NỘI DUNG

CÁC THUẬT TOÁN TÌM KIẾM

Bài toán tìm kiếm

- Input: Cho mảng a có n phần tử
- X: Giá trị cần tìm
- Output: Tìm phần tử có giá trị = x có hay không trong mảng

=> Hai thuật toán tìm kiếm:

- Tìm kiếm tuần tự (áp dụng trên mọi mảng)
- Tìm kiếm nhị phân (áp dụng trên mảng đã có thứ tự)

Thuật toán tìm kiếm tuyến tính

- **Ý tưởng** :Lần lượt so sánh X với từng phần tử trong A cho đến khi tìm thấy hay hết phần tử trong mảng.
- **Các bước tiến hành**

Bước 1: Khởi gán i=1

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

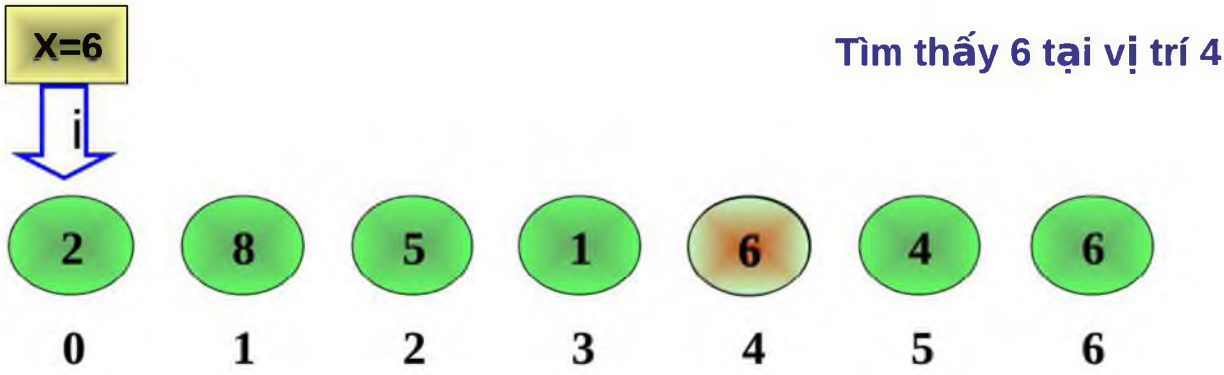
Nếu i >N: Hết mảng. Dừng

Ngược lại: Lặp lại bước 2

Thuật toán tìm kiếm tuyến tính


```
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
}
```

Thuật toán tìm kiếm tuyến tính



Tìm thấy 6 tại vị trí 4

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

- Ý tưởng:
  - So sánh khóa cần tìm với phần tử giữa dãy hiện hành.
  - Nếu nó nhỏ hơn thì tìm bên trái dãy hiện hành.
  - Ngược lại tìm bên phải dãy hiện hành.
  - Lặp lại động tác này.
- Dãy hiện hành là dãy ta tang tìm, chỉ số đầu tiên của phần tử đầu tiên trong dãy là left, và chỉ số của phần tử cuối cùng trong dãy hiện hành là right

Các bước thuật toán tìm kiếm nhị phân

- **Bước 1**: left=1; right=N;//tìm kiếm trên tất cả các phần tử
- **Bước 2**:
  - mid=(left+right)/2;//chỉ số của phầ tử đứng giữa trong dãy hiện hành
  - So sánh a[mid] với x. Có 3 khả năng có:
    - a[mid]= x: tìm thấy. Dừng
    - a[mid]>x
      + Right= mid-1;//Tìm tiếp x trong dãy con a[Left]..a[mid-1]
    - a[mid]<x
      + Left= mid+1;//Tìm tiếp x trong dãy con a[mid+1]..a[right]
- **Bước 3**: Nếu Left <=Right ;//còn phần tử trong dãy hiện hành

  + Lặp lại bước 2

  Ngược lại : Dừng

Thuật toán tìm nhị phân

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

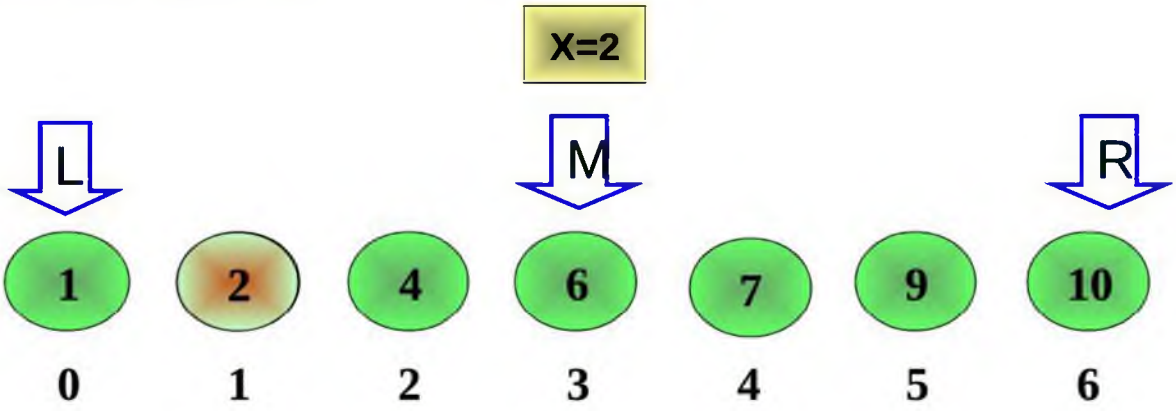
Độ phức tạp thuật toán tìm nhị phân

Tröôøng hôïp	C_{ss}
Tốt nhaát	1
Xaáu nhaát	$\log_2 N$
Trung bình	$\log_2 N / 2$

- Ñoä phöùc taïp $O(\log_2 N)$

Ví dụ thuật toán tìm nhị phân

Tìm thấy 2 tại vị trí 1



NỘI DUNG

CÁC THUẬT TOÁN SẮP XẾP

Sắp xếp

- Cho tập N phần tử có m thuộc tính, được biểu diễn dưới dạng bản ghi.
- Dựa vào 1 (hoặc vài) thuộc tính để sắp xếp các phần tử theo trật tự mới

Sắp xếp

- Gồm 2 bài toán con:
  - Dựa theo khoá sắp xếp định vị lại thứ tự bản ghi
  - Chuyển các bản ghi về vị trí mới.
- Hai thao tác cơ bản
  - So sánh
  - Gán

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

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

Đổi chỗ trực tiếp – Interchange Sort

- ***Khái niệm nghịch thế:***
  - *Xét một mảng các số a_0, a_1, . a_n.*
  - *Nếu có $i<j$ và $a_i > a_j$, thì ta gọi đó là một nghịch thế.*
- Mảng chưa sắp xếp sẽ có nghịch thế
- Mảng đã có thứ tự sẽ không chứa nghịch thế

Đổi chỗ trực tiếp – Interchange Sort

- Tìm tất cả nghịch thế, triệt tiêu chúng bằng cách hoán vị 2 phần tử tương ứng trong nghịch thế

Đổi chỗ trực tiếp – Interchange Sort

- Bước 1 : i = 1;// bắt đầu từ đầu dãy
- Bước 2 : j = i+1;//tìm các phần tử a[j] < a[i], j>i
- Bước 3 :

```
Trong khi j < N thực hiện
    Nếu a[j]<a[i] //xét cặp a[i], a[j]
        Doicho(a[i],a[j]);
j = j+1;
```

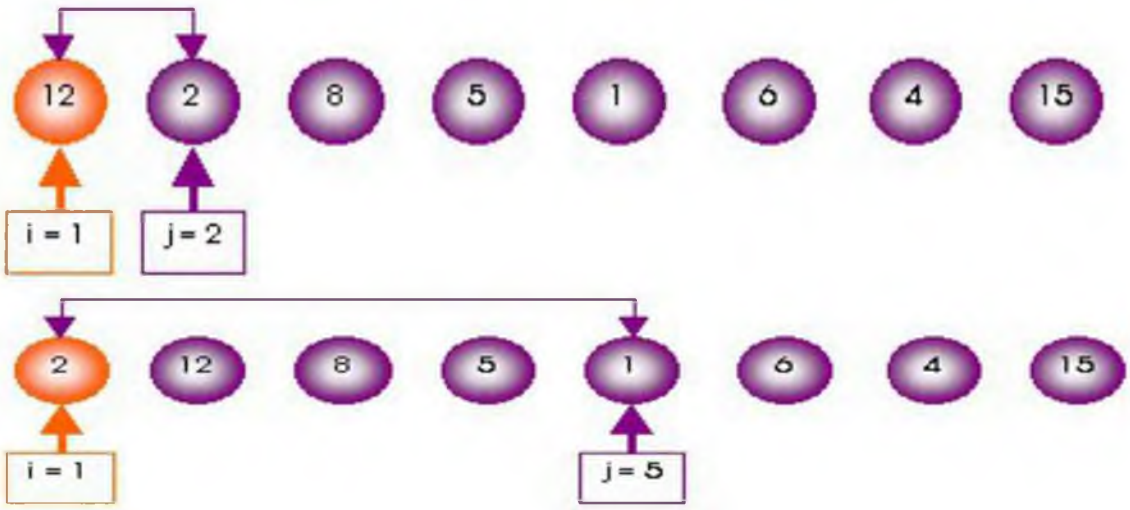
- Bước 4 : i = i+1;

```
Nếu  i < N: Lặp lại Bước 2.
Ngược lại:  Dừng.
```

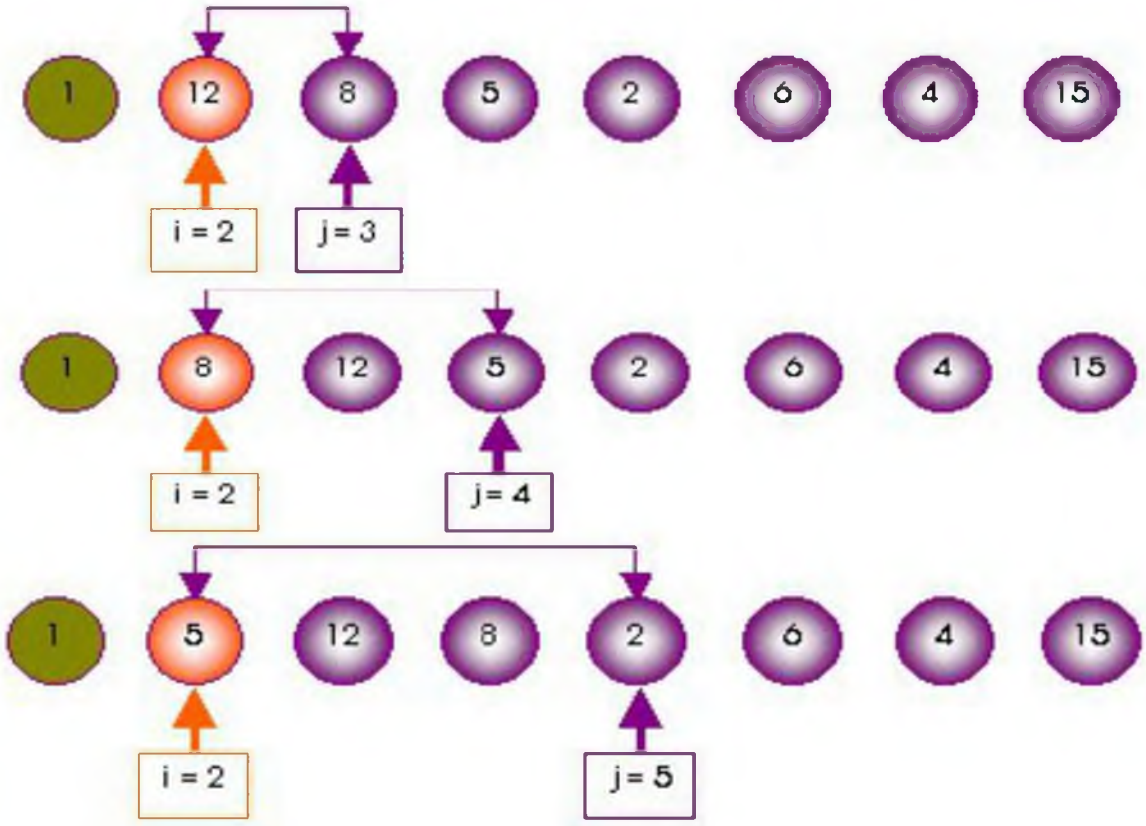
Đổi chỗ trực tiếp – Interchange Sort

- Cho dãy số a:

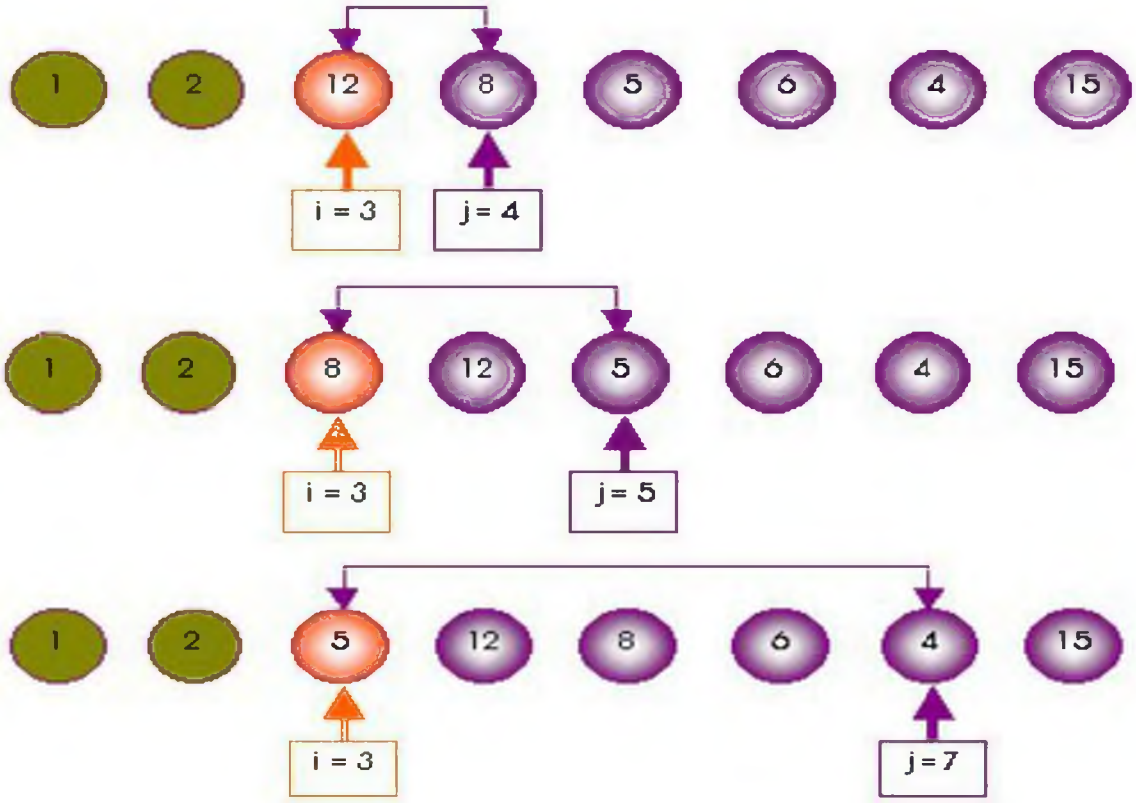
12 2 8 5 1 6 4 15



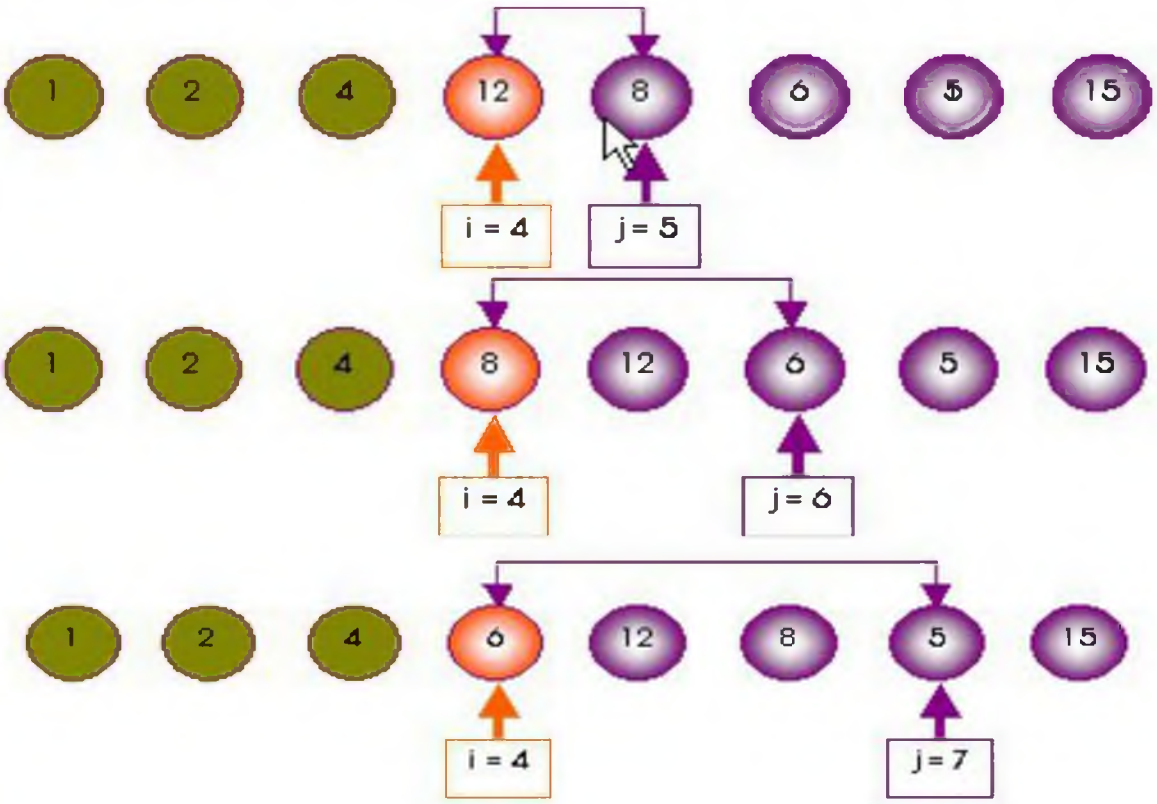
Đổi chỗ trực tiếp – Interchange Sort



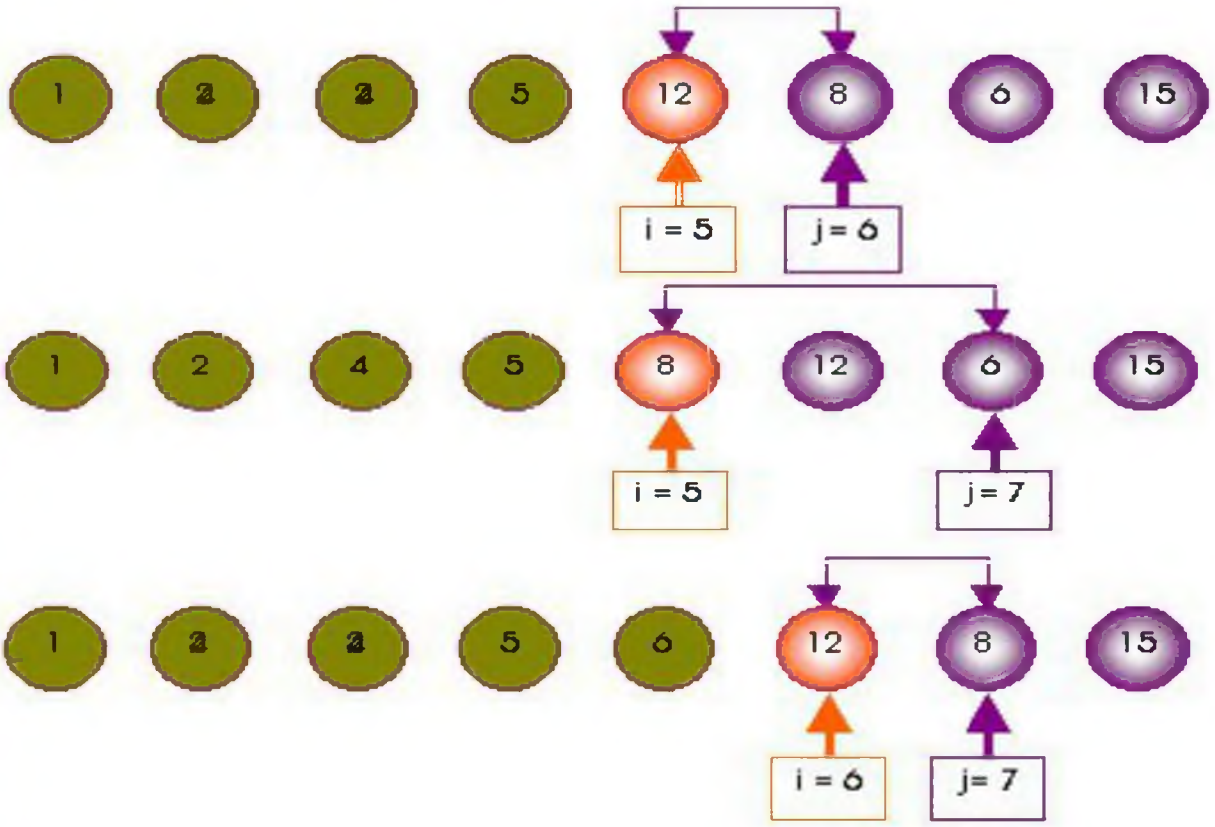
Đổi chỗ trực tiếp – Interchange Sort



Đổi chỗ trực tiếp – Interchange Sort



Đổi chỗ trực tiếp – Interchange Sort

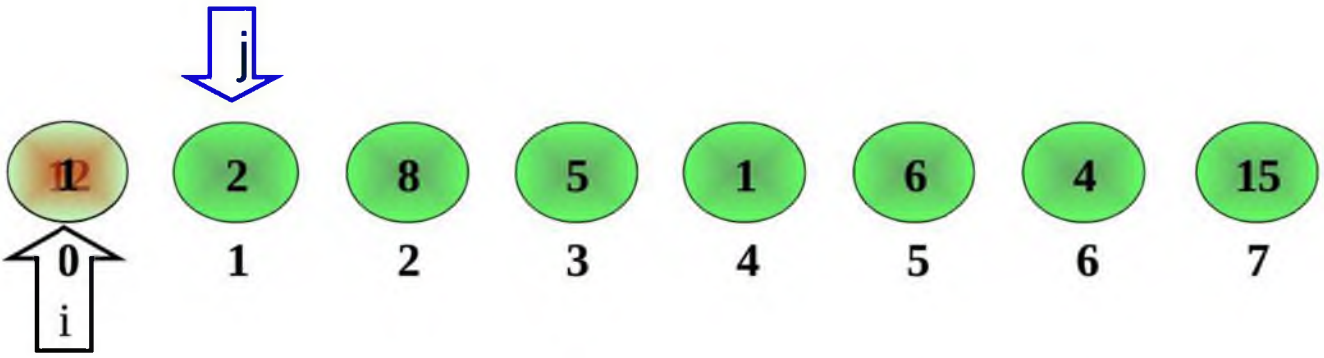


Đổi chỗ trực tiếp – Interchange Sort

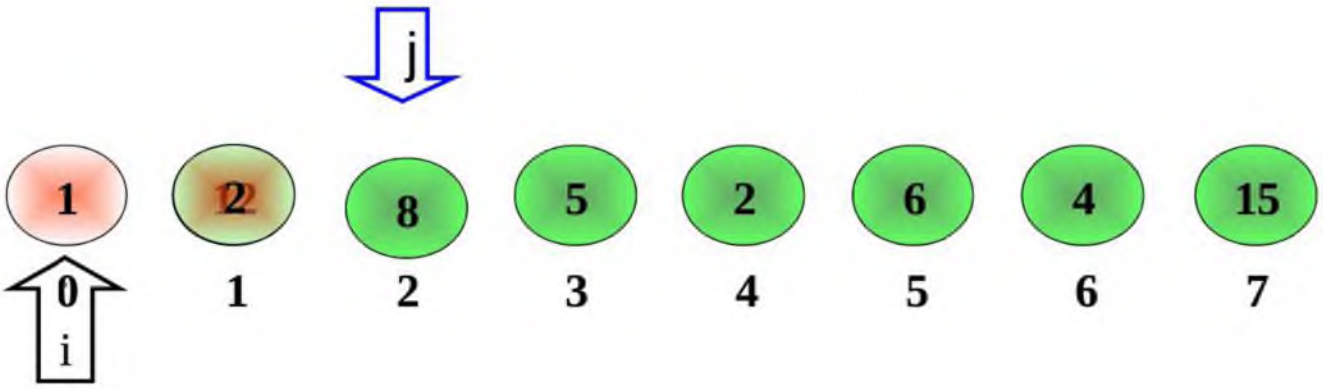
Đổi chỗ trực tiếp – Interchange Sort

```
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])      // nếu có sự sai vị trí thì đổi chỗ
                Doicho(a[i],a[j]);
}
```

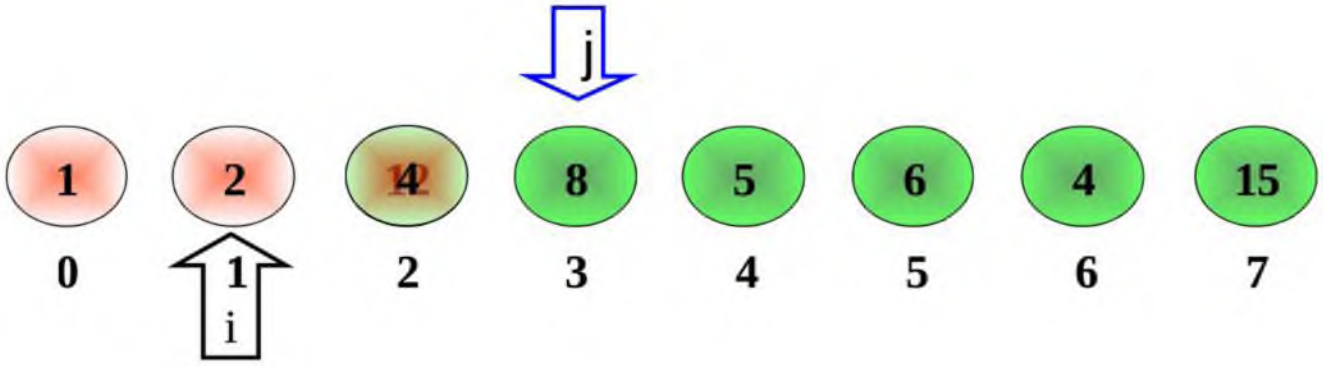
Interchange Sort – Ví dụ



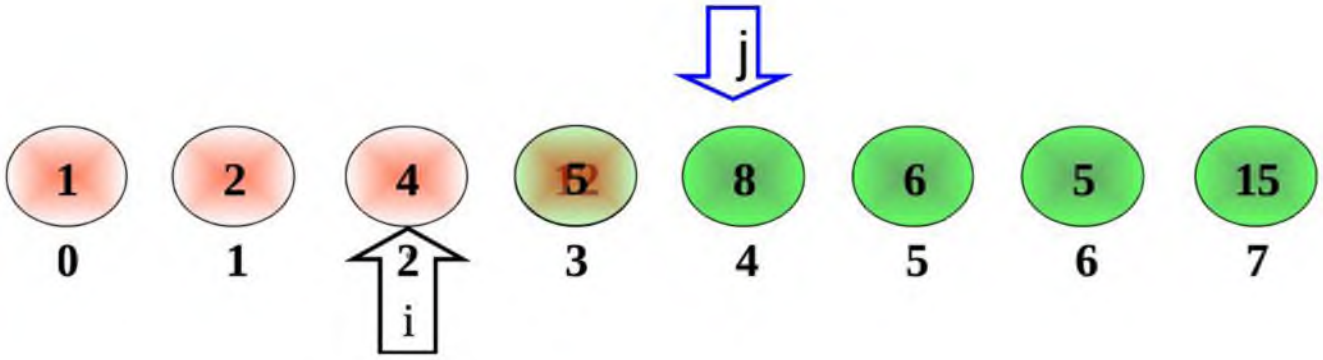
Interchange Sort – Ví dụ



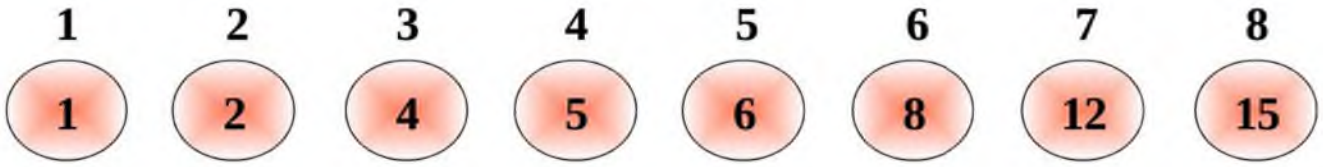
Interchange Sort – Ví dụ



Interchange Sort – Ví dụ



Interchange Sort – Ví dụ



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. **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

Nổi bọt – Bubble Sort

- Ý tưởng chính của giải thuật là 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 .

Nổi bọt – Bubble Sort

- Bước 1 : i = 1; *// lần xử lý đầu tiên*
- Bước 2 : j = N; *//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]);*//xét cặp phần tử kế cận*

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

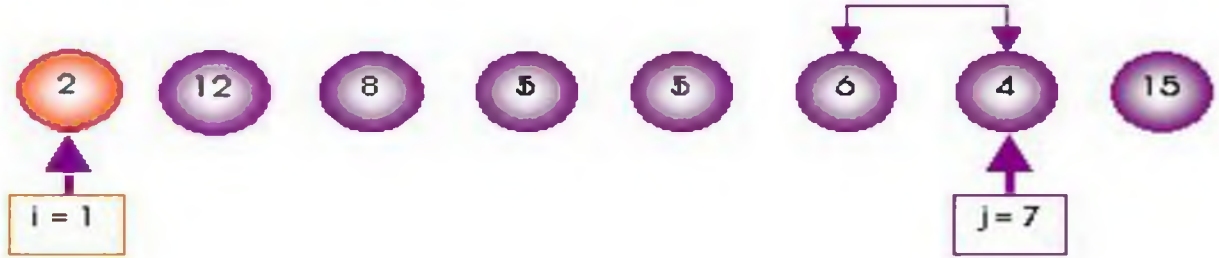
  Nếu i >N-1: Hết dãy. Dừng

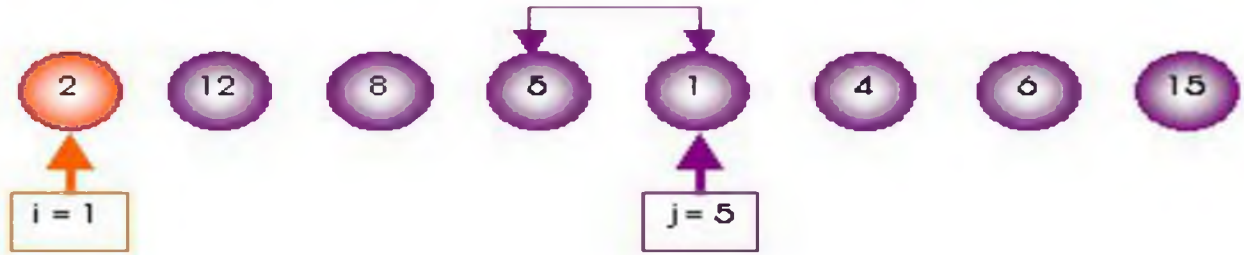
  Ngược lại : Lặp lại Bước 2.

Nổi bọt – Bubble Sort

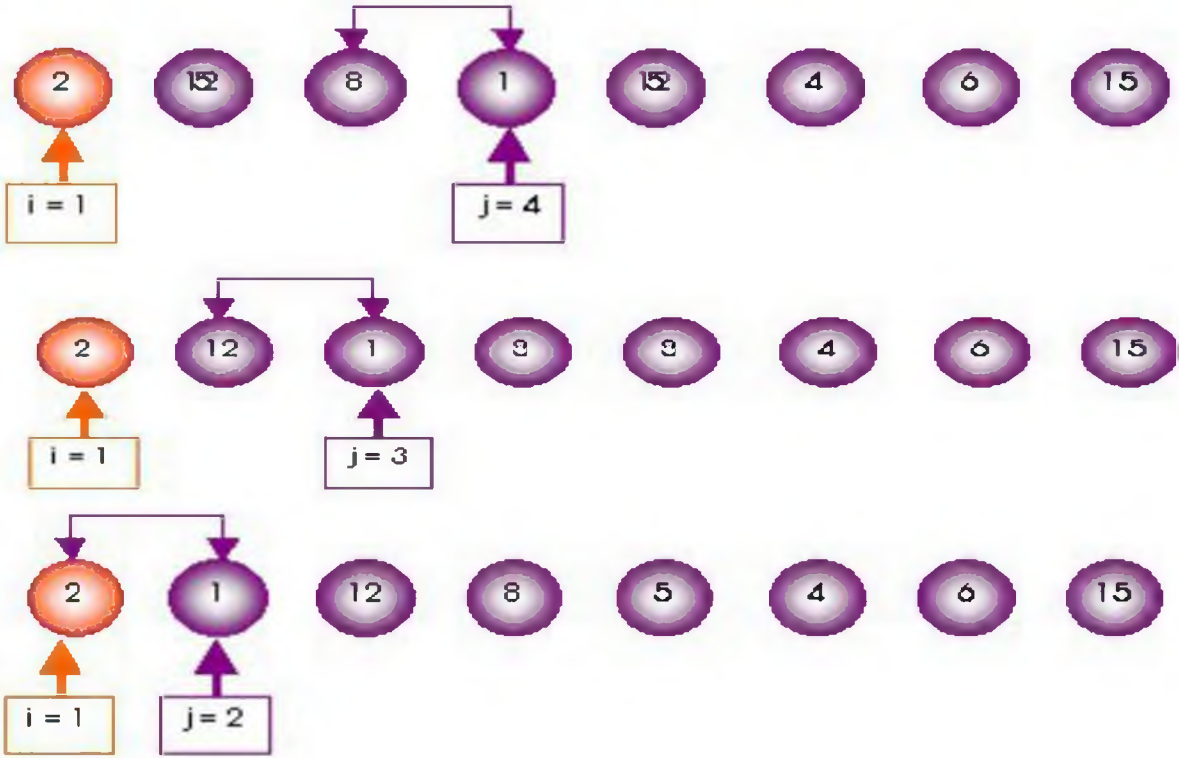
- Cho dãy số a:

2 12 8 5 1 6 4 15

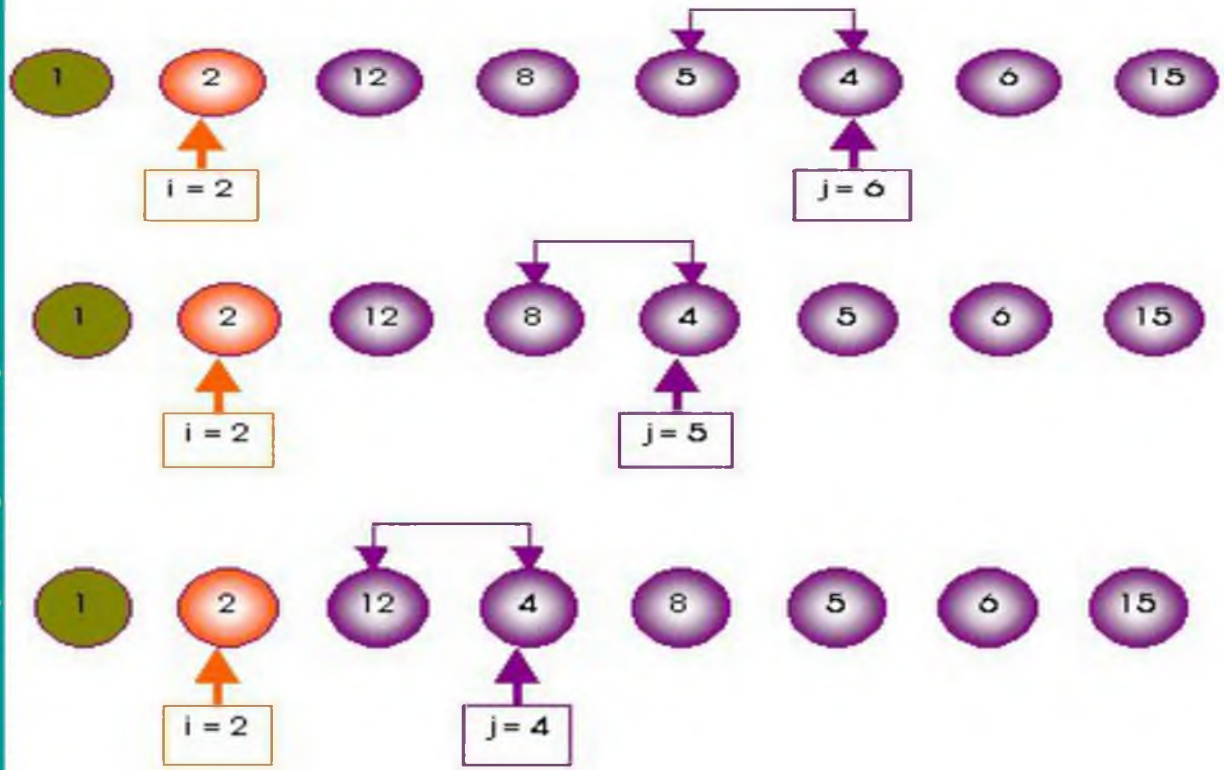




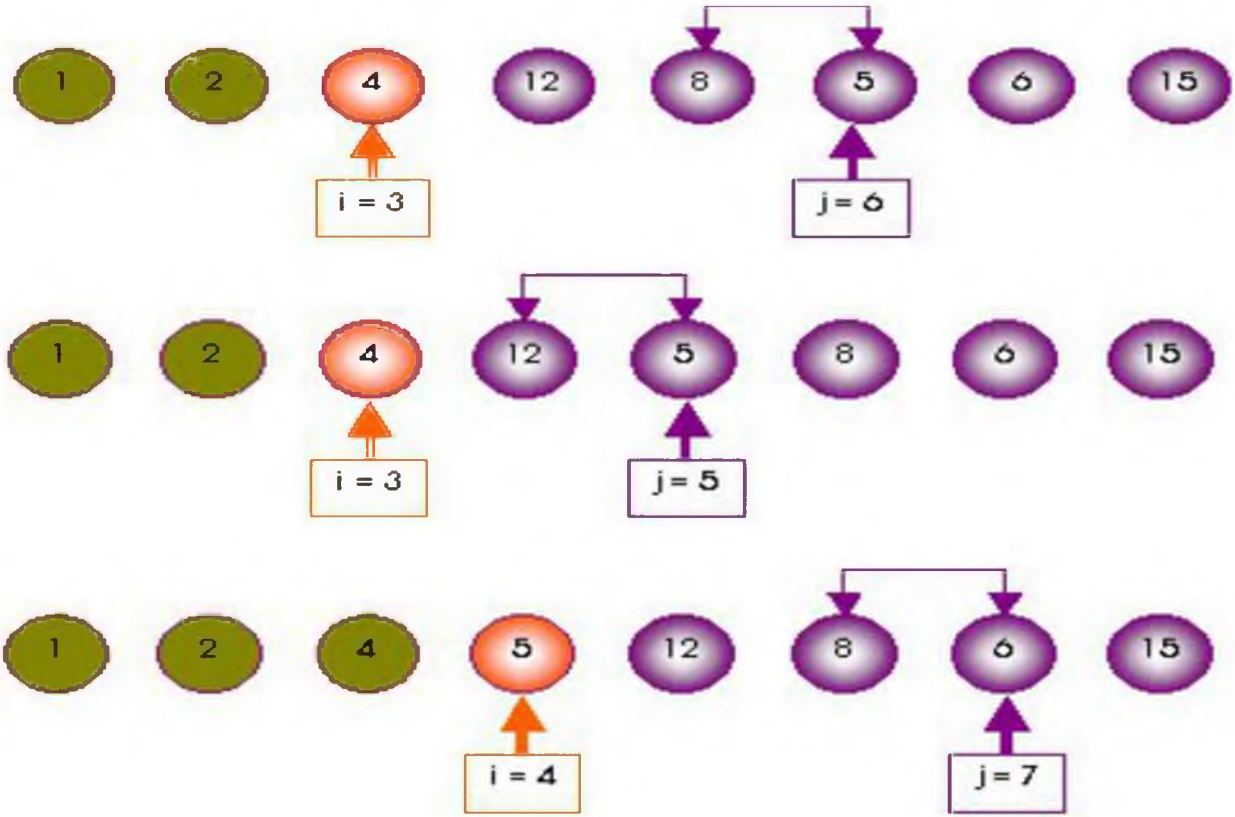
Nổi bọt – Bubble Sort



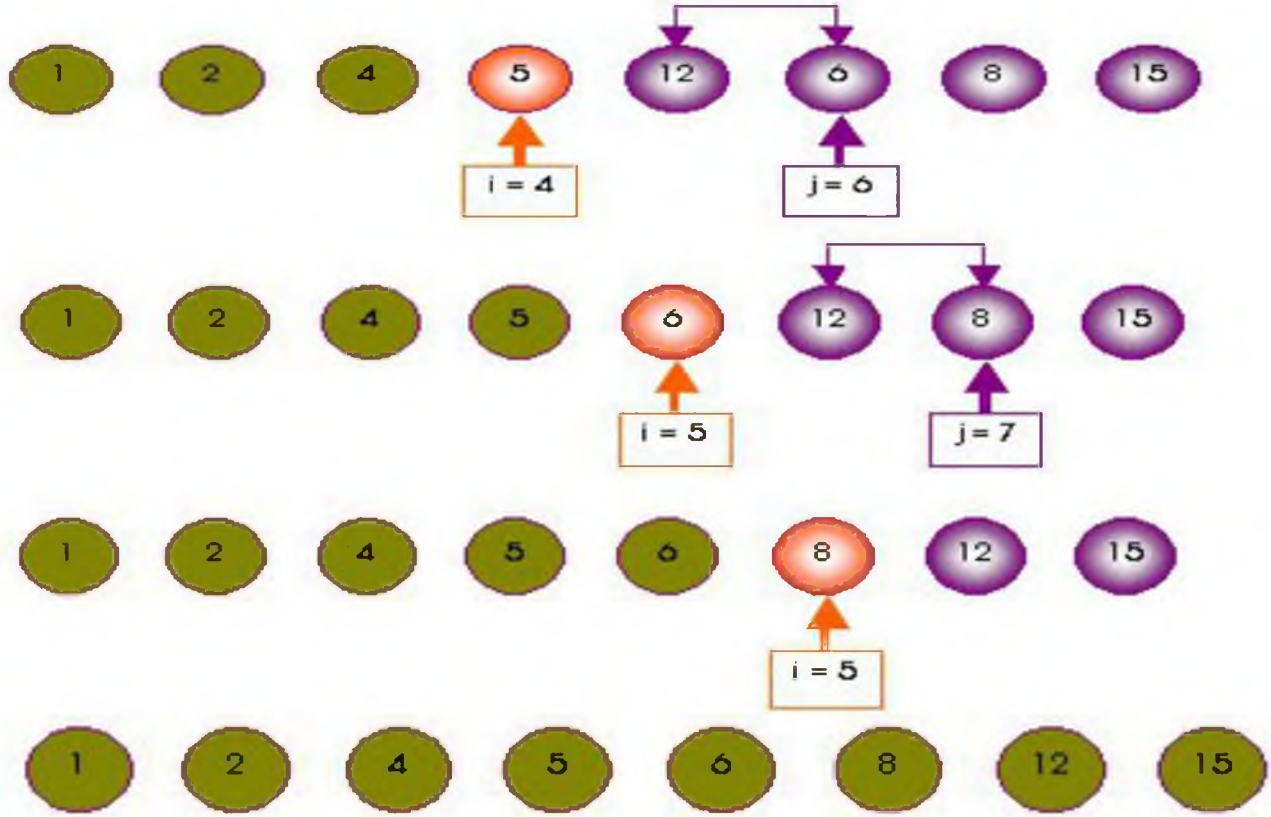
Nổi bọt – Bubble Sort



Nổi bọt – Bubble Sort



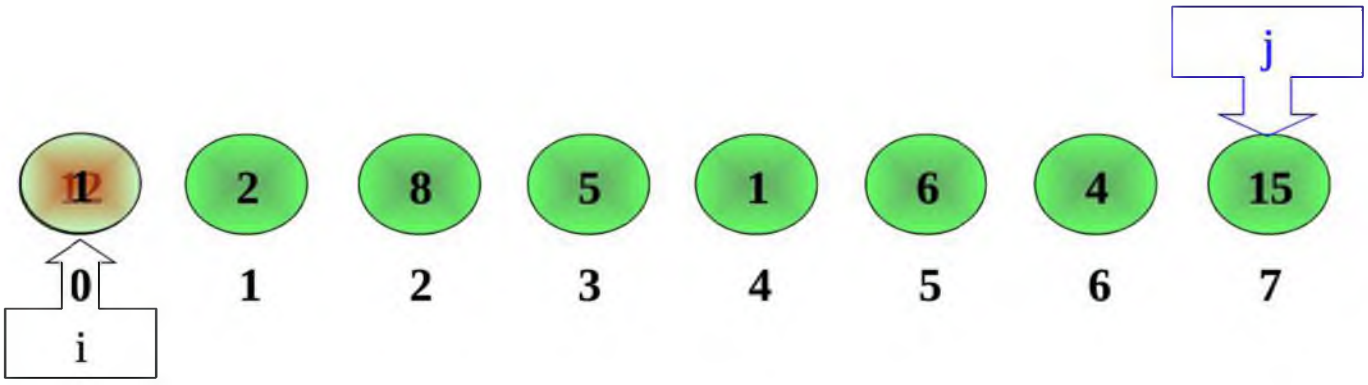
Nổi bọt – Bubble Sort



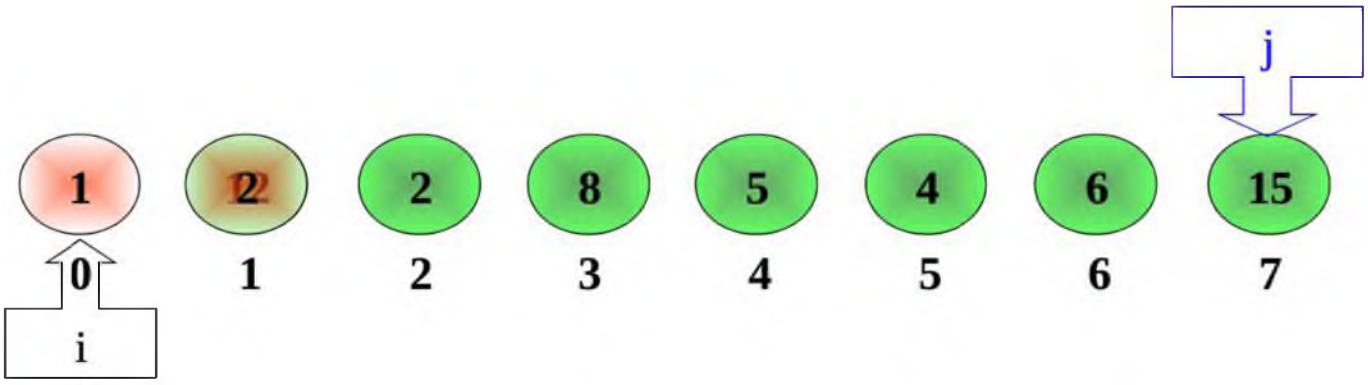
Nổi bọt – Bubble Sort

```
void BubleSort(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
                            Doicho(a[j], a[j-1]);
}
```

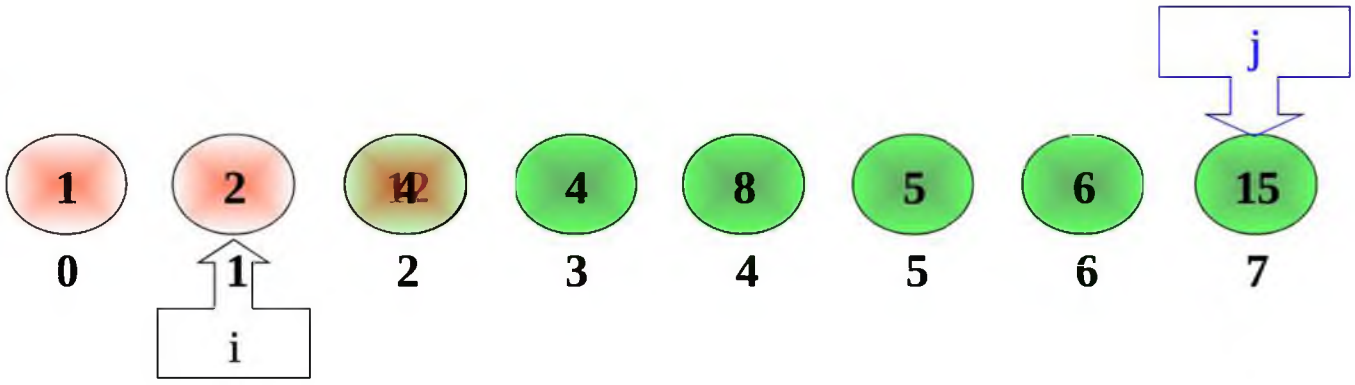
Bubble Sort – Ví dụ



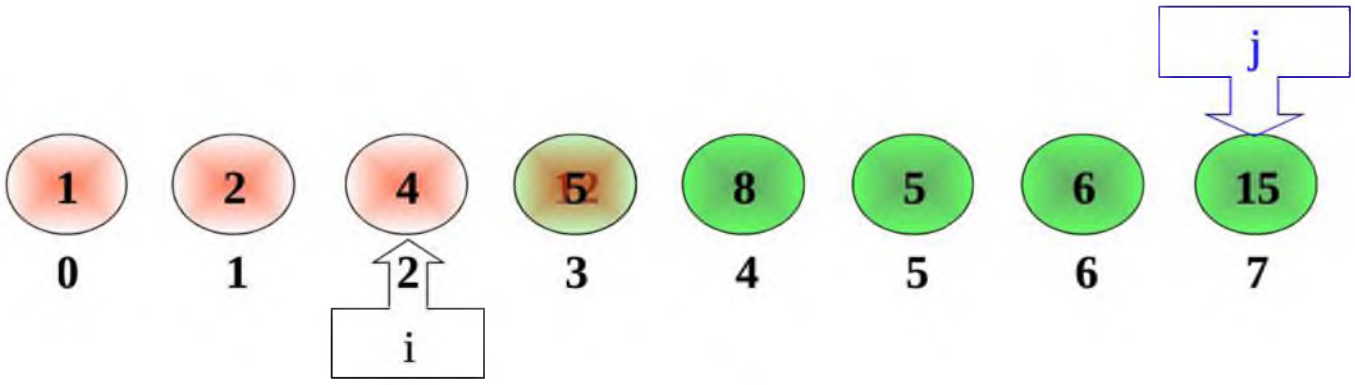
Bubble Sort – Ví dụ



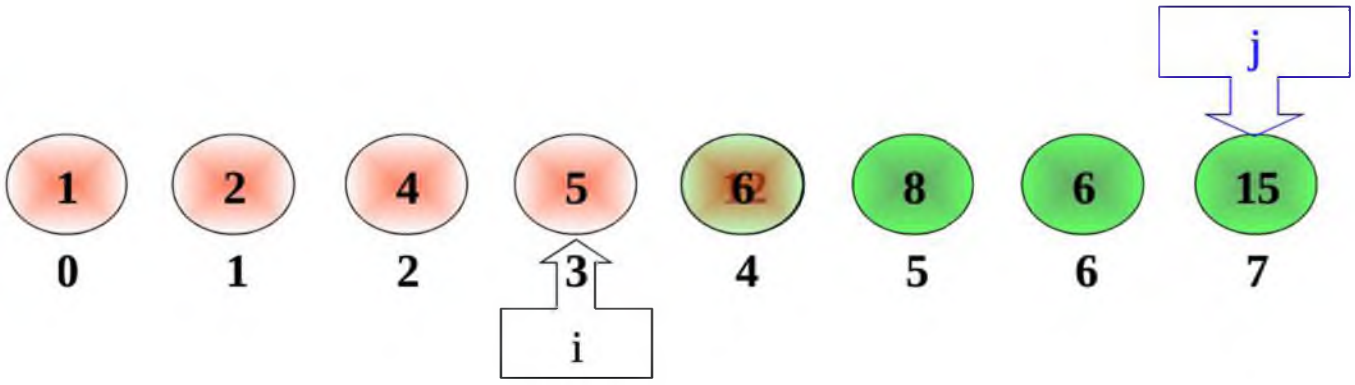
Bubble Sort – Ví dụ



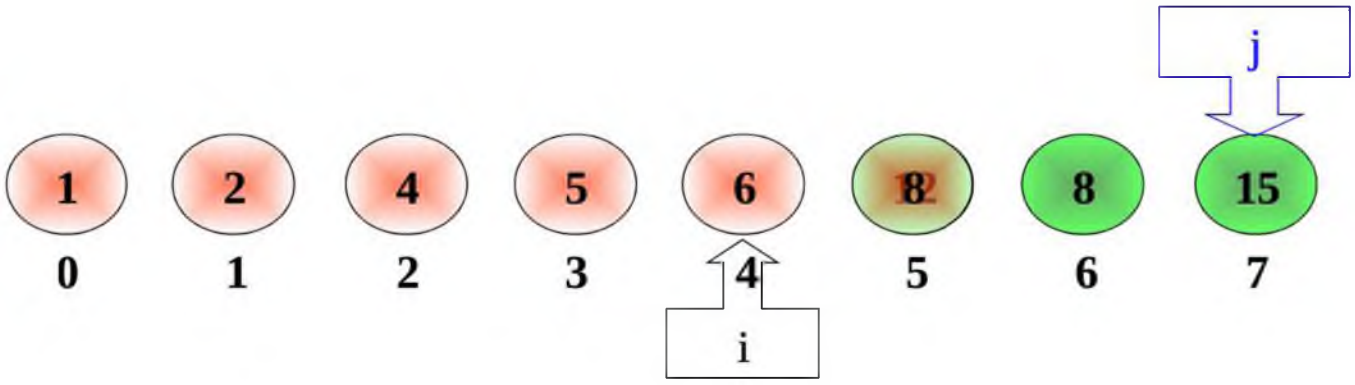
Bubble Sort – Ví dụ



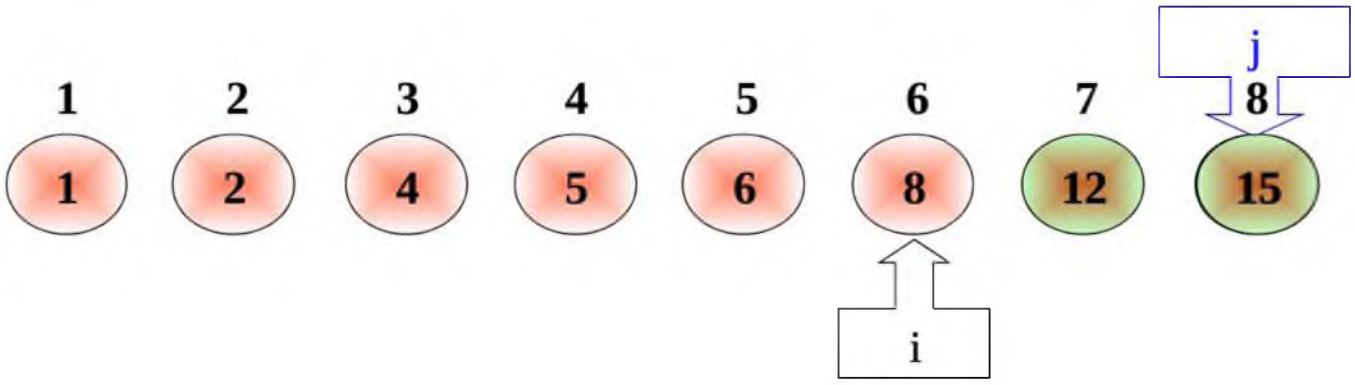
Bubble Sort – Ví dụ



Bubble Sort – Ví dụ



Bubble Sort – Ví dụ



Độ phức tạp của thuật toán sắp xếp Bubble sort

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. 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

Shaker Sort

- Trong mỗi lần sắp xếp, duyệt mảng theo 2 lượt từ 2 phía khác nhau :
  - Lượt đi: đẩy phần tử nhỏ về đầu mảng
  - Lượt về: đẩy phần tử lớn về cuối mảng
- Ghi nhận lại những đoạn đã sắp xếp nhằm tiết kiệm các phép so sánh thừa.

Shaker Sort

```
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])
                      {Doicho(a[j], a[j-1]);k =j;}
          left = k;
          for (j = left; j < right; j ++)
                      if (a[j]> a[j+1])
                      {Doicho(a[j], a[j-1]);k = j; }
          right = k;
     }
}
```

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

Chèn trực tiếp – Insertion Sort

- Giả sử có một dãy $a_1 , a_2 ,... ,a_n$ trong đó i phần tử đầu tiên $a_1 , a_2 ,... ,a_{i-1}$ 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 $a_1 , a_2 ,... ,a_i$ 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 = 2; *// 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;

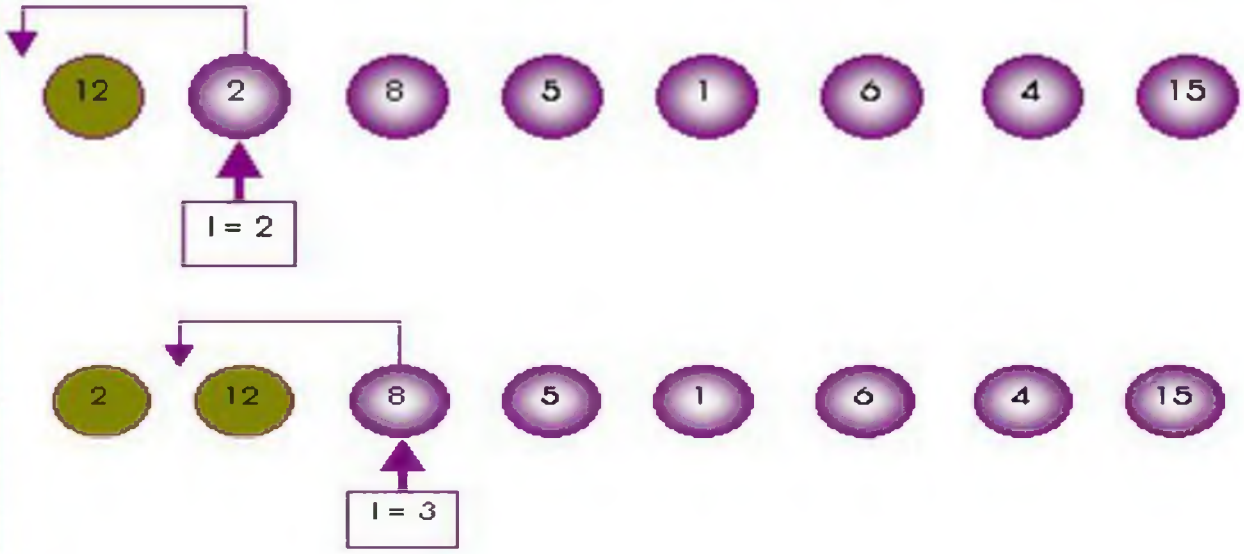
  Nếu i < n : Lặp lại Bước 2.

  Ngược lại : Dừng.

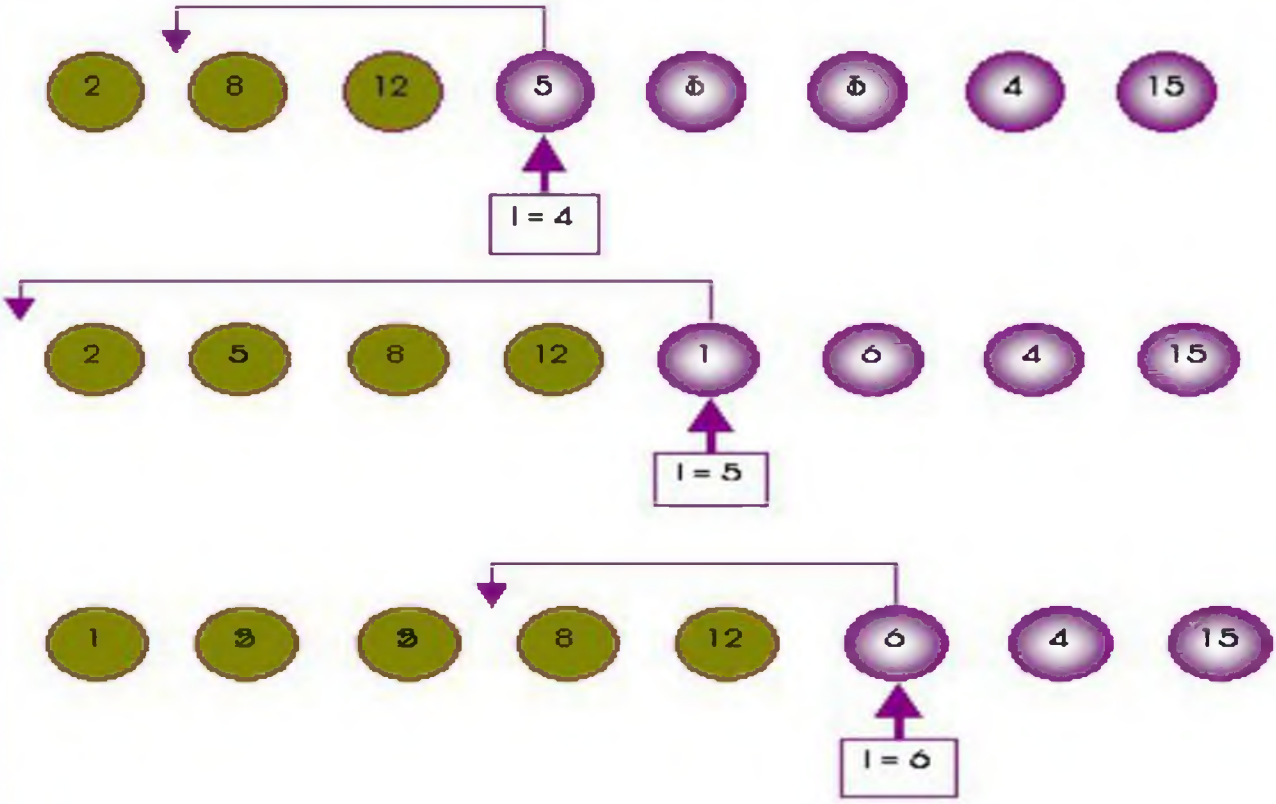
Chèn trực tiếp – Insertion Sort

- Cho dãy số :

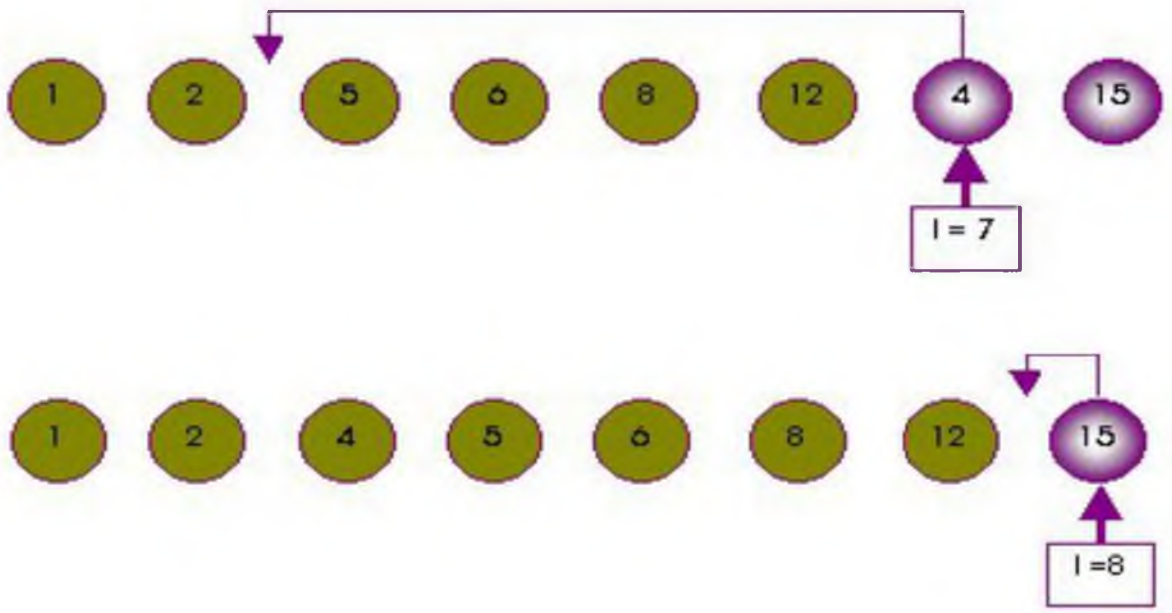
12 2 8 5 1 6 4 15



Chèn trực tiếp – Insertion Sort



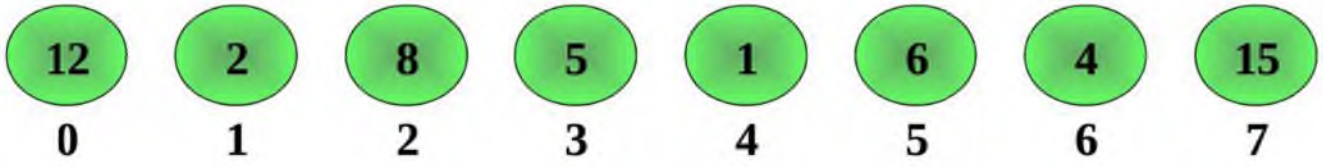
Chèn trực tiếp – Insertion Sort



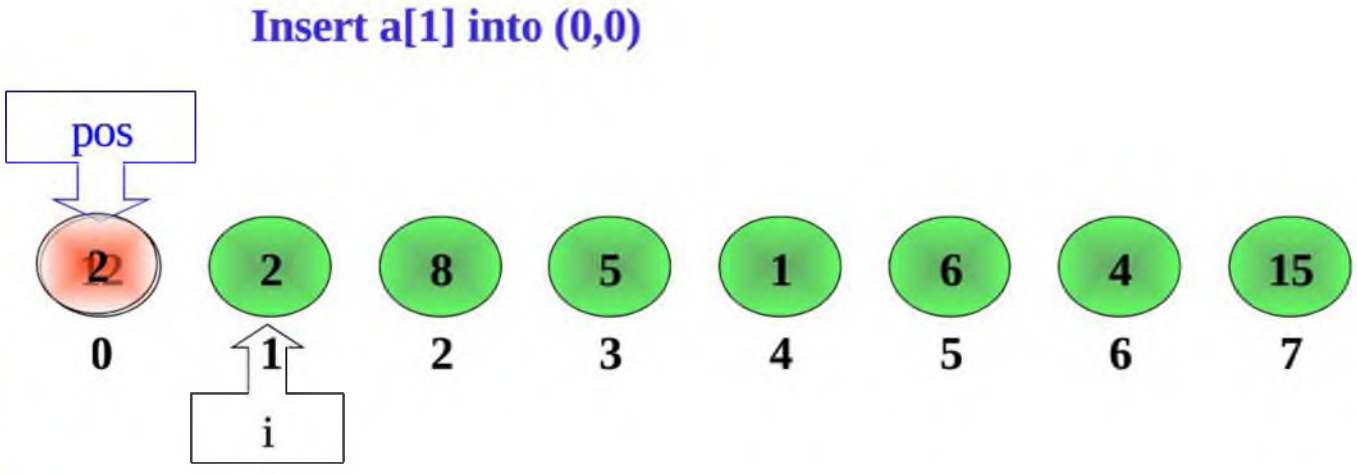
Chèn trực tiếp – Insertion Sort

```
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
        }
   }
```

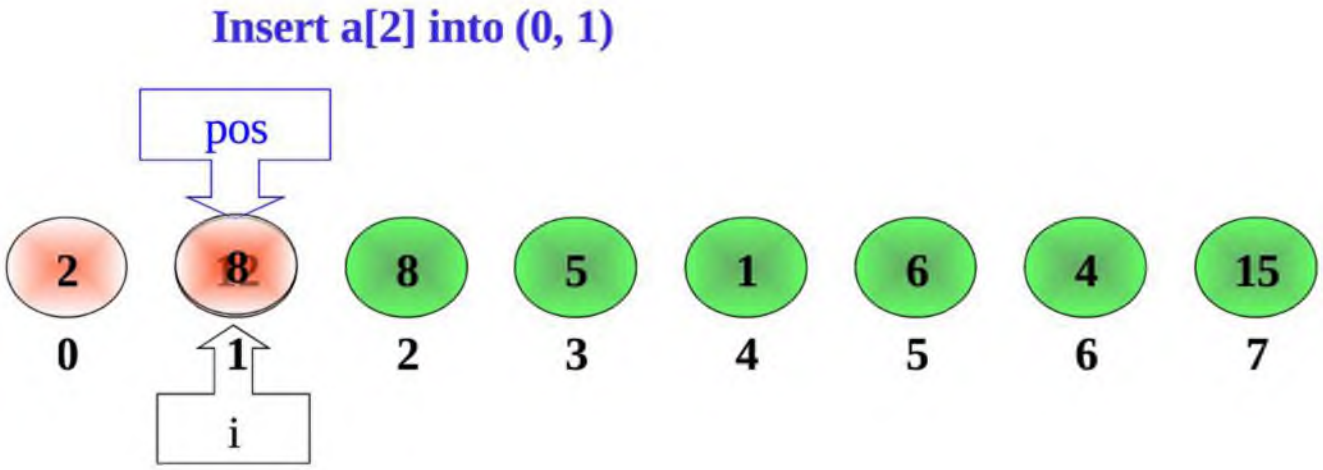
Insertion Sort – Ví dụ



Insertion Sort – Ví dụ



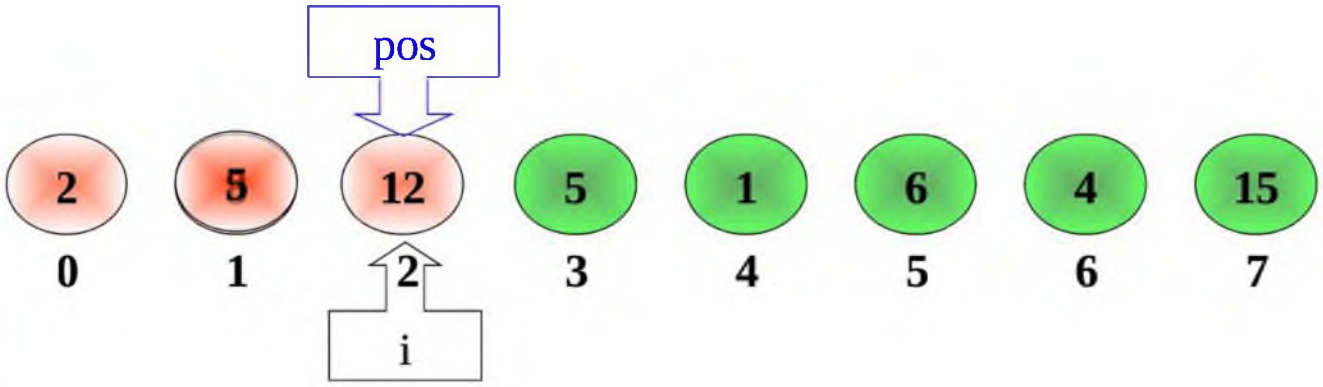
Insertion Sort – Ví dụ



X

Insertion Sort – Ví dụ

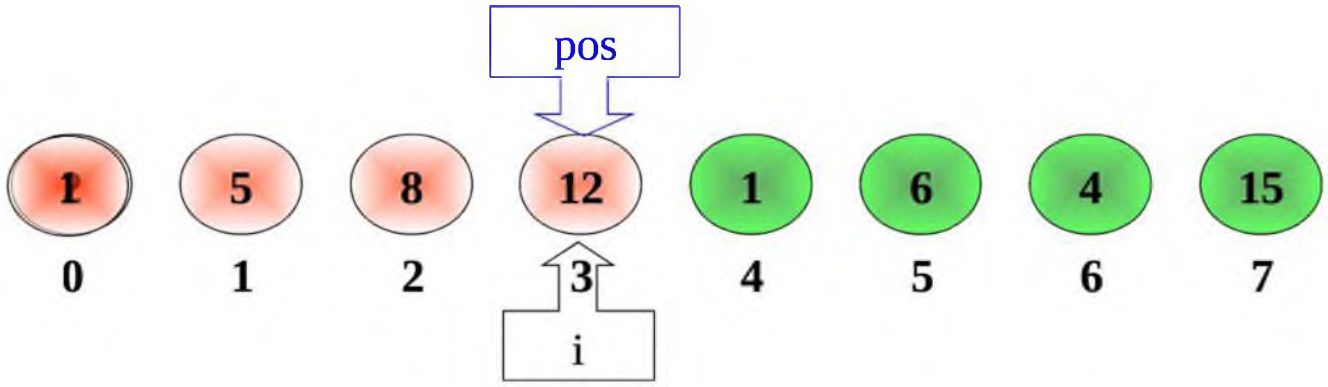
Insert a[3] into (0, 2)



X

Insertion Sort – Ví dụ

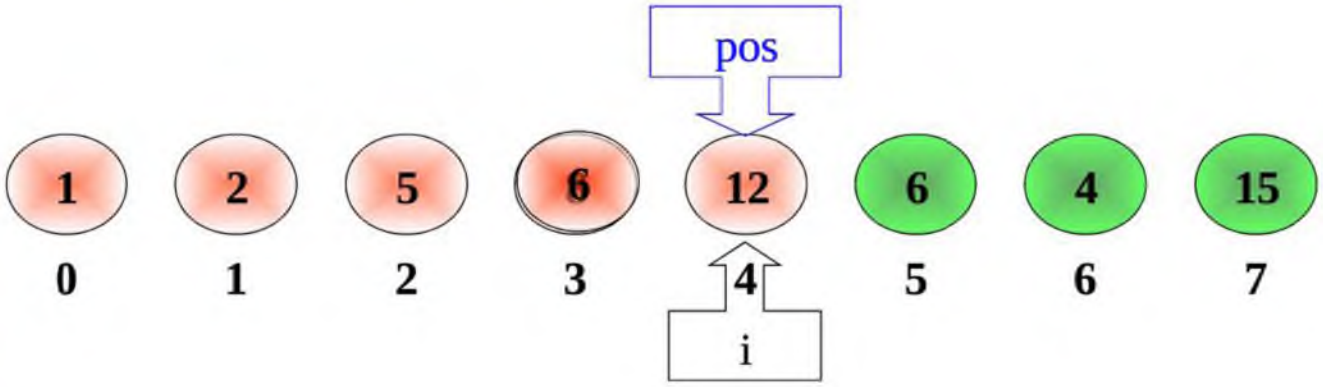
Insert a[4] into (0, 3)



X

Insertion Sort – Ví dụ

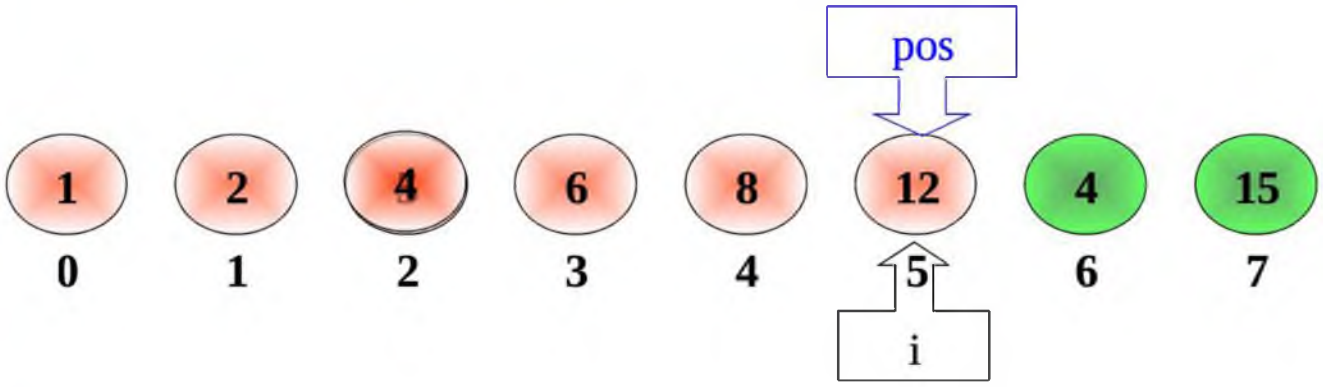
Insert a[5] into (0, 4)



X

Insertion Sort – Ví dụ

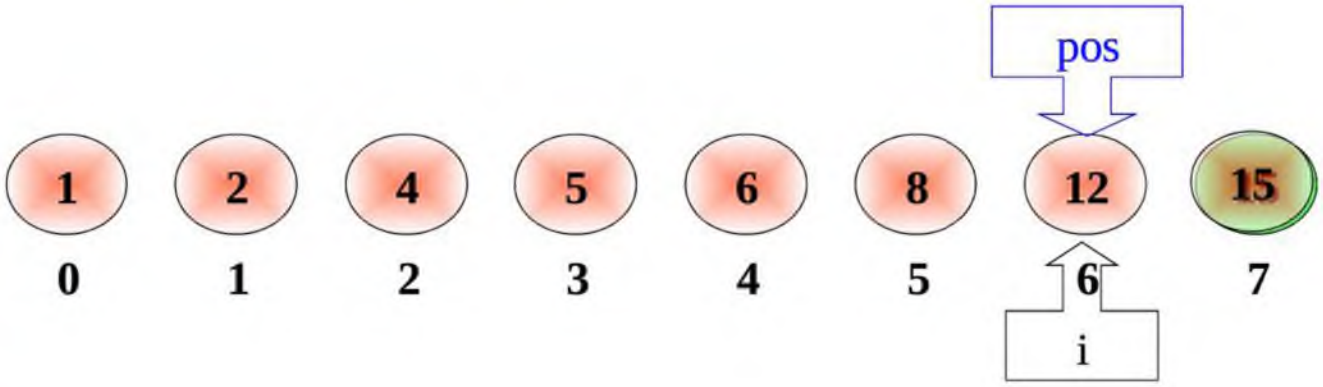
Insert a[6] into (0, 5)



x

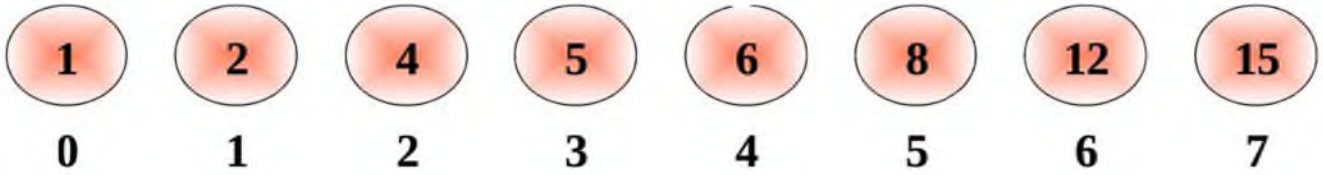
Insertion Sort – Ví dụ

Insert a[8] into (0, 6)



X

Insertion Sort – Ví dụ



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. 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

Chèn nhị phân – Binary Insertion Sort

```
void   BInsertionSort(int a[],int n )
  {
      int l,r,m,i;
      int x;//lưu giá trị a[i] tránh bị ghi đè khi dời chỗ các phần tử.
      for(int i=1 ; i<n ; i++)
      {       x = a[i]; l = 1; r = i-1;
      while(i<=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. 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

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 :

$$h_i > h_{i+1} \text{ và } h_k = 1$$

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

  Ví dụ :127, 40, 13, 4, 1

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

  Ví dụ : 15, 7, 3, 1

Shell Sort

- h có dạng 3i+1: 364, 121, 40, 13, 4, 1
- Dãy 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

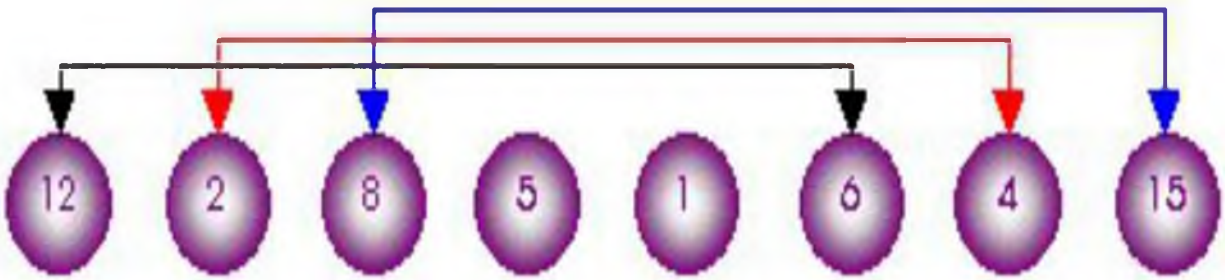
- Cho dãy số a:

  12 2 8 5 1 6 4 15

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

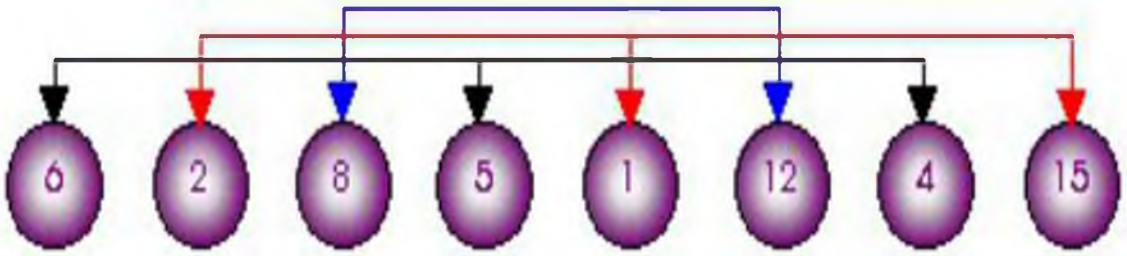
Shell Sort

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



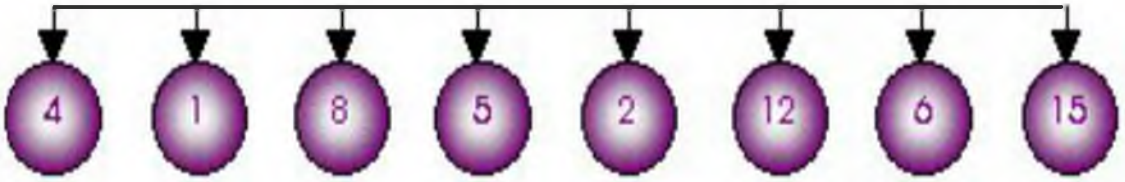
Shell Sort

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

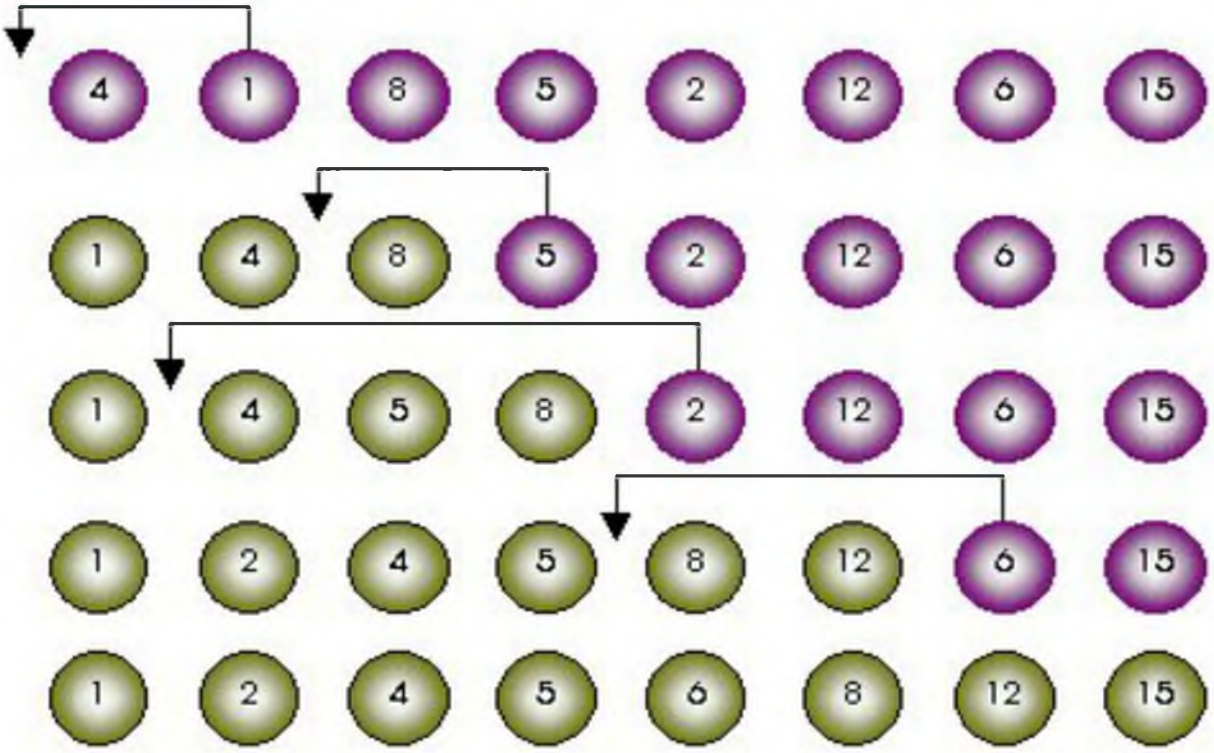


Shell Sort

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



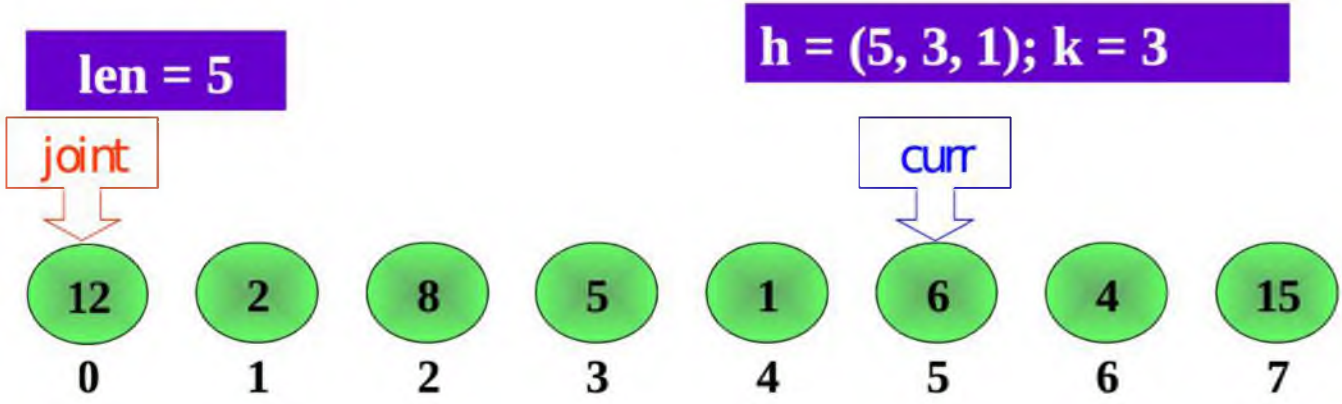
Shell Sort


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

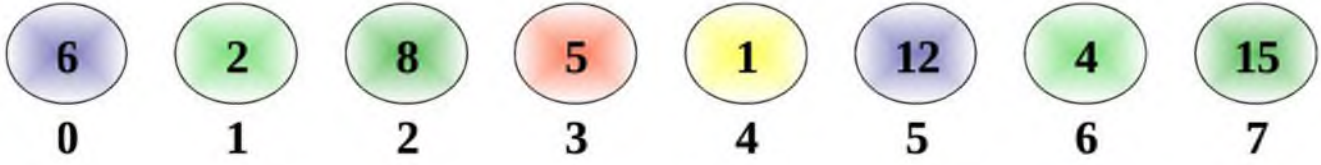
Shell sort – Ví dụ



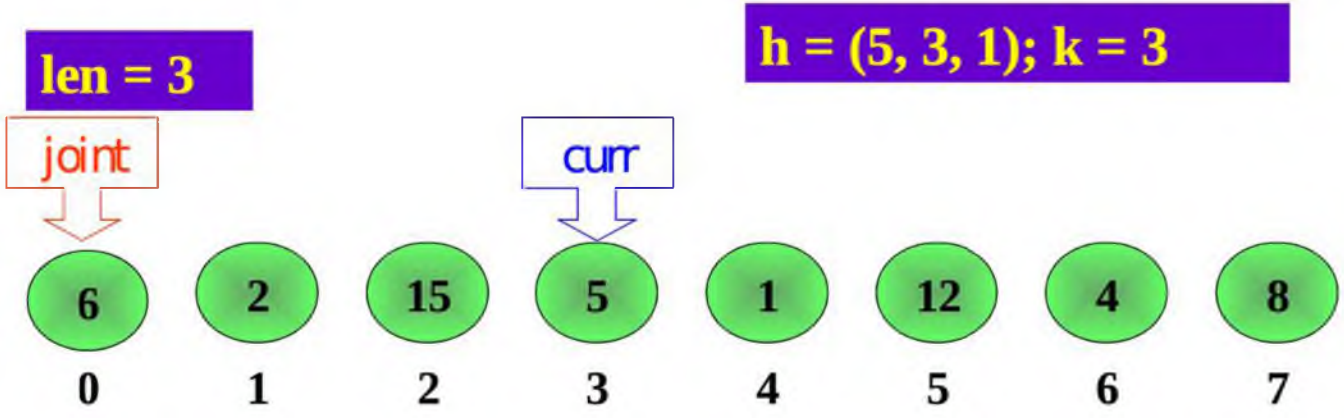
Shell sort – Ví dụ

len = 5;

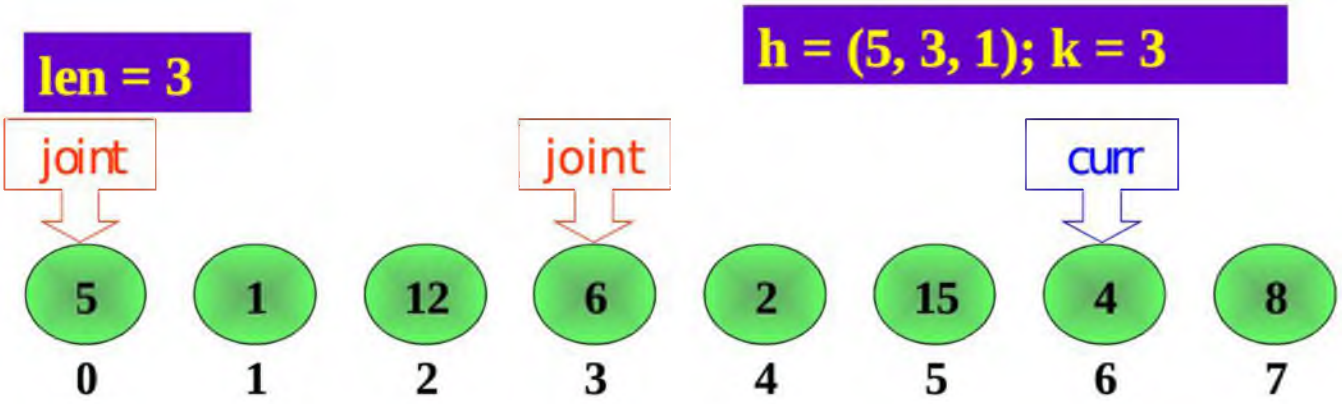
h = (5, 3, 1); k = 3



Shell sort – Ví dụ



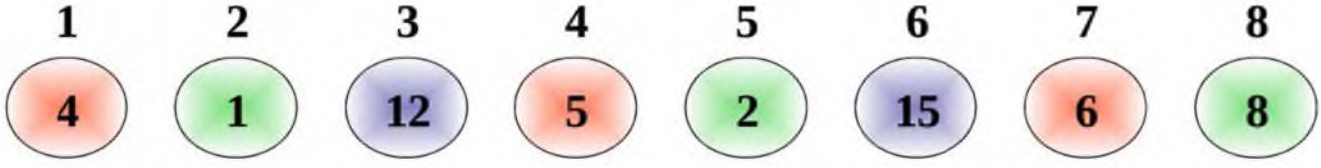
Shell sort – Ví dụ



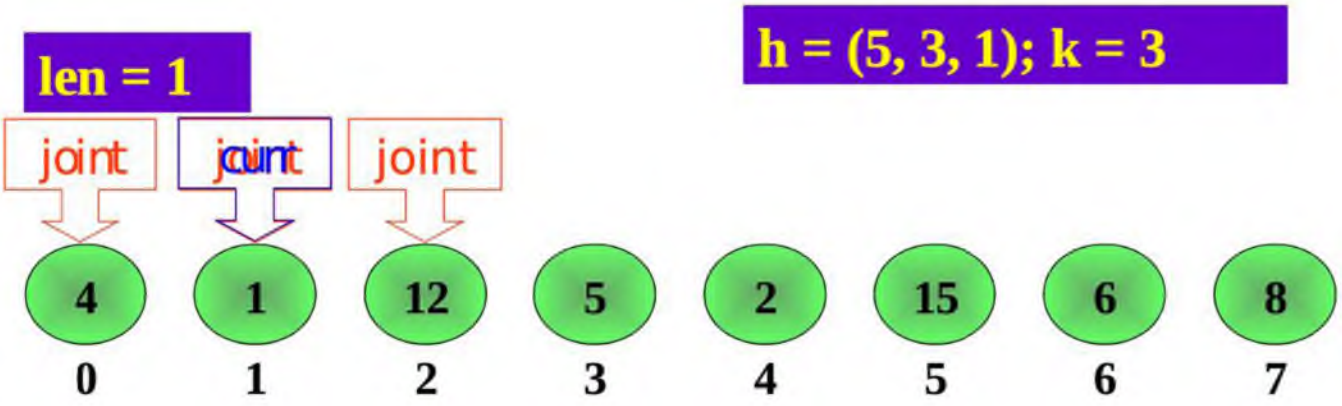
Shell sort – Ví dụ

len = 3

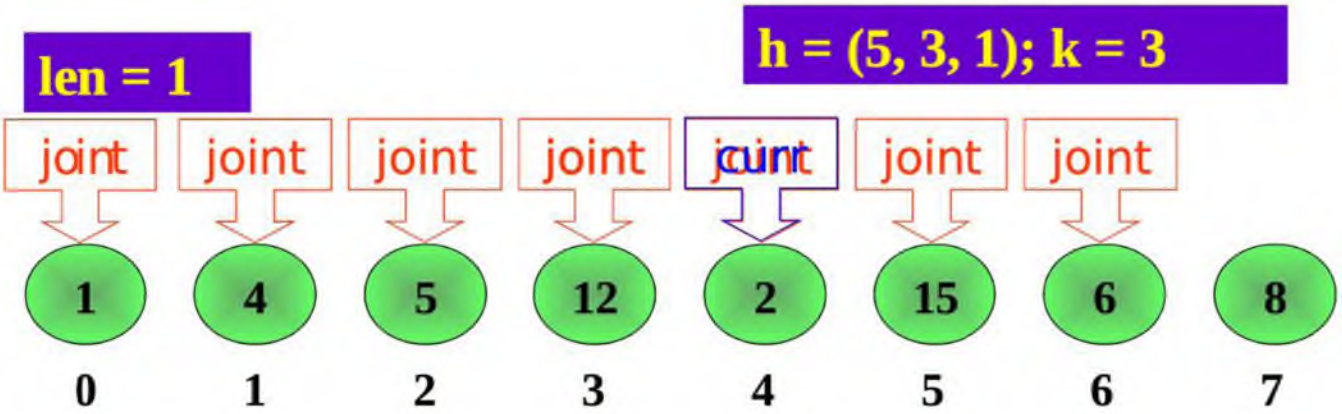
h = (5, 3, 1); k = 3



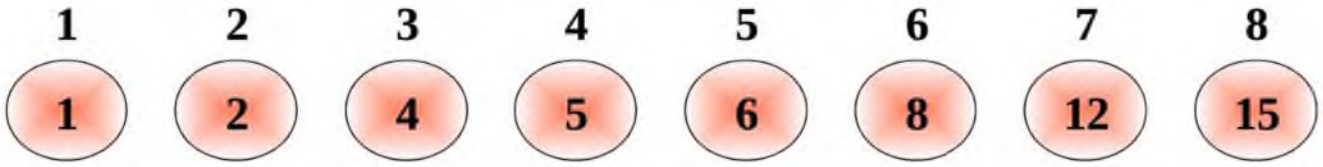
Shell sort – Ví dụ



Shell sort – Ví dụ



Shell sort – Ví dụ



Các thuật toán sắp xếp

1. Đổi chỗ trực tiếp – Interchange Sort
2. 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

Chọn trực tiếp – Selection Sort

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

Chọn trực tiếp – Selection Sort

- Bước 1: i = 1;
- 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

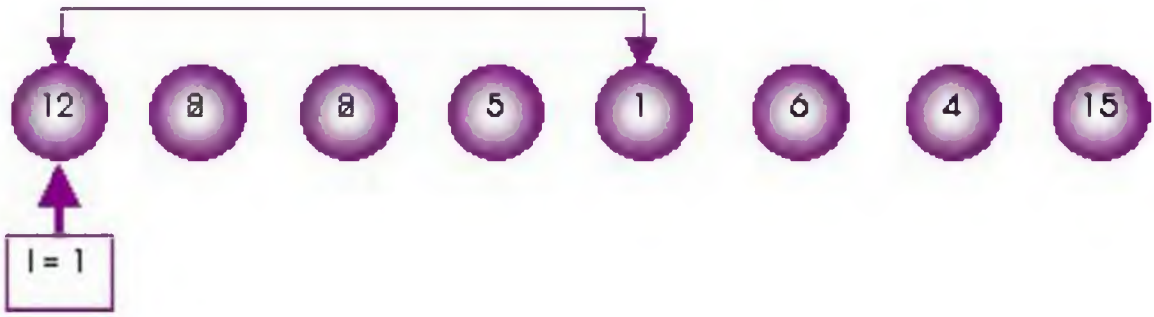
  i = i+1; Lặp lại Bước 2

  Ngược lại: Dừng.

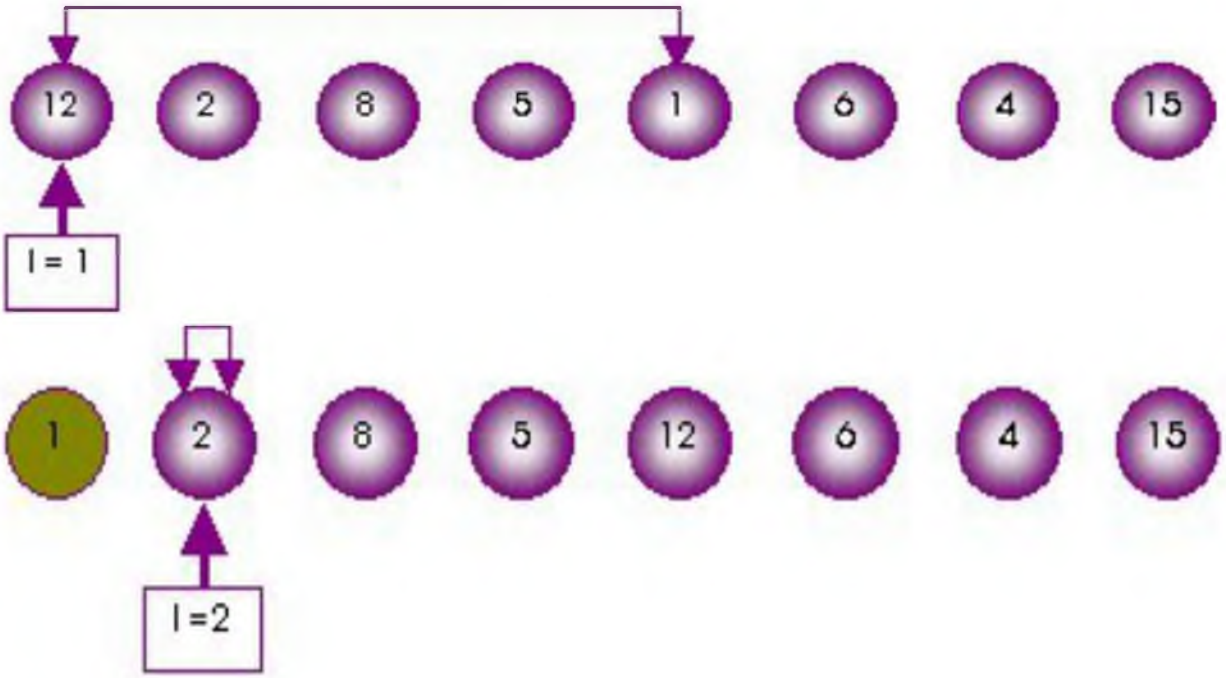
Chọn trực tiếp – Selection Sort

- Cho dãy số a:

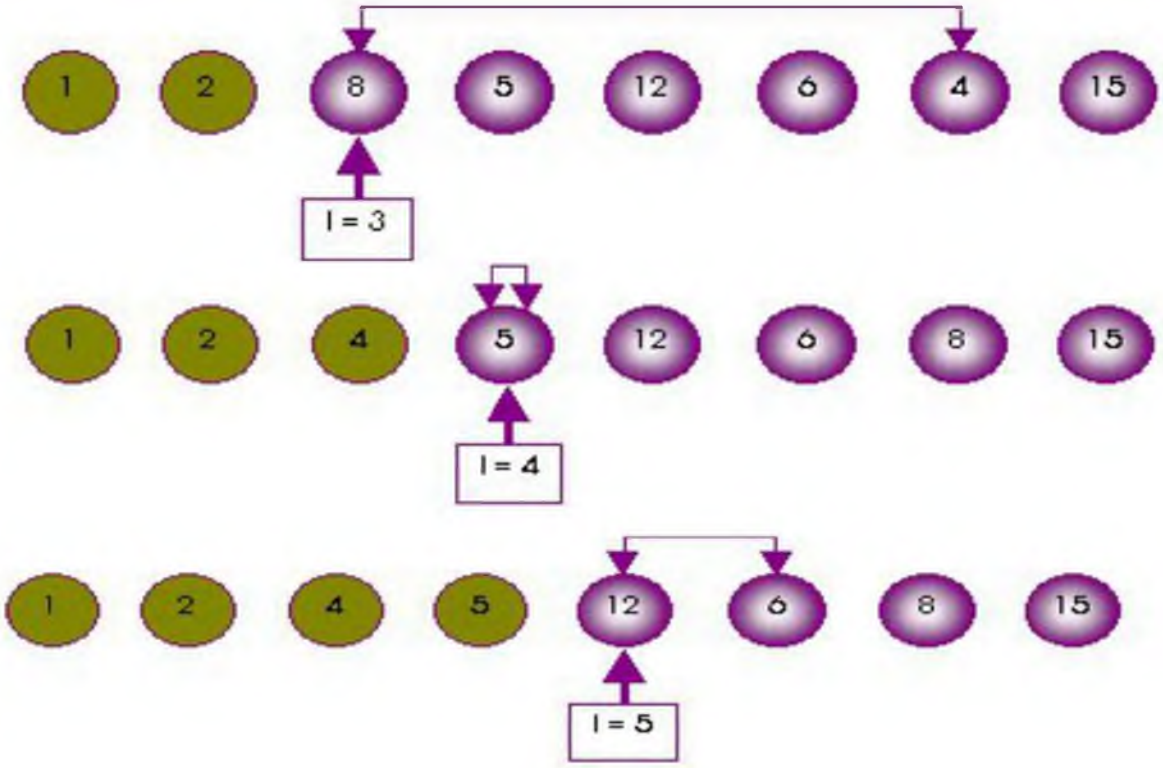
12 2 8 5 1 6 4 15



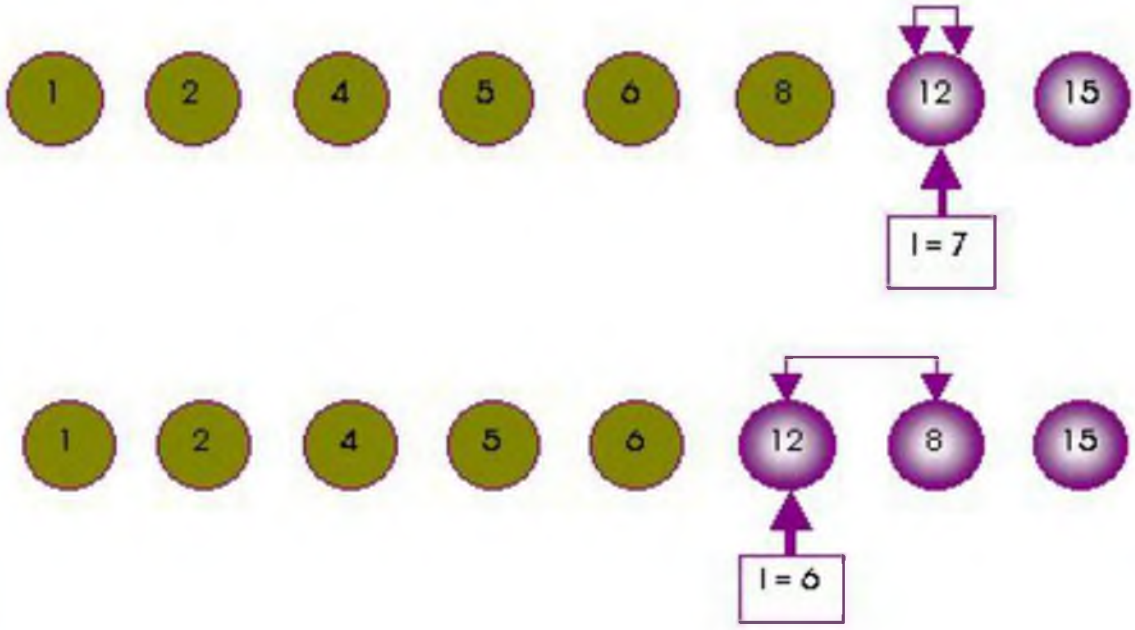
Chọn trực tiếp – Selection Sort



Chọn trực tiếp – Selection Sort



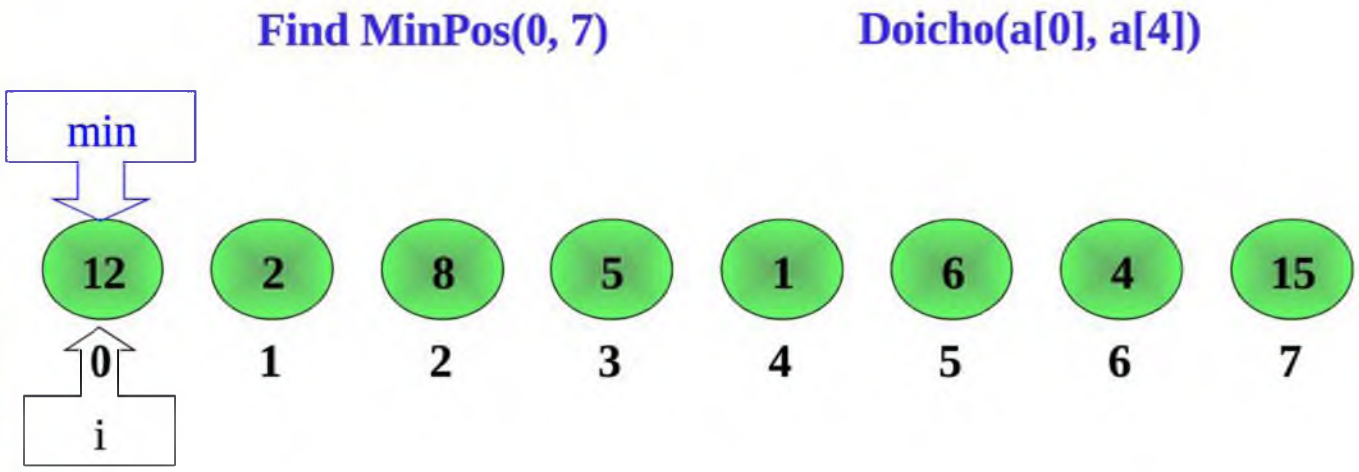
Chọn trực tiếp – Selection Sort



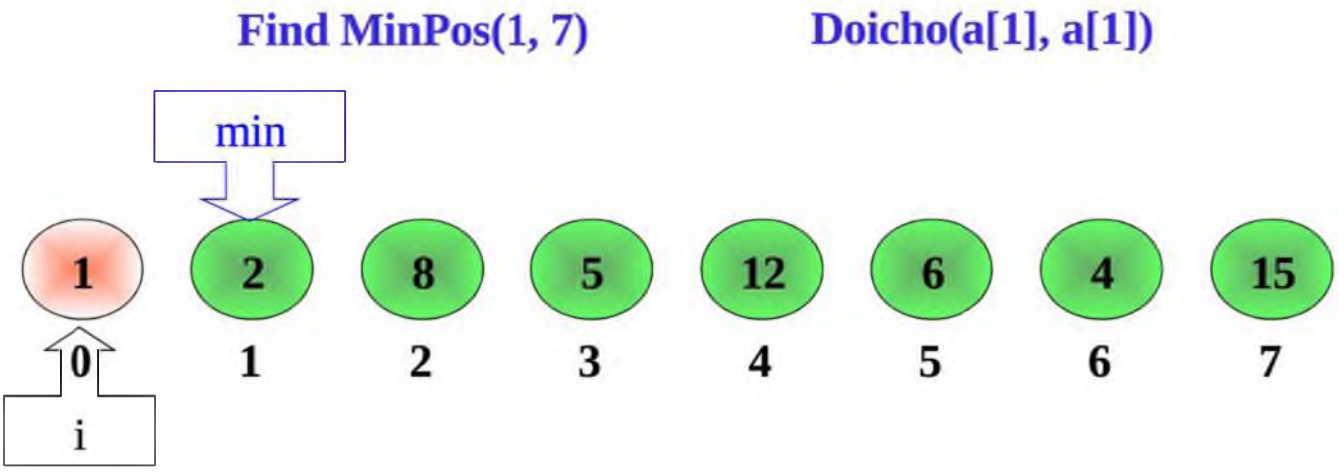
Chọn trực tiếp – Selection Sort

```
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;
                                // ghi nhận vị trí phần tử hiện nhỏ nhất
                                Doicho(a[min],a[i]);
        }
}
```

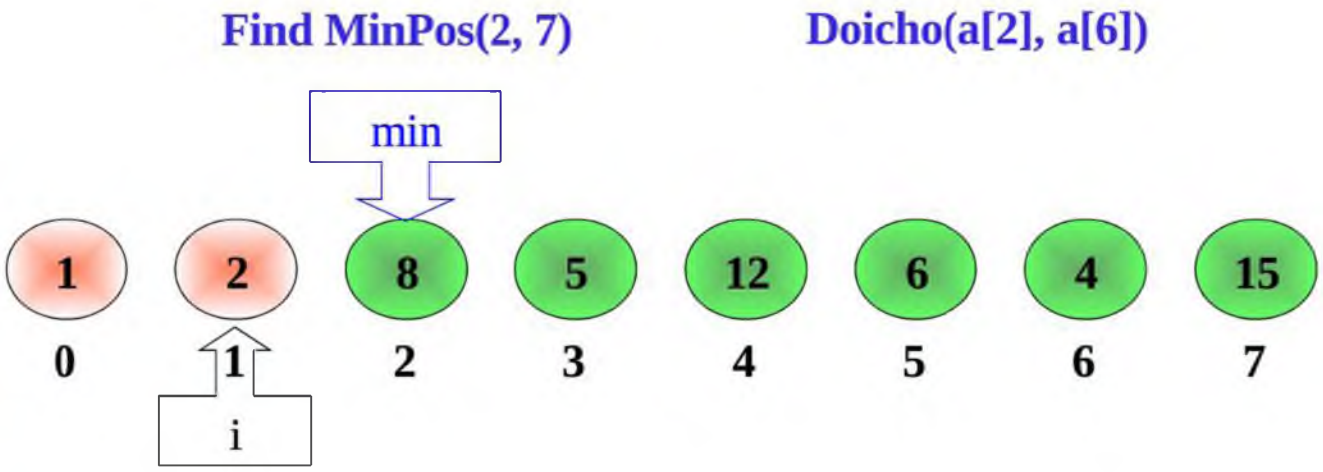
Selection sort – Ví dụ



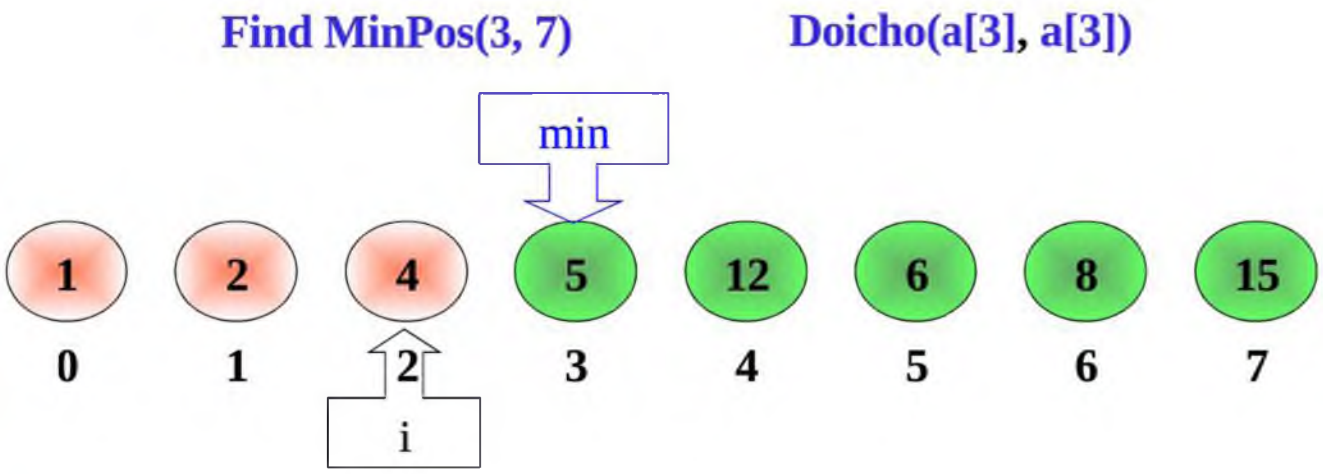
Selection sort – Ví dụ



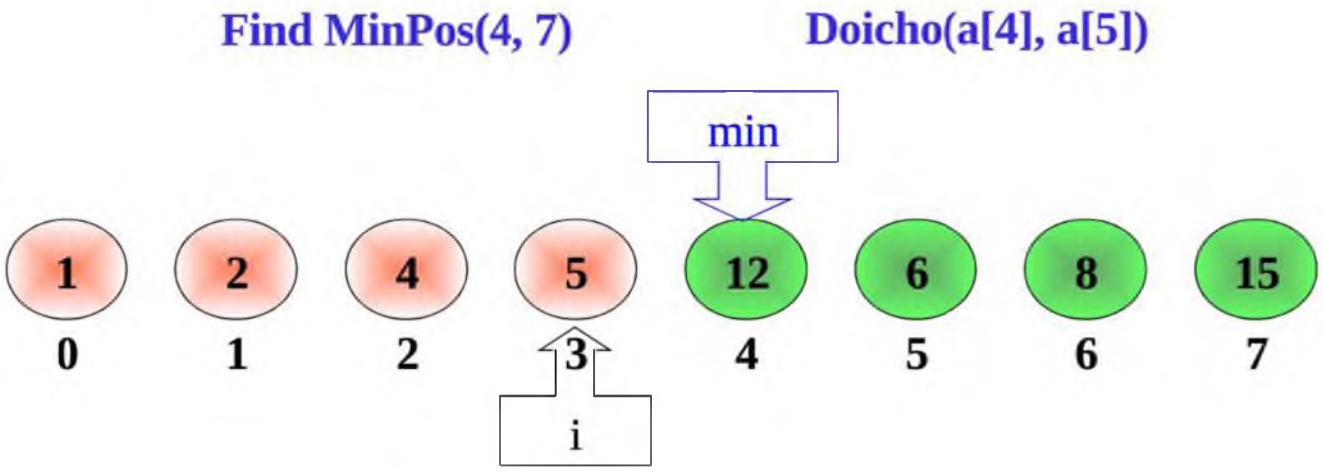
Selection sort – Ví dụ



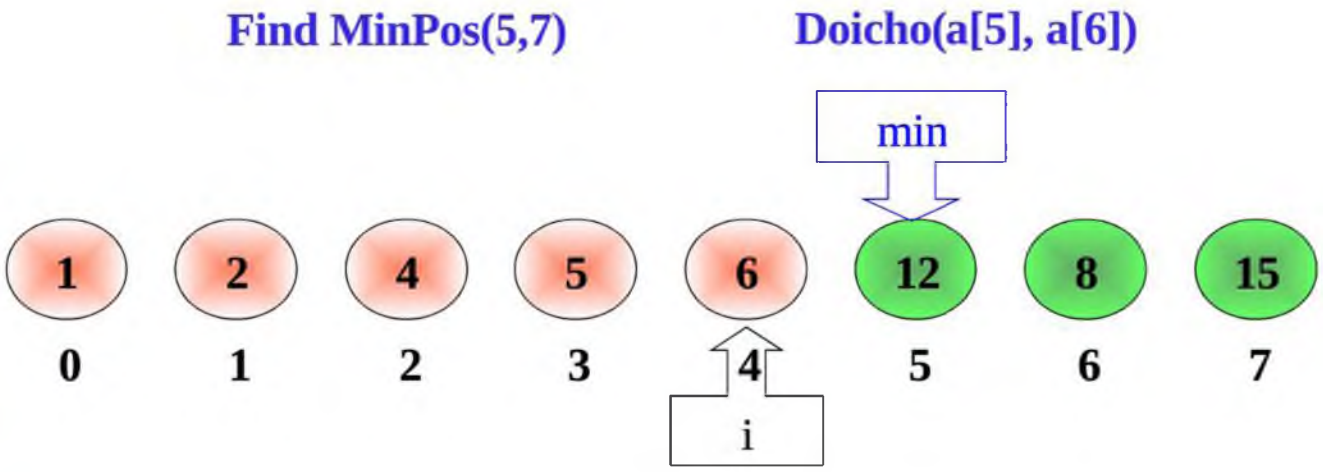
Selection sort – Ví dụ



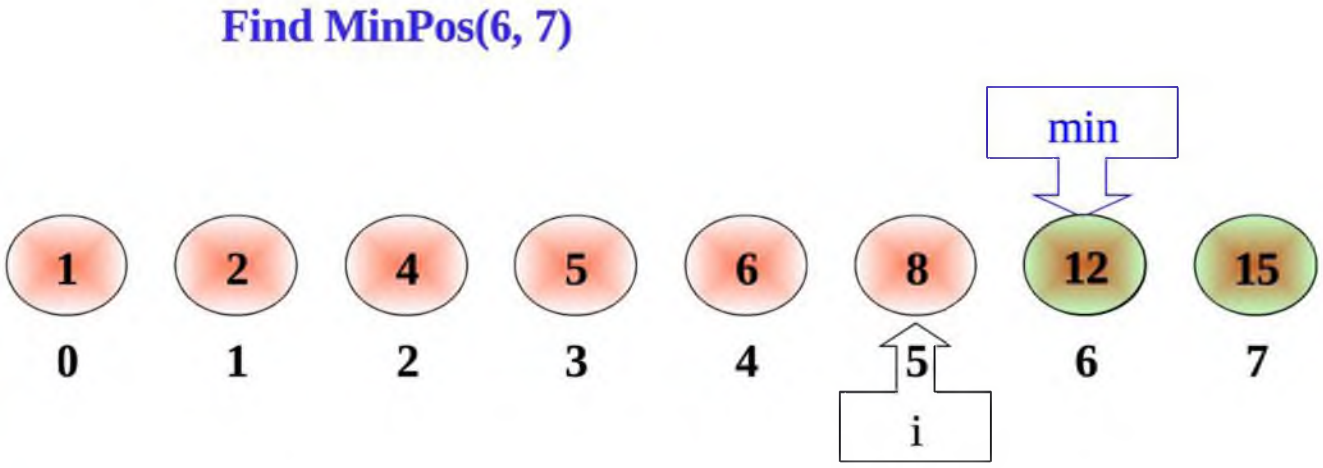
Selection sort – Ví dụ



Selection sort – Ví dụ



Selection sort – Ví dụ



Chọn trực tiếp – Selection Sort

- **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. 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

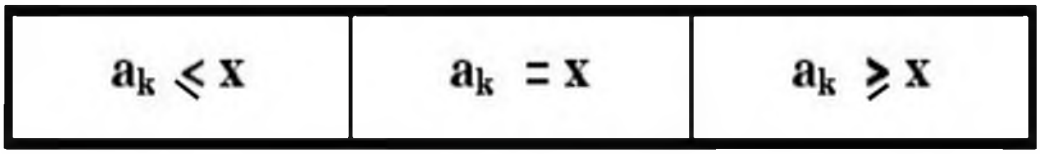
Quick sort – Ý tưởng

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

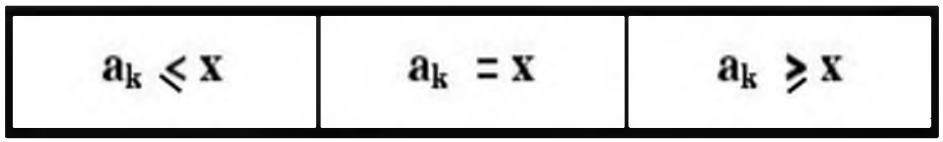
  với x là giá trị của một phần tử tùy ý trong dãy ban đầu.

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



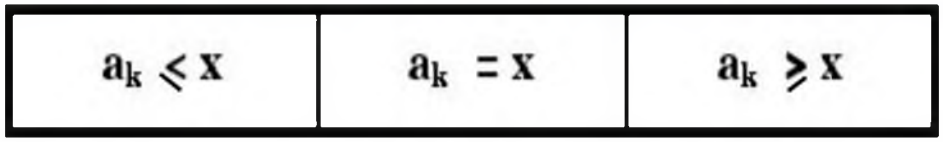
Quick sort – Ý tưởng



- 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 …

Quick sort – Giải thuật

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

  Kết thúc; *//dãy đã được sắp xếp*

- Bước 2: Phân hoạch dãy aleft … aright thành các đoạn: aleft.. aj, aj_{+1}.. ai_{-1}, ai.. Aright

  Đoạn 1 $\leq x$

  Đoạn 2: aj_{+1}.. ai_{-1} $= x$

  Đoạn 3: ai.. aright $\geq$ x

- Bước 3: **Sắp xếp đoạn 1**: aleft.. aj
- Bước 4: **Sắp xếp đoạn 3**: ai.. aright

Quick sort – Giải thuật

- 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;

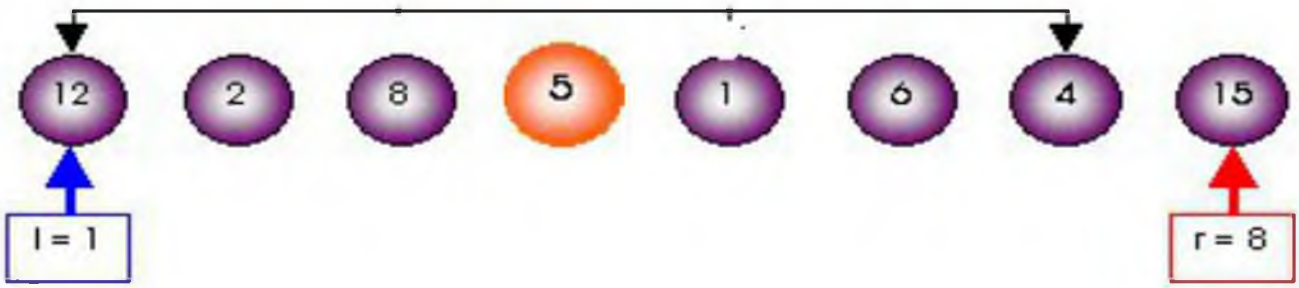
- Bước 2 : Phát hiện và hiệu chỉnh cặp phần tử a[i], a[j] nằm sai chỗ :
  - Bước 2a : Trong khi (a[i]<x) i++;
  - Bước 2b : Trong khi (a[j]>x) j--;
  - Bước 2c : Nếu i< j Đoicho(a[i],a[j]);
- Bước 3 : Nếu i < j: Lặp lại Bước 2.
  Ngược lại: Dừng

Quick sort – Ví dụ

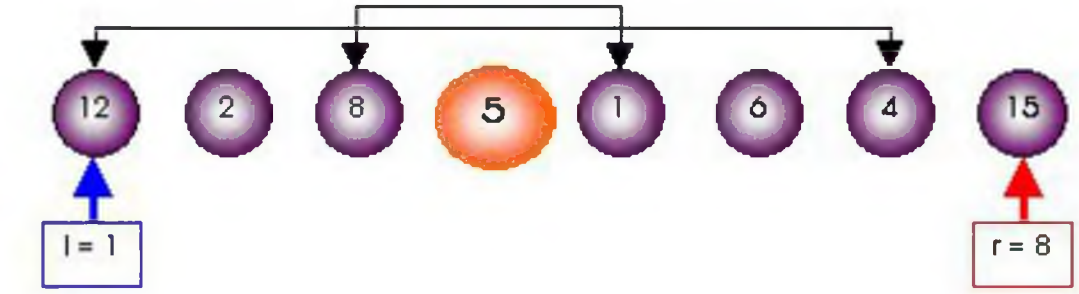
- Cho dãy số a:

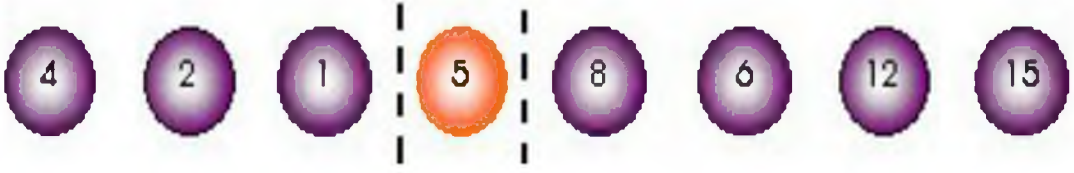
12 2 8 5 1 6 4 15

Phân hoạch đoạn l =1, r = 8: **x = A[4] = 5**



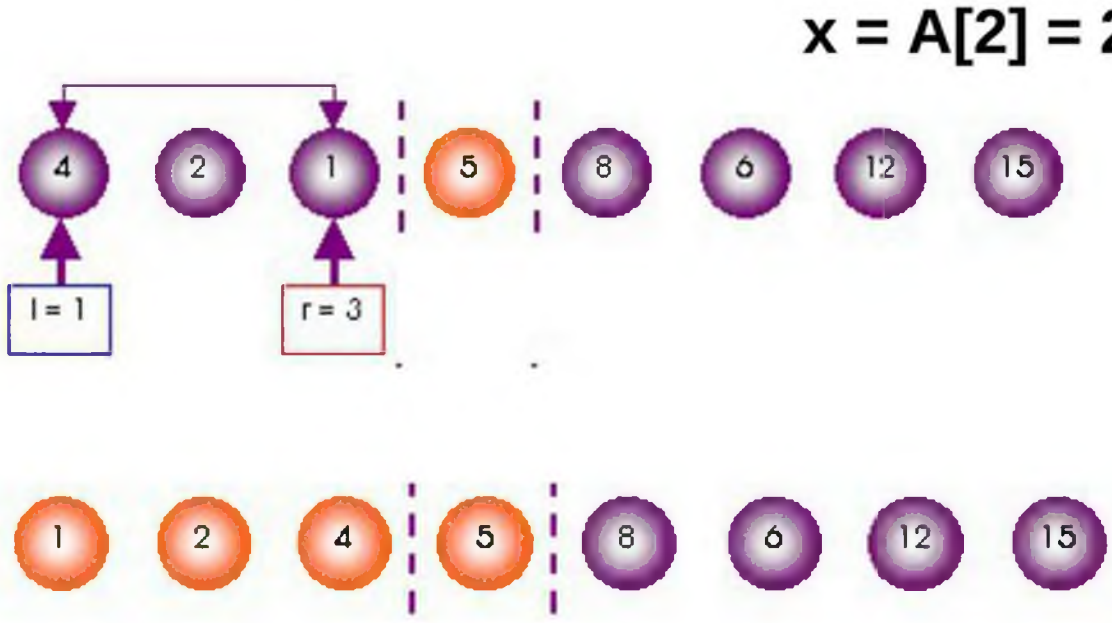
Quick sort – Ví dụ





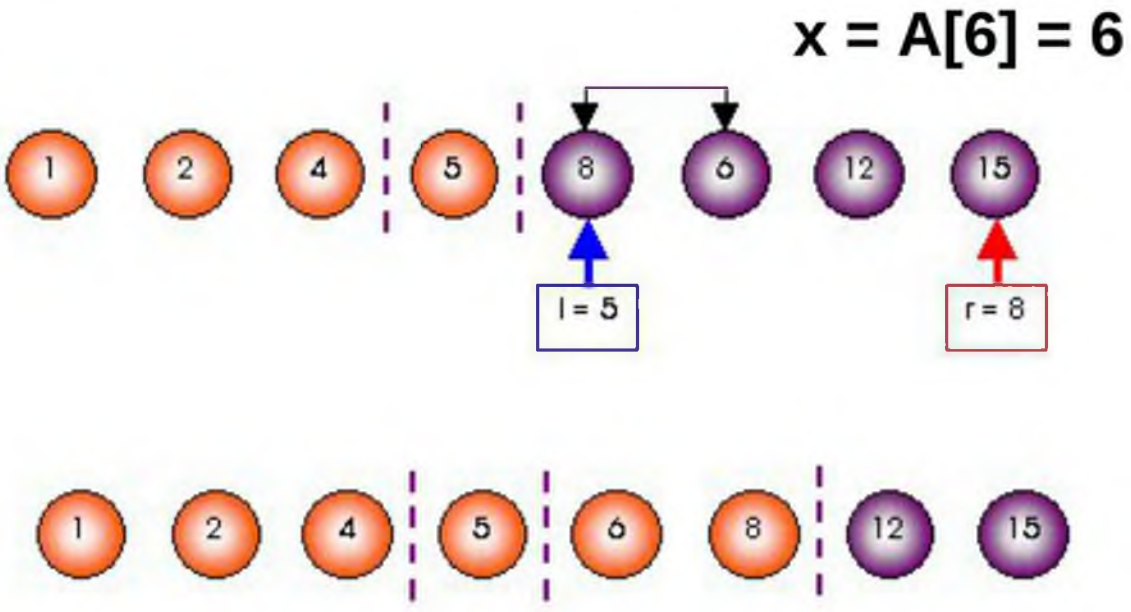
Quick sort – Ví dụ

- Phân hoạch đoạn l =1, r = 3:



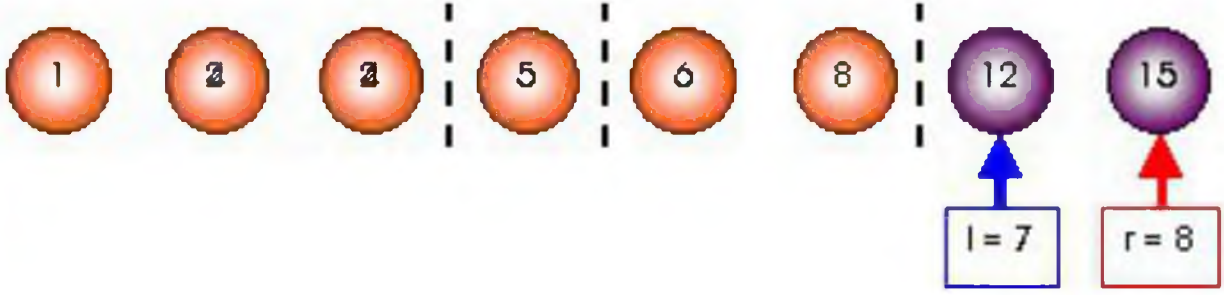
Quick sort – Ví dụ

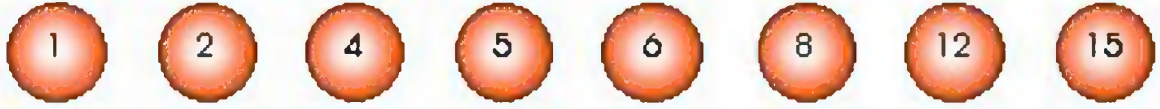
- Phân hoạch đoạn l = 5, r = 8:



- Phân hoạch đoạn l = 7, r = 8:

x = A[7] = 6

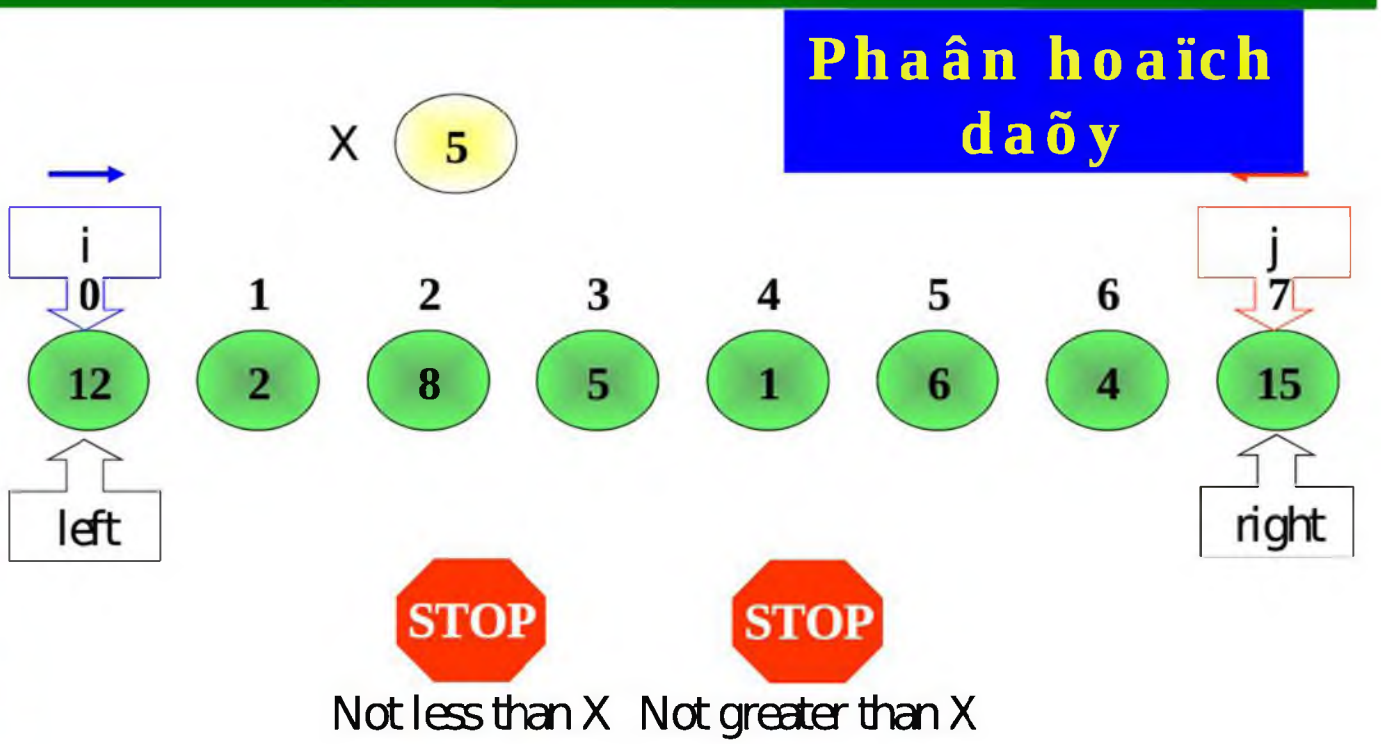




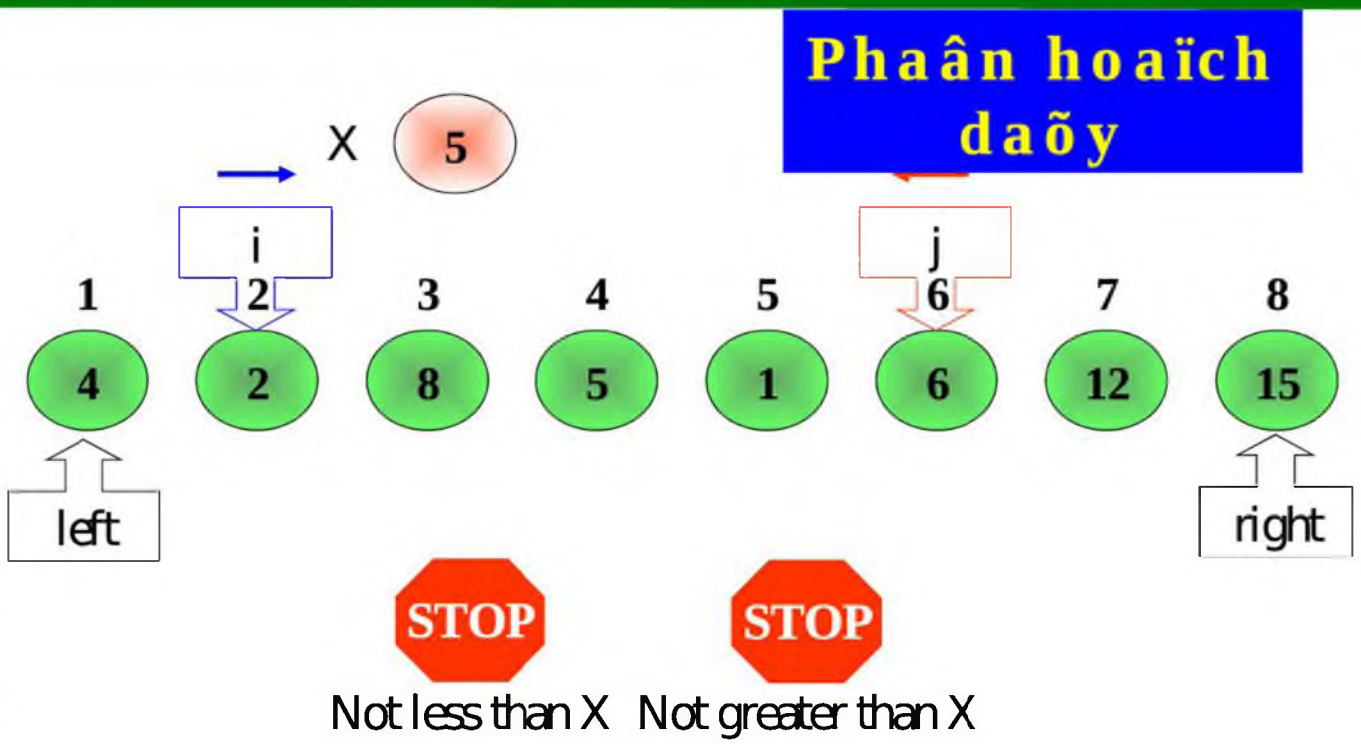
Quick sort


```
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);
}
```

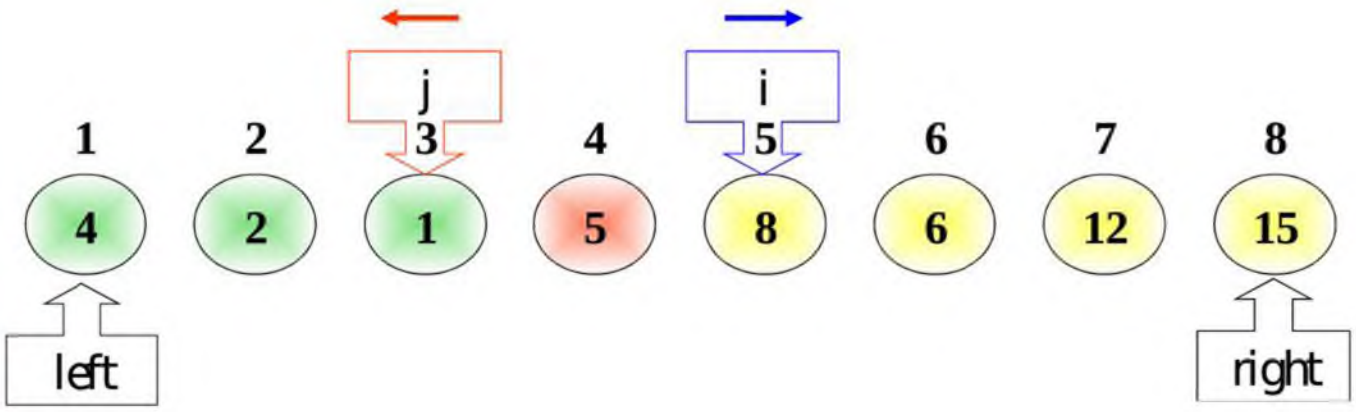
Quick sort – Ví duï



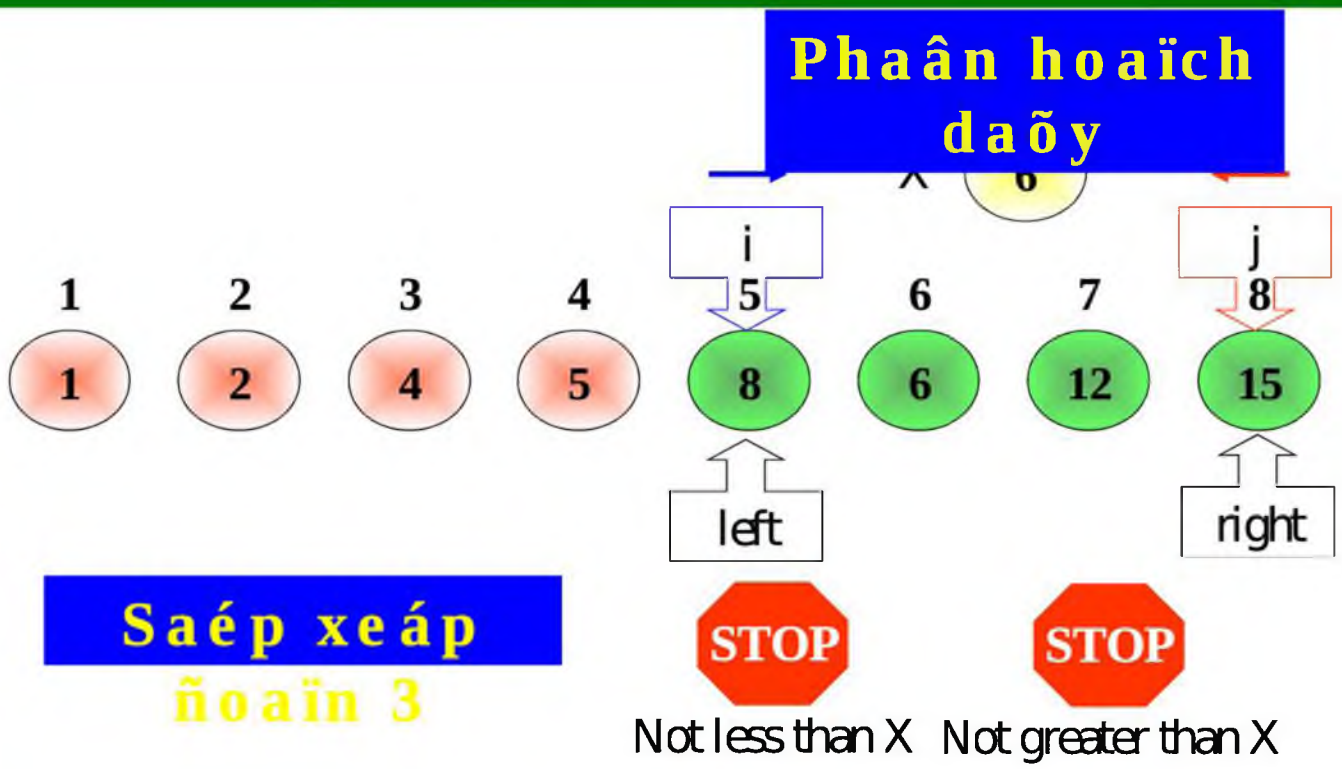
Quick sort – Ví duï



Quick sort – Ví duï

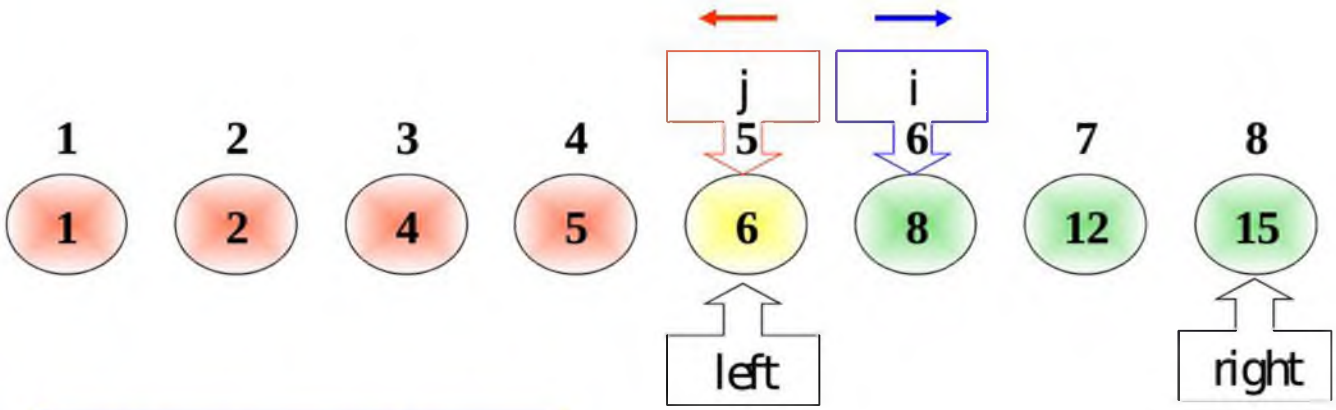


Quick sort – Ví duï



Saép xeáp ñoaïn 3

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. 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

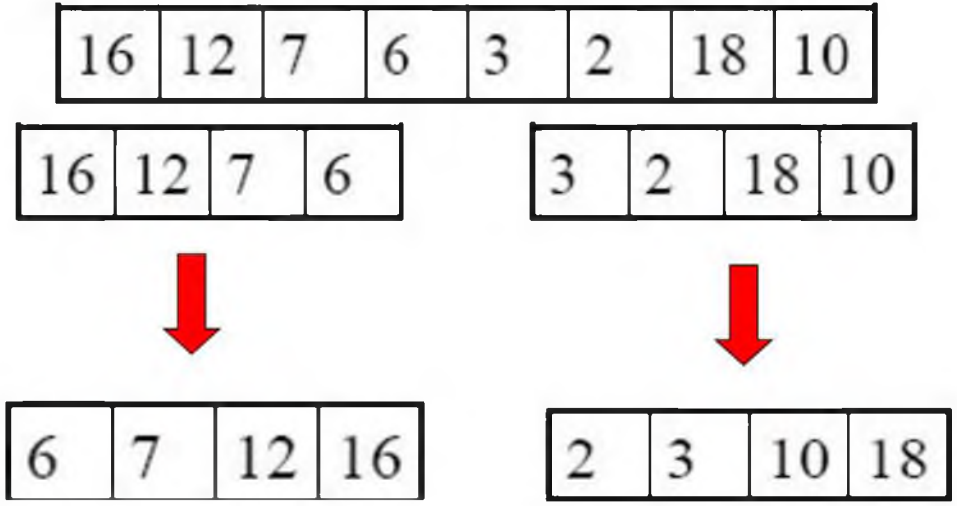
Merge sort – Ý tưởng

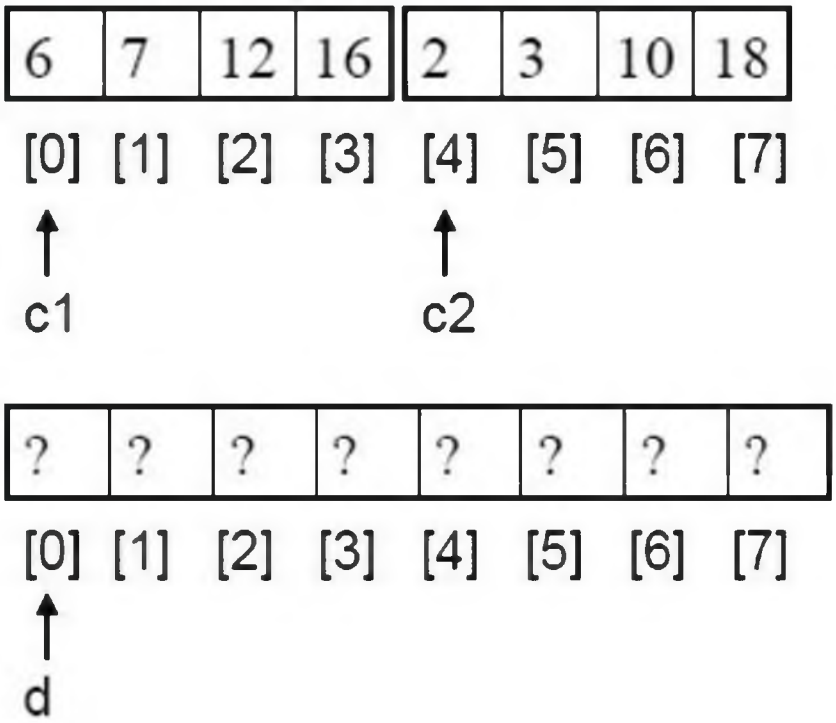
- Giải thuật Merge sort sắp xếp dãy $\mathbf{a_1}$, $\mathbf{a_2}$, ..., **an** dựa trên nhận xét sau:
  - Mỗi dãy $\mathbf{a_1}$, $\mathbf{a_2}$, ..., **an** bất kỳ là một tập hợp các dãy con liên tiếp mà mỗi dãy con đều đã có thứ tự.
    - Ví dụ: dãy 12, 2, 8, 5, 1, 6, 4, 15 có thể coi như gồm 5 dãy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

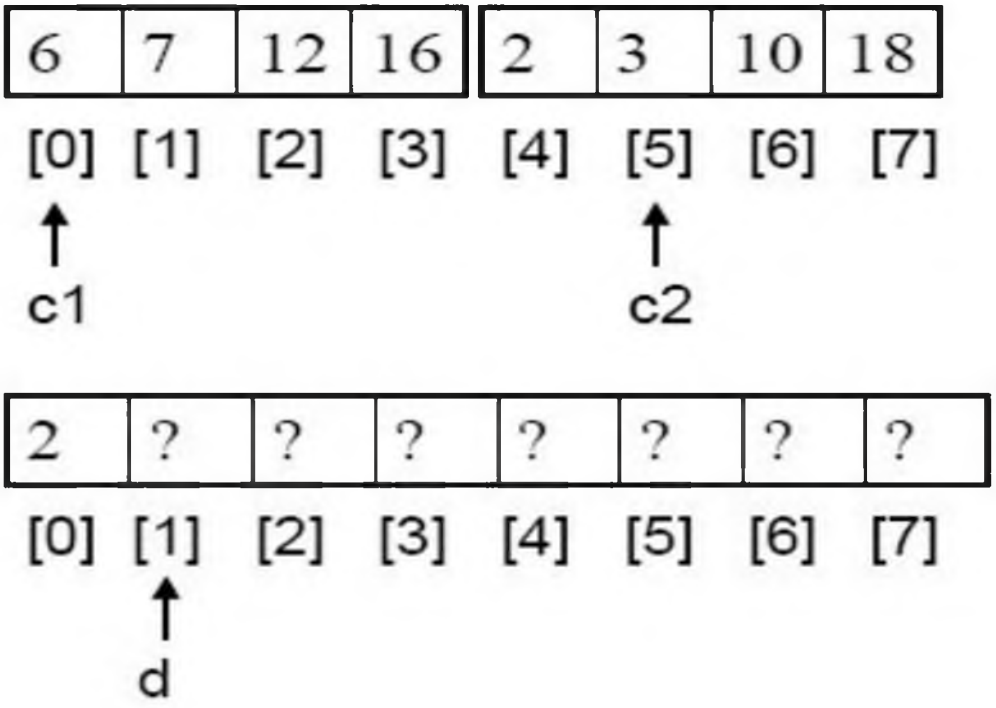
➔Hướng tiếp cận: tìm cách làm giảm số dãy con không giảm của dãy ban đầu.

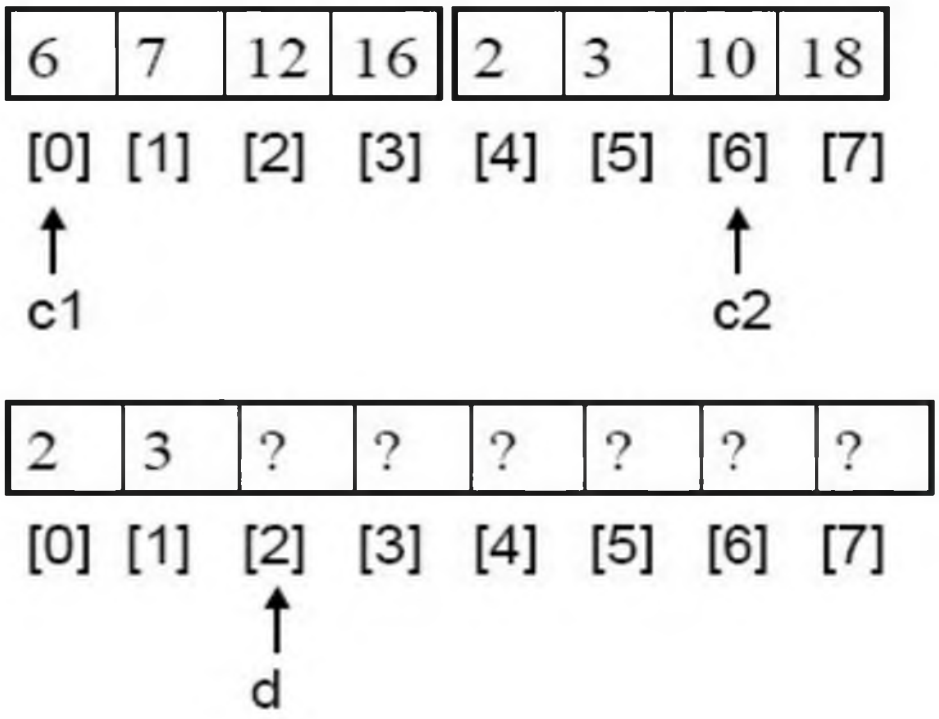
Sắp xếp trộn - Merge Sort

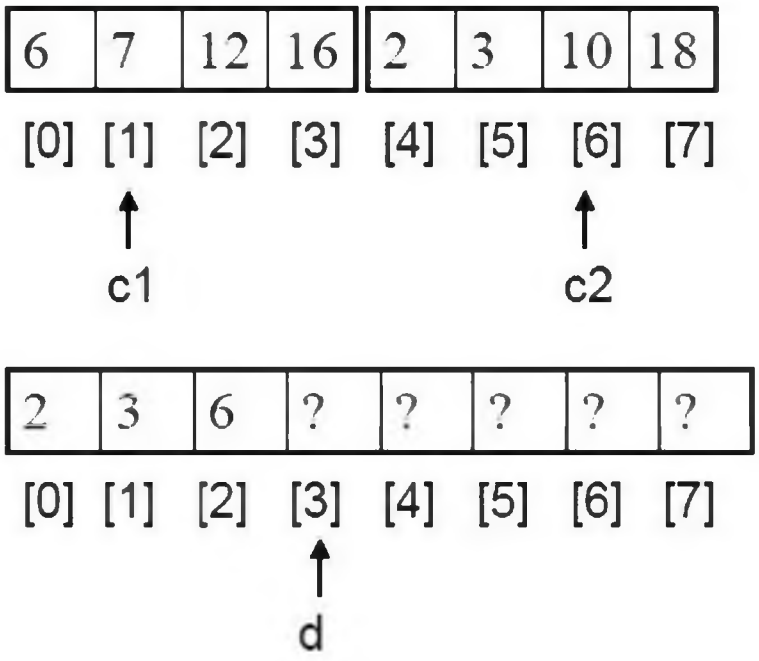
- Mảng A chia làm 02 phần bằng nhau.
- Sắp xếp 02 phần
- Trộn 02 nửa lại


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

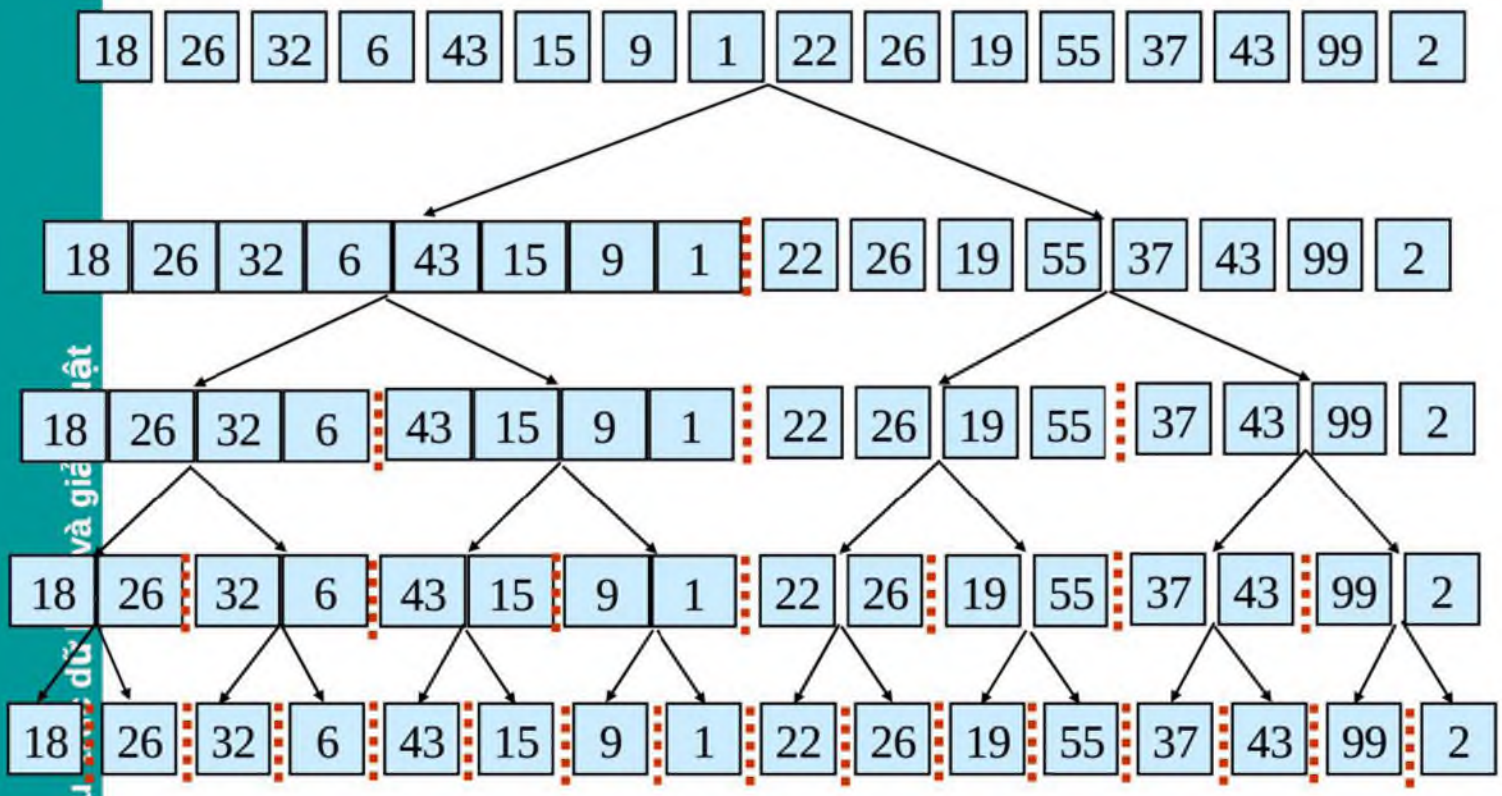

6
7
12
16
2
3
10
18
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
c1
c2
?
?
?
?
?
?
?
?
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
d


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

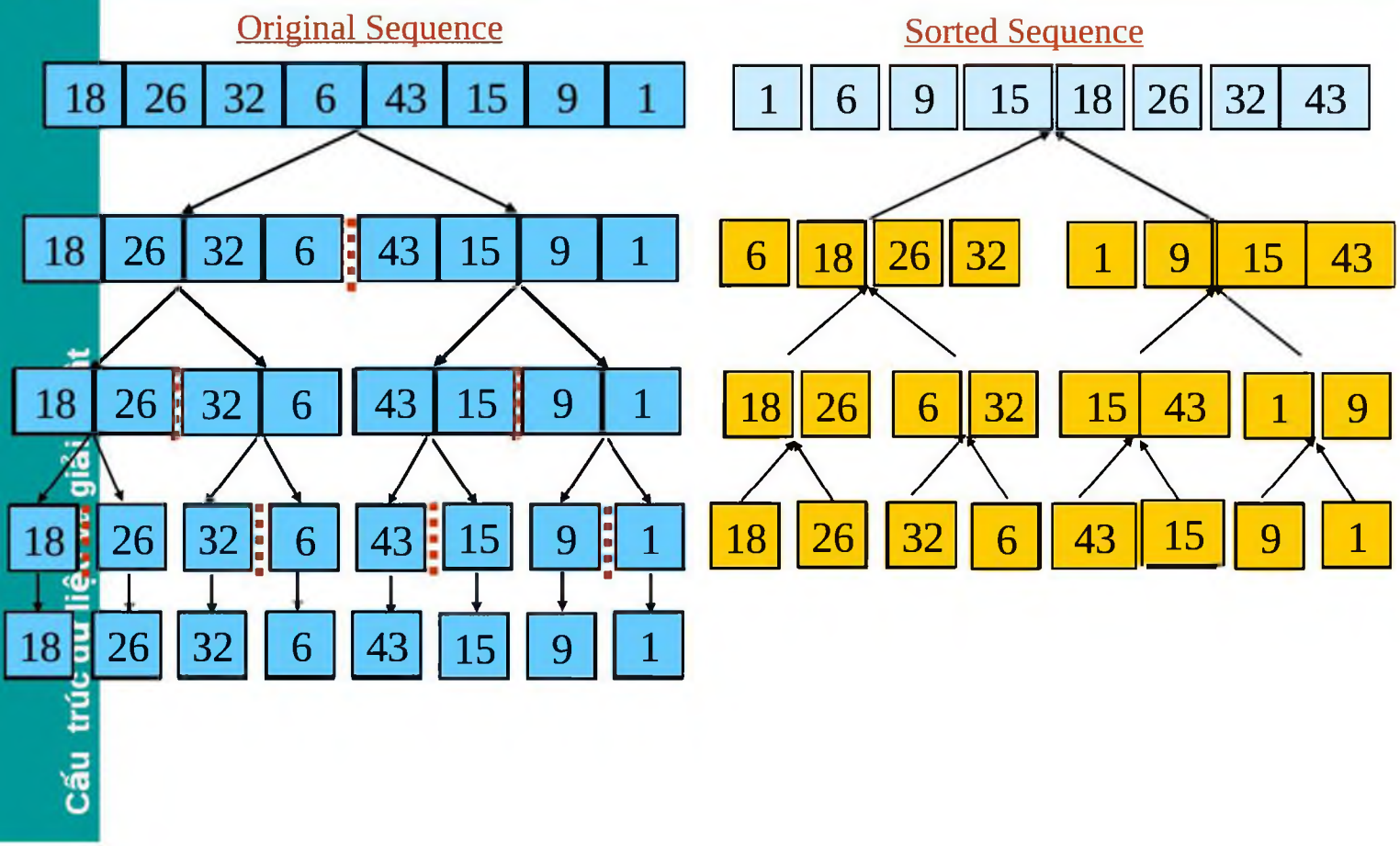

6
7
12
16
2
3
10
18
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
c1
c2
2
3
?
?
?
?
?
?
[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
d


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 – 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

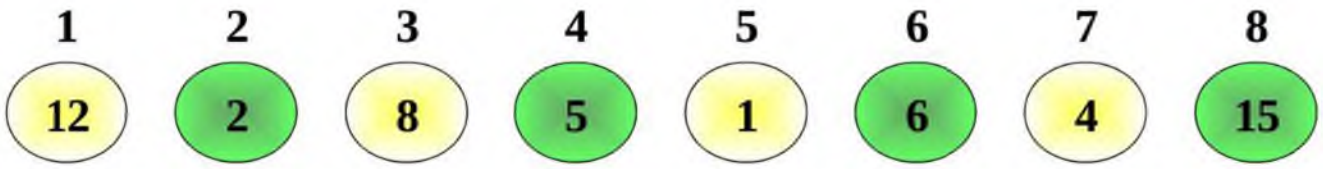
k = 1;

Phaân phoái ñeàu luaân phieân

Merge sort – Ví dụ

k = 1;

Phaân phoái ñeàu luaân phieân

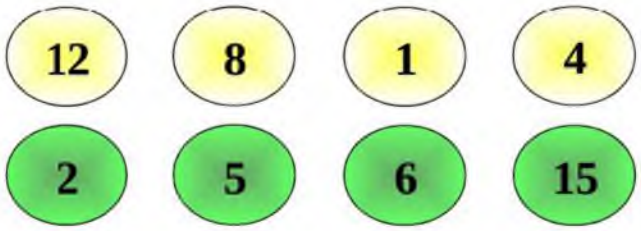


Merge sort – Ví dụ

k = 1;

Troän töøng caëp ñöôøng chaïy

0	1	2	3	4	5	6	7



Merge sort – Ví dụ

k = 1;

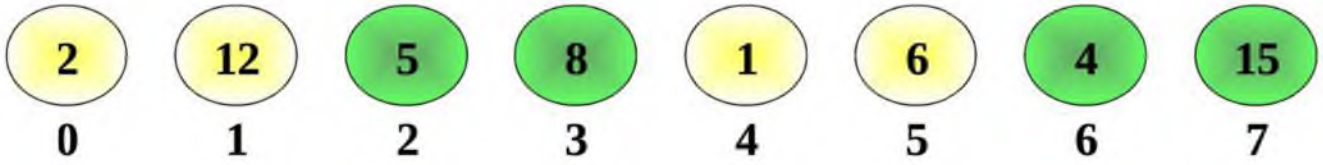
Troän töøng caëp ñöôøng chaïy

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

Merge sort – Ví dụ

k = 2;

Phaân phoái ñeàu luaân phieân

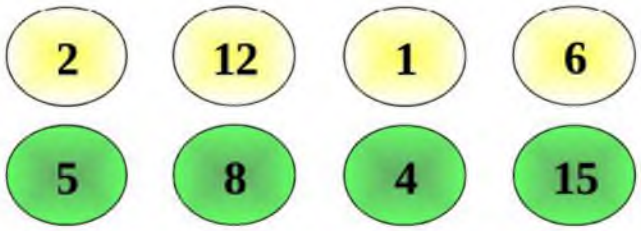


Merge sort – Ví dụ

k = 2;

Troän töøng caëp ñöôøng chaïy

0	1	2	3	4	5	6	7



Merge sort – Ví dụ

k = 2;

Troän töøng caëp ñöôøng chaïy

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

Merge sort – Ví dụ

k = 4;

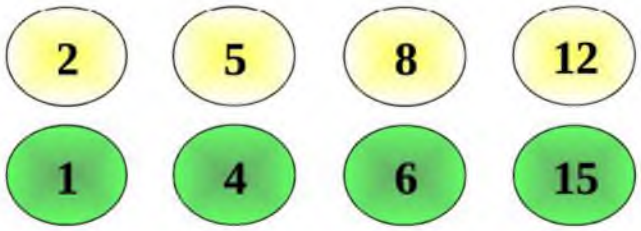
Phaân phoái ñeàu luaân phieân

Merge sort – Ví dụ

k = 4;

Troän töøng caëp ñöôøng chaïy

0	1	2	3	4	5	6	7

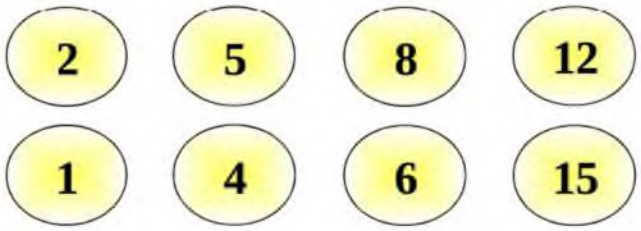


Merge sort – Ví dụ

k = 4;

Troän töøng caëp ñöôøng chaïy

0	1	2	3	4	5	6	7



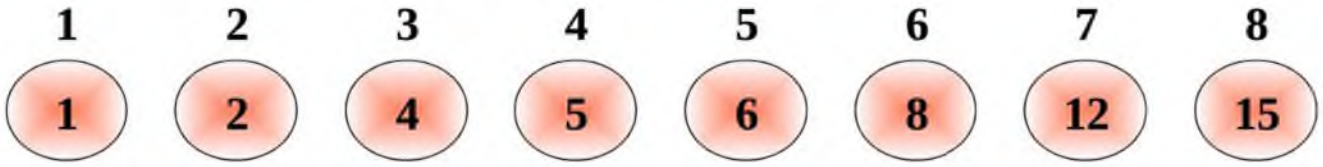
Merge sort – Ví dụ

k = 8;

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

Only one subarray

Merge sort – Ví dụ



Merge Sort – Caøi ñaët

- Döõ lieäu hoã trôï: 2 maûng b, c:

  int b[MAX], c[MAX], nb, nc;
- Caùc haøm caàn caøi ñaët:
  - **void MergeSort(int a[], int N);** : Saép xeáp maûng (a, N) taêng daàn
  - **void Distribute(int a[], int N, int &nb, int &nc, int k);** : Phaân phoái ñeàu luaân phieân caùc daõy con ñoä daøi k töø maûng a vaøo hai maûng b vaø c
  - **void Merge(int a[], int nb, int nc, int k);** : Troän maûng b vaø maûng c vaøo maûng a
  - **void MergeSubarr(int a[], int nb, int nc, int &pa, int &pb, int &pc, int k);** : Troän 1 caëp daõy con töø b vaø c vaøo a

Merge sort – Caøi ñaët

```
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 – Caøi ñaët

```
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*