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

4. Chèn trực tiếp – Insertion Sort
5. Shell Sort
6. Heap Sort
7. Quick Sort

Nhu cầu tìm kiếm và sắp xếp

- Thao tác tìm kiếm được sử dụng nhiều nhất trong các hệ lưu trữ và quản lý dữ liệu.
- Do dữ liệu lớn nên tìm ra giải thuật tìm kiếm nhanh chóng là mối quan tâm hàng đầu. Để đạt được điều này dữ liệu phải được tổ chức theo một thứ tự nào đó thì việc tìm kiếm sẽ nhanh chóng và hiệu quả hơn, vì vậy nhu cầu sắp xếp dữ liệu cũng được lưu ý.
- Tóm lại, bên cạnh những giải thuật tìm kiếm thì các giải thuật sắp xếp dữ liệu không thể thiếu trong hệ quản lý thông tin trên máy tính.

Các giải thuật tìm kiếm

- Có 2 giải thuật thường được áp dụng: Tìm tuyến tính và tìm nhị phân.
- Để đơn giản cho việc minh họa, ta đặc tả như sau:

  - Tập dữ liệu được lưu trữ là dãy số $\mathbf{a_1, a_2, \ldots, a_N}$.
  - Giả sử chọn cấu trúc dữ liệu mảng để lưu trữ dãy số này trong bộ nhớ chính, có khai báo: **int a[N];**
  - Khoá cần tìm là **x**, được khai báo như sau: **int x;**

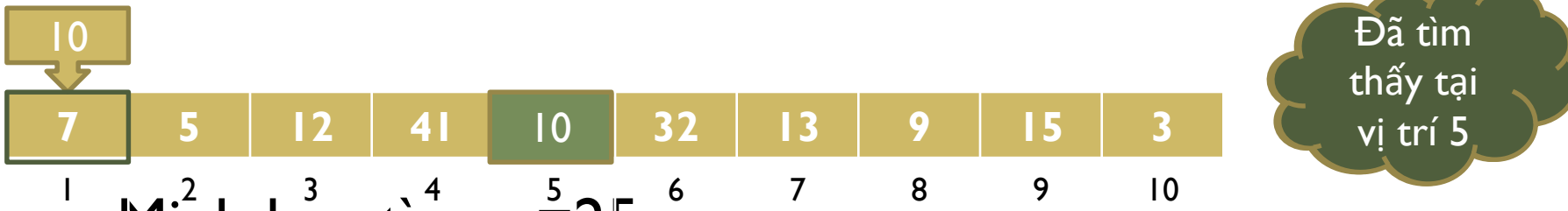
5

Tìm kiếm tuyến tính

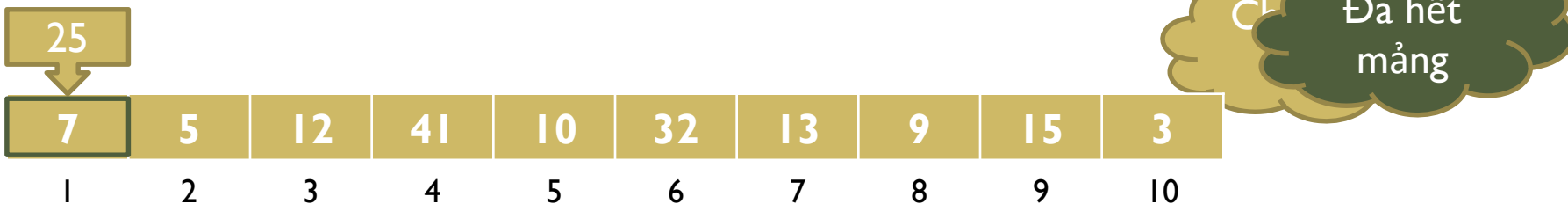
- **Ý tưởng**

  Tiến hành so sánh **x** lần lượt với phần tử thứ nhất, thứ hai, ... của mảng **a** cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy **x**.

- Minh họa tìm x =10



- Minh họa tìm x =25



Giải thuật

Bước 1:

i = 1; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh a[i] với x, có 2 khả năng :

- *a[i] = x :Tìm thấy. Dừng*
- *a[i] != x : Sang Bước 3.*

Bước 3:

- *i = i+1; // xét tiếp phần tử kế trong mảng*
- *Nếu i >N: Hết mảng, không tìm thấy. Dừng*

  Ngược lại: Lặp lại Bước 2.

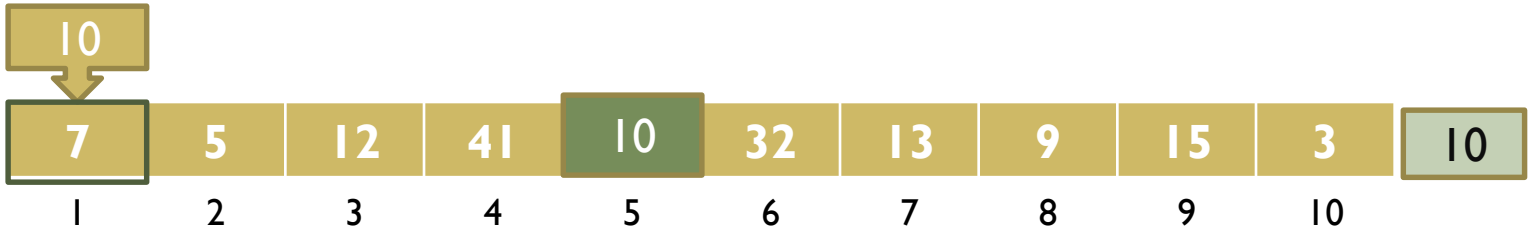
Cài đặt

```
int LinearSearch(int a[], int N, int x)
{
   int i=0;
   while ((i<N) && (a[i]!=x ))
      i++;
   if(i==N)
      return -1;// tìm hết mảng nhưng không có x
   else
      return i;// a[i] là phần tử có khoá x
}
```

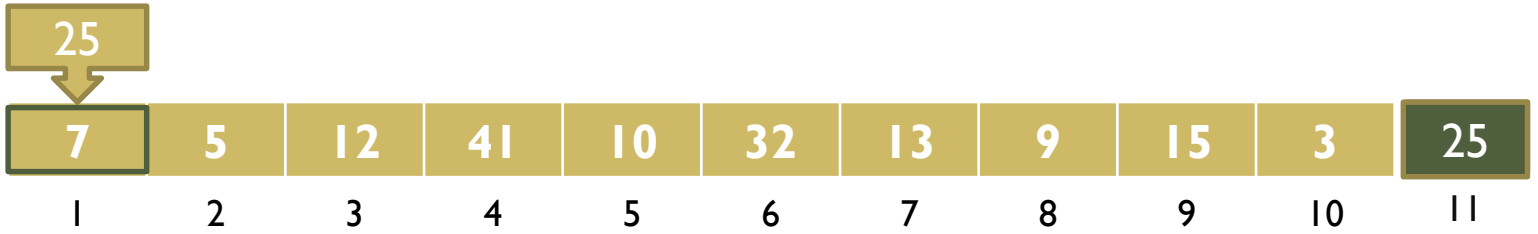
Cải tiến thuật toán tìm kiếm tuyến tính

Cải tiến (dùng phần tử lính canh) giúp giảm bớt một phép so sánh

- Minh họa tìm x =10



- Minh họa tìm x = 25



Cài đặt

```
int LinearSearch2(int a[],int N,int x)
 {   int i=0;    // mảng gồm N phần tử từ a[0]..a[N-1]
     a[N] = x; // thêm phần tử thứ N+1
    while (a[i]!=x )
        i++;
    if (i==N)
         return -1;         // tìm hết mảng nhưng không có x
    else
        return i;           // tìm thấy x tại vị trí i
 }
```

Đánh Giá Thuật Toán Tìm Tuyến Tính

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

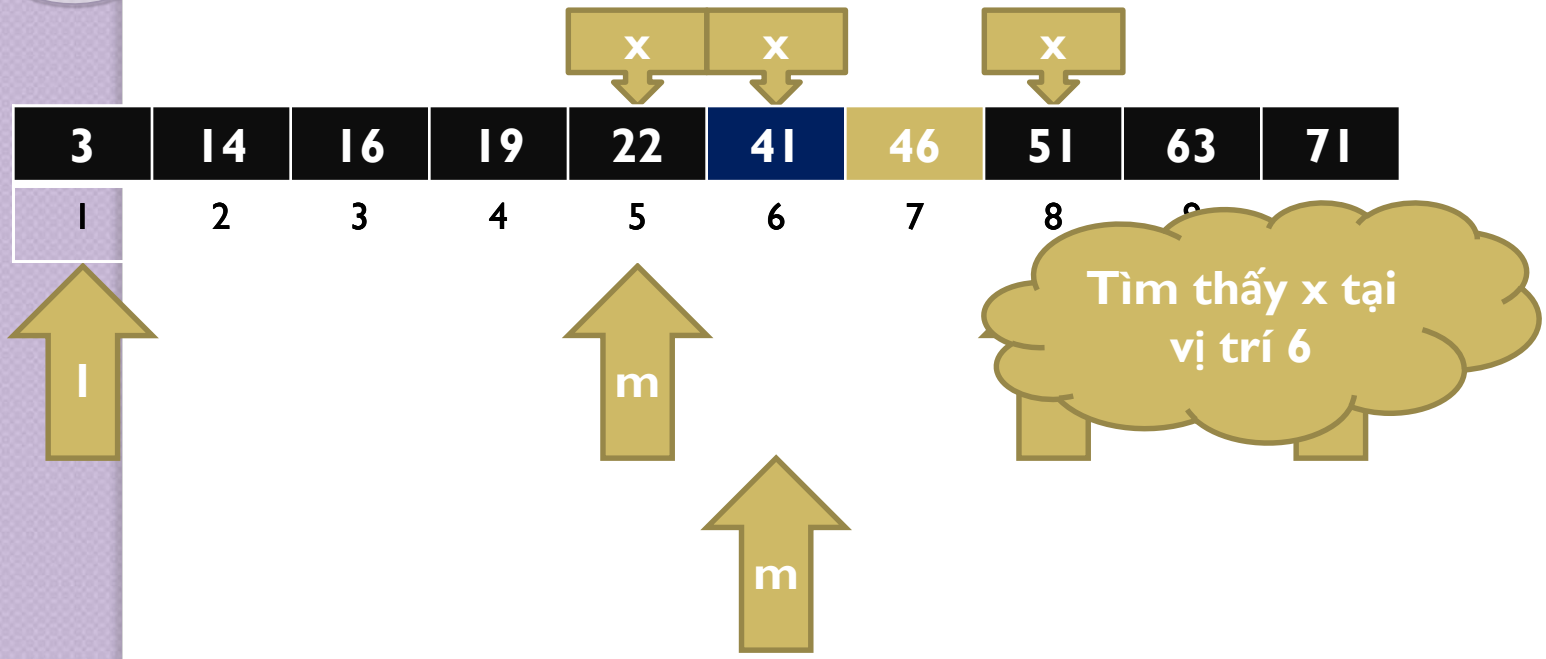
- Độ phức tạp O(N)

Tìm kiếm nhị phân

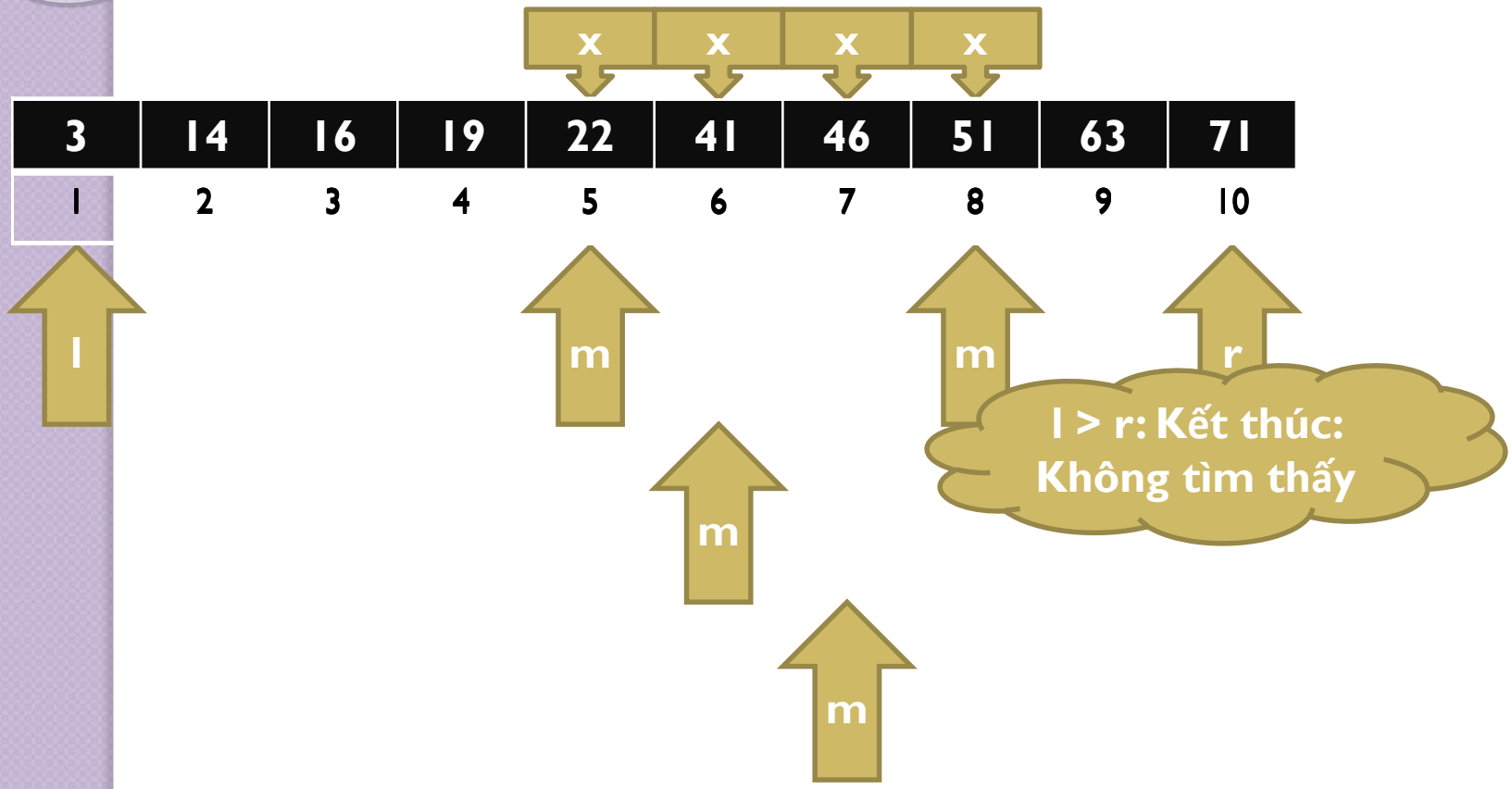
- Được áp dụng trên dãy đã có thứ tự.
- **Ý tưởng**: .
  - Giả xử ta xét mảng có thứ tự tăng, khi ấy ta có

    $a_{i-1} < a_i < a_{i+1}$
  - 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.

Minh họa tìm x = 41



Minh họa tìm x = 45



Giải thuật

Bước 1: left = 1; right = N; // tìm kiếm trên tất cả các phần tử

Bước 2:

mid = (left+right)/2; // lấy mốc so sánh

So sánh a[mid] với x, có 3 khả năng :

- a[mid] = x: Tìm thấy. Dừng
- a[mid] > x: //tìm tiếp x trong dãy con $a_{left} .. a_{mid-1}$

  right =mid - 1;
- a[mid] < x: //tìm tiếp x trong dãy con $a_{mid+1} .. a_{right}$

  left = mid + 1;

Bước 3:

Nếu left <= right //còn phần tử chưa xét tìm tiếp.

Lặp lại Bước 2.

Ngược lại: Dừng //Đã xét hết tất cả các phần tử.

Cài đặt

```
int BinarySearch(int a[],int N,int x )
{       int left =0; right = N-1;
        int mid;
        do{
                mid = (left + right)/2;
                if (x == a[mid])
                        return mid;//Thấy x tại mid
                else if (x < a[mid])
                        right = mid -1;
                      else
                        left = mid +1;
        }while (left <= right);
        return -1; // Tìm hết dãy mà không có x
}
```

Đánh Giá Thuật Toán Tìm Nhị Phân

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

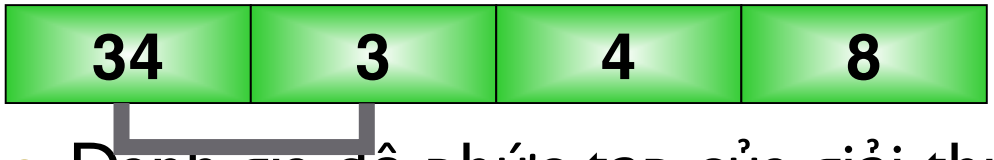
- Độ phức tạp $O(\log_2 N)$

Bài toán sắp xếp

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

Bài toán sắp xếp

- 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

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.

Đổi Chỗ Trực Tiếp–Interchange Sort

- 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;

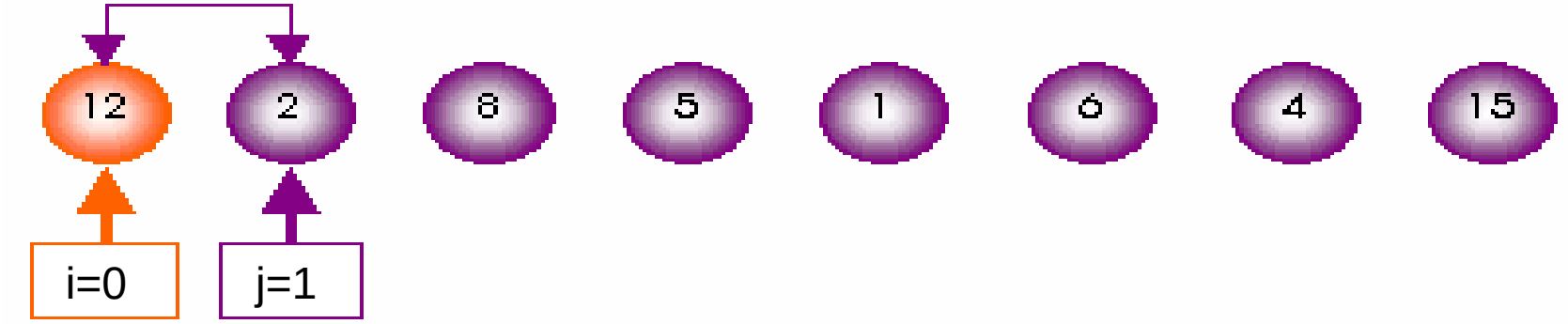
  Nếu i < N-1: Lặp lại Bước 2.

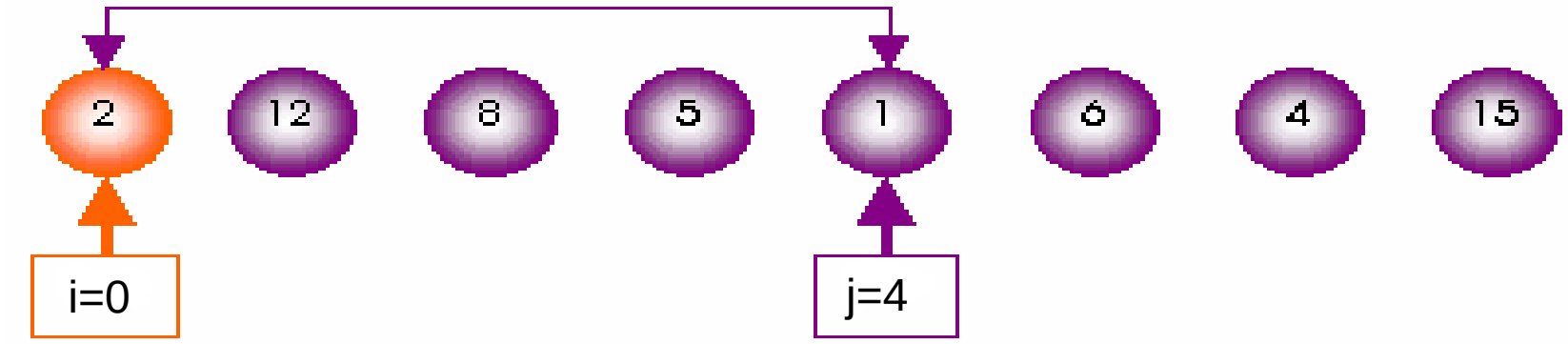
  Ngược lại: Dừng.

Đổi Chỗ Trực Tiếp–Interchange Sort

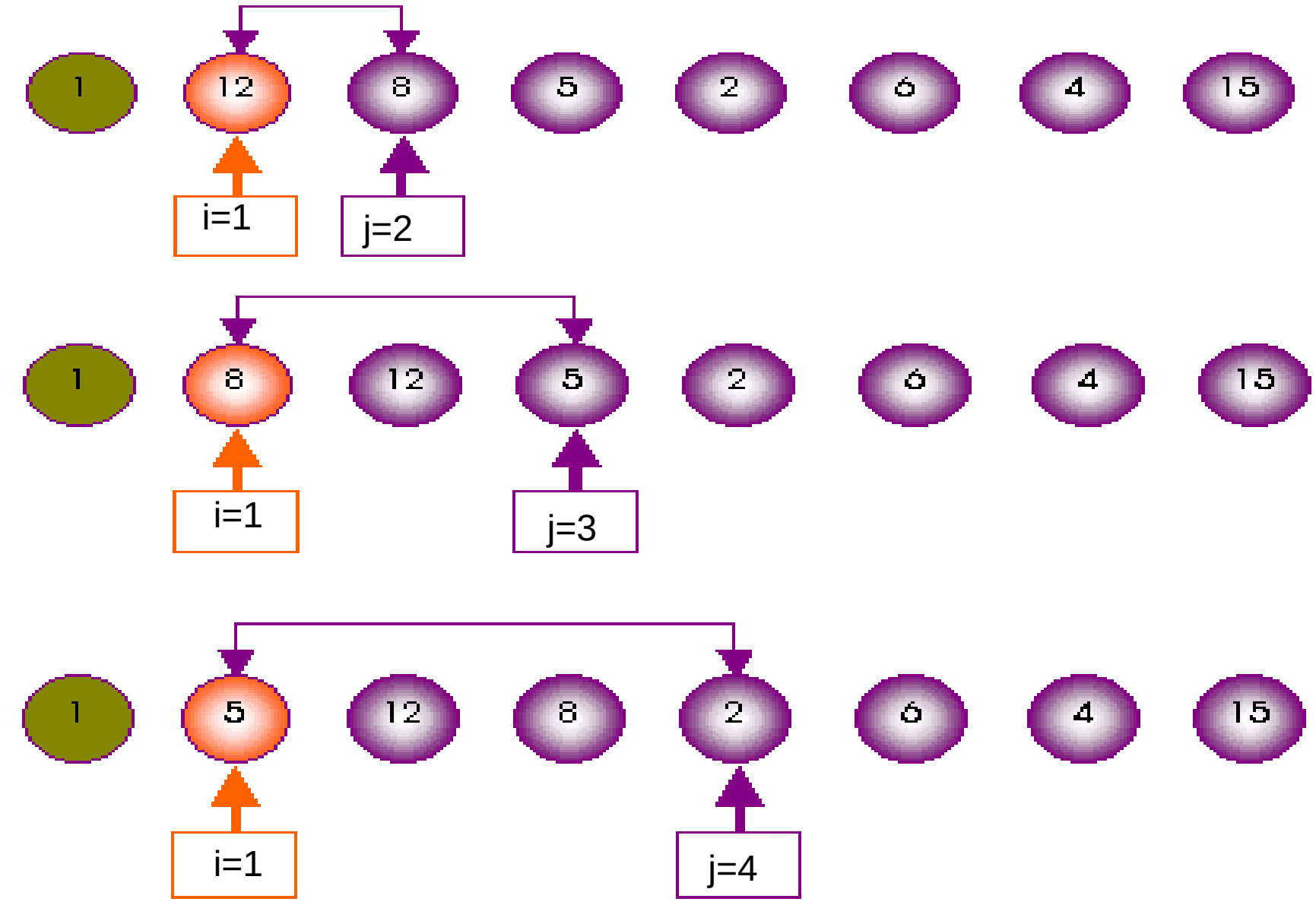
- Cho dãy số a:

12 2 8 5 1 6 4 15

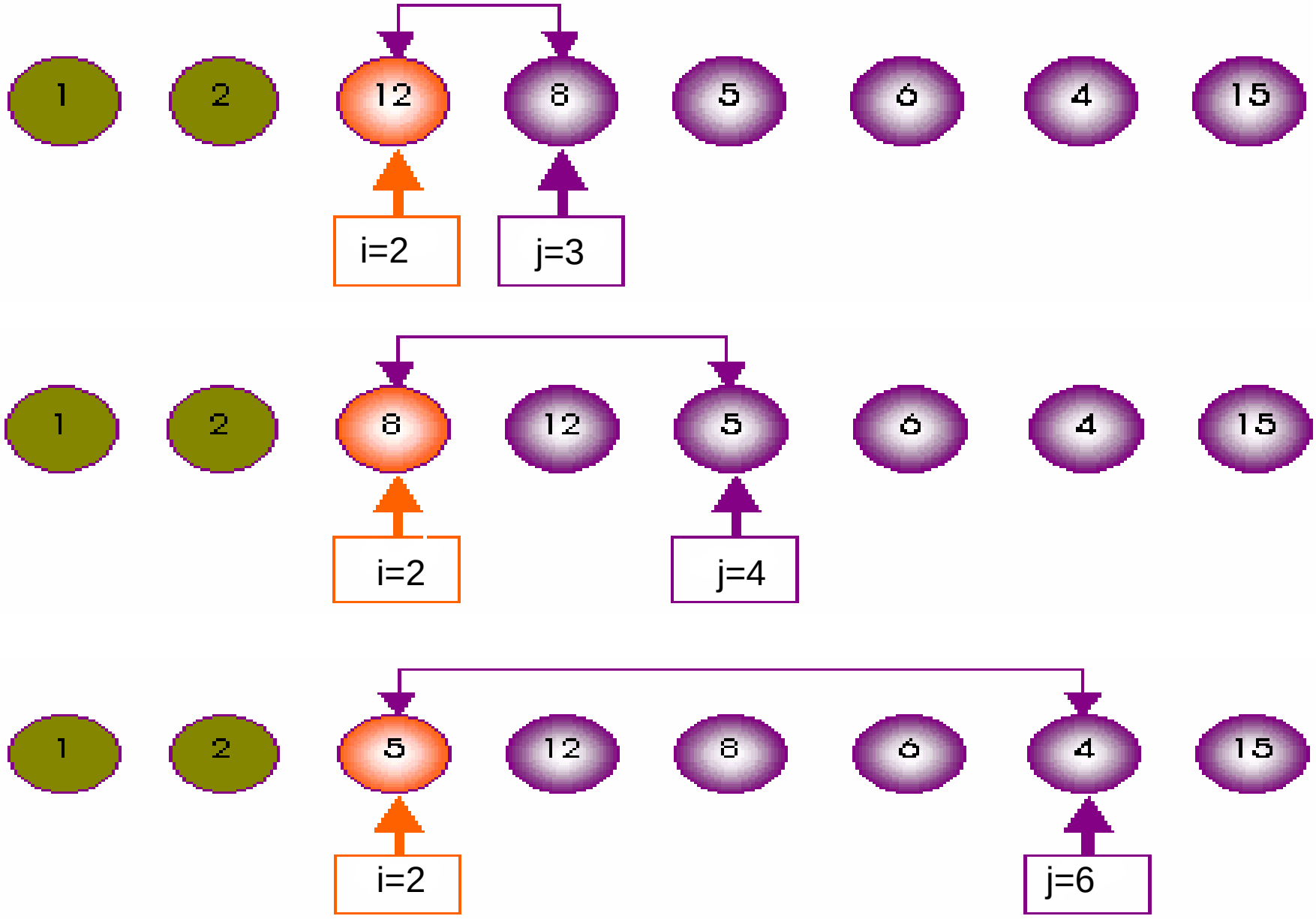




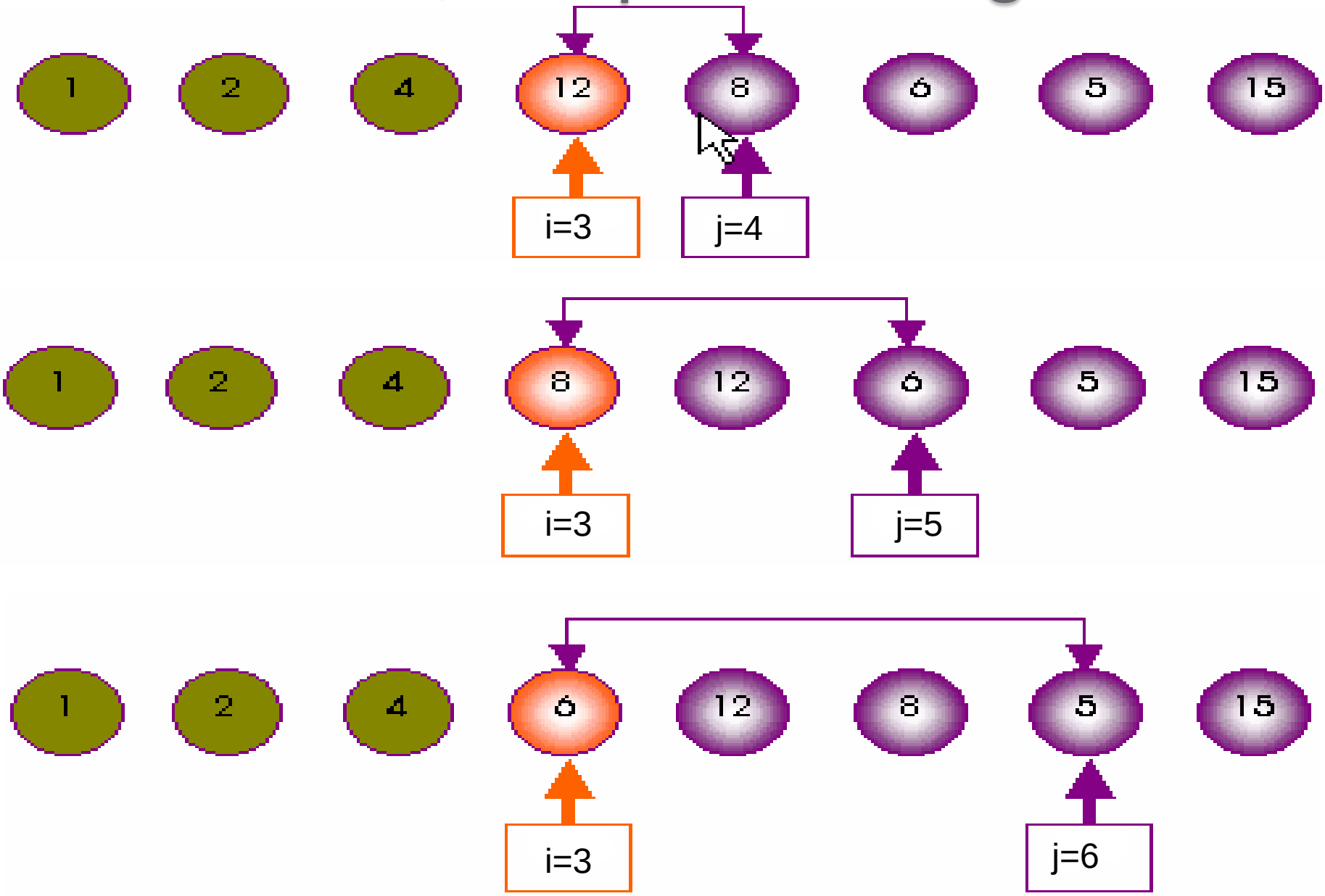
Đổi Chỗ Trực Tiếp–Interchange Sort



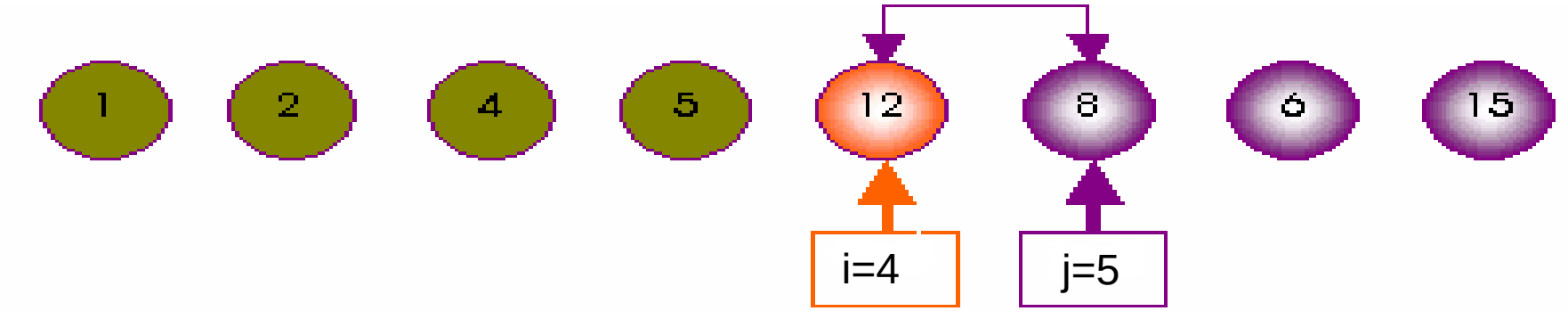
Đổi Chỗ Trực Tiếp–Interchange Sort

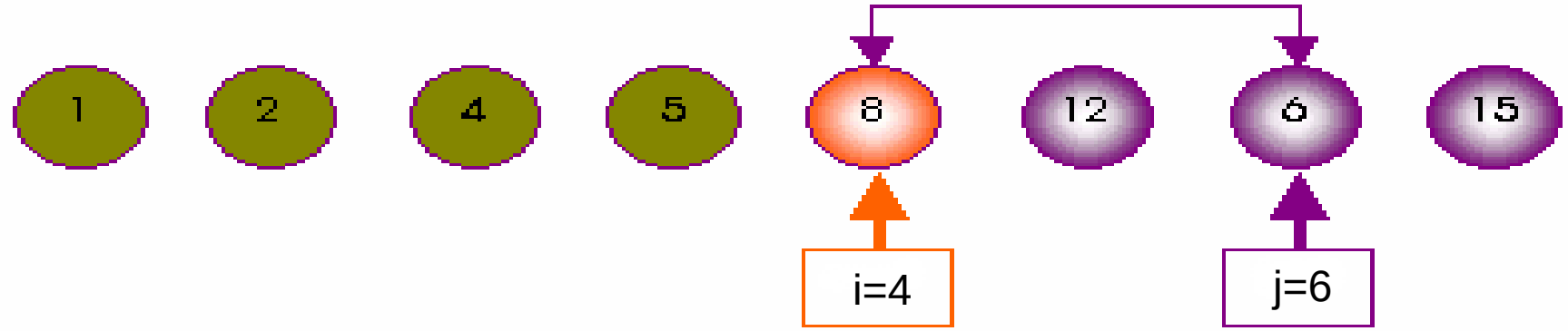


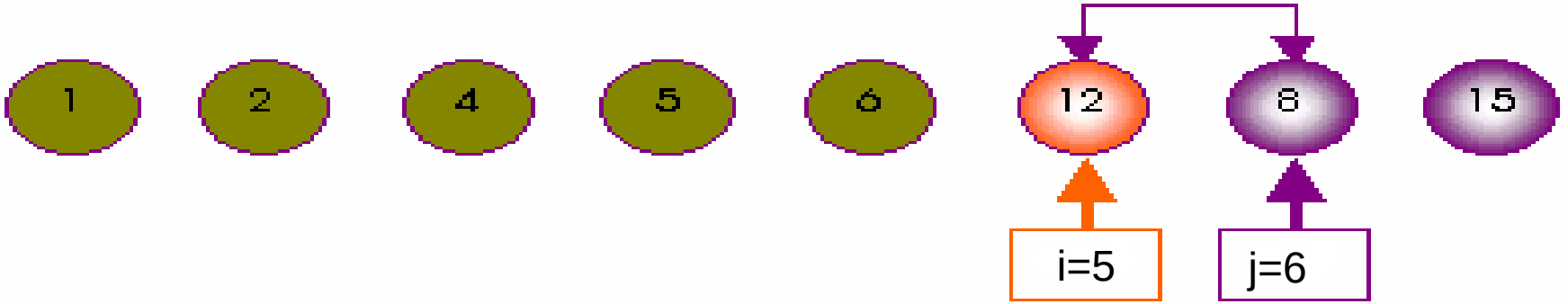
Đổi Chỗ Trực Tiếp–Interchange Sort



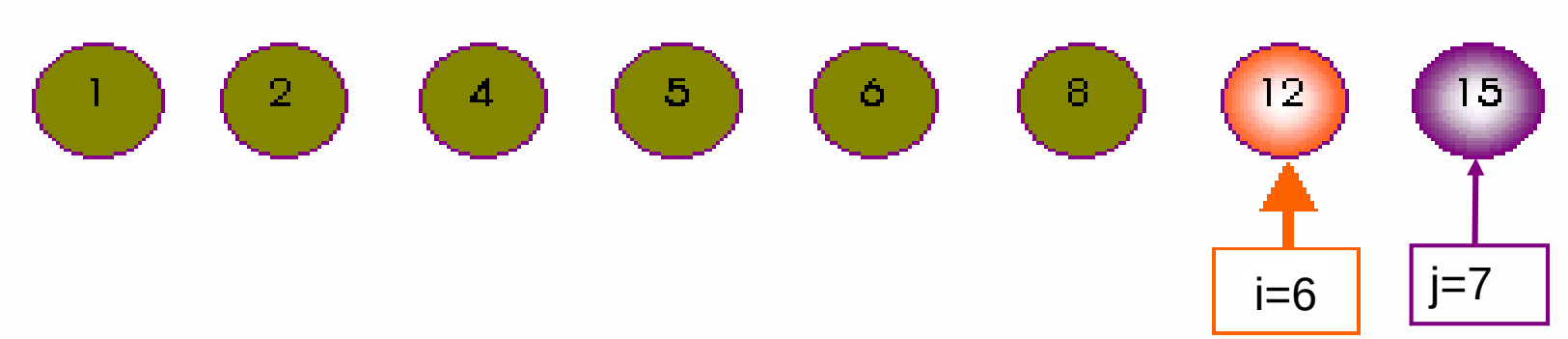
Đổi Chỗ Trực Tiếp–Interchange Sort

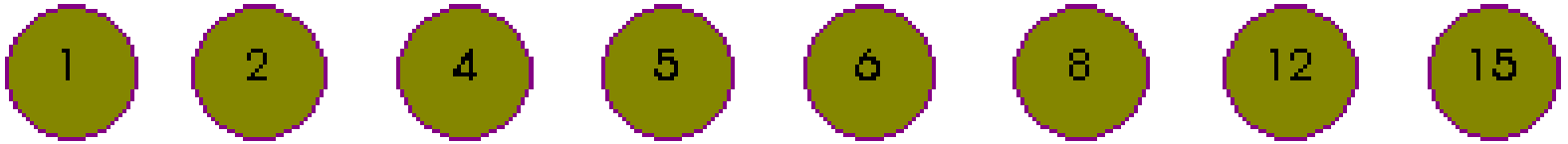






Đổi Chỗ Trực Tiếp–Interchange Sort





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]);
}
```

Độ phức tạp thuật toán Đổi chỗ trực tiếp

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

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

Chọn Trực Tiếp – Selection Sort

- 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

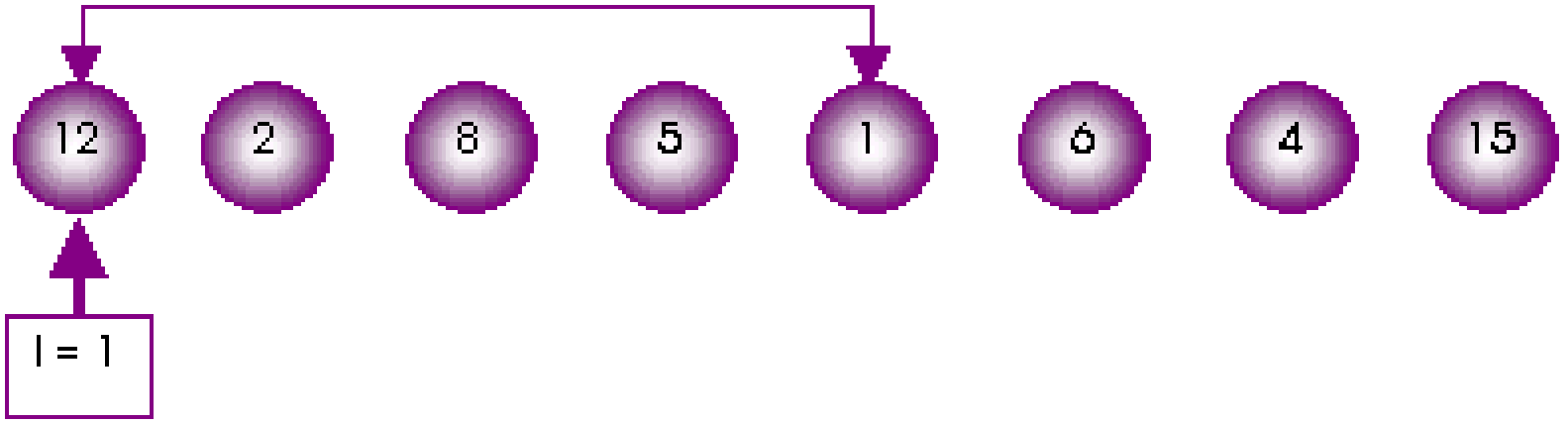
  i = i+1; Lặp lại Bước 2;

  Ngược lại: Dừng.

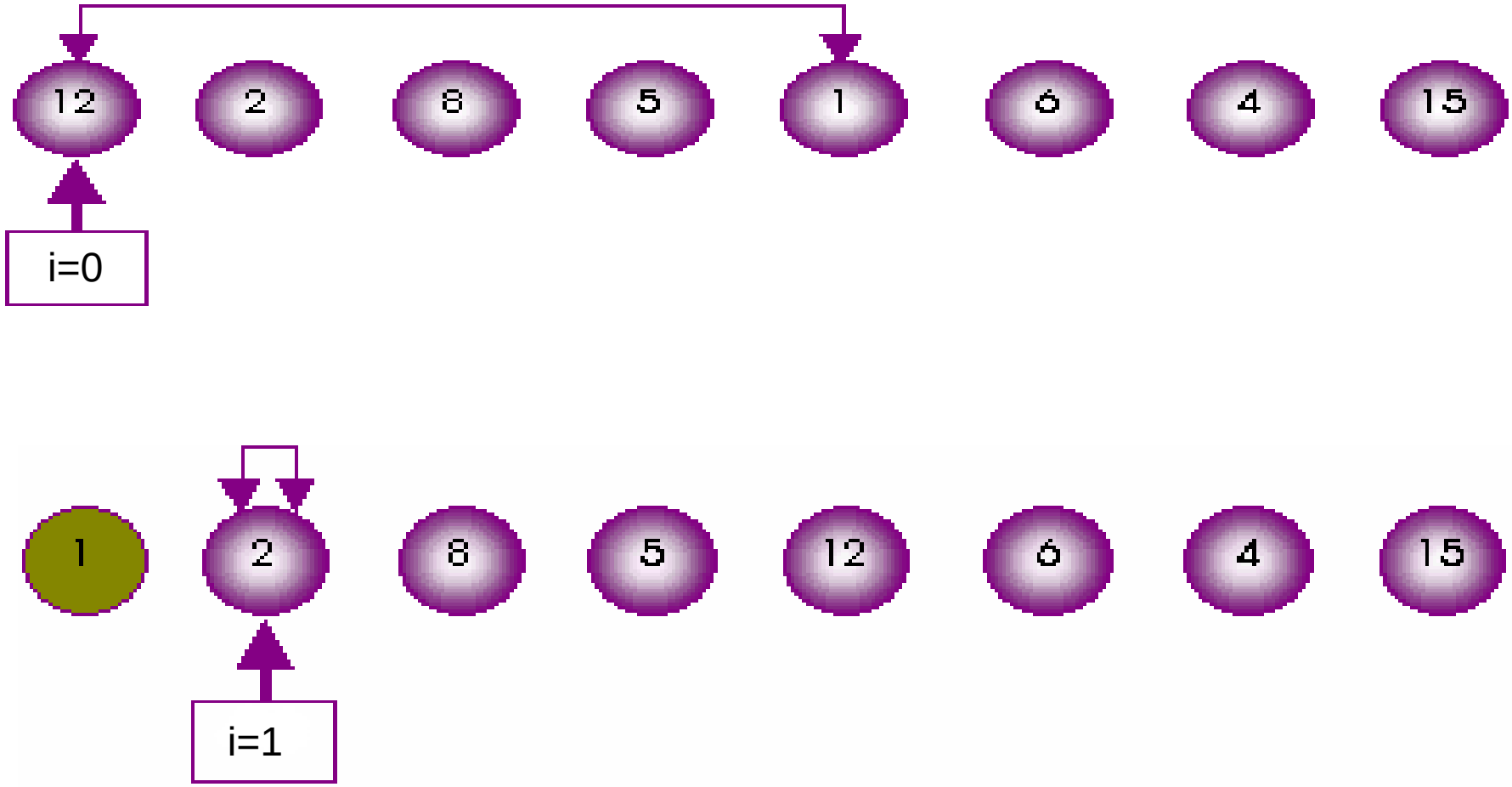
Chọn Trực Tiếp – Selection Sort

- Cho dãy số a:

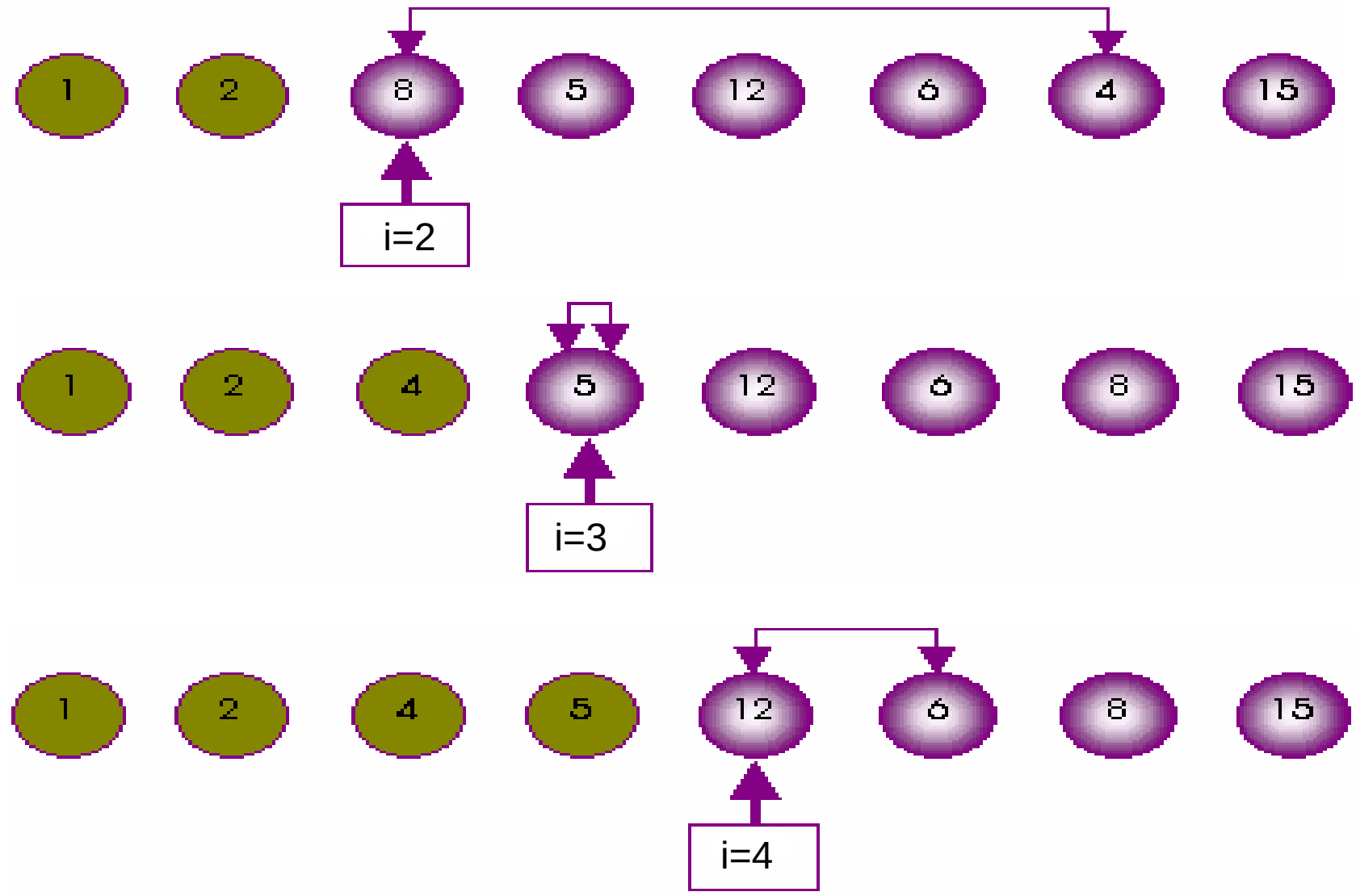
12 2 8 5 1 6 4 15



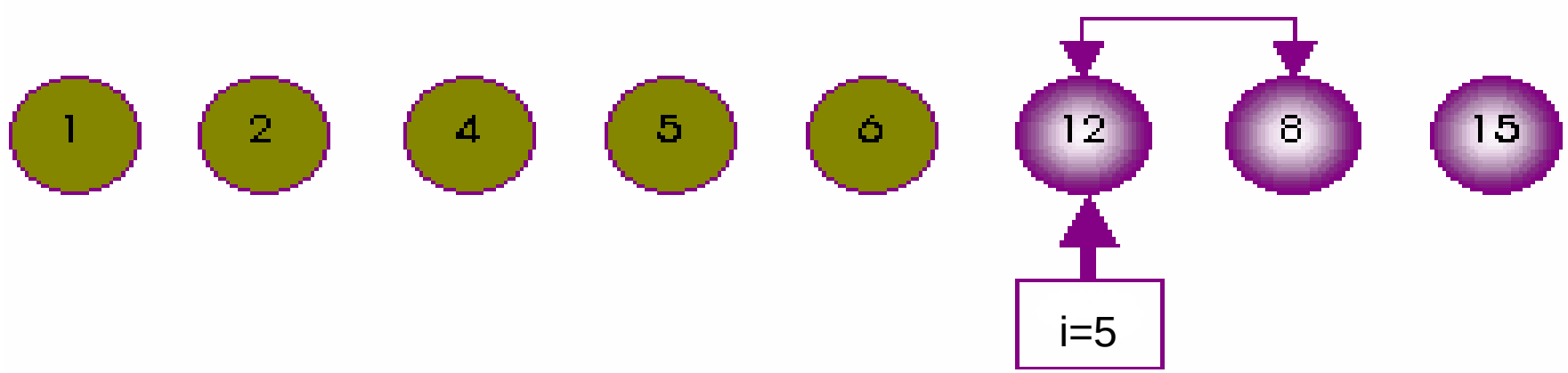
Chọn Trực Tiếp – Selection Sort

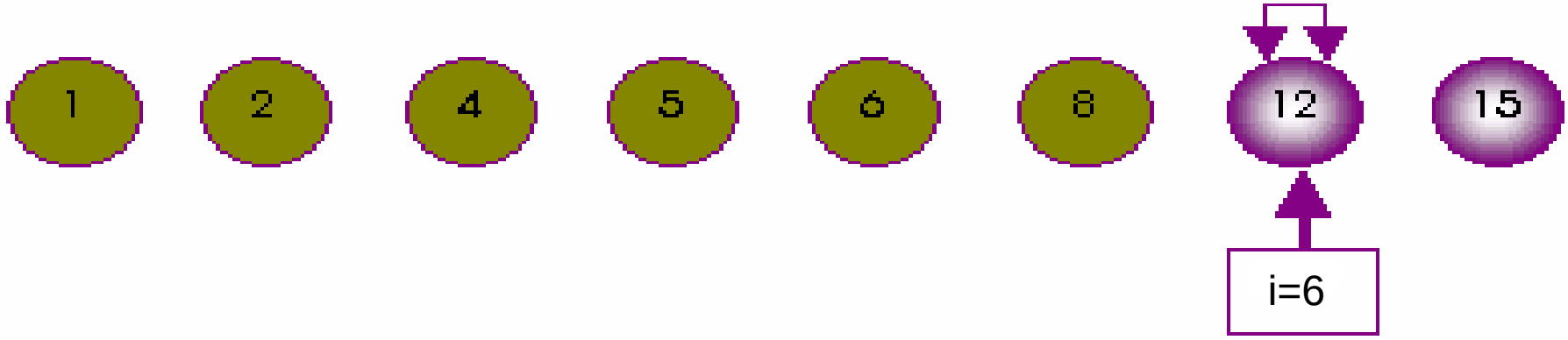


Chọn Trực Tiếp – Selection Sort



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]);
        }
  }
```

Độ phức tạp thuật toán Chọn trực tiếp

- **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

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

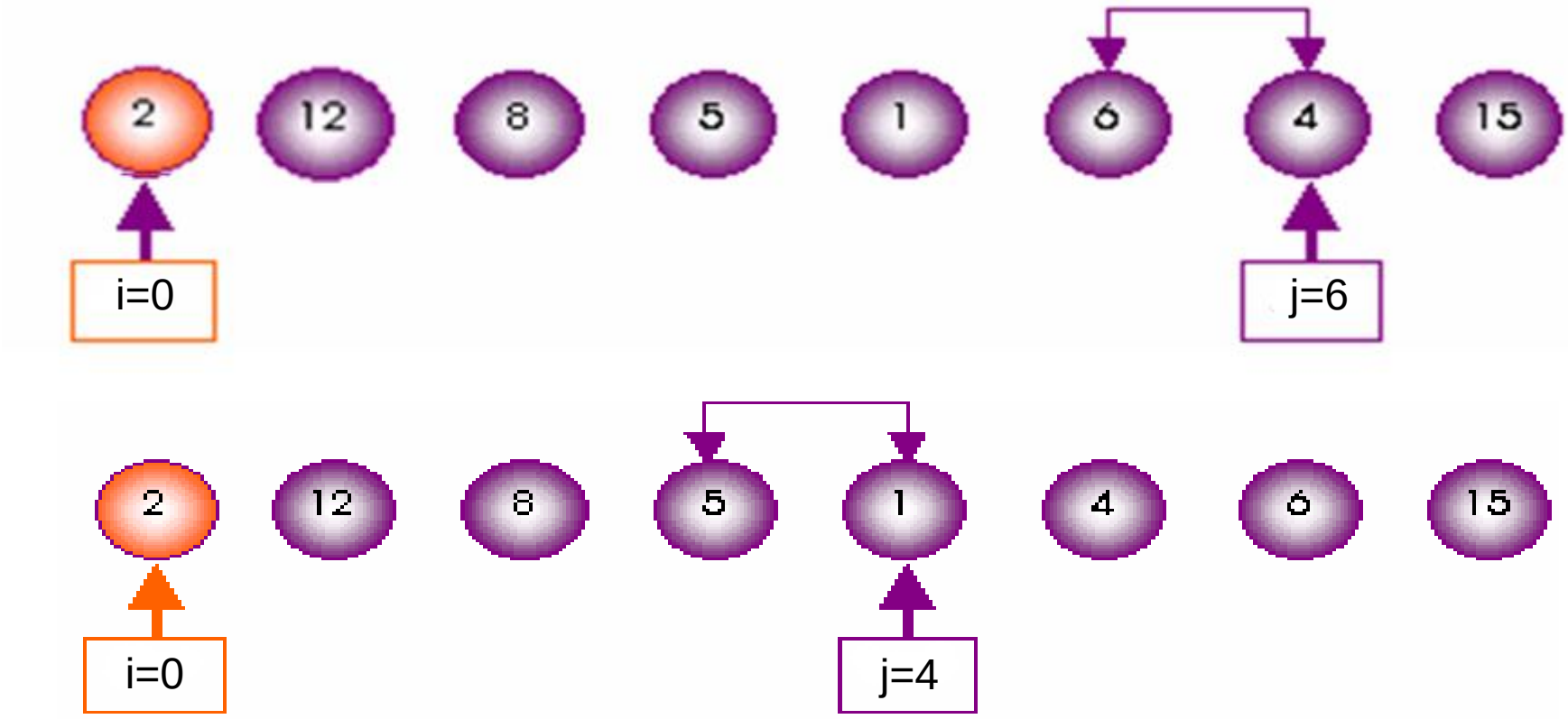
  Nếu i =N: Hết dãy. Dừng

  Ngược lại : Lặp lại Bước 2.

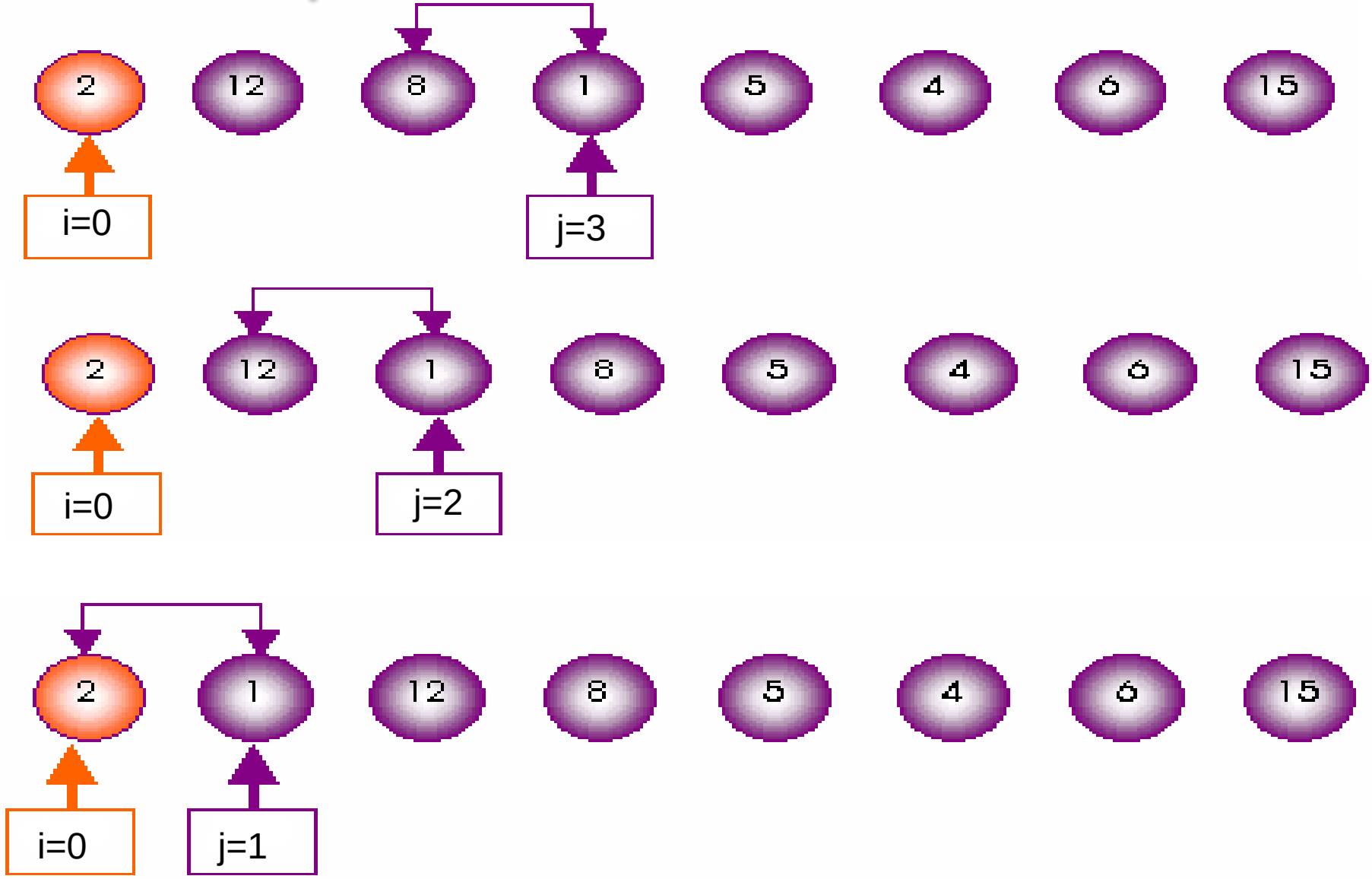
Nổi Bọt – Bubble Sort

- Cho dãy số a:

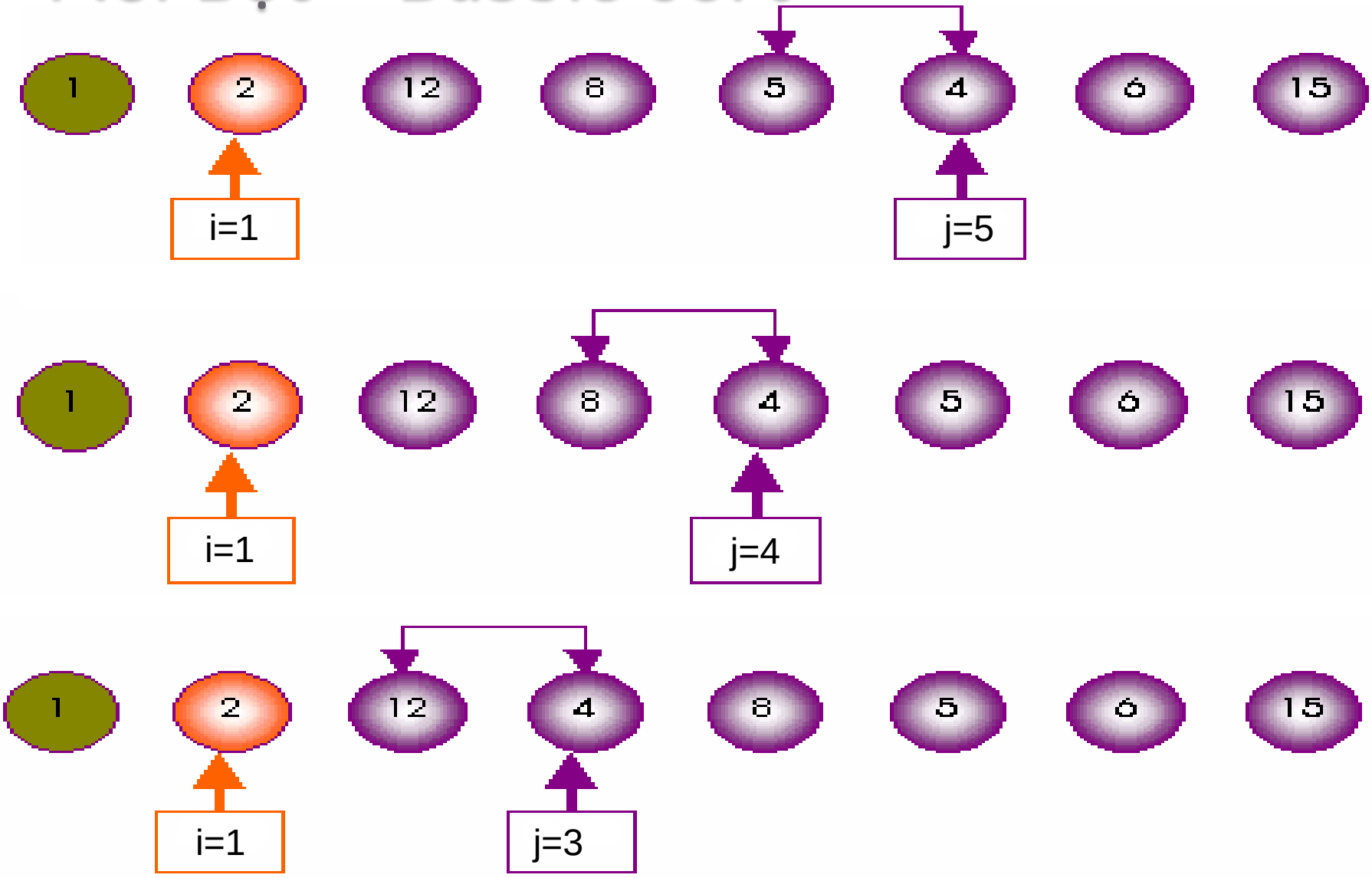
2 12 8 5 1 6 4 15



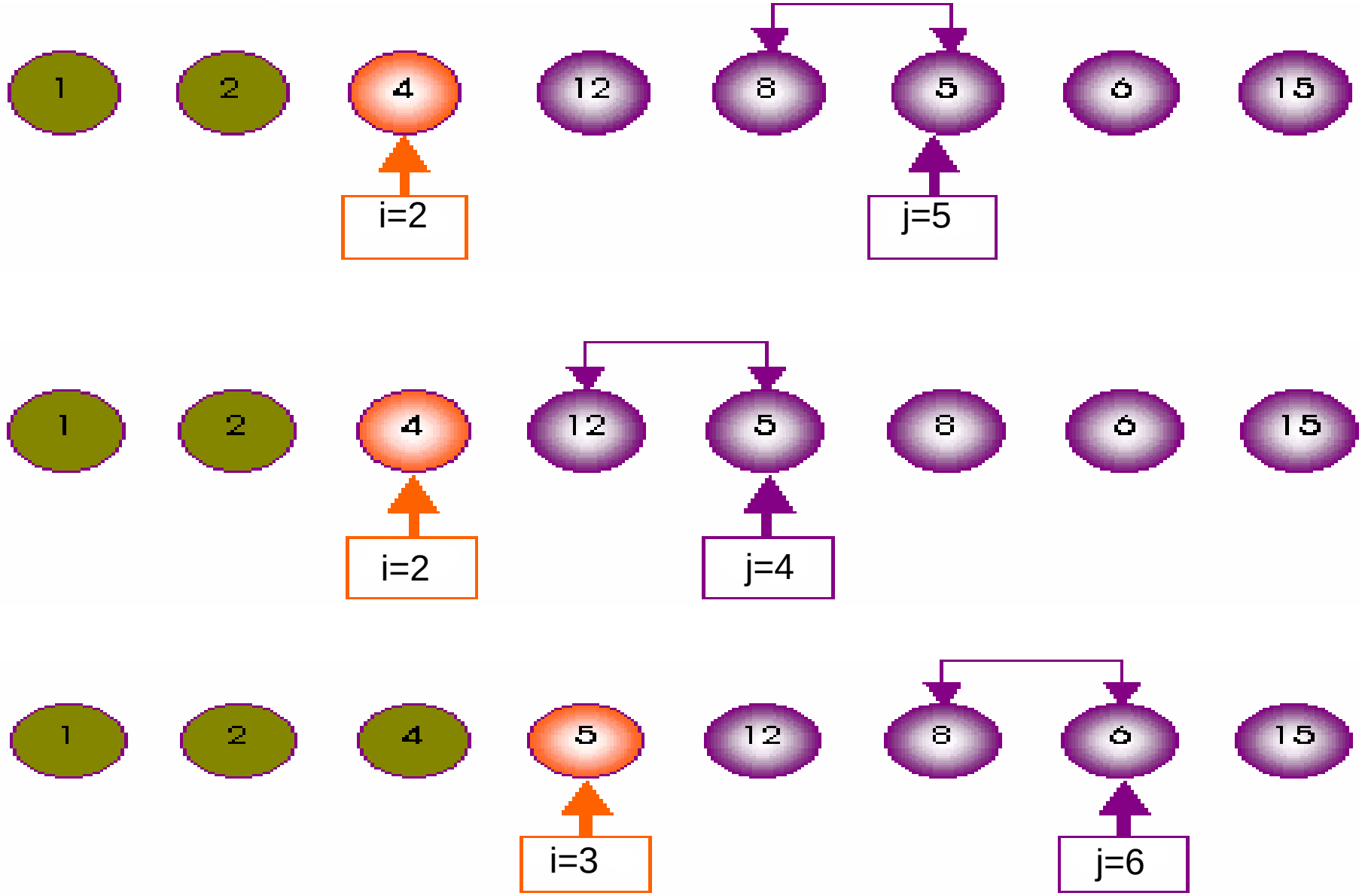
Nổi Bọt – Bubble Sort



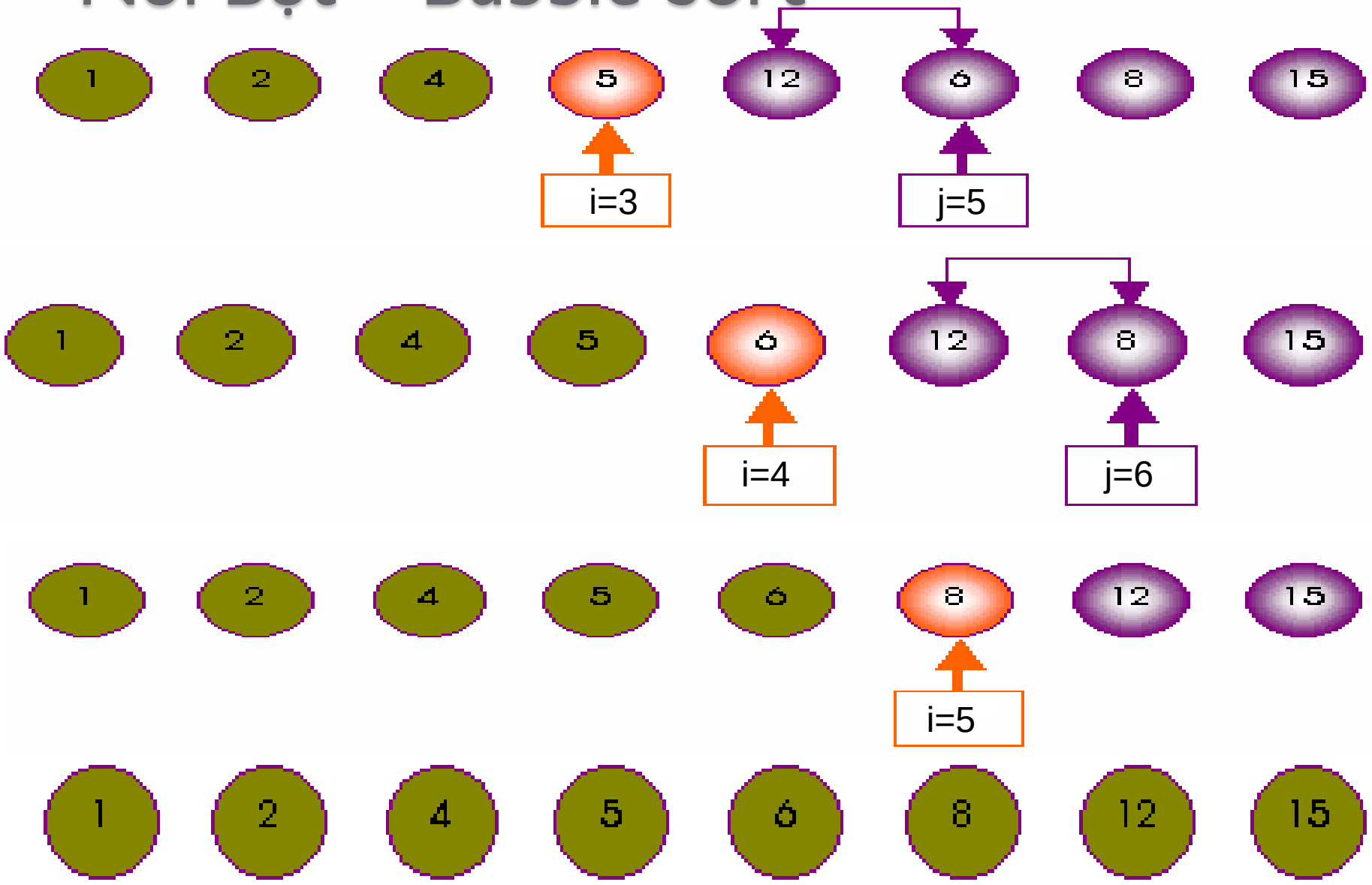
Nổi Bọt – Bubble Sort



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]);
}
```

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

ShakerSort

- 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.

ShakerSort

- 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]): 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 ShakerSort

```
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;
    }
}
```

Chèn Trực Tiếp – Insertion Sort

- Giả sử có một dãy $a_0, a_1, \ldots, a_{n-1}$ trong đó i phần tử đầu tiên $a_0, a_1, \ldots, a_{i-1}$ 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 $a_0, a_1, \ldots, a_i$ 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 chỗ cho a[i]
- Bước 4: a[pos] = x; //có đoạn a[1]..a[i] đã được sắp
- Bước 5: i = i+1;

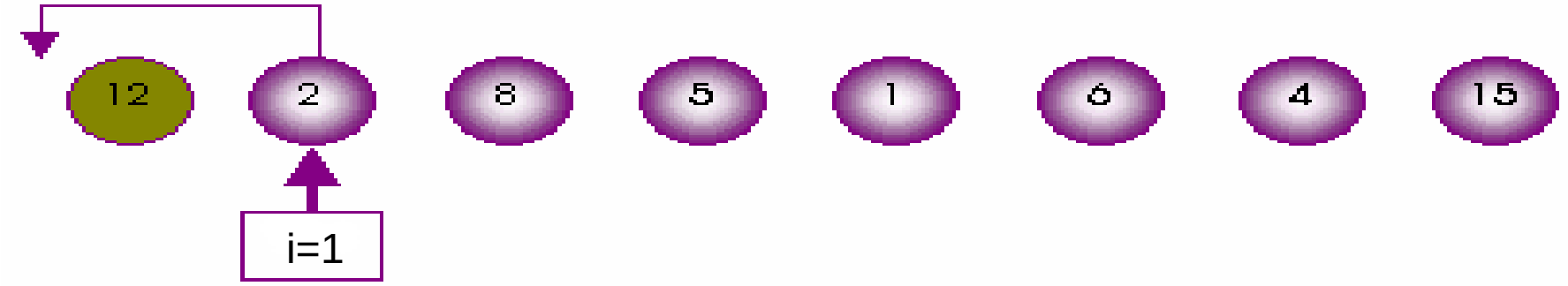
  Nếu i < n : Lặp lại Bước 2

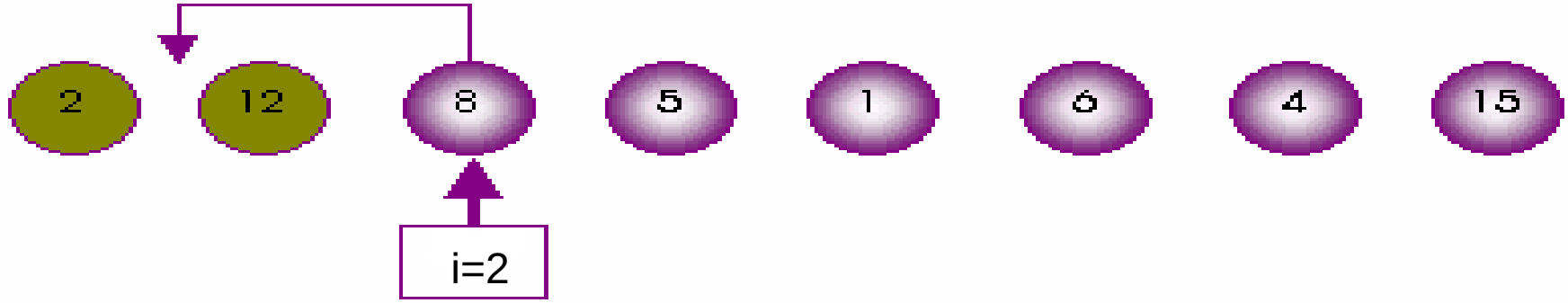
  Ngược lại : Dừng

Chèn Trực Tiếp – Insertion Sort

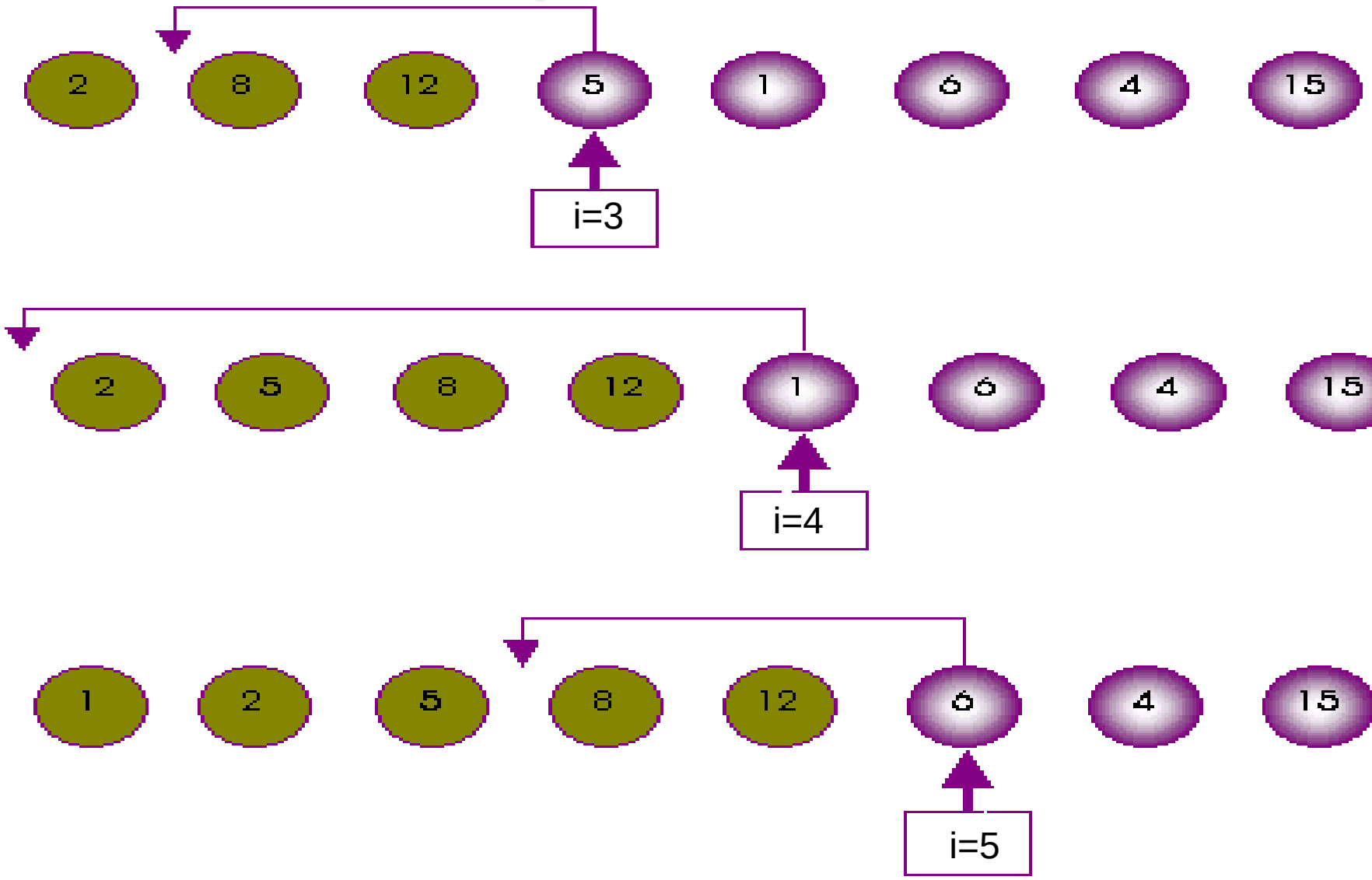
- Cho dãy số :

12 2 8 5 1 6 4 15

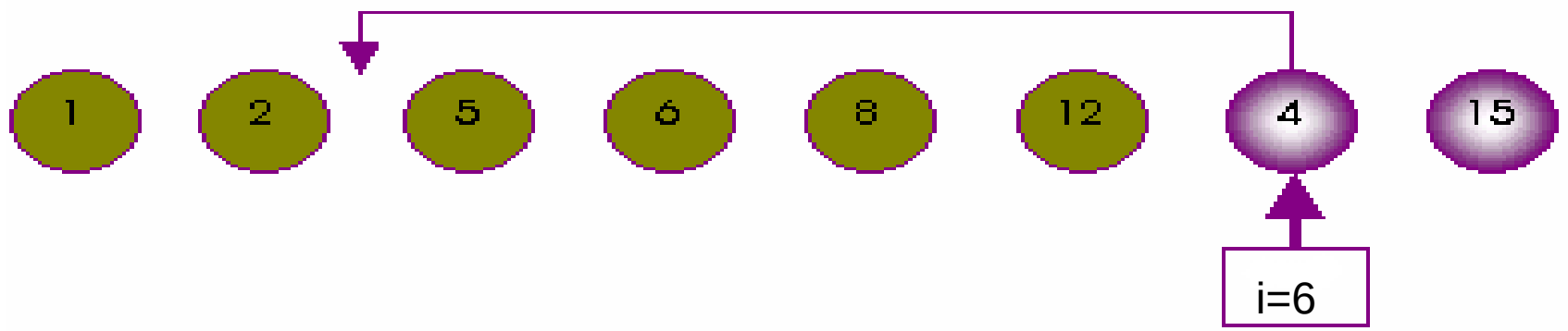


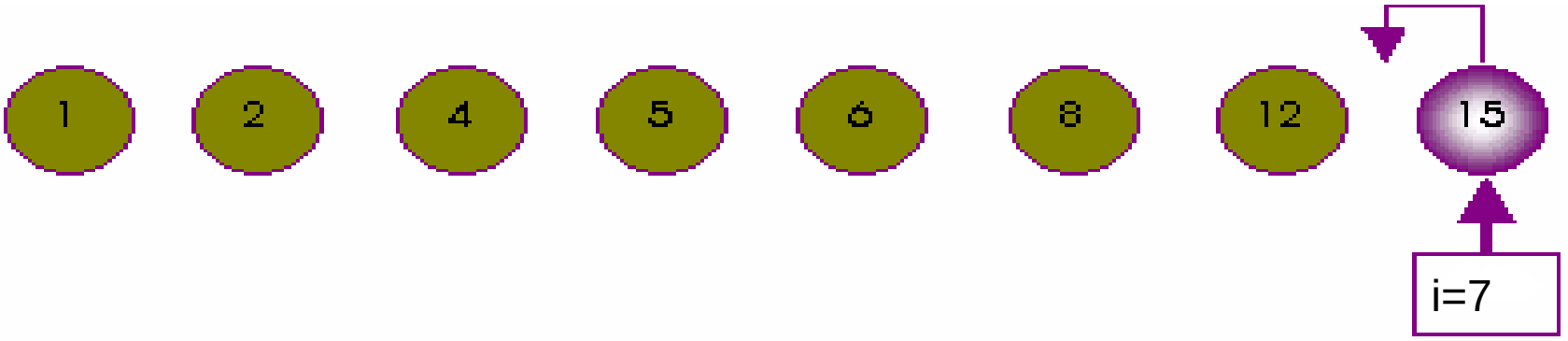


Chèn Trực Tiếp – Insertion Sort



Chèn Trực Tiếp – Insertion Sort





Cài Đặt Thuật Toán Chèn Trực Tiếp

```
void InsertionSort(int 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
         }
   }
```

Độ Phức Tạp Thuật Toán 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	$\sum_{i=1}^{n-1} (i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1$

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

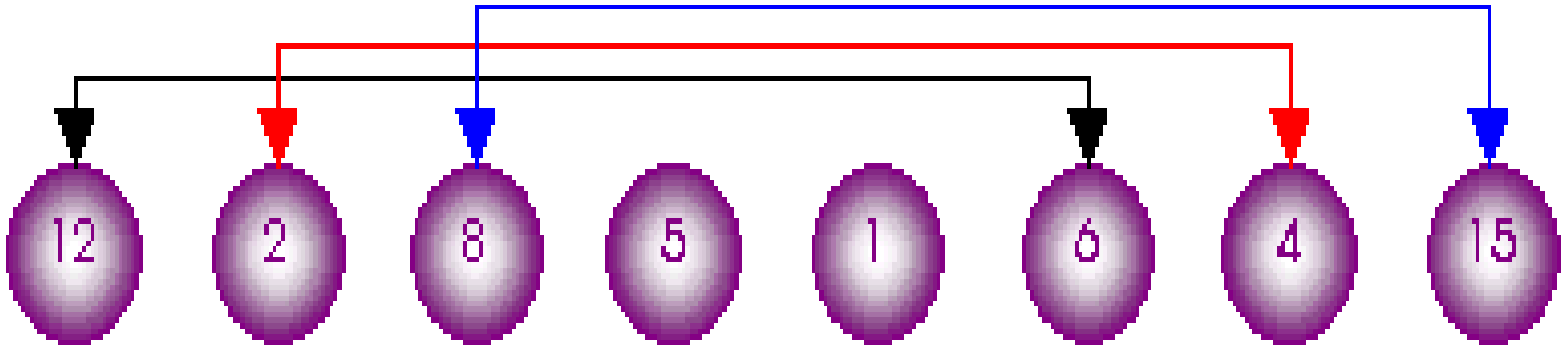
- Cho dãy số a:

  12 2 8 5 1 6 4 15

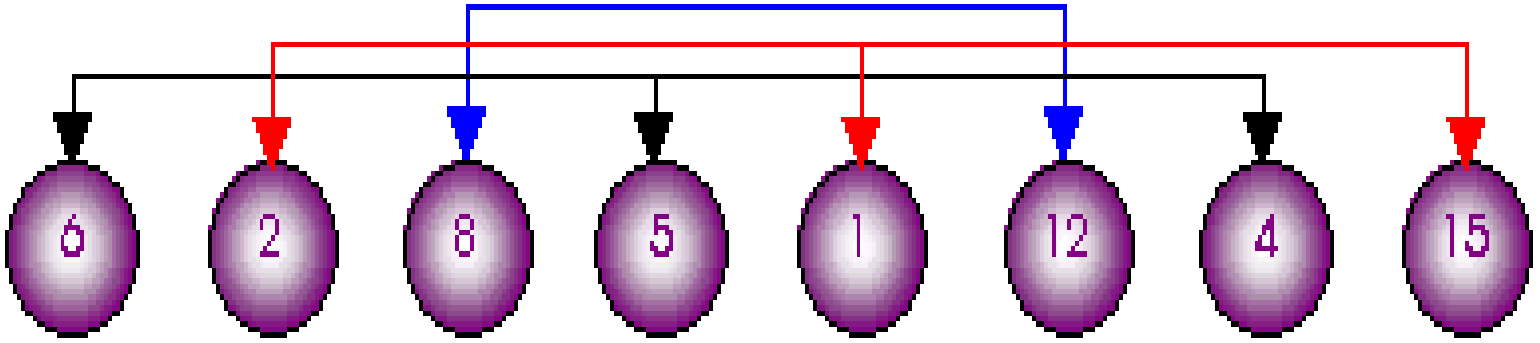
- Giả sử chọn các khoảng cách là 5, 3, 1

Shell Sort

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

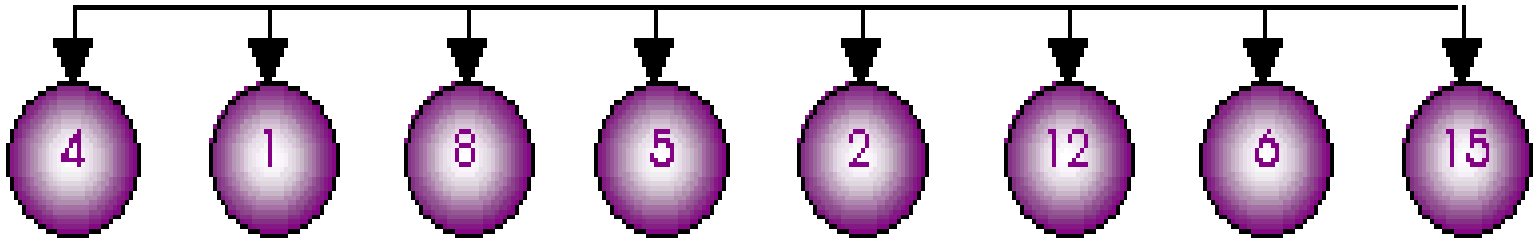


- h = 3 : (sau khi đã sắp xếp các dãy con ở bước trước)

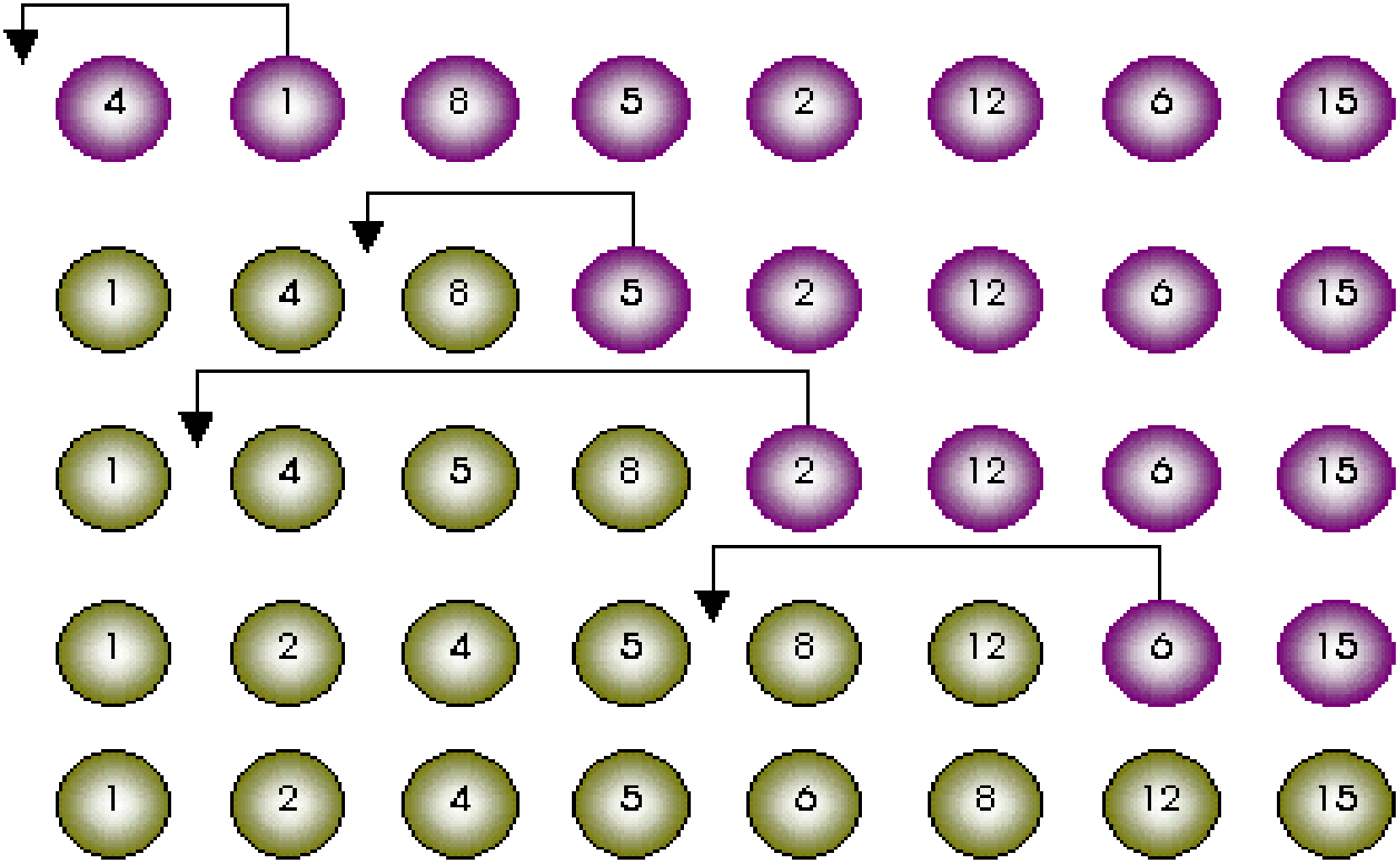


Shell Sort

- 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

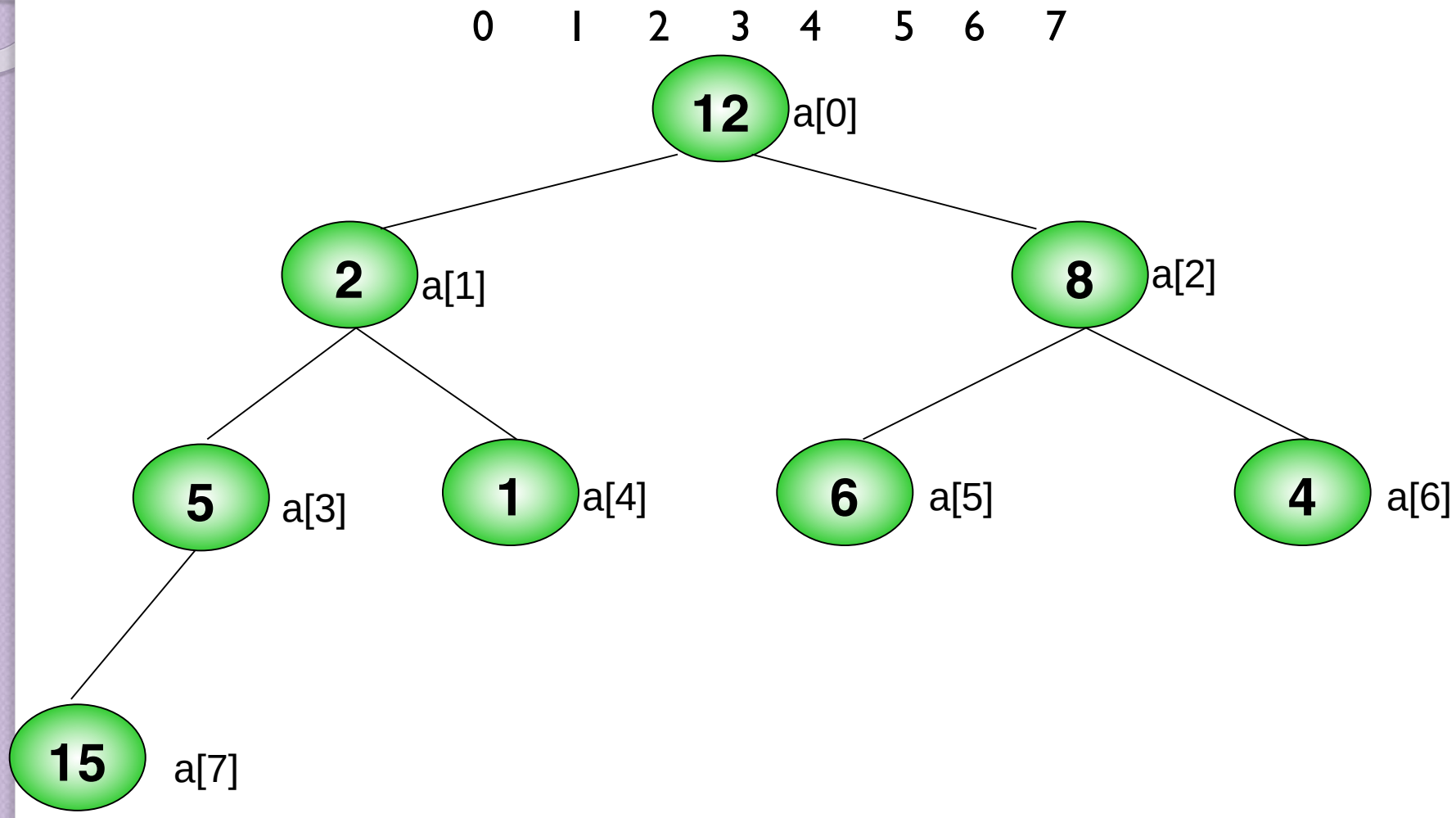
```
void ShellSort(int a[],int n, int h[], int k)
{  int   step,i,j, x,len;
   for (step = 0 ; step <k; step++)
   {     len = h[step];
         for (i = len; i<n; i++)
         {
                  x = a[i];
                  j = i-len; // a[j] đứng kề trước a[i] trong cùng dãy con
                  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;
         }
   }
}
```

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



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; 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 a1 , a2 ... ar thành một heap.
  - Bước 3:

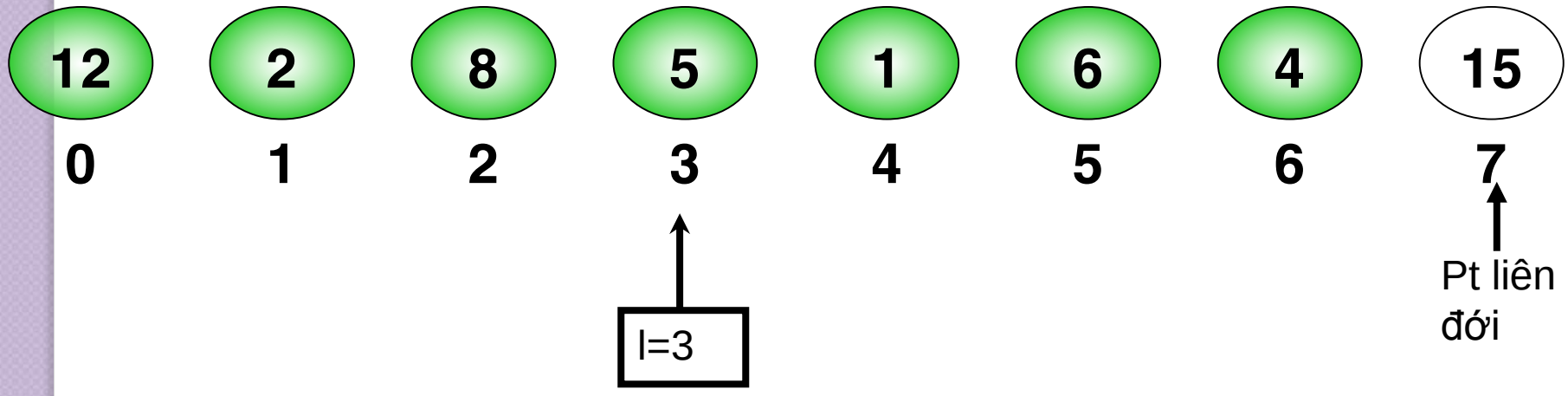
    Nếu r>1 (heap còn phần tử): Lặp lại Bước 2

    Ngược lại : Dừng

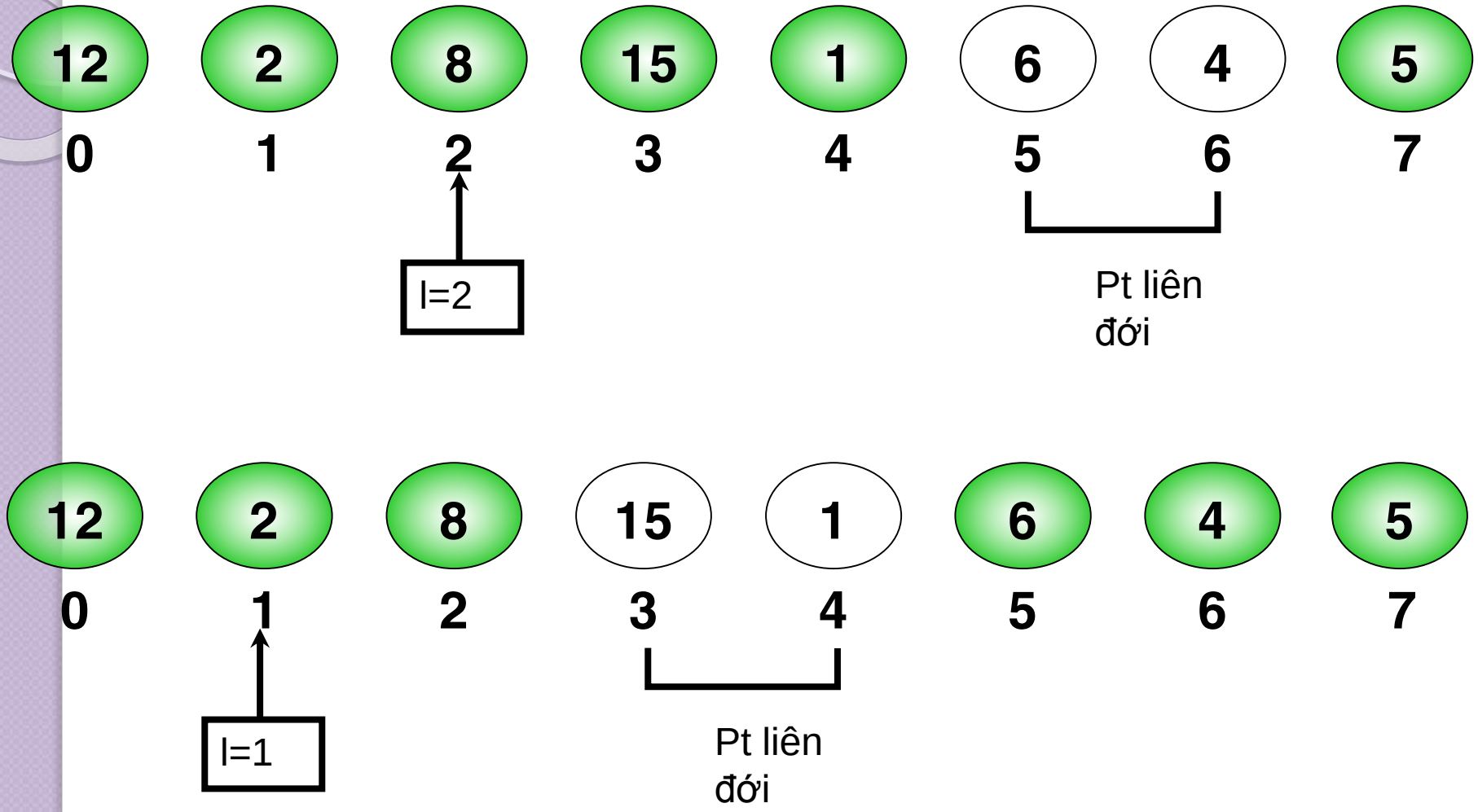
Minh Họa Thuật Toán

- **Heap:** Là một dãy các phần tử $a_l, a_{l+1}, \ldots, a_r$ 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

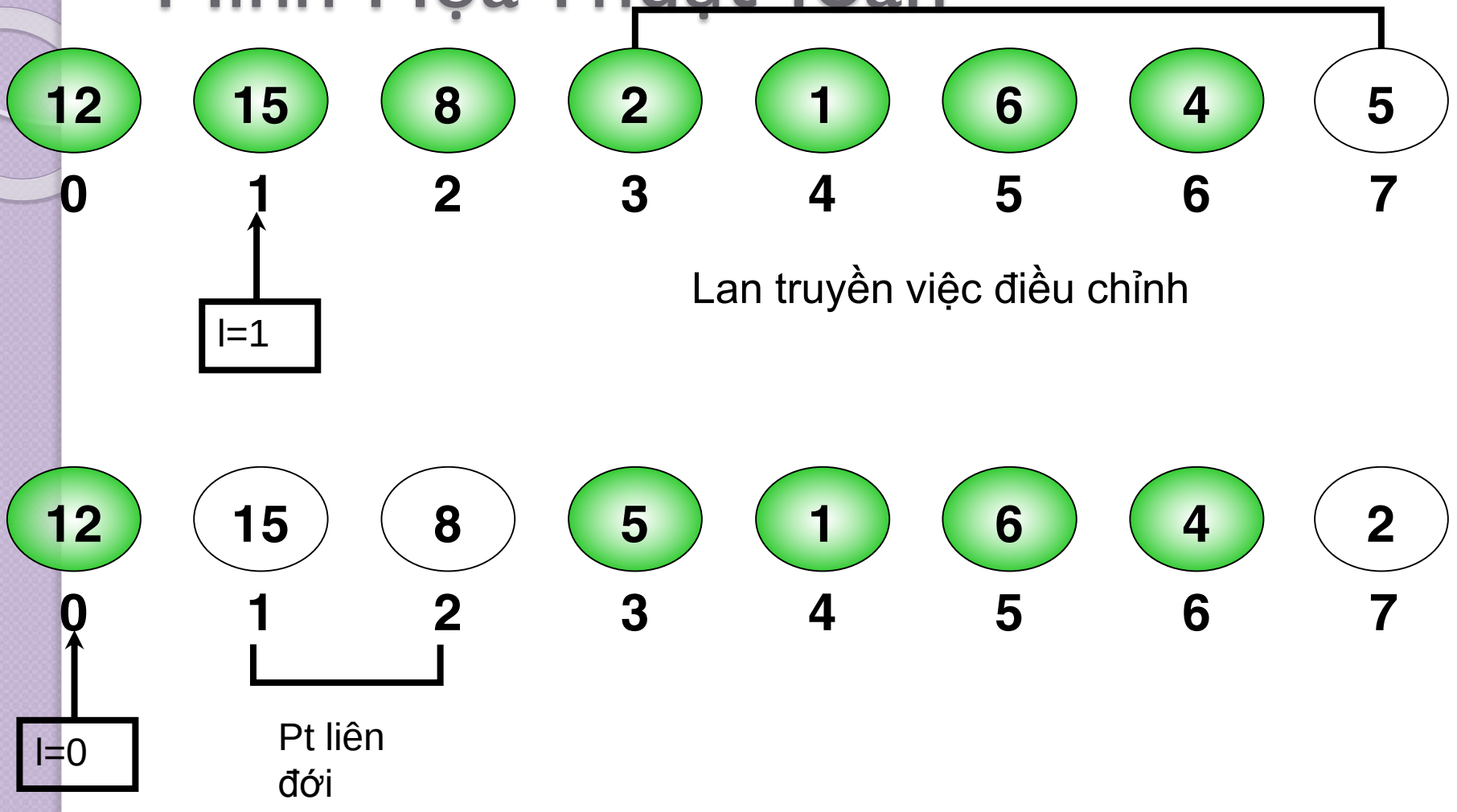
➢ Giai đoạn 1: Hiệu chỉnh dãy ban đầu thành Heap



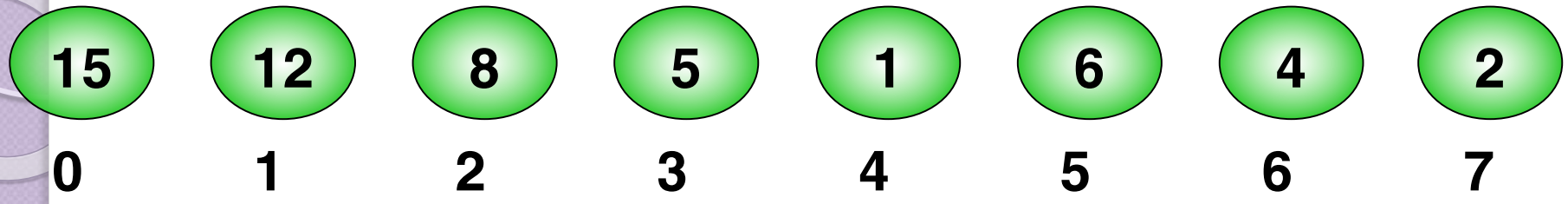
Minh Họa Thuật Toán



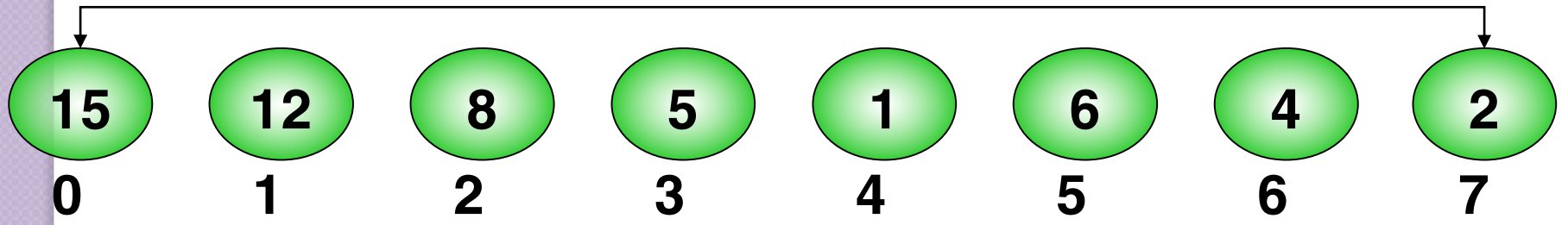
Minh Họa Thuật Toán

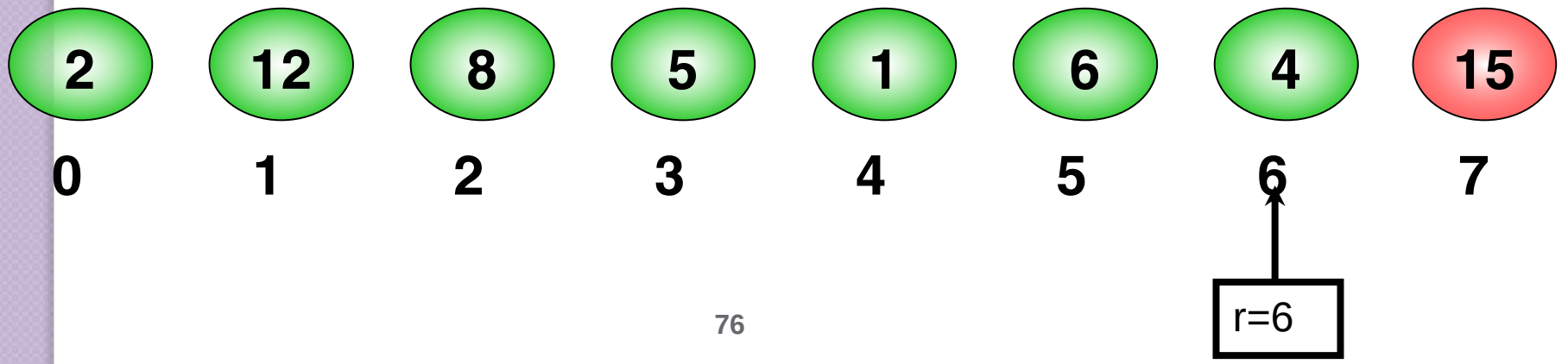


Minh Họa Thuật Toán



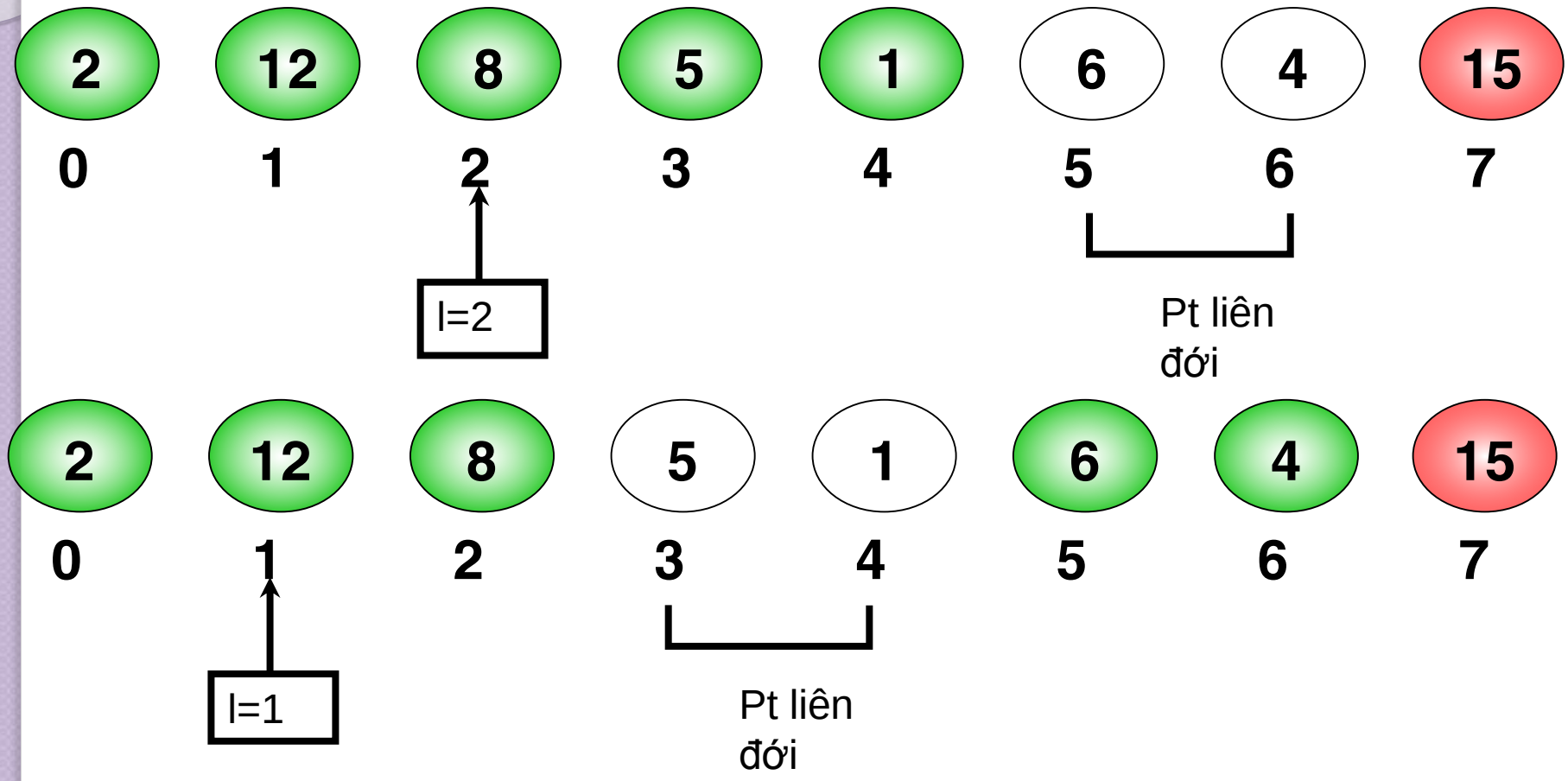
- Giai đoạn 2: Sắp xếp dãy số dựa trên Heap



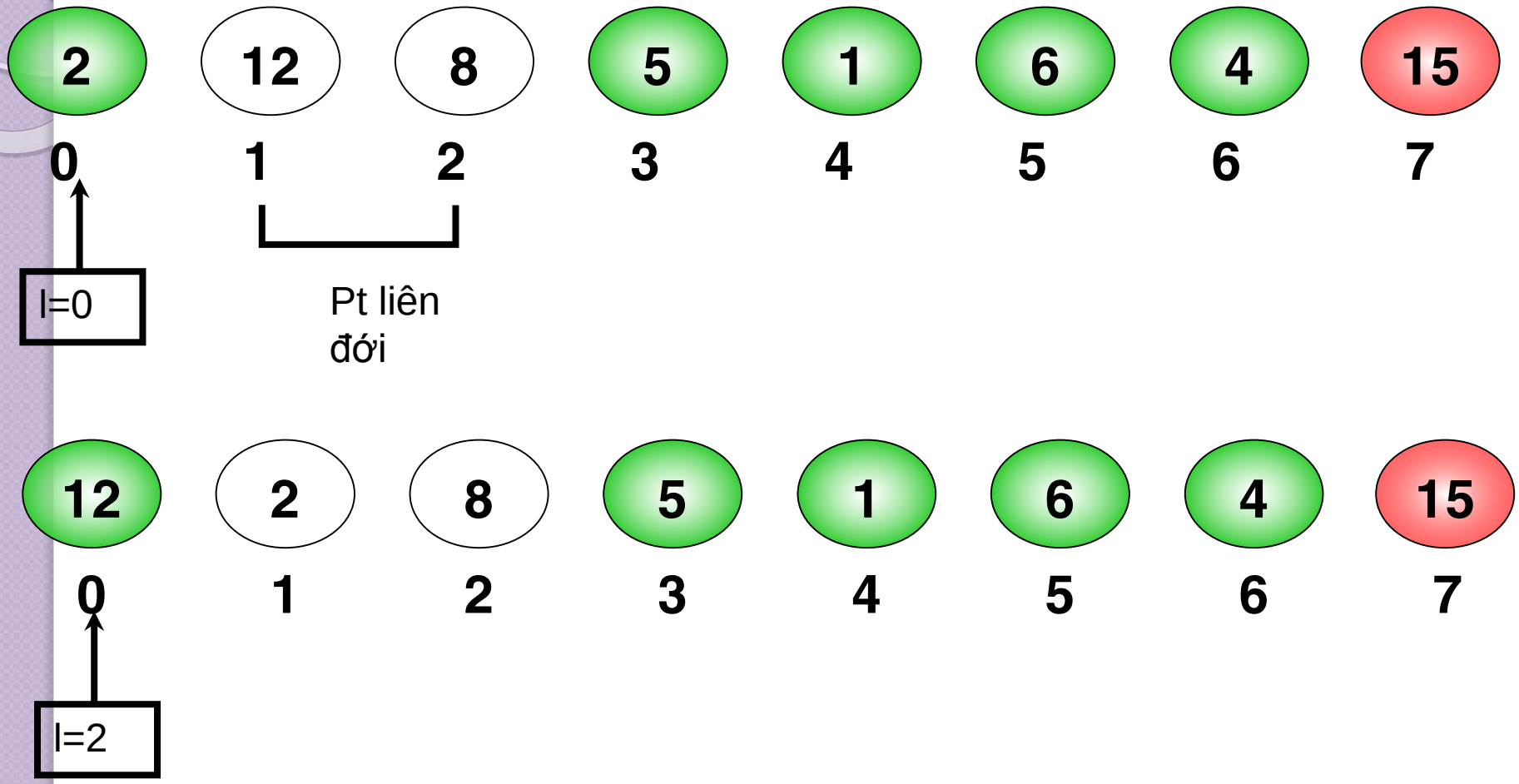


Minh Họa Thuật Toán

- Hiệu chỉnh Heap

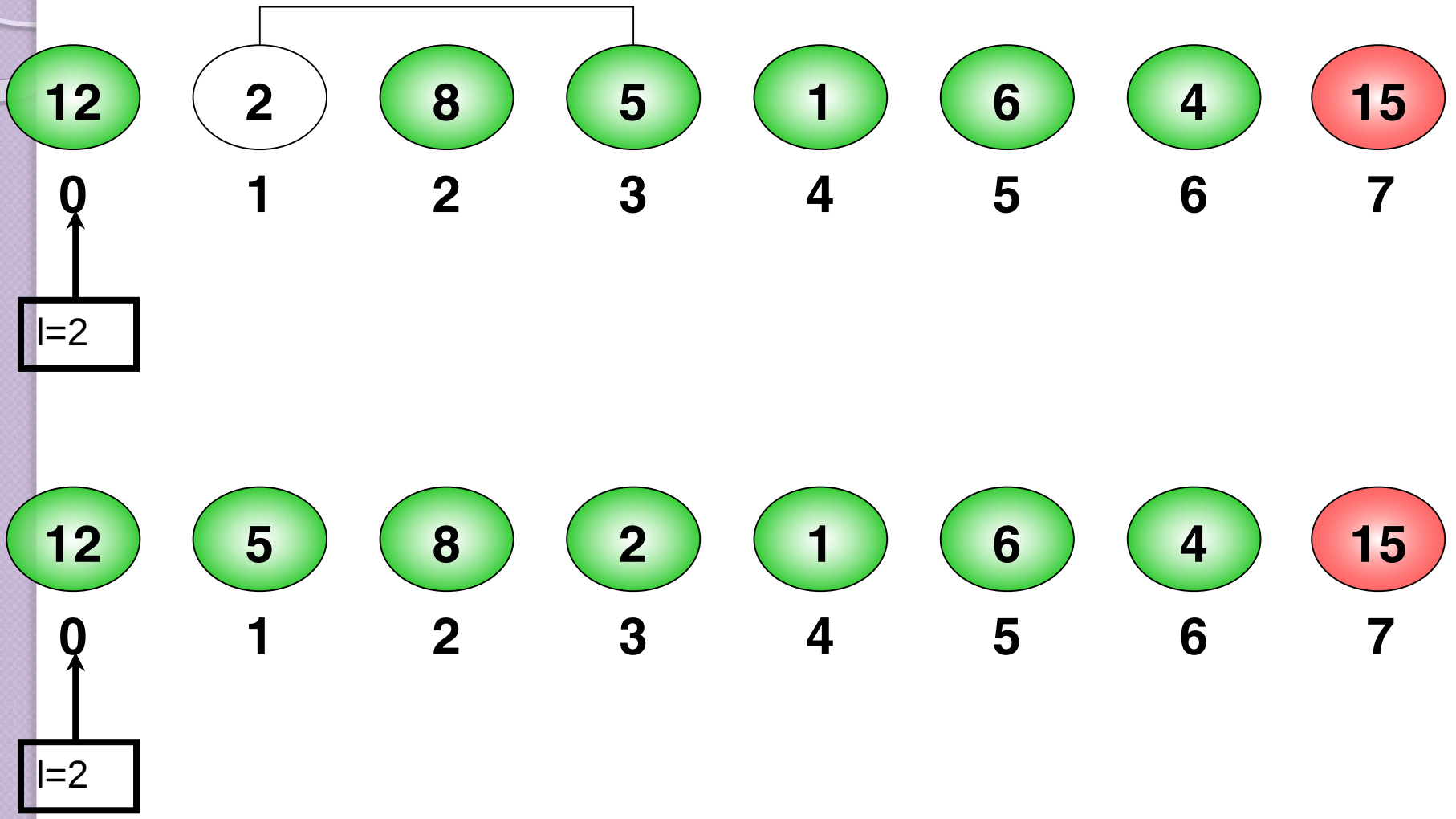


Minh Họa Thuật Toán

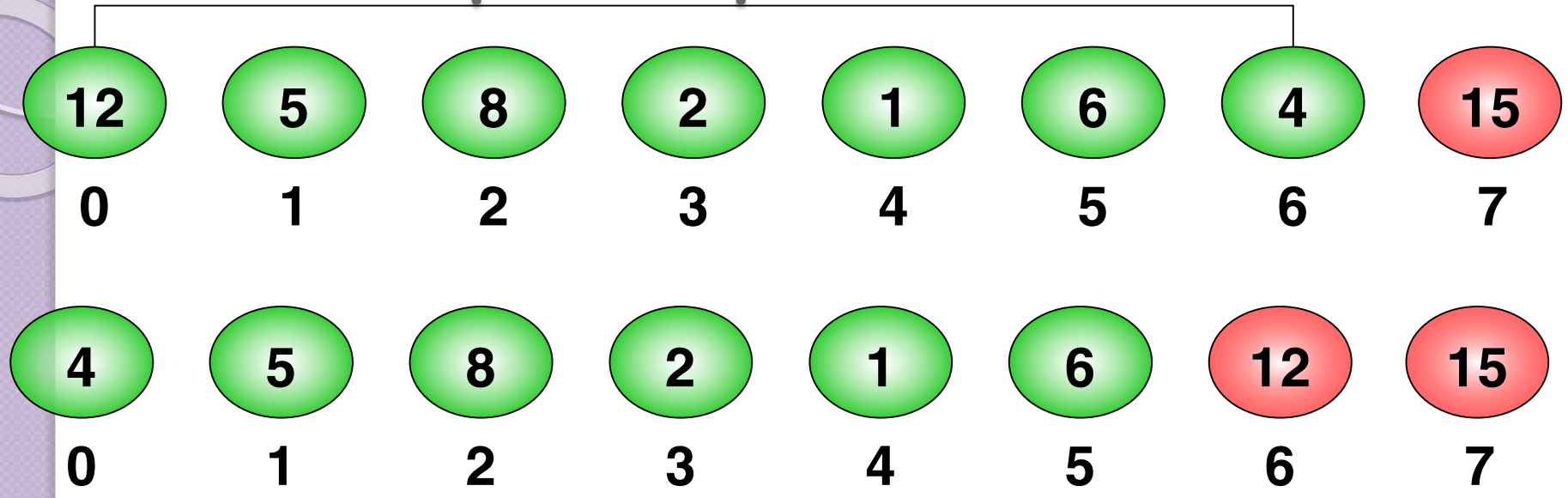


Minh Họa Thuật Toán

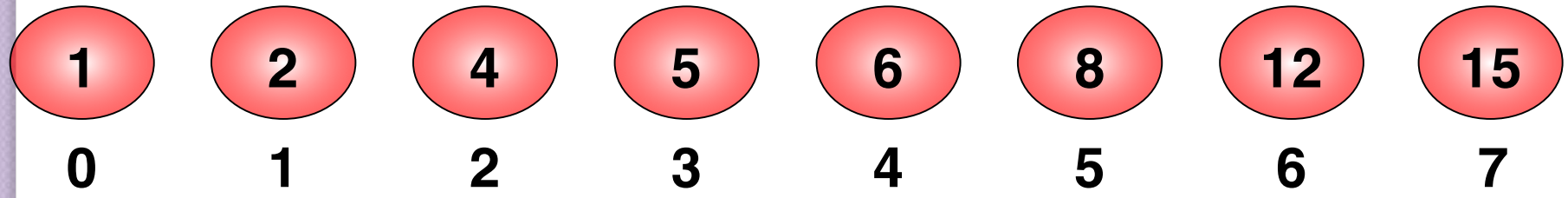
Lan truyền việc điều chỉnh



Minh Họa Thuật Toán



- Thực hiện với r= 5,4,3,2 ta được



Cài Đặt Thuật Toán

- Hiệu chỉnh $a_l, a_{l+1},..,a_r$ 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);
   }
}
```

Quick Sort

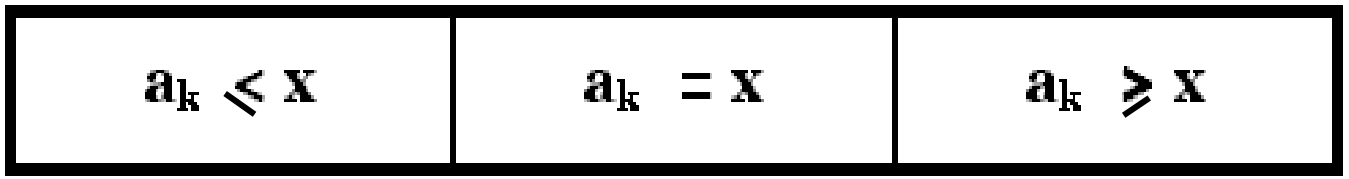
- Ý tưởng:

- Giải thuật QuickSort sắp xếp dãy $\mathbf{a_1, a_2 ..., a_N}$ 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

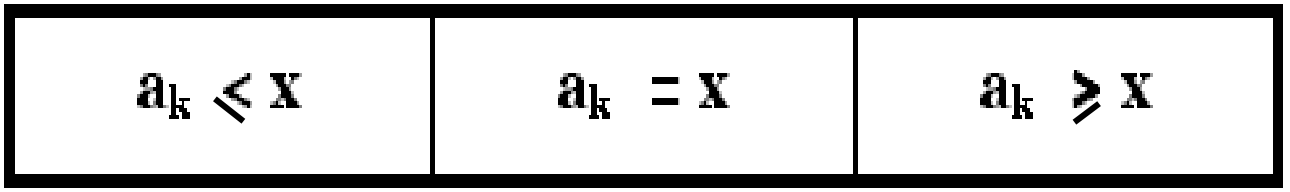
- Sau khi thực hiện phân hoạch, dãy ban đầu được phân thành 3 đoạn:
  - 1. $a_k \leq x$, với k = 1 .. j
  - 2. $a_k = x$, với k = j+1 .. i-1
  - 3. $a_k \geq x$, với k = i..N



  - Đoạn thứ 2 đã có thứ tự.
  - Nếu các đoạn 1 và 3 chỉ có 1 phần tử : đã có thứ tự

    → khi đó dãy con ban đầu đã được sắp.

Quick Sort



- 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 $a_{left} \dots a_{right}$ 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;

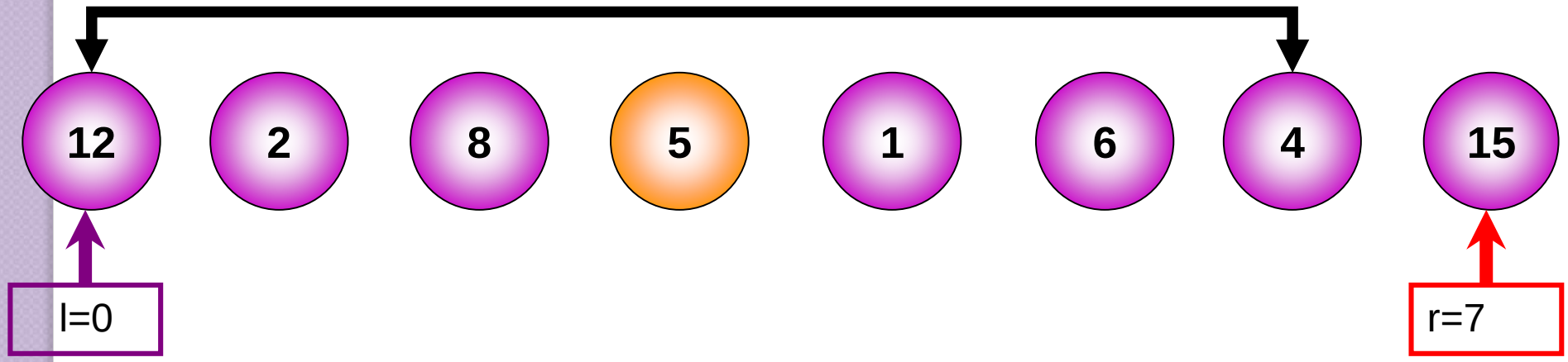
- Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :
  - Bước 2a : Trong khi (a[i]<x) i++;
  - Bước 2b : Trong khi (a[j]>x) j--;
  - Bước 2c : Nếu i< j Swap(a[i],a[j]);
- Bước 3 : Nếu i < j: Lặp lại Bước 2.
  Ngược lại: Dừng

Quick Sort – Ví Dụ

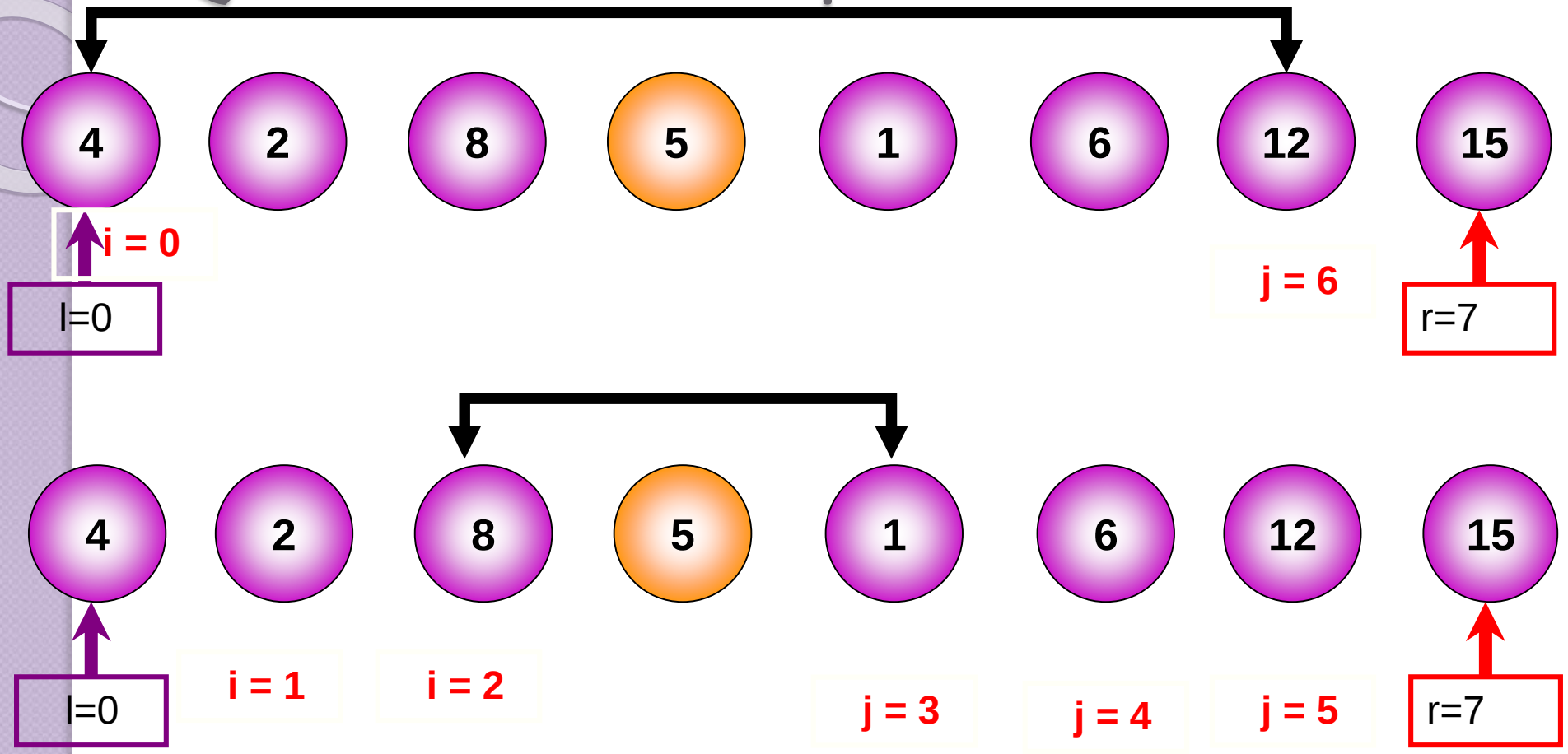
- Cho dãy số a:

12 2 8 5 1 6 4 15

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

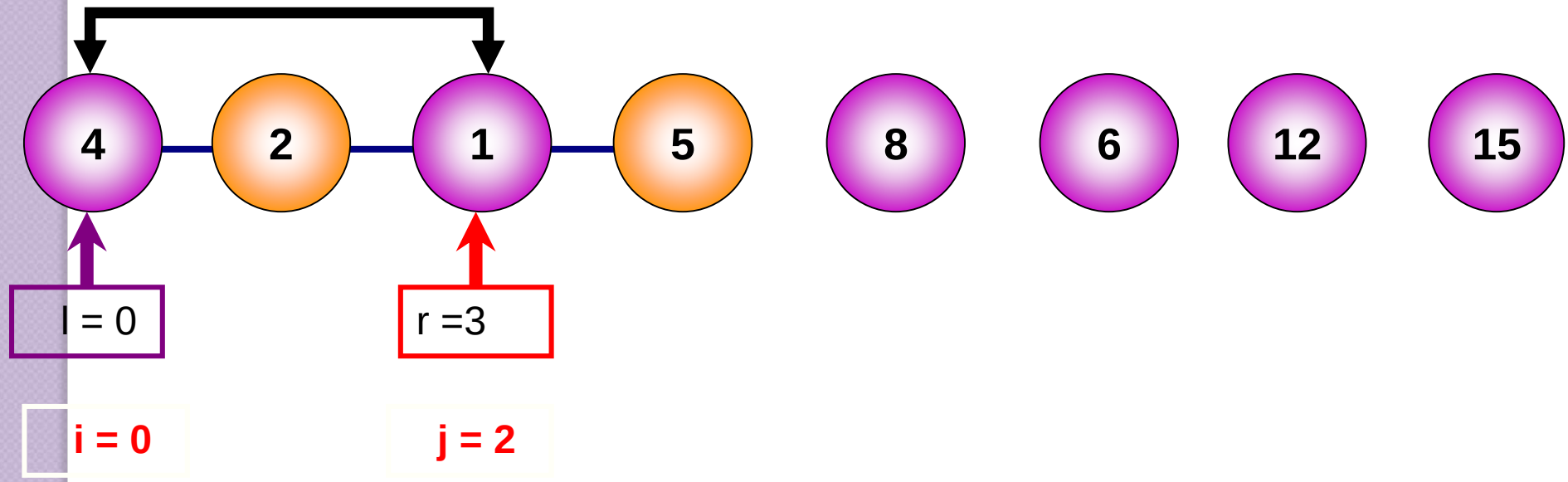


Quick Sort – Ví Dụ



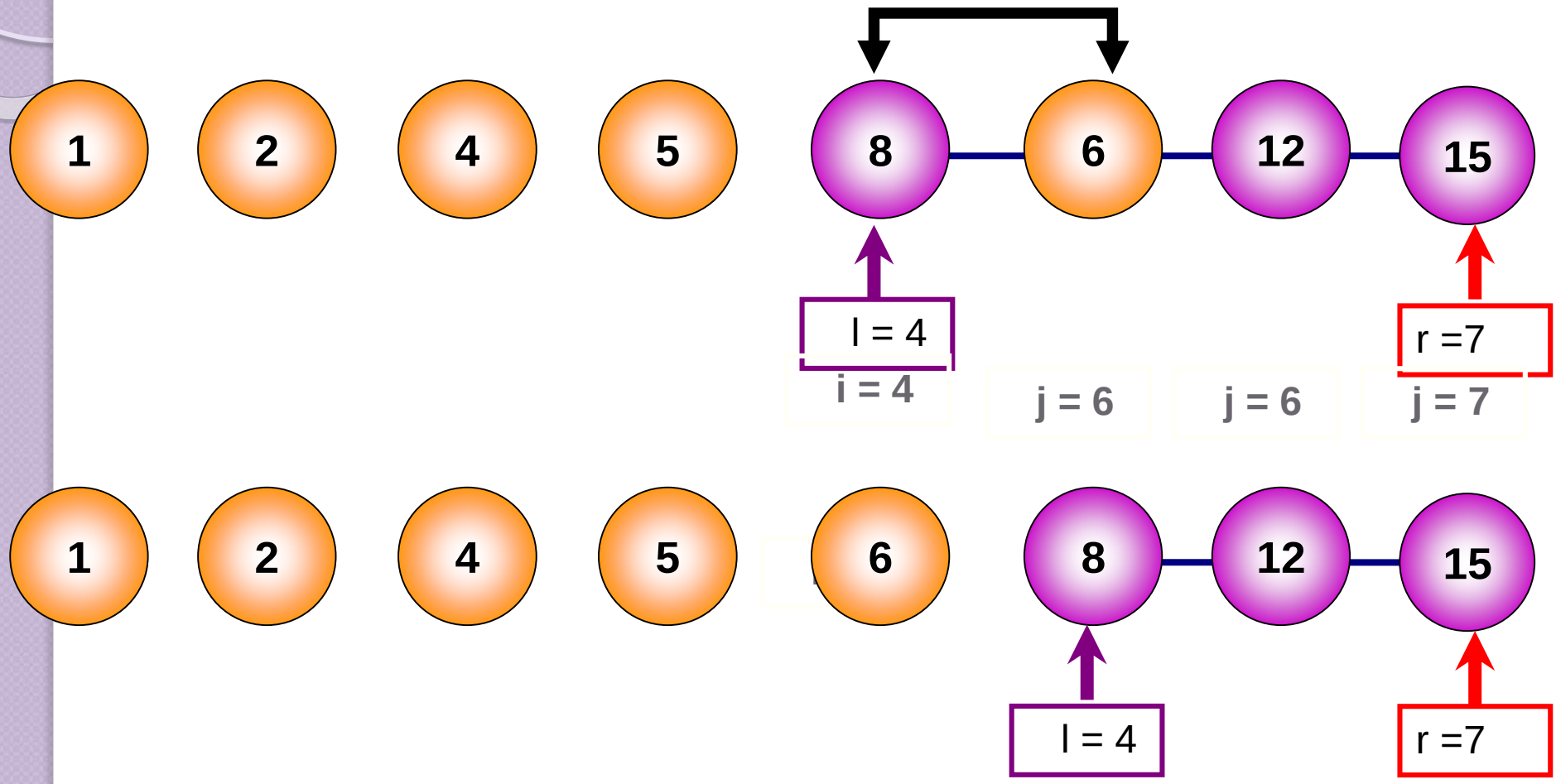
Quick Sort – Ví Dụ

- Phân hoạch đoạn l = 0, r = 2:



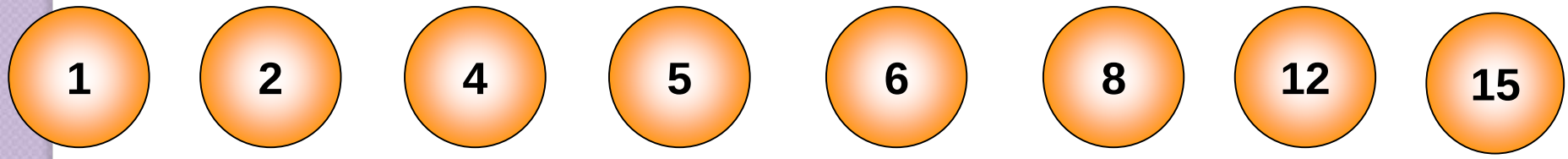
Quick Sort – Ví Dụ

- Phân hoạch đoạn l =4, r = 7:



Quick Sort – Ví Dụ

- Phân hoạch đoạn l =6, r = 7:



Quick Sort

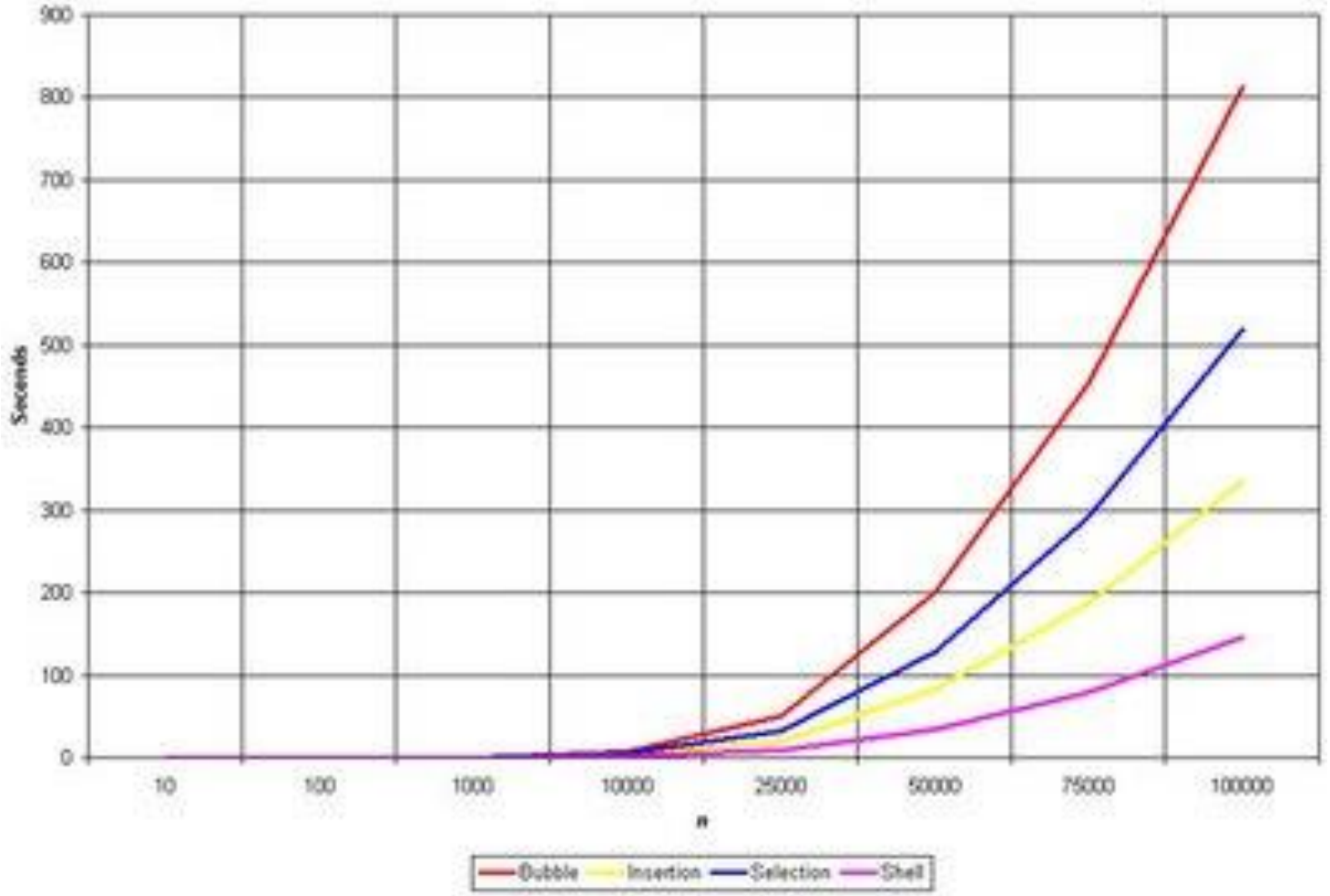
```
void QuickSort(int a[], int left, int right)
{  int   i, j, x;
   x = a[(left+right)/2];
   i = left; j = right;
   do
   {
        while(a[i] < x) i++;
        while(a[j] > x) j--;
                if(i <= j)
        {
                Swap(a[i],a[j]);
                i++ ; j--;
        }
   } while(i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

   if(left<j)
        QuickSort(a, left, j);
   if(i<right)
        QuickSort(a, i, right);
}
```

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

So sánh các thuật toán O(n^2)



So sánh các thuật toán O(nlog2n)