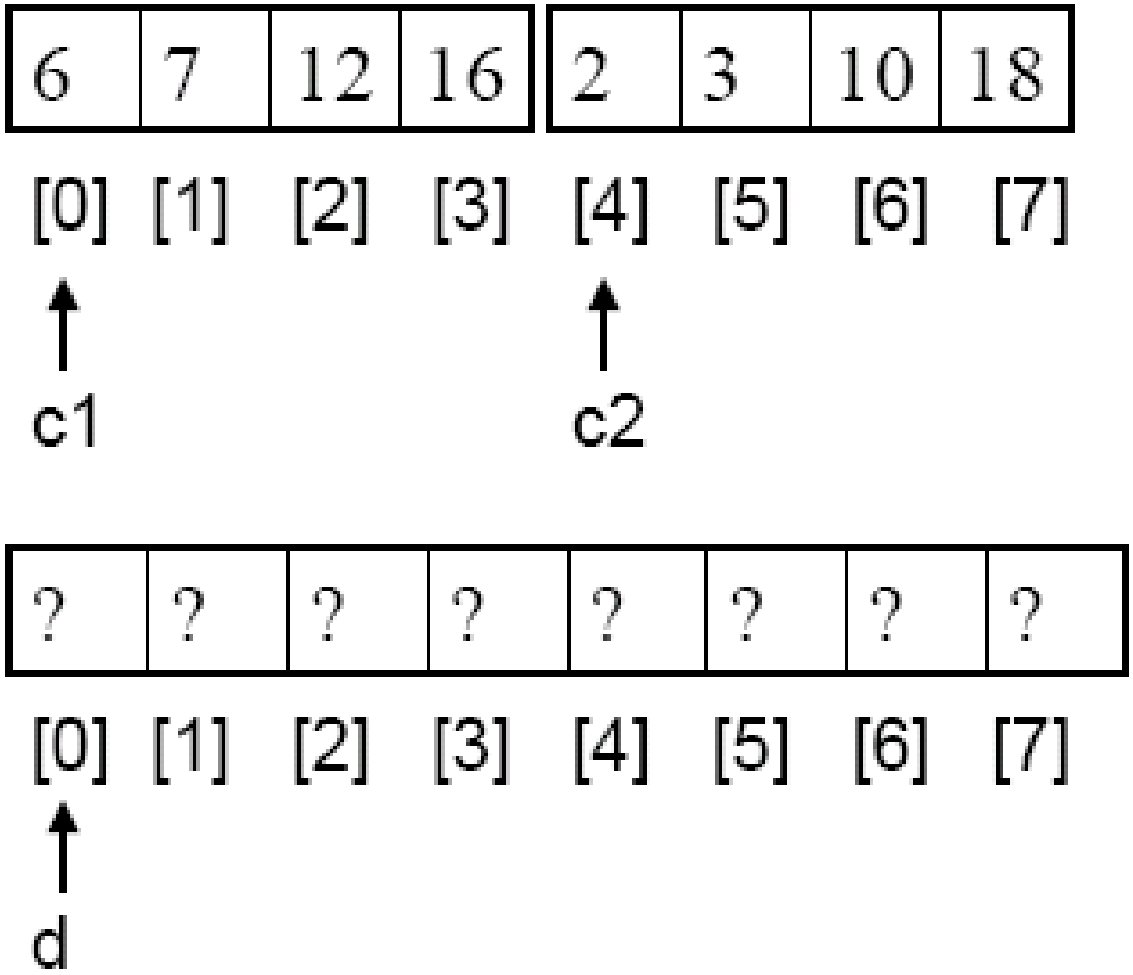
- Giải thuật Merge sort sắp xếp dãy **a_1, a_2, ..., an** dựa trên nhận xét sau:
  - Mỗi dãy **a_1, a_2, ..., an** bất kỳ là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự.
    - Ví dụ: dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy con không giảm (12); (2, 8); (5); (1, 6); (4, 15).
  - Dãy đã có thứ tự coi như có 1 dãy con.

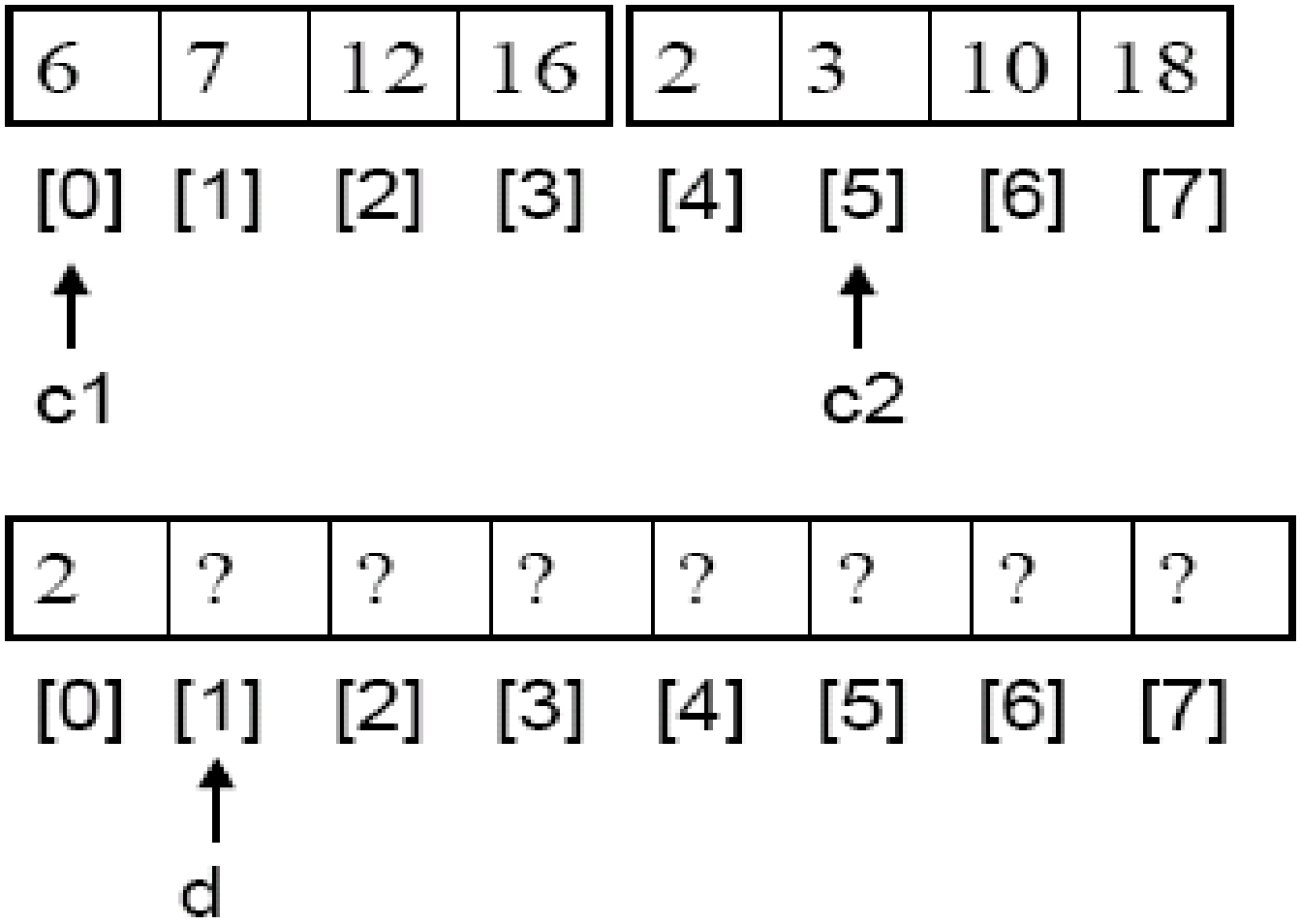
➔ Hướng tiếp cận: tìm cách làm giảm số dãy con không giảm của dãy ban đầu.

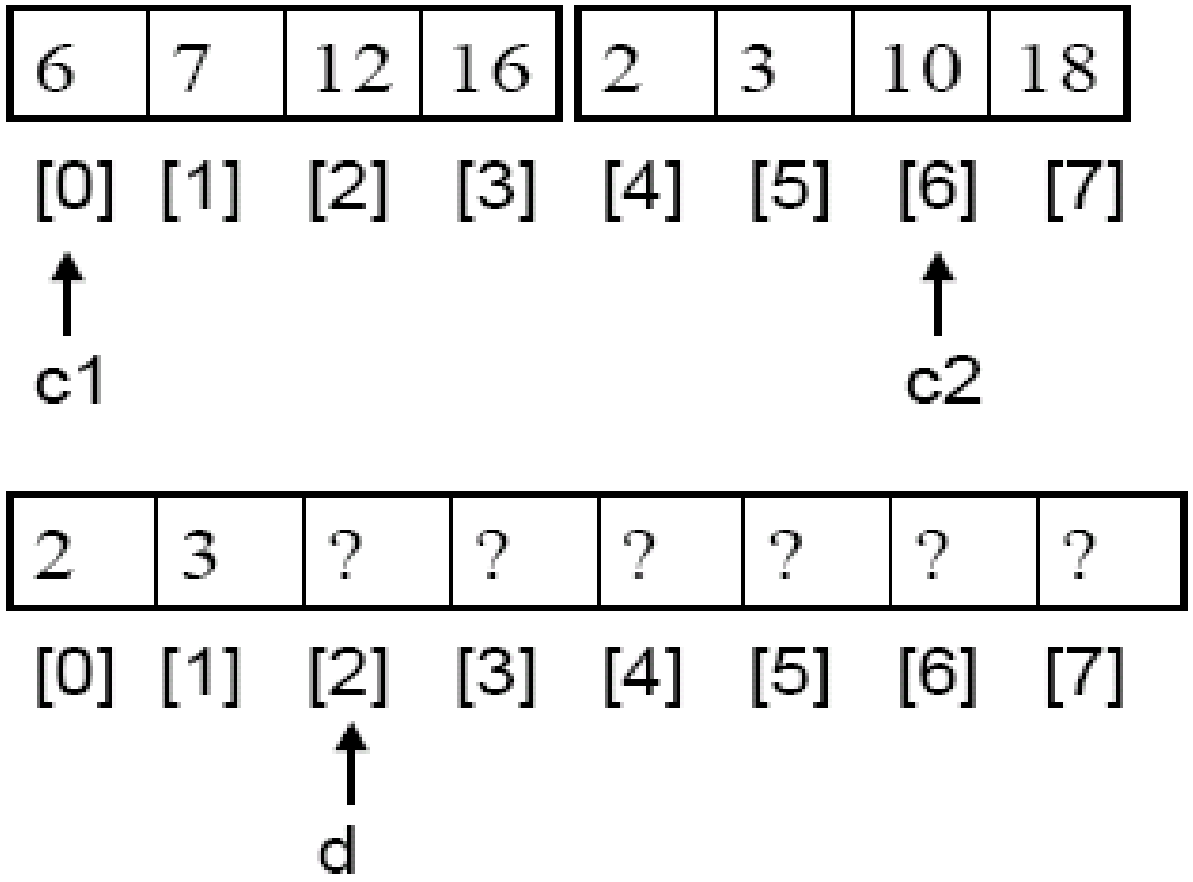
Sắp Xếp Trộn - Merge Sort

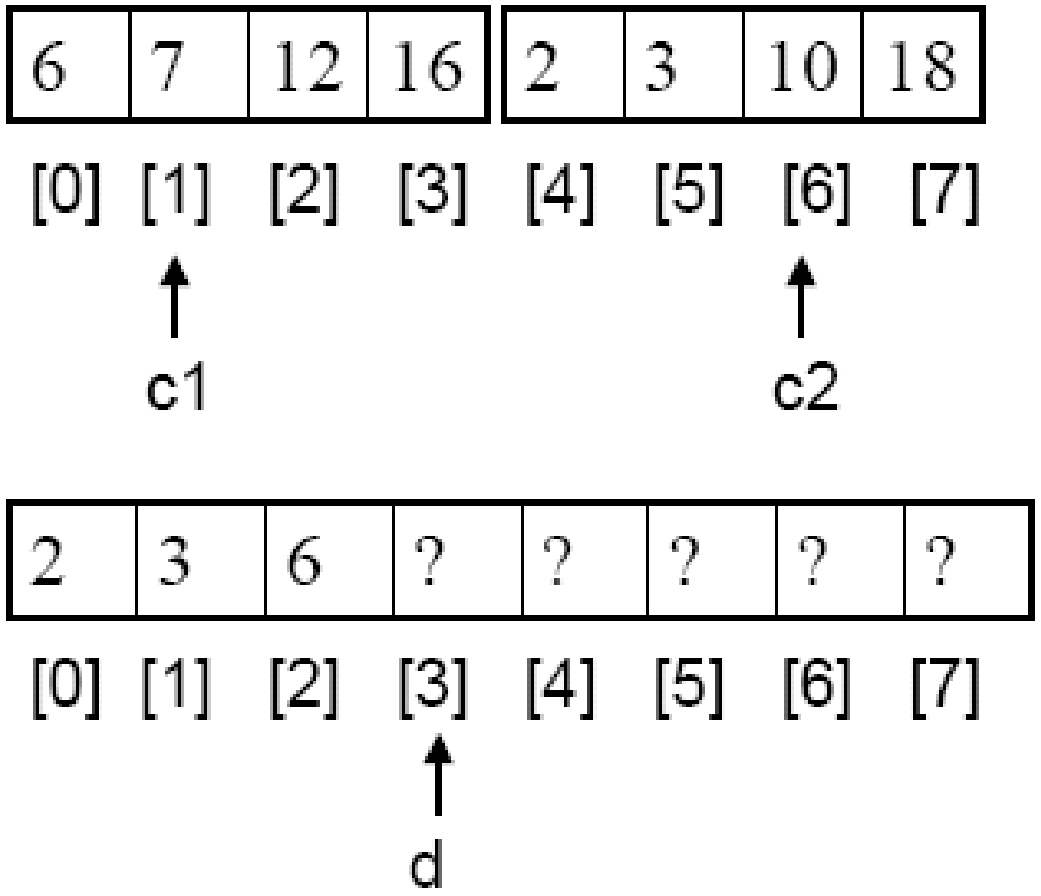
- Mảng A chia làm 02 phần bằng nhau.
- Sắp xếp 02 phần
- Trộn 02 nửa lại


16 12 7 6 3 2 18 10
16 12 7 6
3 2 18 10
6 7 12 16
2 3 10 18

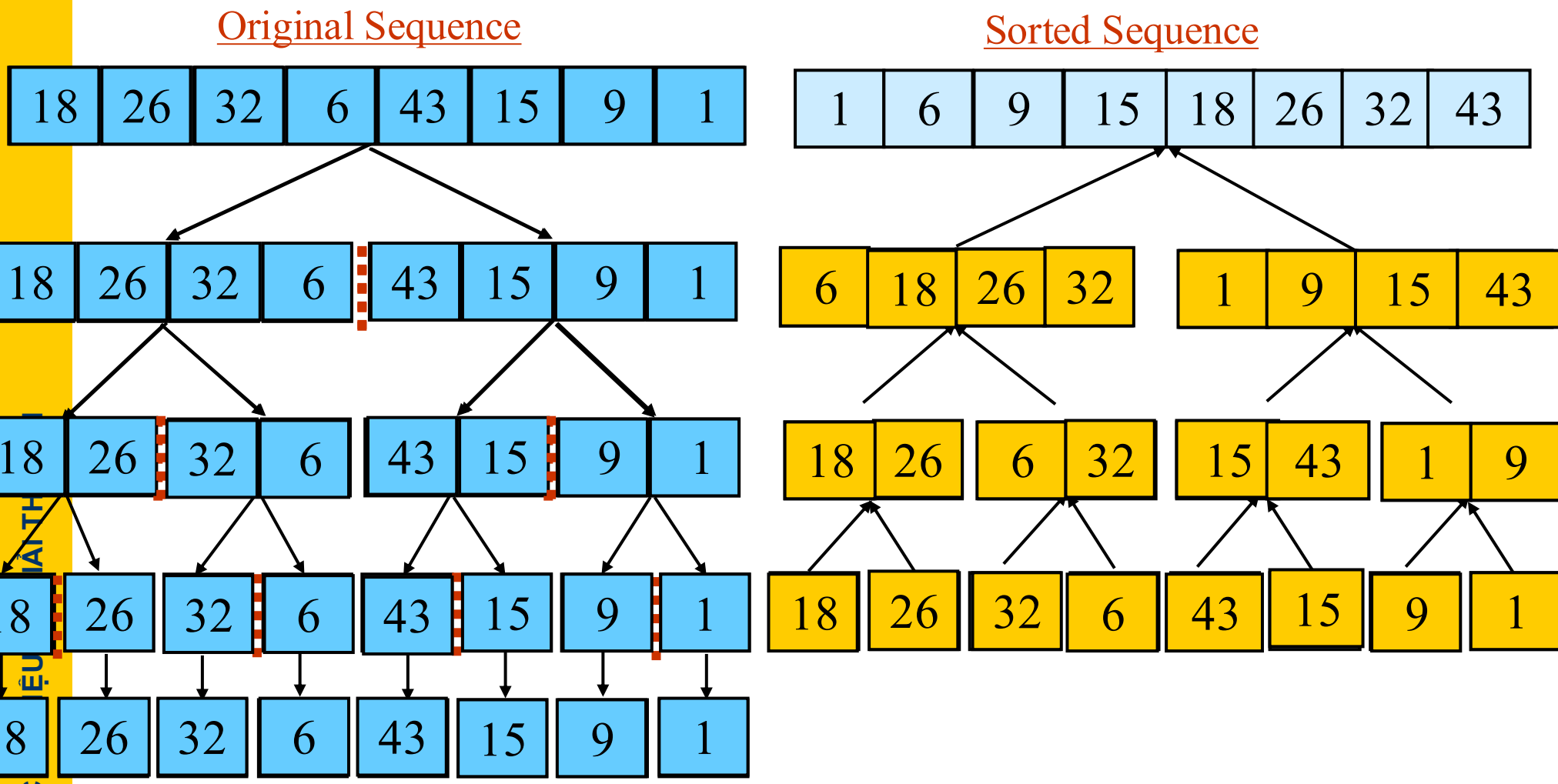

6
7
12
16
2
3
10
18
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
c1
c2
?
?
?
?
?
?
?
?
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
d


6 7 12 16
2 3 10 18
[0] [1] [2] [3] [4] [5] [6] [7]
c1
c2
2 ? ? ? ? ? ? ?
[0] [1] [2] [3] [4] [5] [6] [7]
d


6
7
12
16
2
3
10
18
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
c1
c2
2
3
?
?
?
?
?
?
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
d


6
7
12
16
2
3
10
18
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
c1
c2
2
3
6
?
?
?
?
?
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
d

Merge Sort – Ví Dụ



Merge Sort

```
void MergeSort (Day &d, p, r)
{
    if p < r
    {
        q = (p+r)/2
        MergeSort (A, p, q)
        MergeSort (A, q+1, r)
        Merge (A, p, q, r);
    }
}
```


Merge Sort

- Các dãy con tăng dần sẽ được tách ra 2 dãy phụ theo nguyên tắc **phân phối đều luân phiên**.
- Trộn từng cặp dãy con của hai dãy phụ thành một dãy con của dãy ban đầu → dãy mới có số lượng dãy con giảm đi so với dãy ban đầu.

Merge Sort

- Bước 1 : k = 1; *// dãy con có 1 phần tử là dãy không giảm*
- Bước 2 : Lặp trong khi (k < N) *// dãy còn hơn 1 dãy con*
  - Bước 21: Phân phối đều luân phiên dãy a_1, a_2, …, an thành 2 dãy b, c theo từng nhóm k phần tử liên tiếp nhau.

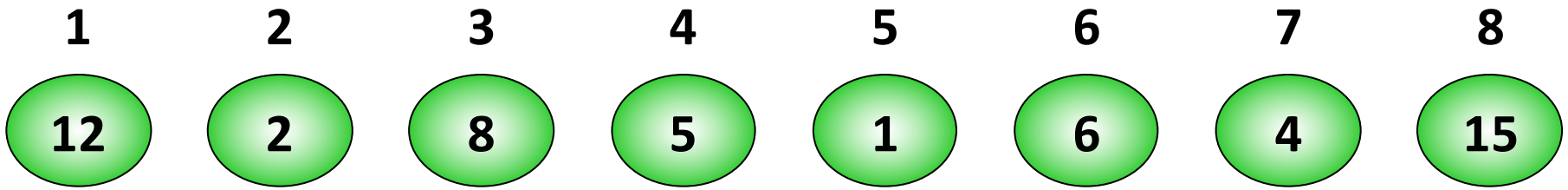
    //b = a_1, …, ak, a_{2k+1}, …, a_{3k}, …

    //c = ak_{+1}, …, a_{2k}, a_{3k+1}, …, a_{4k}, …
  - Bước 22: Trộn từng cặp dãy con gồm k phần tử của 2 dãy b, c vào a.
  - Bước 23: k = k*2;

Merge Sort – Ví Dụ

p = 1

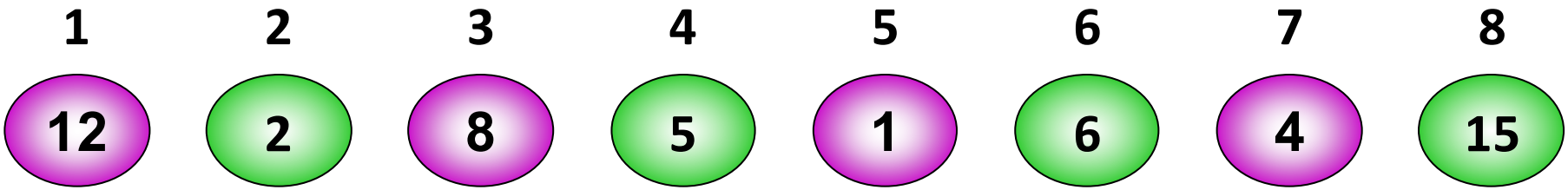
Phân phối đều luân phiên p hần tử



Merge Sort – Ví Dụ

p = 1

Phân phối đều luân phiên p hần tử

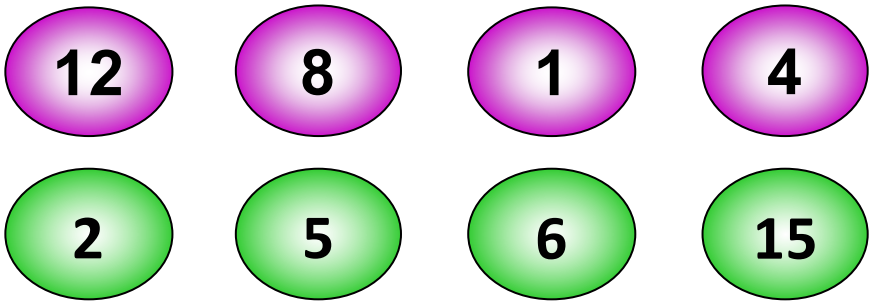


Merge Sort – Ví Dụ

k = 1

Trộn p phần tử

0	1	2	3	4	5	6	7



Merge Sort – Ví Dụ

k = 1

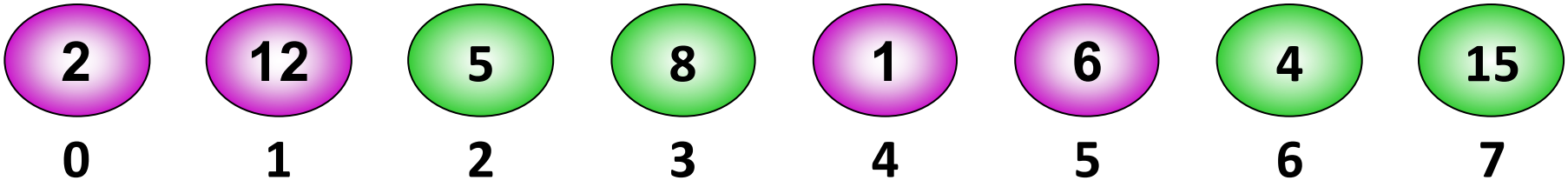
Trộn từng cặp p phần tử

0	1	2	3	4	5	6	7
12	8	1	4				
2	5	6	15				

Merge Sort – Ví Dụ

p = 2

Phân phối p phần tử

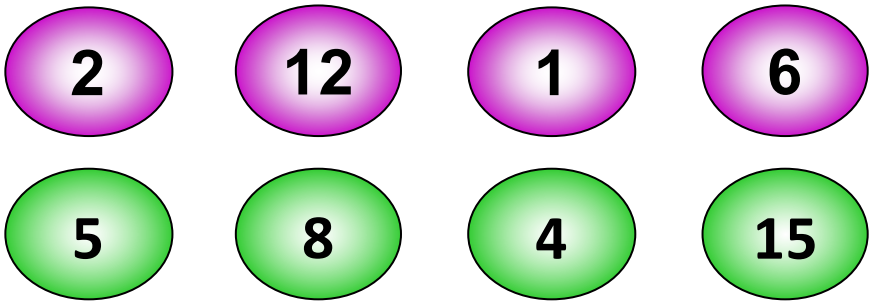


Merge Sort – Ví Dụ

p = 2

Trộn từng cặp p phần tử

0	1	2	3	4	5	6	7



Merge Sort – Ví Dụ

p = 2

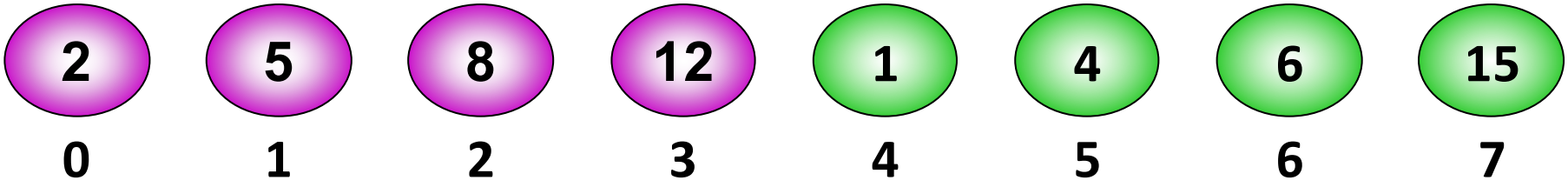
Trộn từng cặp p phần tử

0	1	2	3	4	5	6	7
2	12	1	6				
5	8	4	15				

Merge Sort – Ví Dụ

p = 4

Phân phối luân phiên p phần tử



Merge Sort – Ví Dụ

p = 4

Trộn từng cặp p phần tử

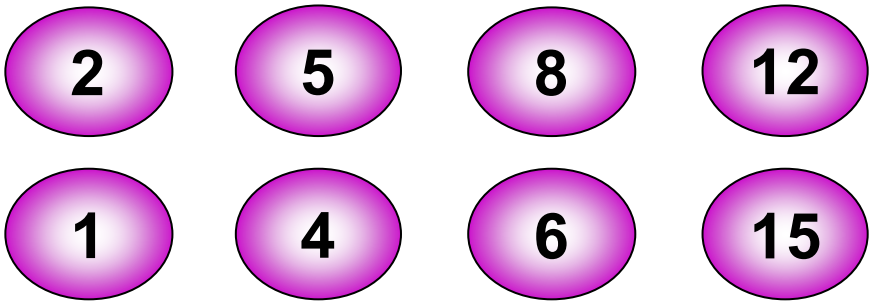
0	1	2	3	4	5	6	7
2	5	8	12				
1	4	6	15				

Merge Sort – Ví Dụ

p = 4

Trộn từng cặp p phần tử

0	1	2	3	4	5	6	7



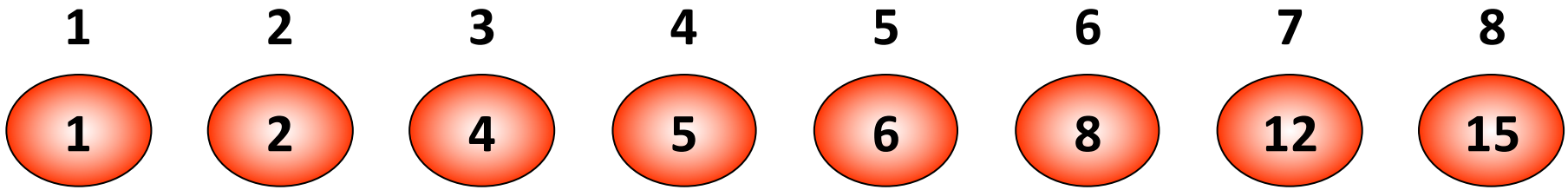
Merge Sort – Ví Dụ

p = 8

0	1	2	3	4	5	6	7
1	2	4	5	6	8	12	15

Chỉ một mảng con

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;
}
```


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 Số 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, ..., an, giải thuật Radix Sort thực hiện như sau:
  - Trước tiên, ta có thể giả sử mỗi phần tử ai trong dãy a_1, a_2, ..., an 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, …, B_9 rỗng;

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

- Bước 3 :
  - For i = 1 .. n do
    - Đặt ai vào lô Bt với t: chữ số thứ k của ai;
- Bước 4 :
  - Nối B_0, B_1, …, 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	0701										
11	1725										
10	0999										
9	9170										
8	3252										
7	4518										
6	7009										
5	1424										
4	0428										
3	1239										0999
2	8425						1725			4518	7009
1	7013	9170	0701	3252	7013	1424	8425			0428	1239
CS	A	0	1	2	3	4	5	6	7	8	9

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	0999										
11	7009										
10	1239										
9	4518										
8	0428										
7	1725										
6	8425										
5	1424										
4	7013			0428							
3	3252			1725							
2	0701	7009	4518	8425							
1	9170	0701	7013	1424	1239		3252		9170		0999
CS	A	0	1	2	3	4	5	6	7	8	9

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	0999										
11	9170										
10	3252										
9	1239										
8	0428										
7	1725										
6	8425										
5	1424										
4	4518										
3	7013					0428					
2	7009	7013		3252		8425			1725		
1	0701	7009	9170	1239		1424	4518		0701		0999
CS	A	0	1	2	3	4	5	6	7	8	9

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	0999										
11	1725										
10	0701										
9	4518										
8	0428										
7	8425										
6	1424										
5	3252										
4	1239										
3	9170	0999	1725								
2	7013	0701	1424						7013		
1	7009	0428	1239		3252	4518			7009	8425	9170
CS	A	0	1	2	3	4	5	6	7	8	9

Sắp Xếp Theo Phương Pháp Cơ Số Radix Sort

12	9170										
11	8425										
10	7013										
9	7009										
8	4518										
7	3252										
6	1725										
5	1424										
4	1239										
3	0999										
2	0701										
1	0428										
CS	A	0	1	2	3	4	5	6	7	8	9

Bài Tập

- 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.*