CẤU TRÚC DỮ LIỆU

Chương 2. MỘT SỐ GIẢI THUẬT CƠ BẢN

Created by Bich Ngan

02/2012

1. Giải thuật tìm kiếm trên mảng số

1.1. Tìm kiếm tuyến tính (Linear Search)

- **Ý tưởng:**

  Thuật toán tiến hành so sánh x lần lượt với các phần tử thứ 1, thứ 2,… của mảng a cho đến khi gặp phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy x.

- **Ví dụ:** Cho dãy số sau:

  5 3 6 8 9 1 2

  Tìm phần tử có giá trị x = 9, x= 10

Minh họa ví dụ

Xét dãy số A có 7 phần tử:

5	3	6	8	9	1	2

Tìm x = 9

Vị trí trong dãy (i)	0	1	…	4
Giá trị a[i]	5	3	…	9
Kết quả so sánh a[i] và x	5!= 9	3!=9	…	9= = 9

Tìm được x=9 tại vị trí 4.

Kết thúc quá trình

Minh họa ví dụ

Xét dãy số A có 7 phần tử:

5 3 6 8 9 1 2

Tìm x = 10

Vị trí trong dãy (i)	0	1	…	6	7
Giá trị a[i]	5	3	…	2	null
Kết quả so sánh a[i] và x	5!= 10	3!=10	…	2!= 10	

i=7, a[i] không tồn tại. →Không có 10 trong dãy A.

Kết thúc quá trình

1.1. Tìm kiếm tuyến tính (Linear Search) (tt)

- **Cài đặt**

```
int LinearSearch (int a[], int n, int x)
{
        for(int i=0;i<n;i++)
            if(a[i]==x)
                    return i;          // trả về vị trí của x trong a
        return -1; // trả về -1 báo là không có x trong a
}
```

1.2. Thuật toán tìm kiếm nhị phân (Binary Search)

- **Ý tưởng**
  - Giả sử dãy số a đã có thứ tự tăng.
  - Tại mỗi bước tiến hành so sánh x với phần tử nằm vị trí giữa của dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này để quyết định giới hạn dãy tìm ở bước kế tiếp là nửa trên hay nửa dưới của dãy tìm kiếm hiện hành.
- **Ví dụ:**
  - Cho dãy a có 7 phần tử: 3 4 6 8 9 10 13
  - Tìm x = 10 và x = 2

Minh họa ví dụ

- Cho dãy số a: 3 4 6 8 9 10 13

tìm x= 10.

Bước	left	right	Mid = [(left+right)/2]	a[mid]	Kết quả so sánh x và a[mid]
1	0	6	[(0+6)/2] = 3	8	8<10
2	Left = mid +1 = 3+1 = 4	6	[(4+6)/2] = 5	10	10 = = 10

→ **Tìm được x=10 tại vị trí 5.**

Kết thúc quá trình

Minh họa ví dụ

- Cho dãy số a: -1 0 6 8 9 10 13

tìm x= 2.

Bước	left	right	Mid = [(left+right)/2]	a[mid]	Kết quả so sánh x và a[mid]
1	0	6	[(0+6)/2] = 3	8	8>2
2	0	Right = mid – 1 = 3-1 = 2	[(0+2)/2] = 1	0	0 < 2
3	Left = mid +1 = 1+1 = 2	2	2	6	6>2
4	2	Mid-1 = 2-1=1	?		

Tại bước 4 có left>right -> vô lý.
Kết luận không có x = 2 trong a

1.2. Tìm kiếm nhị phân (Binary Search) (tt)

- **Cài đặt**

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

Bài tập lý thuyết

- Cho dãy số sau:

7 9 13 17 27 30 31 35 38 40

a. Tìm x= 17, x=35, x=40

b. Tìm x = 23, x=10, x=36

- Cho dãy ký tự

Z R L K H F E C A

a. Tìm x = R, x = C

b. Tìm x = D, x = Q

Bài tập thực hành

Cho mảng 1 chiều a chứa n số nguyên. Viết chương trình thực hiện các yêu cầu sau:

1. Viết hàm nhập/xuất mảng a.
2. Tìm max/min của a.
3. Đếm số phần tử chẵn/lẻ trong a.
4. Tìm kiếm phần tử x trong a theo 2 dạng (trả về vị trí/xuất câu thông báo) với giải thuật tìm kiếm tuyến tính/ tìm kiếm nhị phân.
5. Tìm trên a có bao nhiêu phần tử x.

Bài tập (tt)

6. Tìm trên a có bao nhiêu phần tử lớn hơn x.
7. Tính tổng các phần tử của a.
8. Xuất các số nguyên tố trong a.
9. Xuất các số hoàn thiện trong a.
10. Xuất các phần tử ở vị trí chẵn/ vị trí lẻ trong a.
11. Xuất giá trị max/min kèm theo vị trí của giá trị đó trong mảng a.
12. Cho 2 dãy số b có m phần tử, dãy c có n phần tử. Ghép b và c thành dãy a được xếp tăng dần.

2 Các giải thuật sắp xếp cơ bản

Đường link xem demo của các giải thuật sắp xếp:

http://www.cs.ubc.ca/~harrison/Java/sorting-demo.html

http://maven.smith.edu/~thiebaut/java/sort/demo.html

2. CÁC GIẢI THUẬT SẮP XẾP

(Các thuật toán minh họa sắp xếp dãy không tăng trên mảng 1 chiều chứa dữ liệu là các số nguyên)

Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu trữ tại mỗi phần tử.

Các phương pháp sắp xếp thông dụng

2.1. Sắp xếp đổi chỗ trực tiếp – Interchange Sort

2.2. Sắp xếp chọn trực tiếp – Selection Sort

2.3. Sắp xếp chèn trực tiếp – Insertion Sort

2.4. Sắp xếp Nổi bọt – Bubble Sort

2.1. Sắp xếp đổi chỗ trực tiếp – interchange Sort

- **Khái niệm nghịch thế:**
  - Xét dãy các số a: a_1, a_2, … a_n, với a là dãy không giảm.
    Nếu i<j và a_i>a_j thì ta gọi đó là 1 nghịch thế.
  - **Ví dụ**: Cho dãy số a như sau:

    14 5 7 8 3.

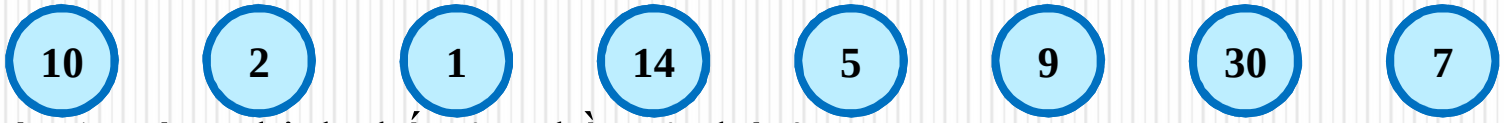
Vậy dãy trên trên có các cặp nghịch thế sau: (14, 5); (7, 3); (8, 3)….

- **Ý tưởng thuật toán:**

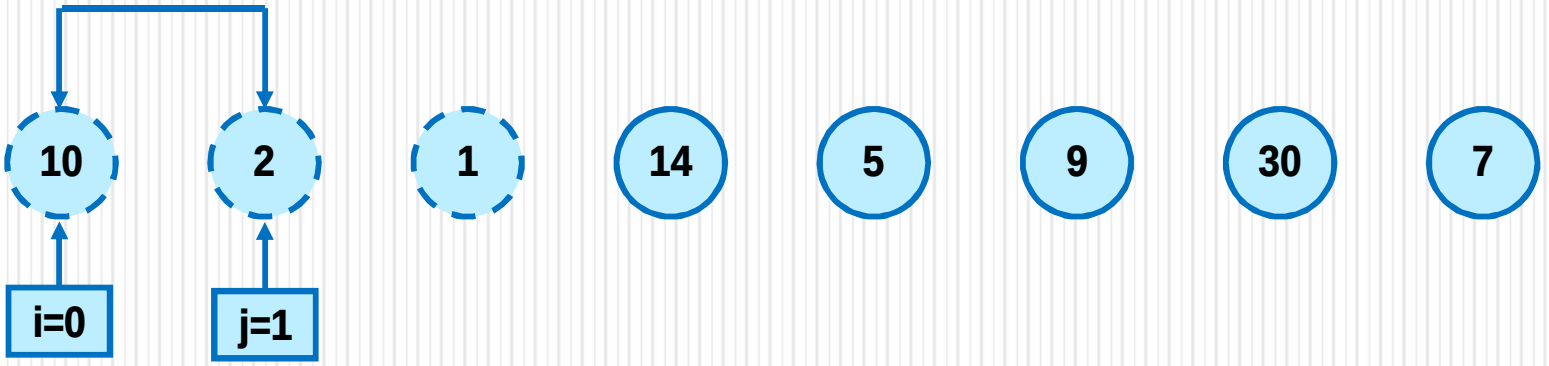
Xuất phát từ đầu dãy, lần lượt xét từng phần tử cho đến cuối dãy. Tại mỗi phần tử **tìm tất cả nghịch thế** chứa phần tử này, **đổi chỗ phần tử này với các phần tử trong cặp nghịch thế**.

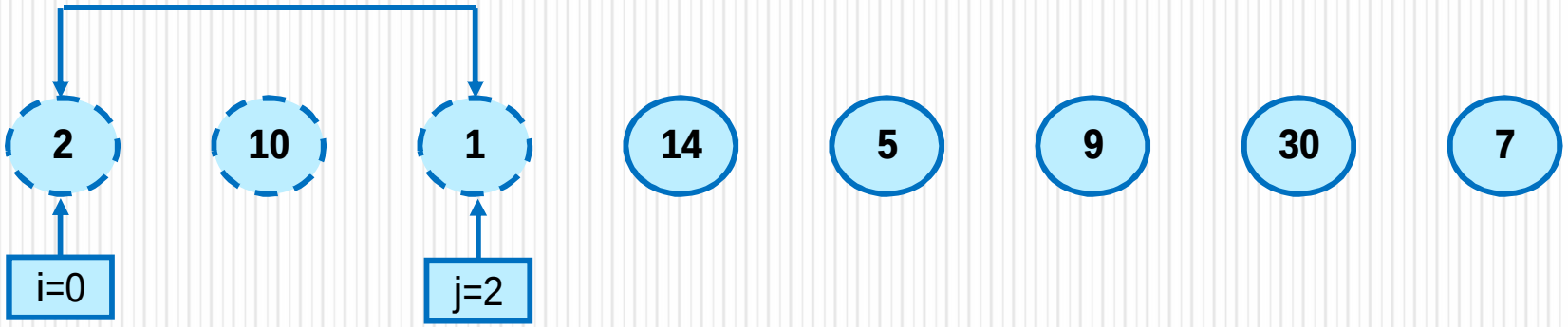
Minh họa ví dụ interchange sort

- Cho dãy số a



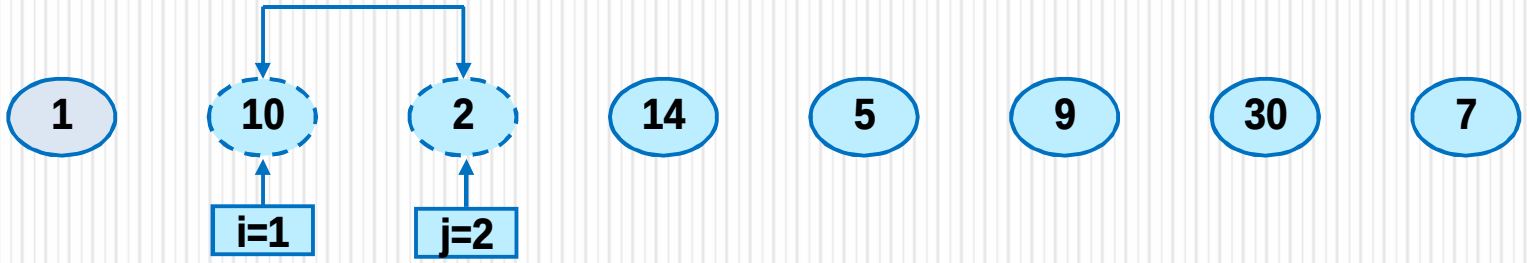
- Bước 1: xét nghịch thế của phần tử thứ 0 – a_0



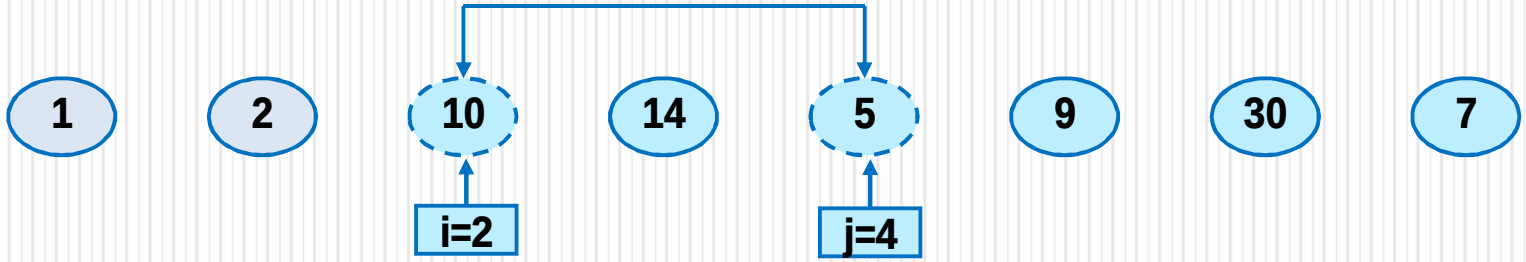


Minh họa ví dụ interchange sort (tt)

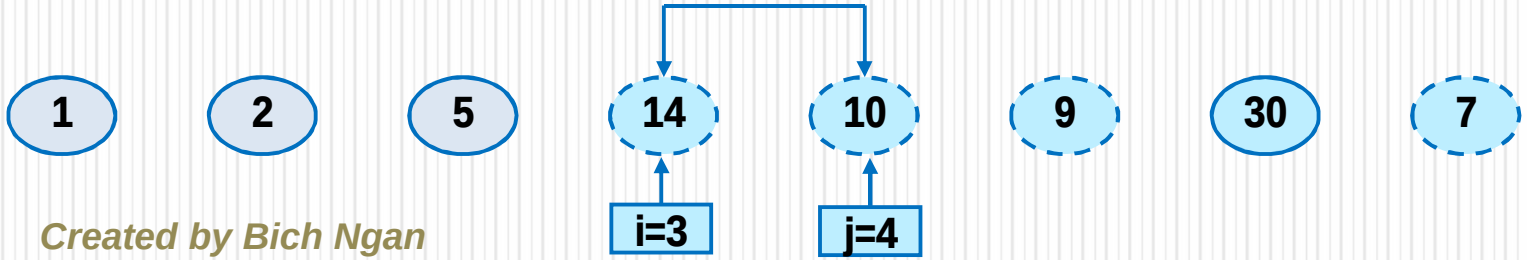
❖ Bước 2: xét nghịch thế của phần tử thứ 1 – a_1



❖ Bước 3: xét nghịch thế của phần tử thứ 2 – a_2

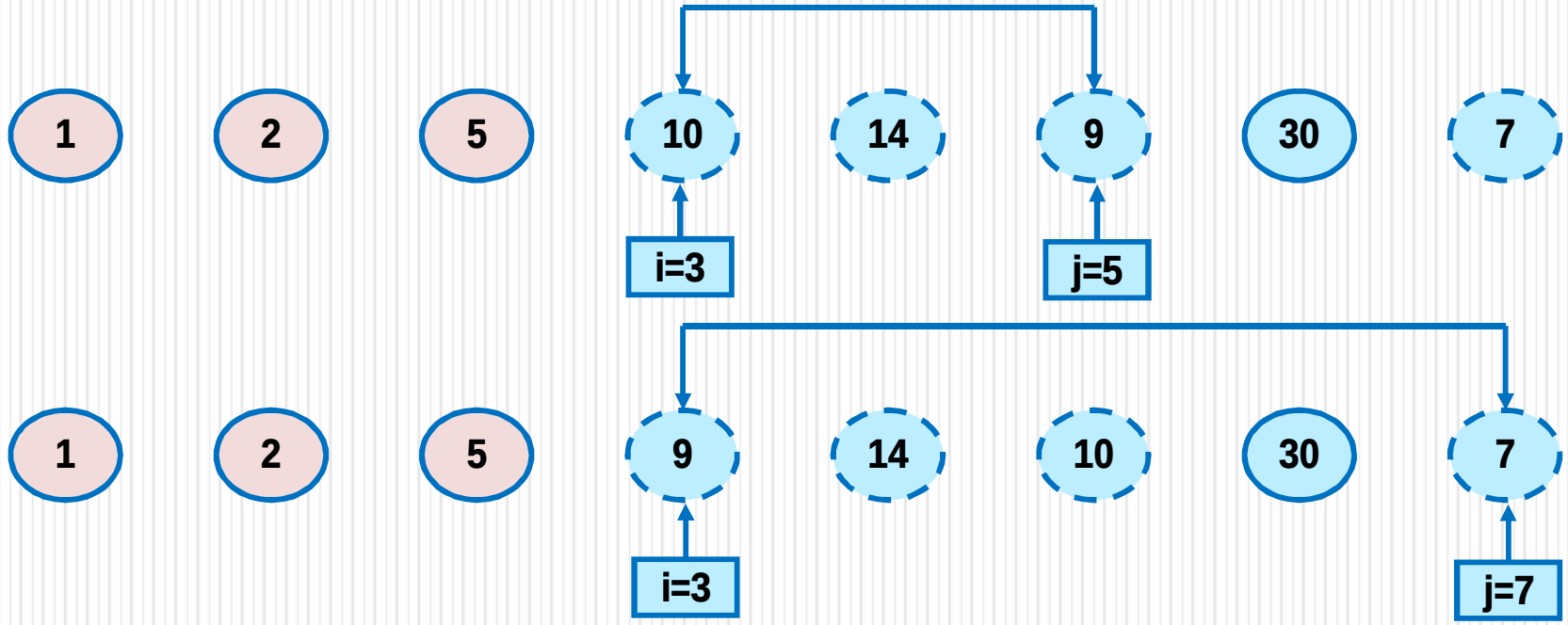


❖ Bước 4: xét nghịch thế của phần tử thứ 3 – a_3

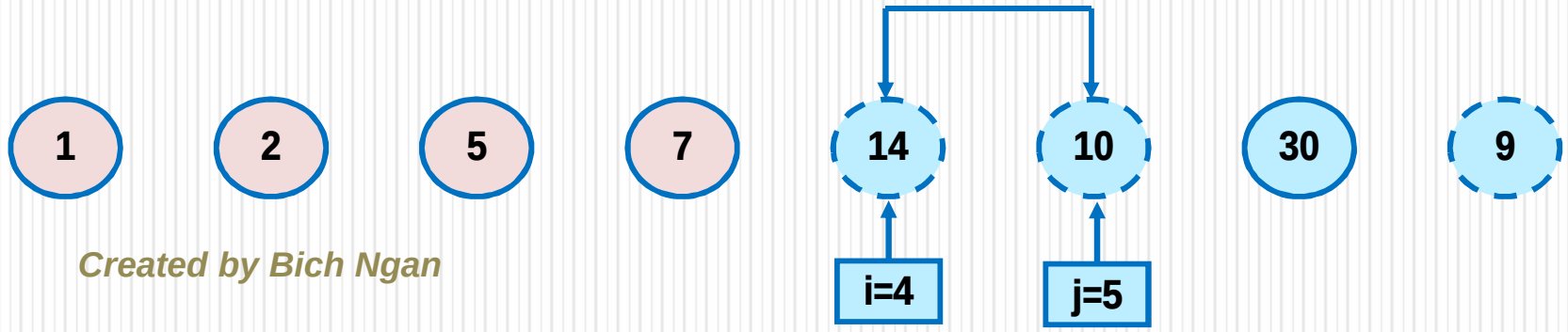


Minh họa ví dụ interchange sort (tt)

❖ Bước 4: xét nghịch thế của phần tử thứ 3 – a_3 tiếp theo

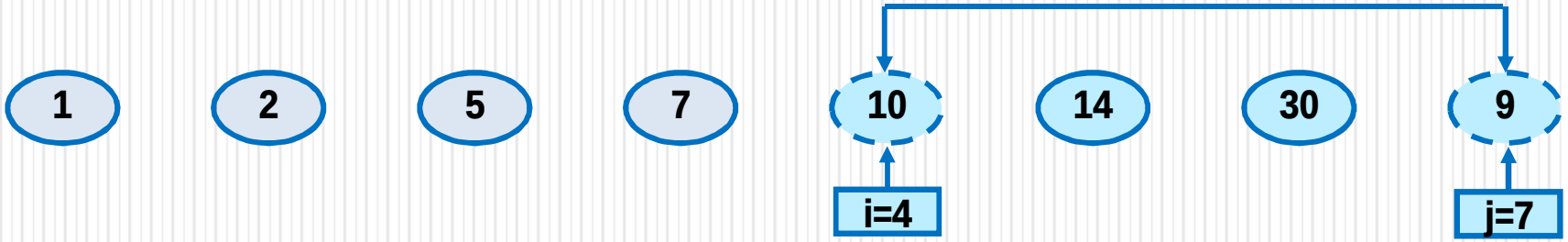


❖ Bước 5: xét nghịch thế của phần tử thứ 4 – a_4

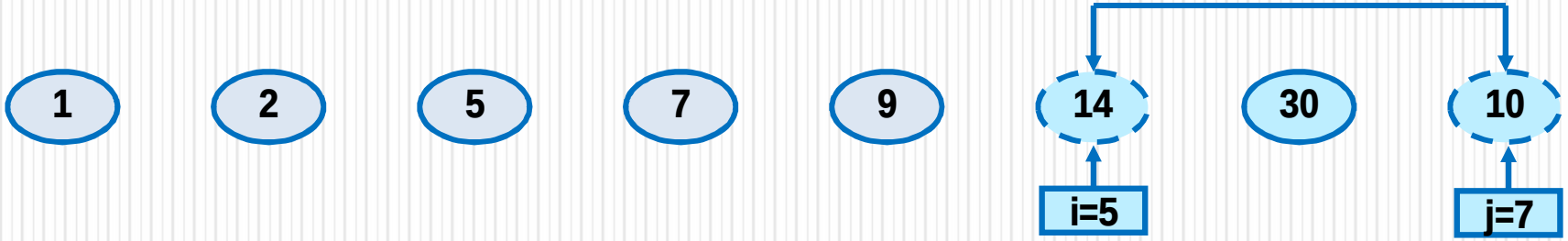


Minh họa ví dụ interchange sort (tt)

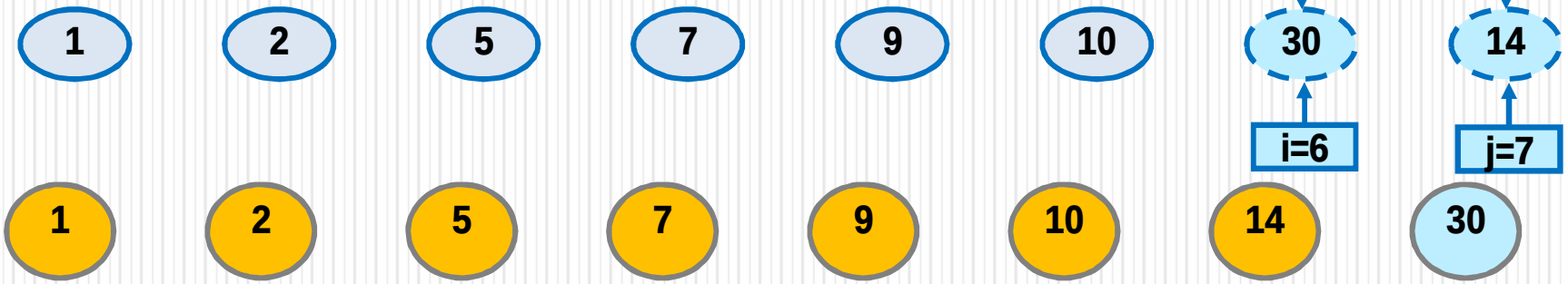
❖ Bước 4: xét nghịch thế của phần tử thứ 4 – a_4 tiếp theo



❖ Bước 5: xét nghịch thế của phần tử thứ 5 – a_5 tiếp theo



❖ Bước 5: xét nghịch thế của phần tử thứ 6 – a_6



Dừng. Vậy dãy đã được sắp xếp.

Ví dụ

- Minh họa thao tác sắp xếp dữ liệu theo phương pháp interchange Sort cho các dãy dữ liệu sau:
  - Sắp xếp tăng:
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**
  - Sắp xếp giảm
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**

Cài đặt

```
void Interchangesort(int a[],int n)
{
      for(int i=0;i<n-1;i++)
            for(int j=i+1;j<n;j++)
                  if(a[i]>a[j])
                  swap(a[i],a[j]); //ham hoan doi gia
                                   //tri 2 so nguyen
}
```

```
void swap(int &x, int &y)
{
      int t = x;
      x = y;
      y = t;
}
```

Bài tập cài đặt

- Viết bổ sung các hàm vào chương trình xử lý mảng 1 chiều các hàm thực hiện những yêu cầu sau:

1. Viết hàm sắp xếp tăng theo PP **interchange sort** cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.
2. Viết hàm sắp xếp tăng theo PP interchange sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.

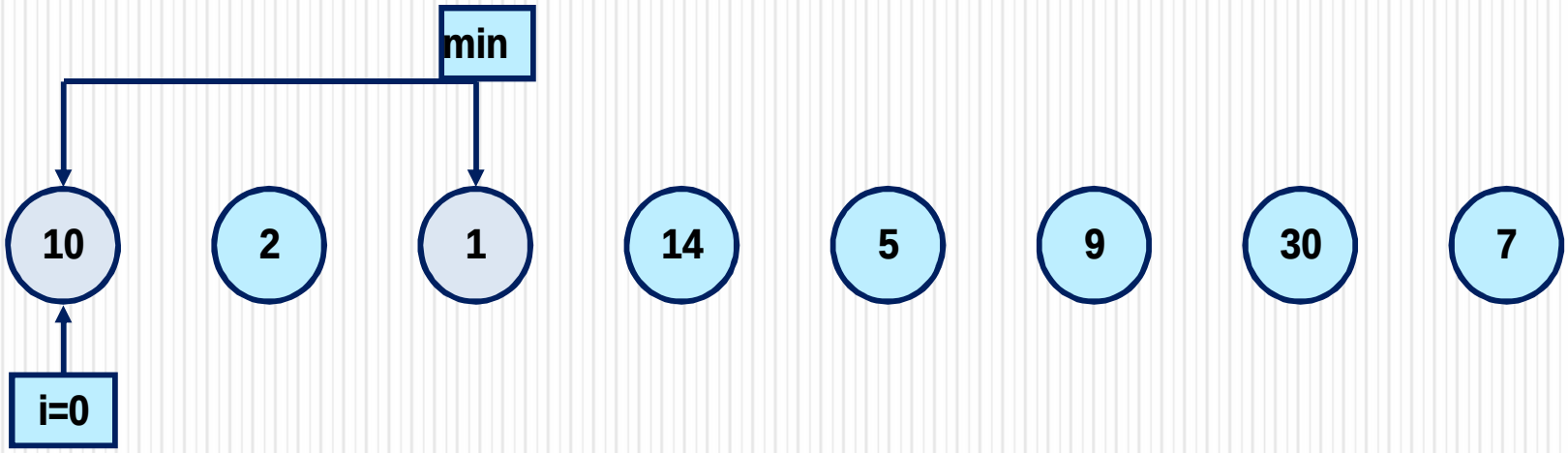
2.2. Sắp xếp chọn trực tiếp – Selection sort

- **Ý tưởng giải thuật**
  - Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí thứ 0 của dãy hiện hành; sau đó không quan tâm đến nó nữa, xem dãy hiện hành chỉ còn (n – 1) phần tử, bắt đầu từ vị trí thứ 1; lặp lại quá trình đó trên dãy hiện hành… đến khi dãy hiện hành chỉ còn 1 phần tử thì dừng

Minh họa ví dụ Selection sort

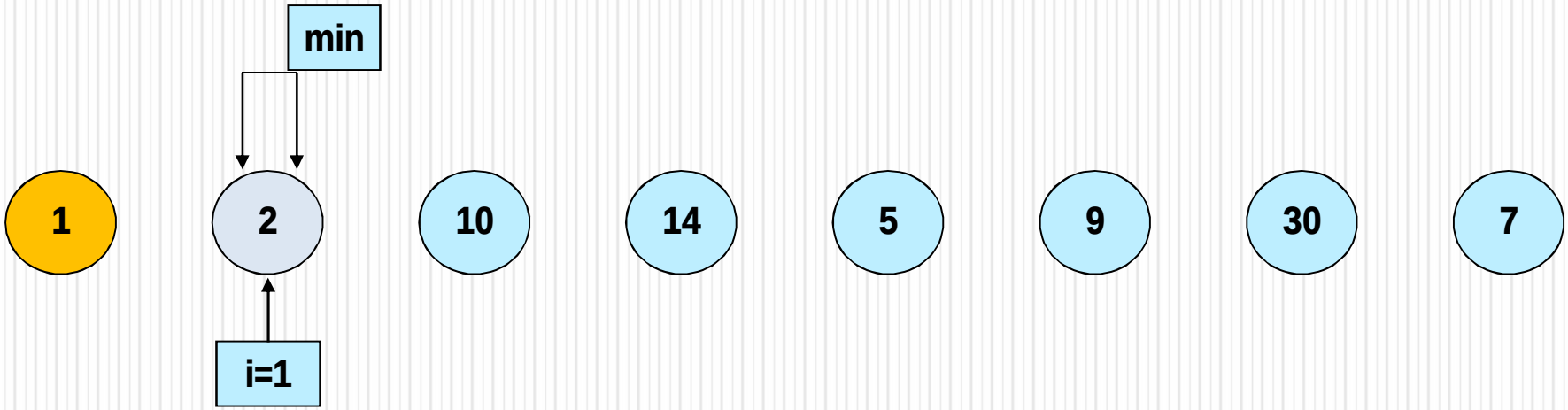
- Cho dãy số a

- Bước 1: Tìm min của dãy số từ $a_0 – a_{n-1}$. Sau đó hoán đổi min với a_0

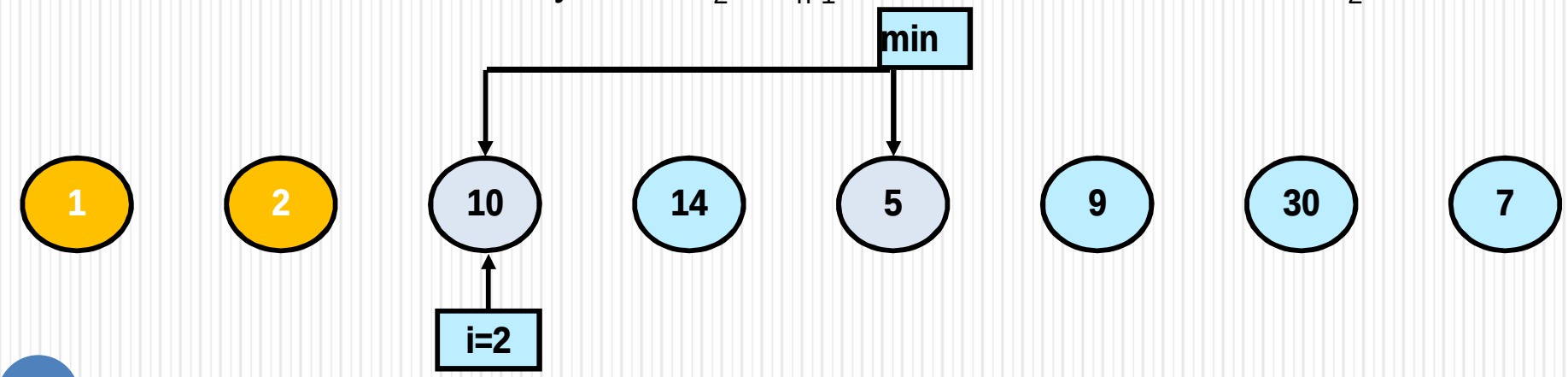


Minh họa ví dụ Selection sort (tt)

❖ Bước 2: Tìm min của dãy số từ $a_1 – a_{n-1}$. Sau đó hoán đổi min với a_1

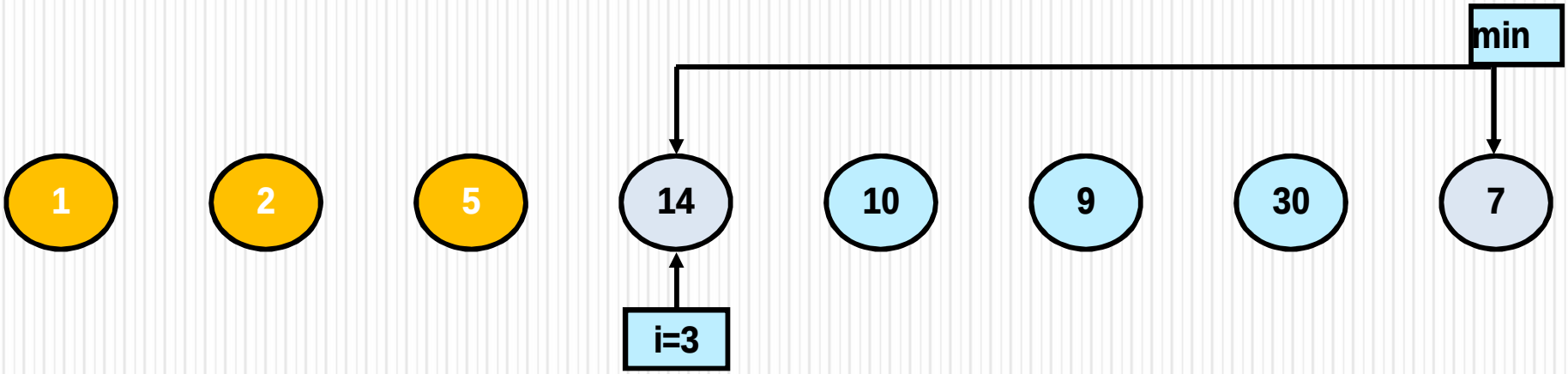


❖ Bước 3: Tìm min của dãy số từ $a_2 – a_{n-1}$. Sau đó hoán đổi min với a_2

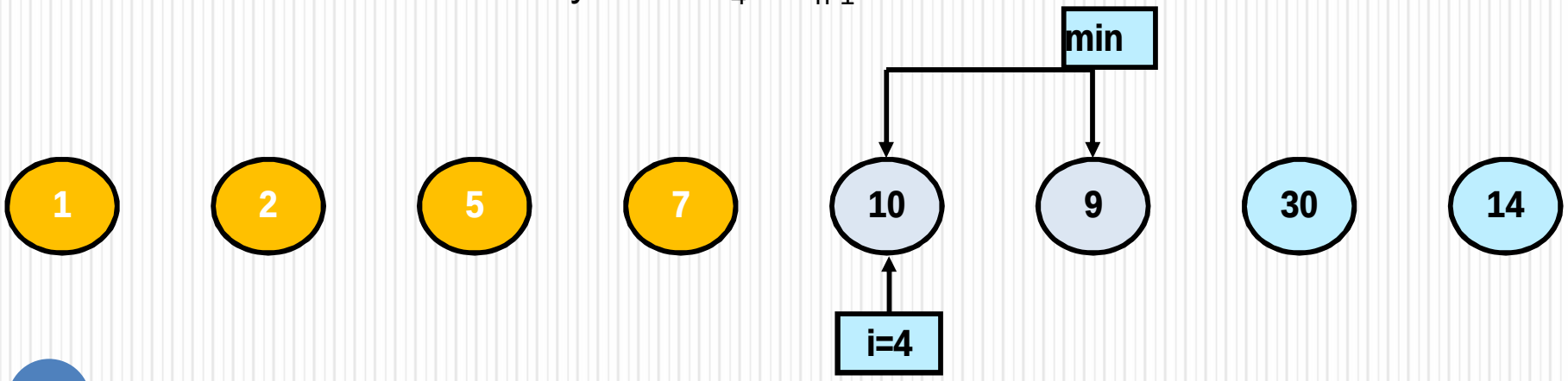


Minh họa ví dụ Selection sort (tt)

❖ Bước 4: Tìm min của dãy số từ $a_3 – a_{n-1}$. Sau đó hoán đổi min với a_3

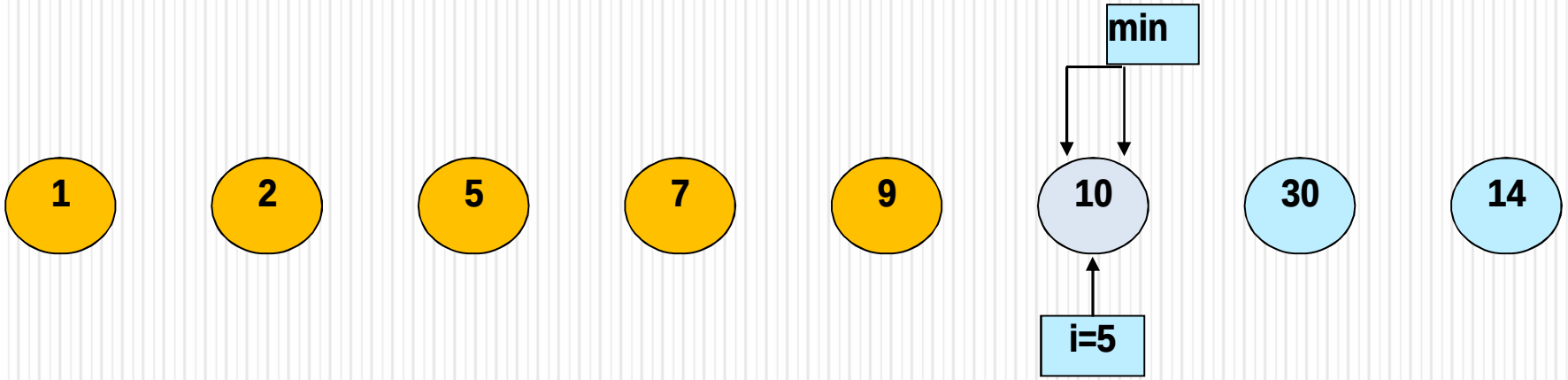


❖ Bước 5: Tìm min của dãy số từ $a_4 – a_{n-1}$. Sau đó hoán đổi min với a4

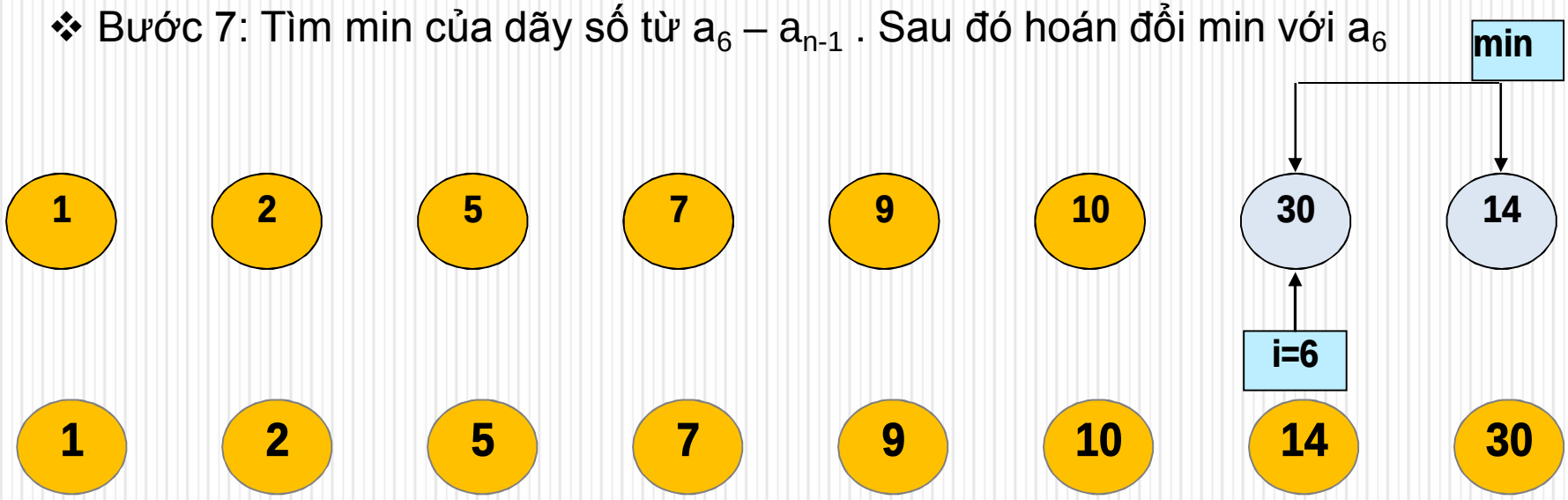


Minh họa ví dụ Selection sort (tt)

❖ Bước 6: Tìm min của dãy số từ $a_5 – a_{n-1}$. Sau đó hoán đổi min với a_5



❖ Bước 7: Tìm min của dãy số từ $a_6 – a_{n-1}$. Sau đó hoán đổi min với a_6



Dừng. Vậy dãy đã được sắp xếp.

Ví dụ

- Minh họa thao tác sắp xếp dữ liệu theo phương pháp Selection Sort cho các dãy dữ liệu sau:
  - Sắp xếp tăng:
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**
  - Sắp xếp giảm
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**

Cài đặt

```
void SelectionSort(int a[],int n)
{
        for(int i=0;i<n-1;i++)
        {
                int min=i;
                for(int j=i+1;j<=n-1;j++)  //tìm min của dãy số
                        if(a[j]<a[min])    //từ a_i → a_n-1
                                min=j;
                swap(a[min],a[i]);
        }
}
```

Bài tập cài đặt

- Viết bổ sung các hàm vào chương trình xử lý mảng 1 chiều các hàm thực hiện những yêu cầu sau:

1. Viết hàm sắp xếp tăng theo PP **selection sort** cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.
2. Viết hàm sắp xếp tăng theo PP interchange sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.

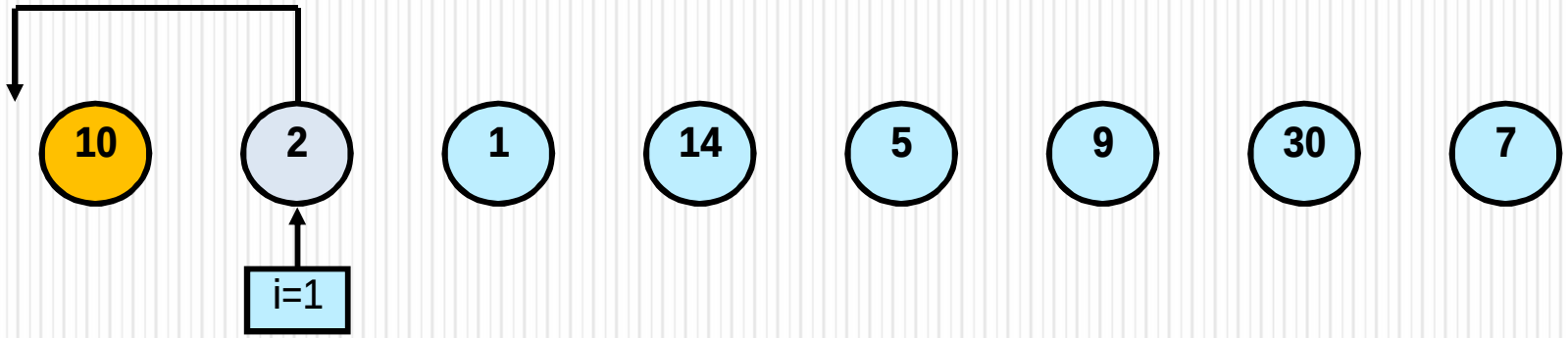
2.3. Sắp xếp chèn trực tiếp – Insertion Sort

- **Ý tưởng**
  - Giả sử có 1 dãy $a_1, a_2,\ldots,a_n$ trong đó (i – 1) phần tử đầu tiên $a_1, a_2,\ldots, a_{i-1}$ đã có thứ tự. Ý tưởng của giải thuật là tìm cách chèn phần tử a_i vào vị trí thích hợp của đoạn đã sắp xếp để có dãy mới $a_1, a_2,\ldots, a_i$ đã có thứ tự. Cứ như thế các phần tử tiếp theo cho đến hết dãy. Vậy ta được 1 dãy sắp xếp.

Minh họa ví dụ Insertion sort

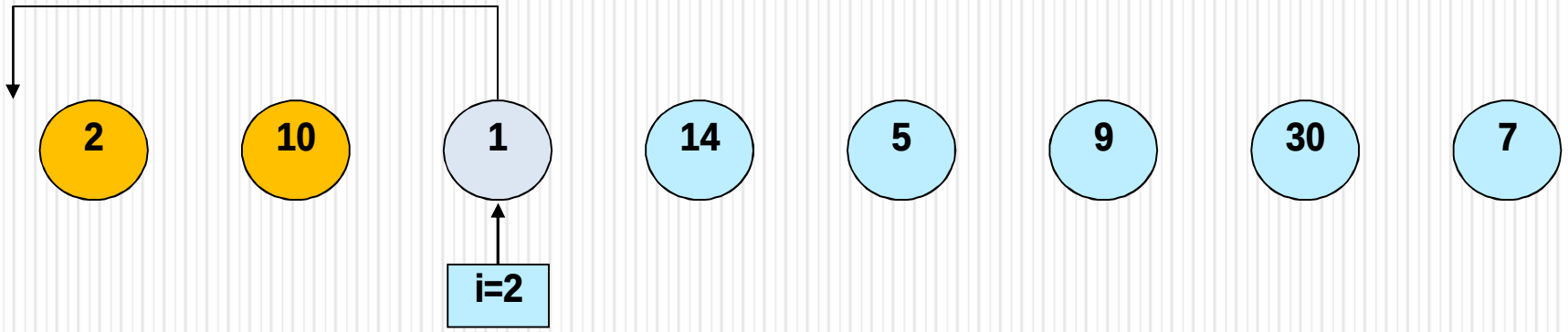
- Cho dãy số a

- Bước 1: Dãy a_0-a_0 đã có thứ tự. Cần chèn a_1 vào dãy này để dãy vẫn có thứ tự

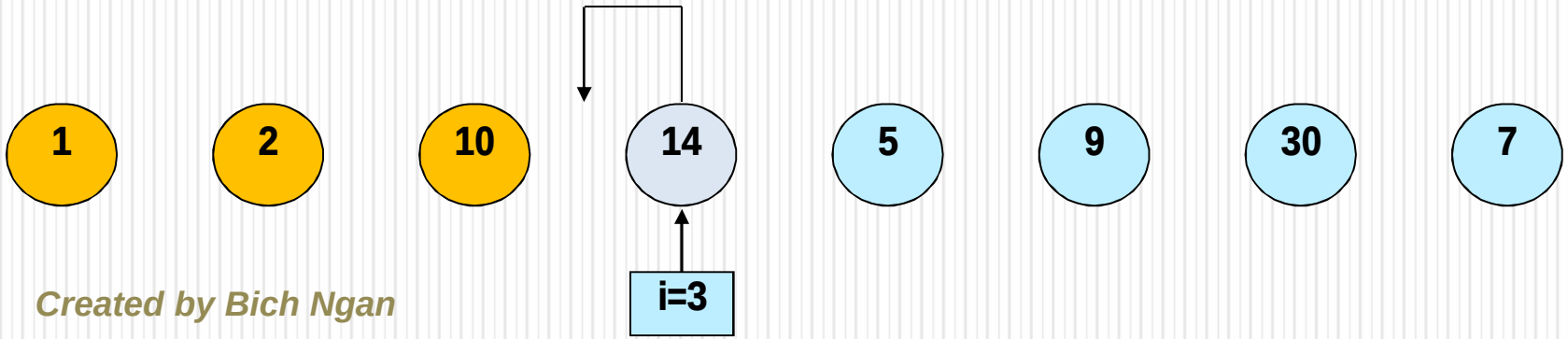


Minh họa ví dụ Insertion sort (tt)

Bước 2: Dãy a_1-a_0 đã có thứ tự. Cần chèn a_2 vào dãy này để dãy vẫn có thứ tự

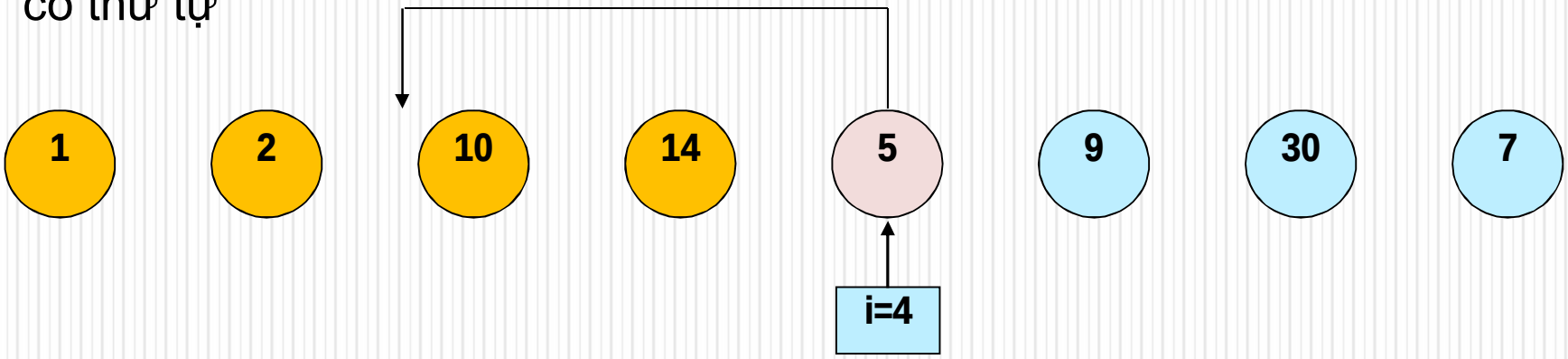


Bước 3: Dãy a_2-a_0 đã có thứ tự. Cần chèn a_3 vào dãy này để dãy vẫn có thứ tự

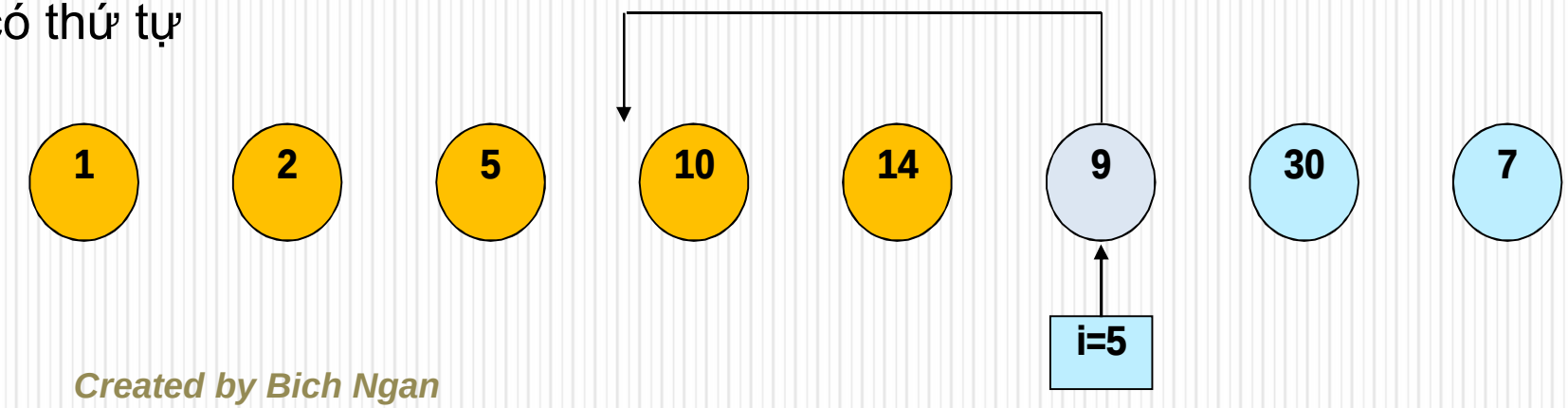


Minh họa ví dụ Insertion sort (tt)

Bước 4: Dãy a_3-a_0 đã có thứ tự. Cần chèn a_4 vào dãy này để dãy vẫn có thứ tự

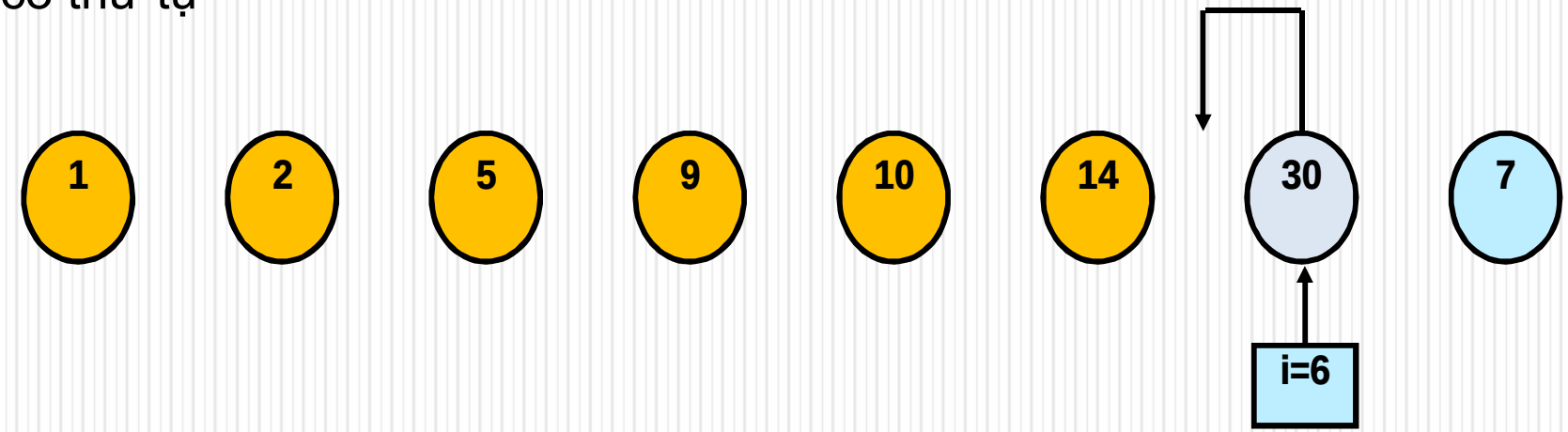


Bước 5: Dãy a_4-a_0 đã có thứ tự. Cần chèn a_5 vào dãy này để dãy vẫn có thứ tự

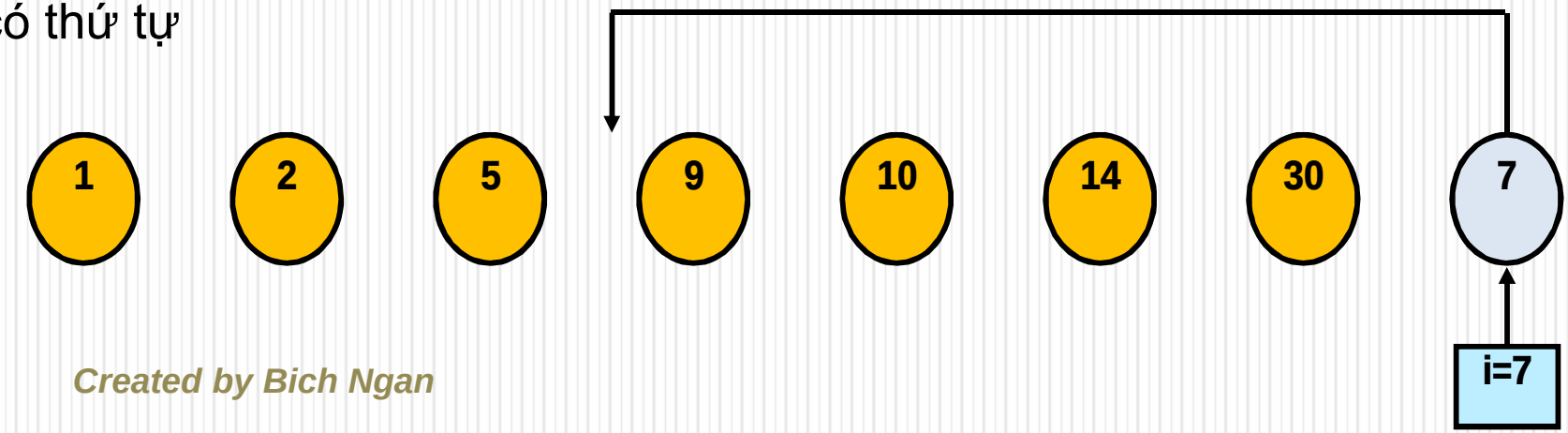


Minh họa ví dụ Insertion sort (tt)

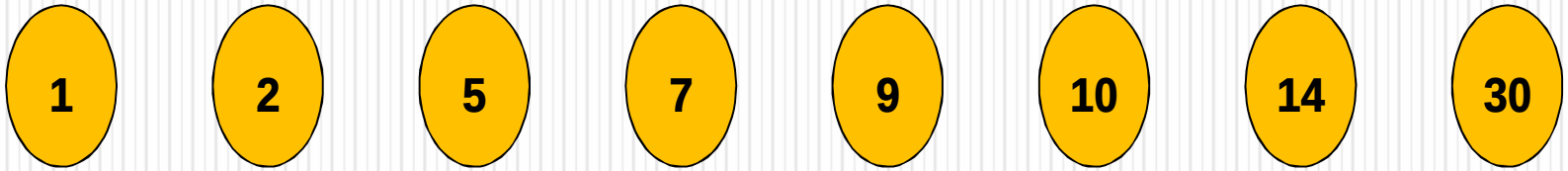
Bước 6: Dãy a_5-a_0 đã có thứ tự. Cần chèn a_6 vào dãy này để dãy vẫn có thứ tự



Bước 7: Dãy a_6-a_0 đã có thứ tự. Cần chèn a_7 vào dãy này để dãy vẫn có thứ tự



Minh họa ví dụ Insertion sort (tt)



Dừng. Vậy dãy đã được sắp xếp.

Ví dụ

- Minh họa thao tác sắp xếp dữ liệu theo phương pháp Insertion Sort cho các dãy dữ liệu sau:
  - Sắp xếp tăng:
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**
  - Sắp xếp giảm
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**

Cài đặt

```
void InsertionSort(int a[],int n)
{
        for(int i=1;i<n;i++)
        {
                int x=a[i];
                for(int j=i-1;j>=0;j- -)
                {
                        if(a[j]>x)      a[j+1]=a[j];
                        else                    break;
                }
                a[j+1]=x;
        }
}
```

Bài tập cài đặt

- Viết bổ sung các hàm vào chương trình xử lý mảng 1 chiều các hàm thực hiện những yêu cầu sau:

1. Viết hàm sắp xếp tăng theo PP **insertion sort** cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.
2. Viết hàm sắp xếp tăng theo PP interchange sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.

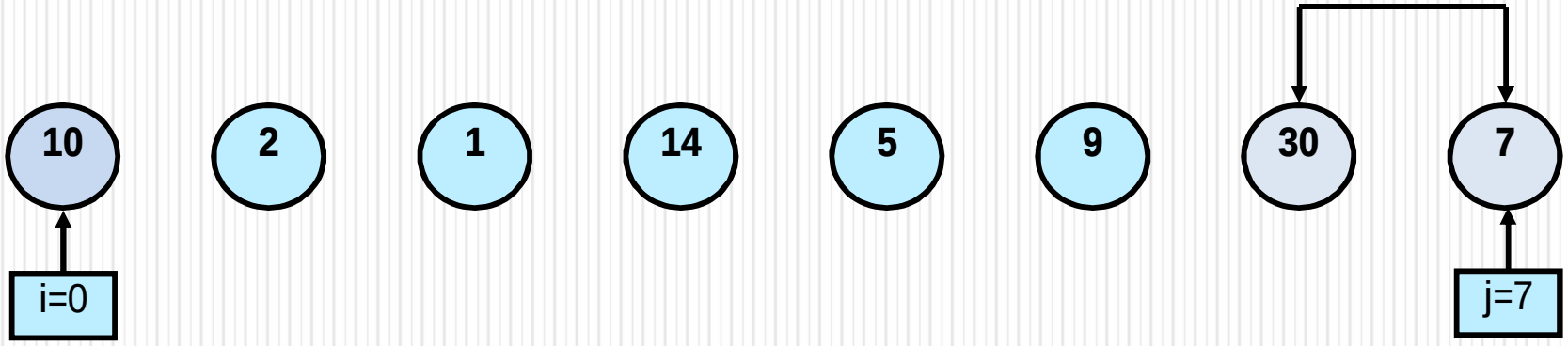
2.4. Sắp xếp Nổi bọt – Bubble Sort.

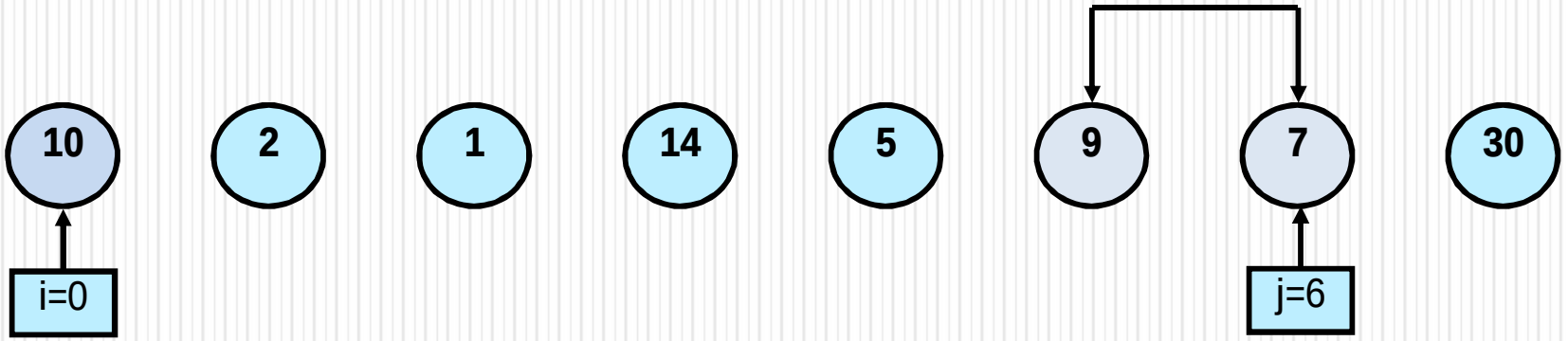
- **Ý tưởng:** (sắp tăng)
  - Dựa vào ý tưởng đưa phần tử nhỏ lên đầu mảng và lớn về phía sau mảng.
  - Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đứng đầu dãy hiện hành, sau đó không xét tới nó ở bước tiếp theo, do vậy ở lần lặp thứ i có vị trí đầu dãy là i.

Minh họa ví dụ Bubble sort

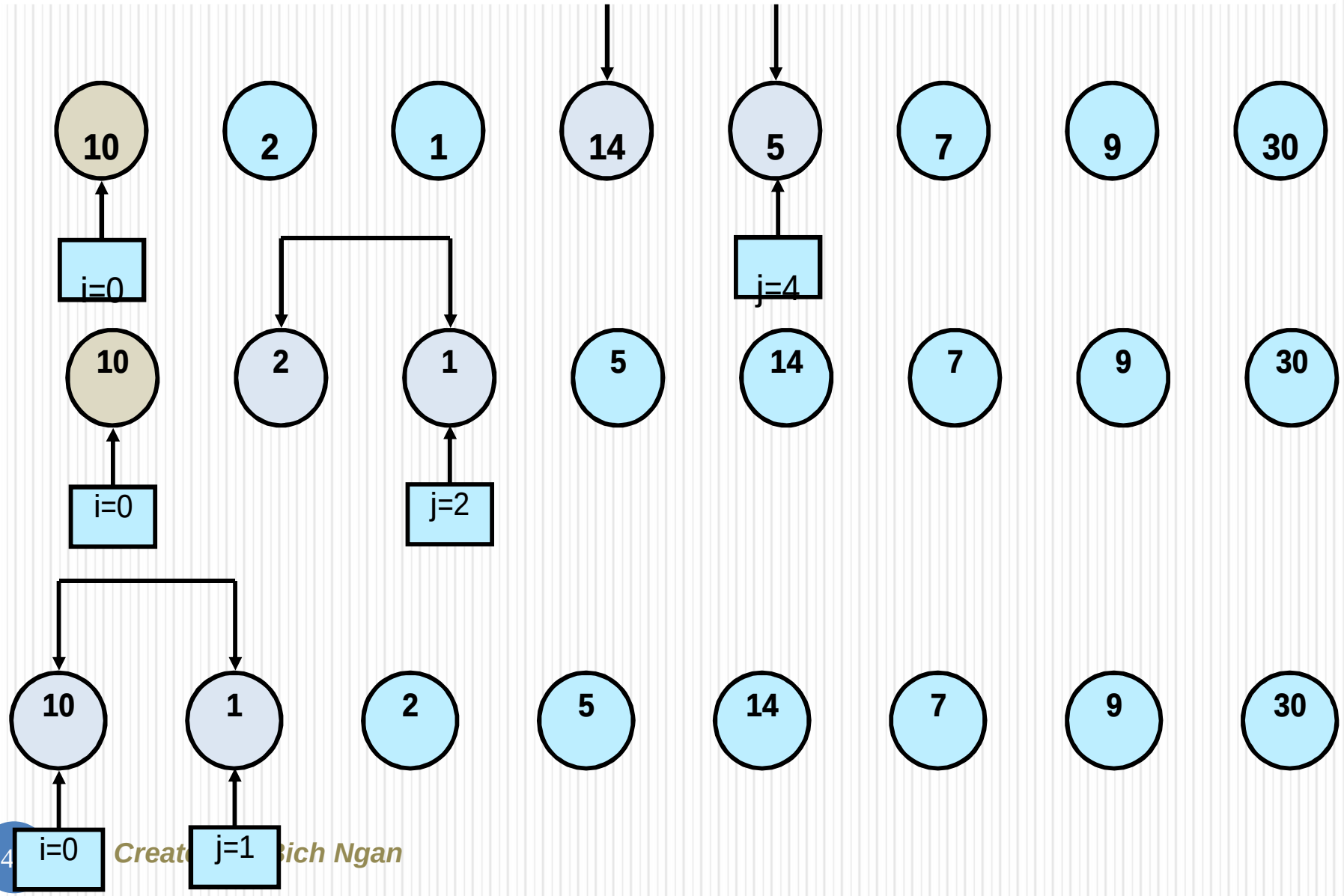
- Cho dãy số a

- Bước 1: Đưa phần tử nhỏ bắt đầu từ cuối mảng (j=7) lên vị trí i = 0



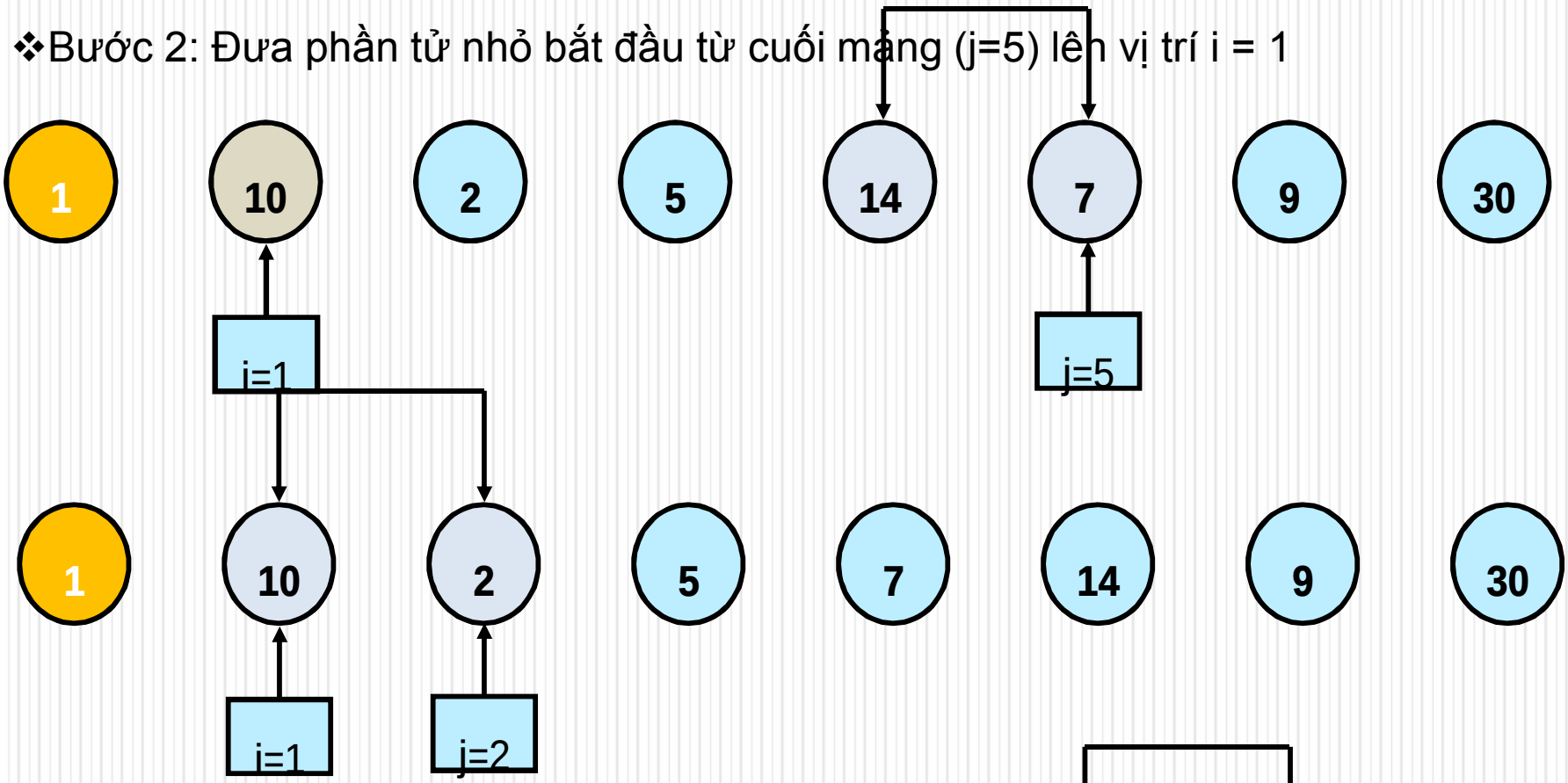


Minh họa ví dụ Bubble sort (tt)

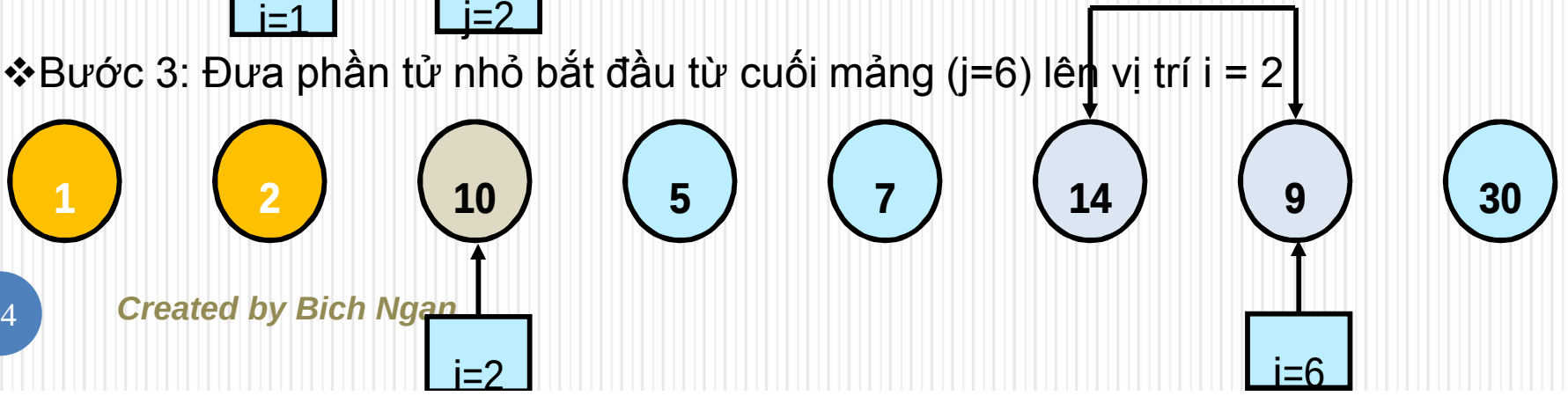


Minh họa ví dụ Bubble sort (tt)

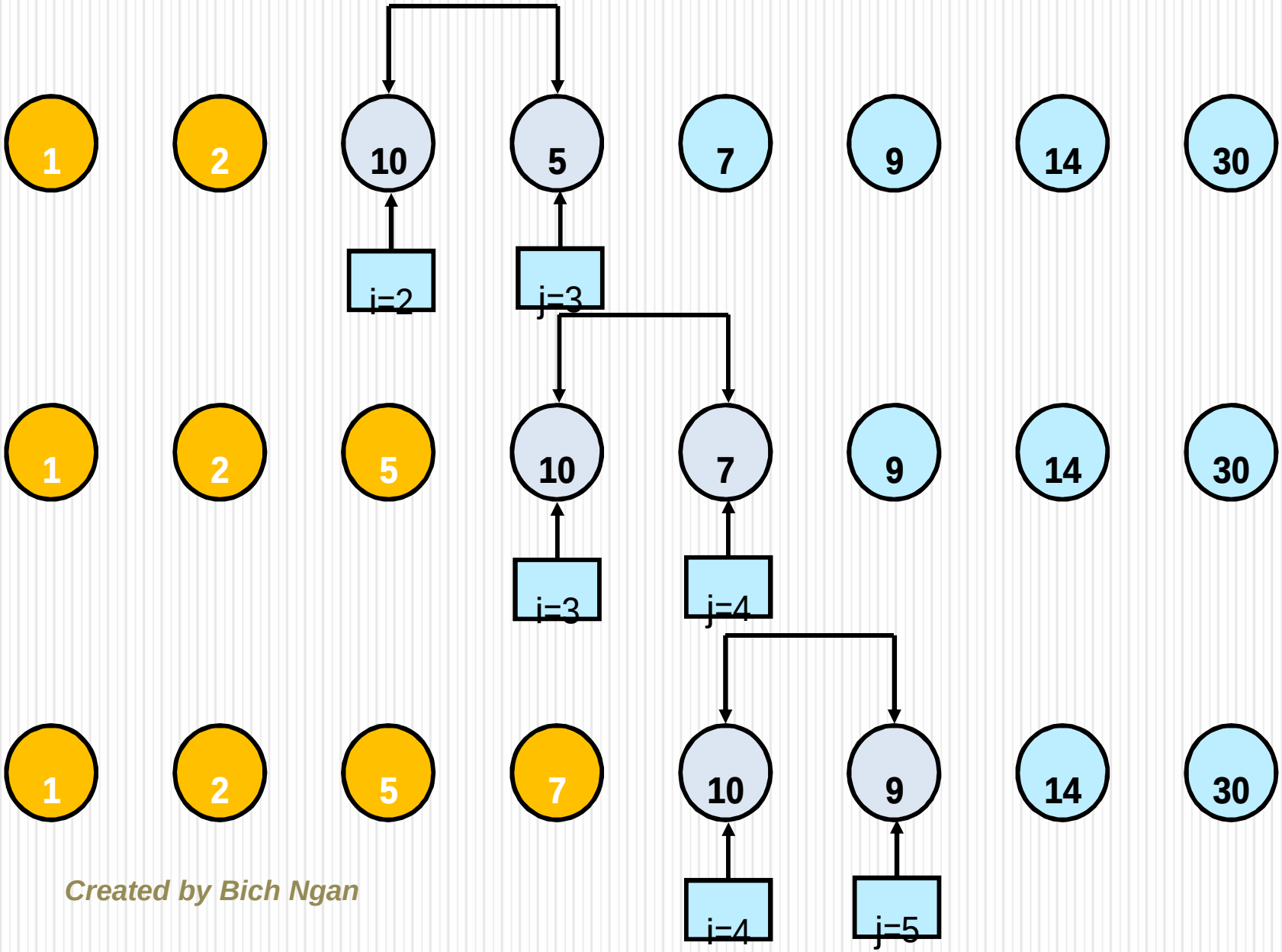
❖Bước 2: Đưa phần tử nhỏ bắt đầu từ cuối mảng (j=5) lên vị trí i = 1



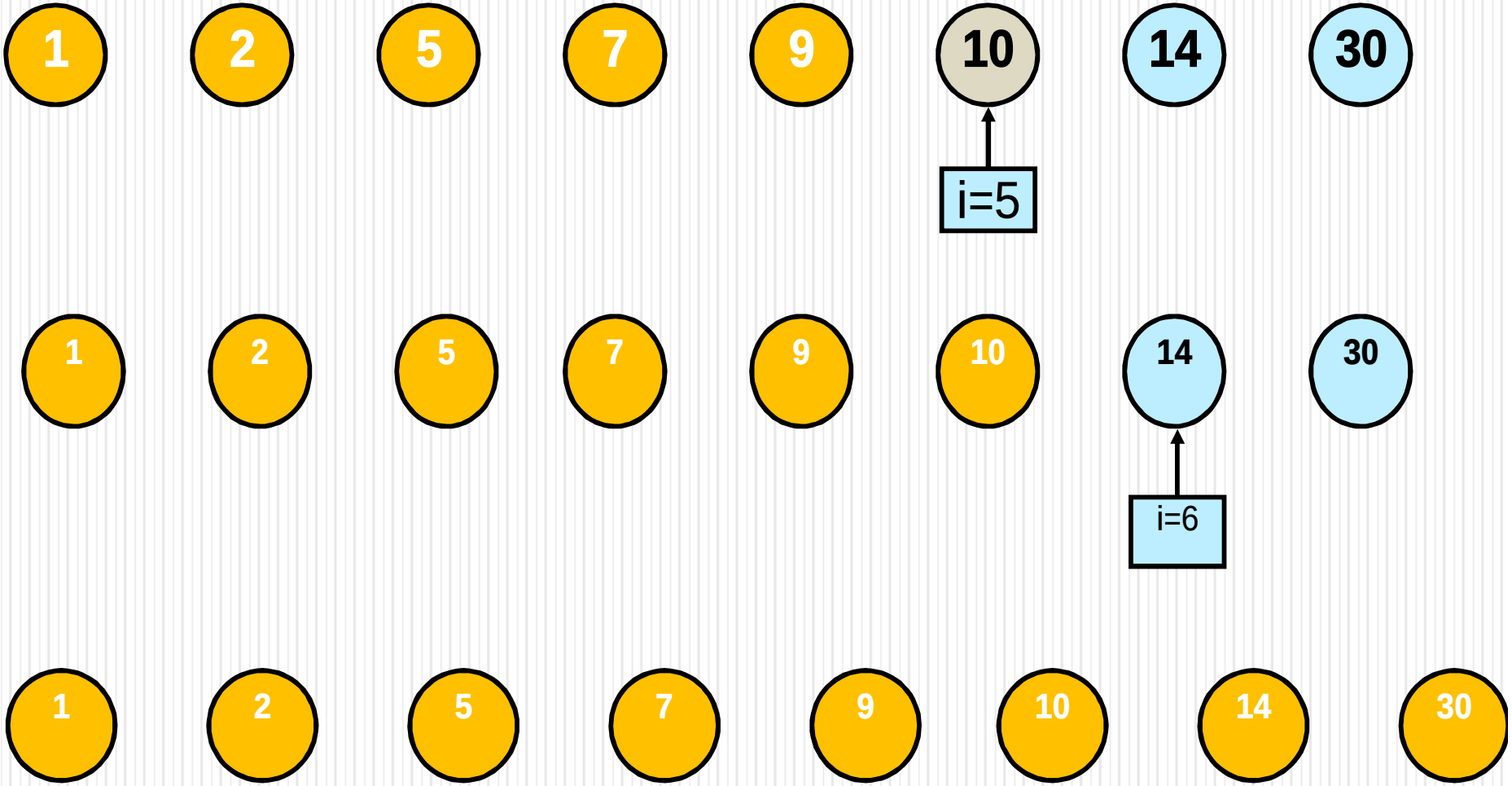
❖Bước 3: Đưa phần tử nhỏ bắt đầu từ cuối mảng (j=6) lên vị trí i = 2



Minh họa ví dụ Bubble sort (tt)



Minh họa ví dụ Bubble sort (tt)



Ví dụ

- Minh họa thao tác sắp xếp dữ liệu theo phương pháp Bubble Sort cho các dãy dữ liệu sau:
  - Sắp xếp tăng:
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**
  - Sắp xếp giảm
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**

Cài đặt

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

Bài tập cài đặt

- Viết bổ sung các hàm vào chương trình xử lý mảng 1 chiều các hàm thực hiện những yêu cầu sau:

1. Viết hàm sắp xếp tăng theo PP **Bubble sort** cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.
2. Viết hàm sắp xếp tăng theo PP **Bubble** sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.

Bài tập (bổ sung Mảng 1 chiều)

1. Viết chương trình để đảo ngược vị trí các phần tử trong mảng một chiều.
2. Nhập mảng a gồm n phần tử sao cho các số chẵn và lẻ xen kẽ nhau.
3. Viết hàm tìm phần tử chẵn lớn nhất trong mảng số nguyên n phần tử.
4. Viết hàm tìm phần tử lẻ nhỏ nhất trong mảng số nguyên n phần tử.
5. Sắp xếp mảng a có n phần tử theo thứ tự tăng dần ở các vị trí chẵn/ giảm dần ở vị trí lẻ.
6. Xóa phần tử thứ i trong mảng a có n phần tử.
7. Chèn một phần tử x vào vị trí thứ i của mảng a.

2.5. Sắp xếp Quick Sort

- **Ý tưởng (sắp tăng)**
  - Phân hoạch dãy $a_0 – a_{n-1}$ thành 2 phần:
    - Dãy con 1: a_1-a_i có giá trị < x
    - Dãy con 1: a_j-a_{n-1} có giá trị > x

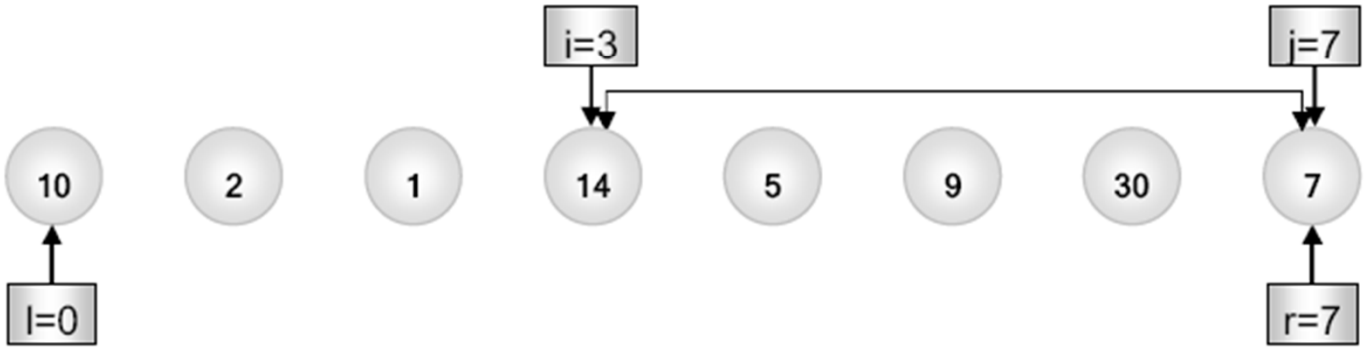
    với x là giá trị tùy ý trong dãy ban đầu.
  - Tuy nhiên thuật toán sẽ chạy nhanh nếu chọn x là phần tử trung bình của dãy. Nhưng thực tế việc tìm ra phần tử trung bình của dãy sẽ tốn nhiều "chi phí", nên x thường được chọn là phần tử nằm giữa dãy: x = a[k], với k = (left + right)/2.

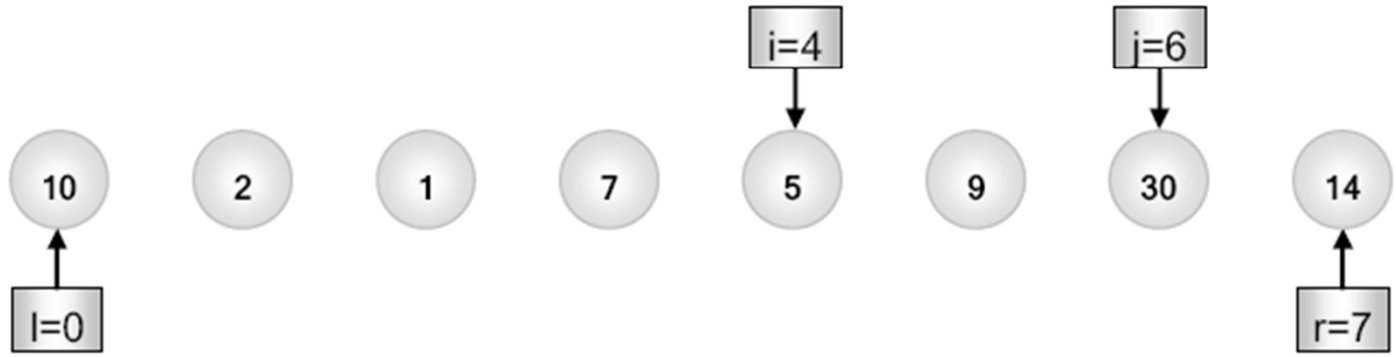
2.5. Sắp xếp Quick Sort

- **Thuật toán Quick Sort**
  - Bước 1: Chọn tùy ý 1 phần tử a[k] trong dãy làm giá trị chuẩn, ($k \in [0: n-1]$). Đặt : x= a[k], i = *l* (left), j = r (right).
  - Bước 2: Tìm và hoán đổi cặp phần tử a[i], a[j] nằm sai chỗ (a[i]>x và a[j]<x)
    - Bước 2.1: trong khi a[i]< x thì i++;
    - Bước 2.2: trong khi a[j]> x thì j--;
    - Bước 2.3: nếu i<j // (a[i]>x và a[j]<x) thì hoán đổi a[i] và a[j]
  - Nếu i<j thì lặp lại B2 cho các dãy $a_{left} - a_j$, a_i - a_{right}

Minh họa ví dụ cho TT Quick Sort

Phân hoạch đoạn `l=0,r=7,mid=3,x=14.`

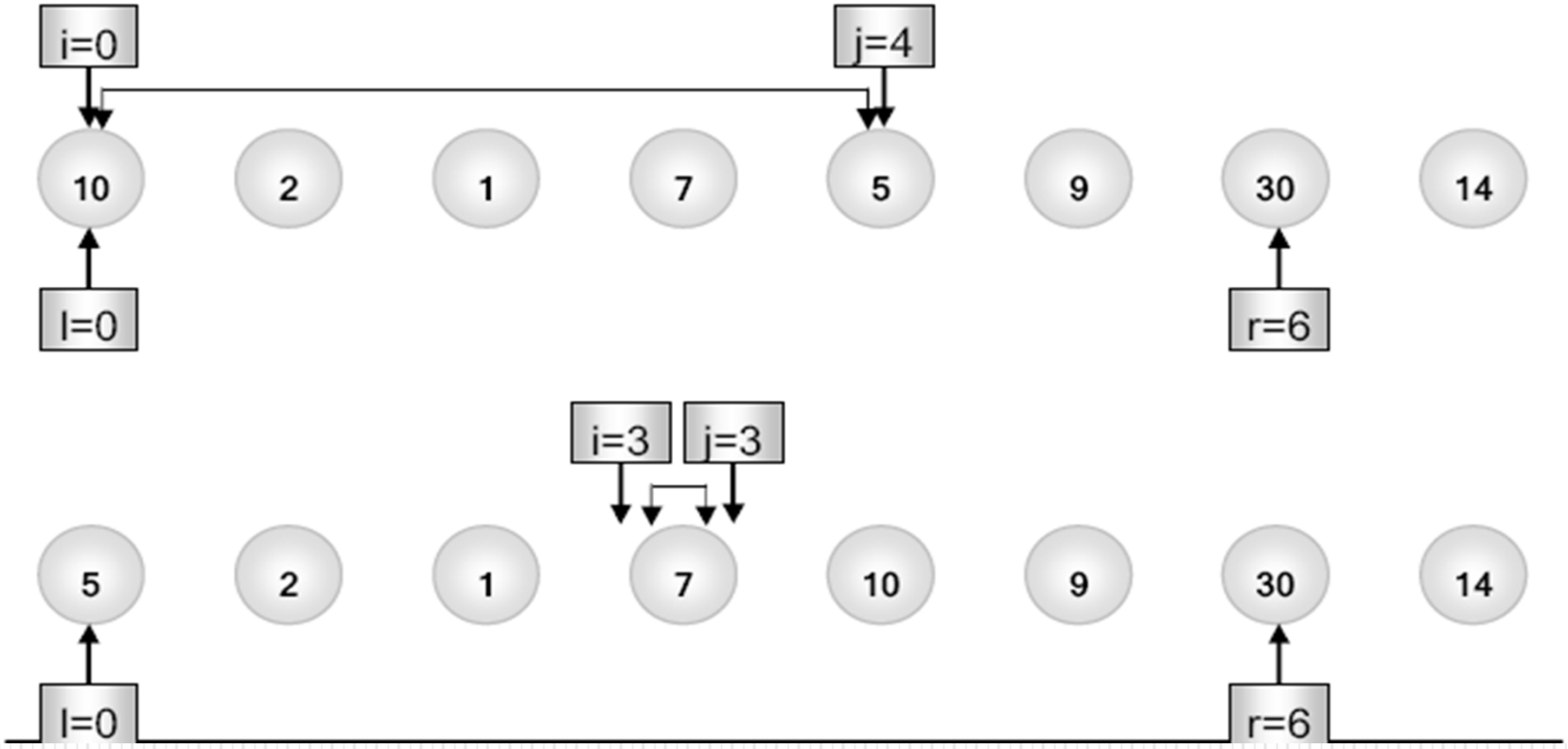




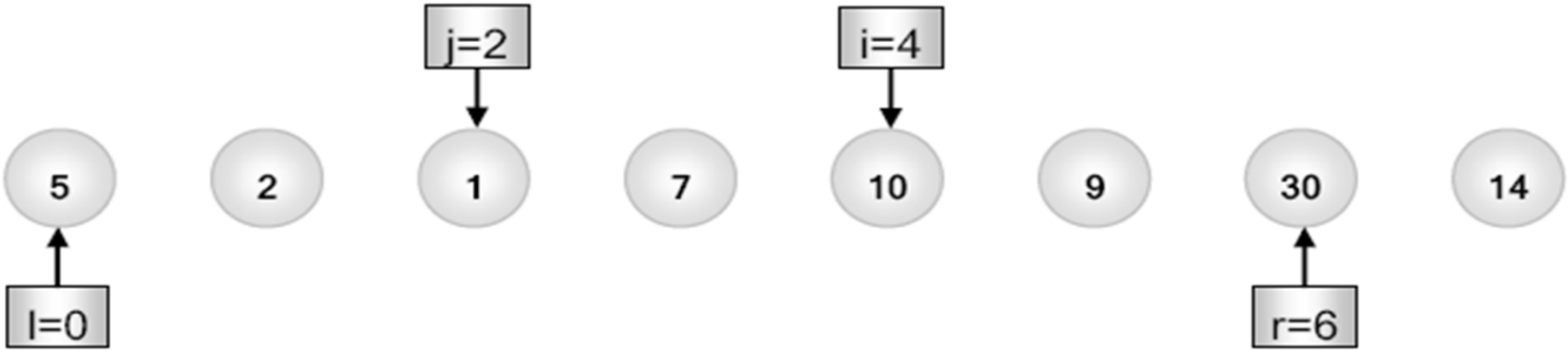
Dừng lặp.

Minh họa ví dụ cho TT Quick Sort (tt)

Phân hoạch đoạn `l=0,r=j=6,mid=3,x=7`. (chưa làm phân hoạch đoạn `l=i=6,r=7`).

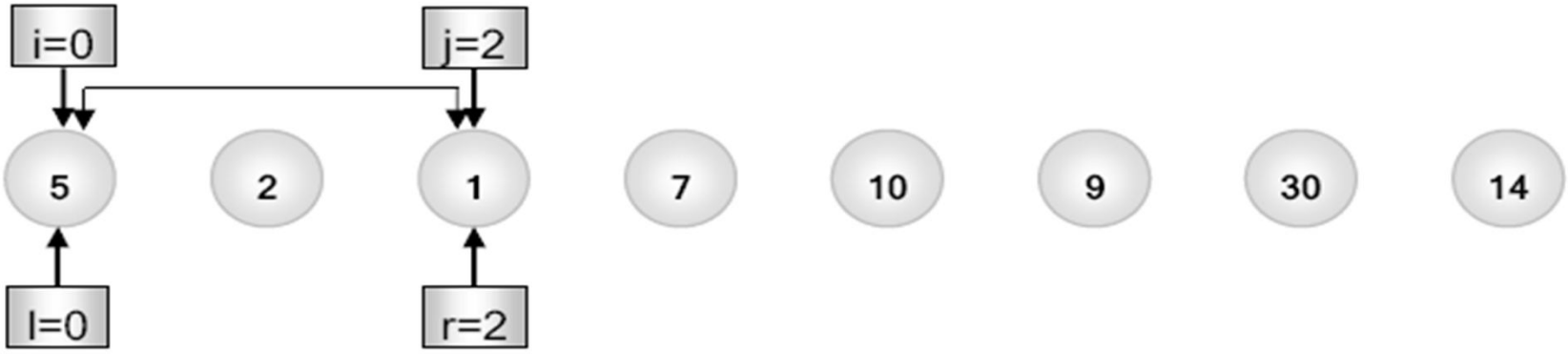


Minh họa ví dụ cho TT Quick Sort (tt)

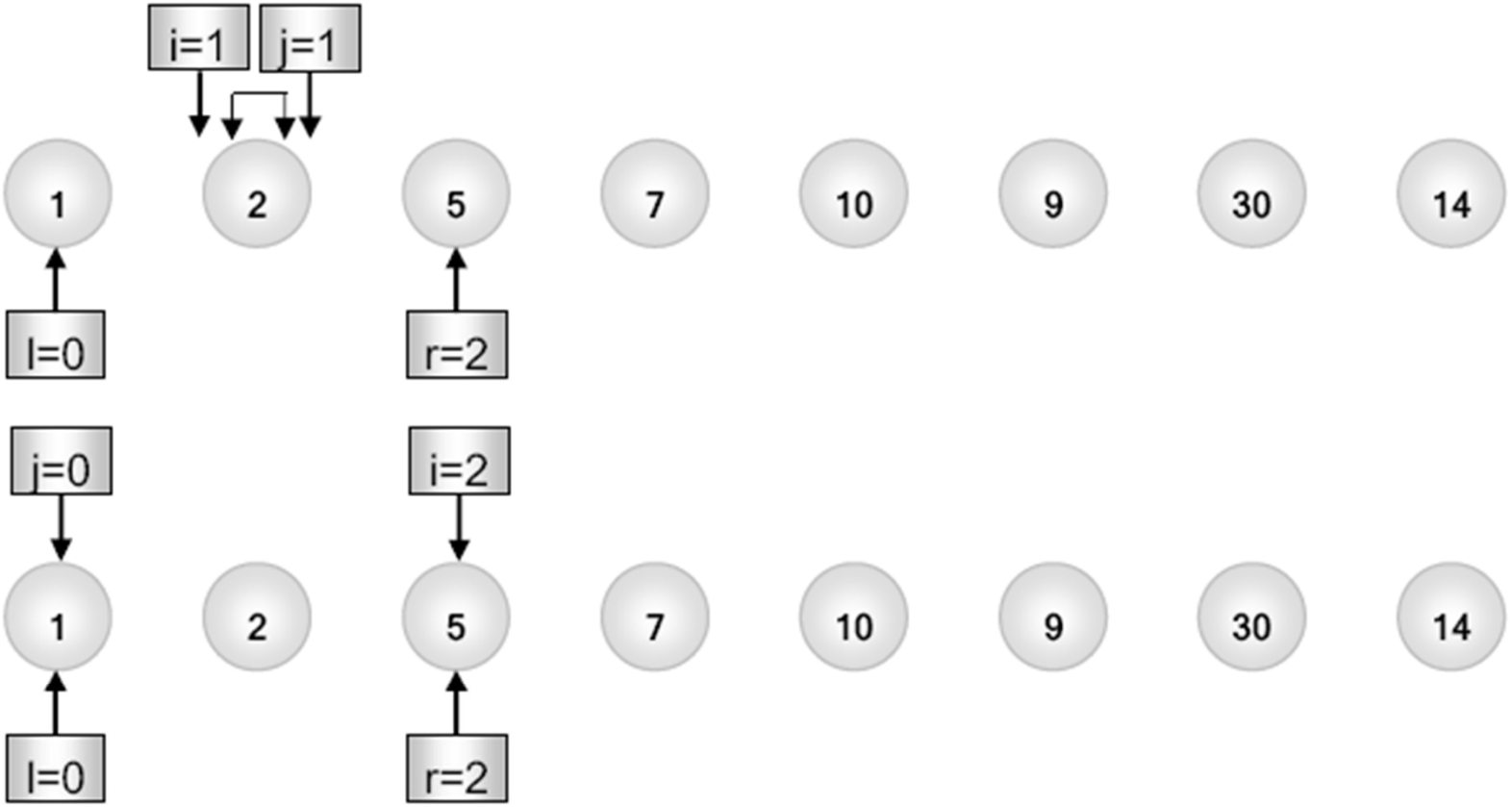


Dừng lặp.

Phân hoạch đoạn `l=0,r=j=2,mid=1,x=2`.(chưa làm phân hoạch đoạn `l=i=4,r=6`).



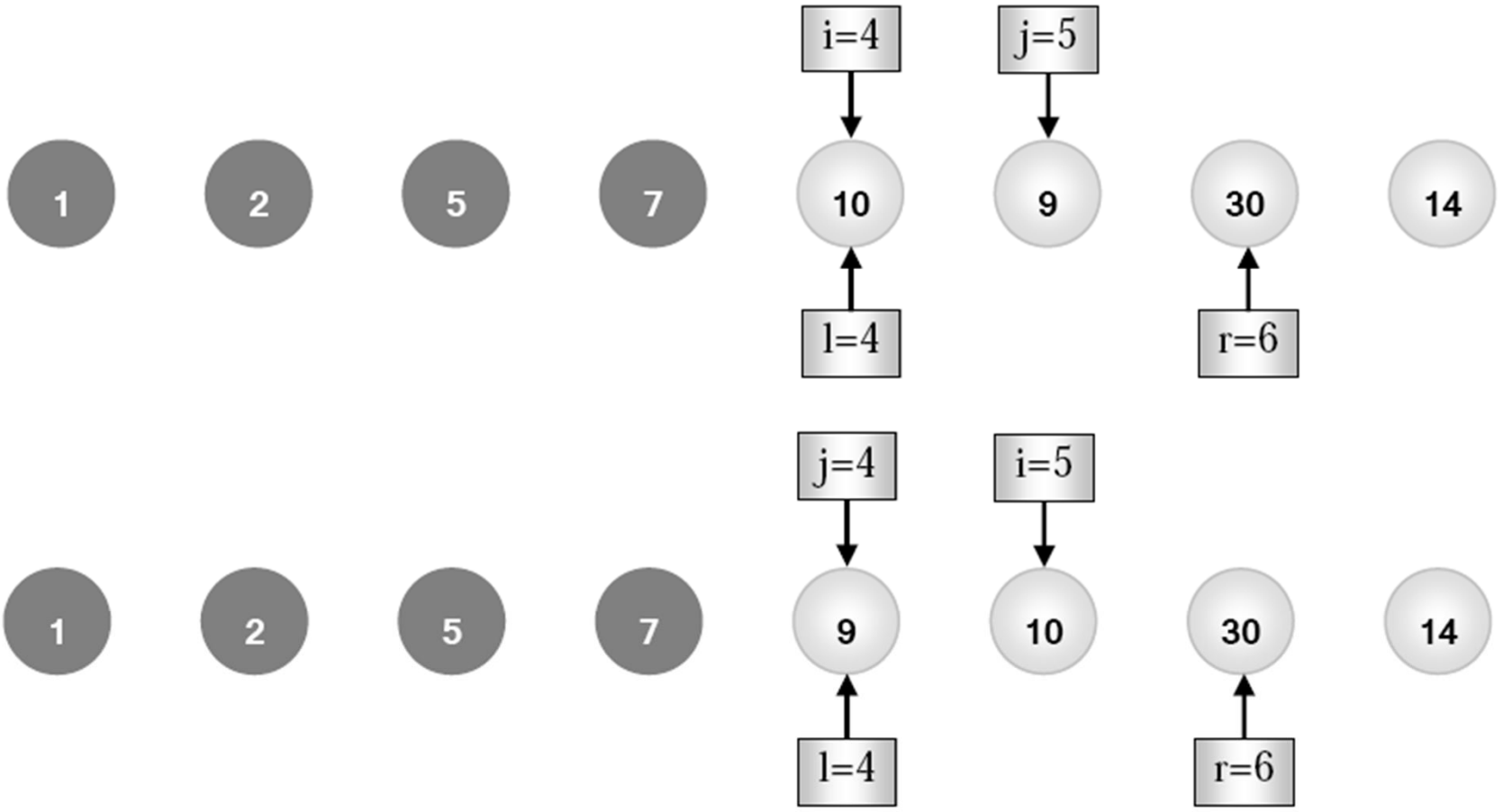
Minh họa ví dụ cho TT Quick Sort (tt)



Dừng lặp.

Minh họa ví dụ cho TT Quick Sort (tt)

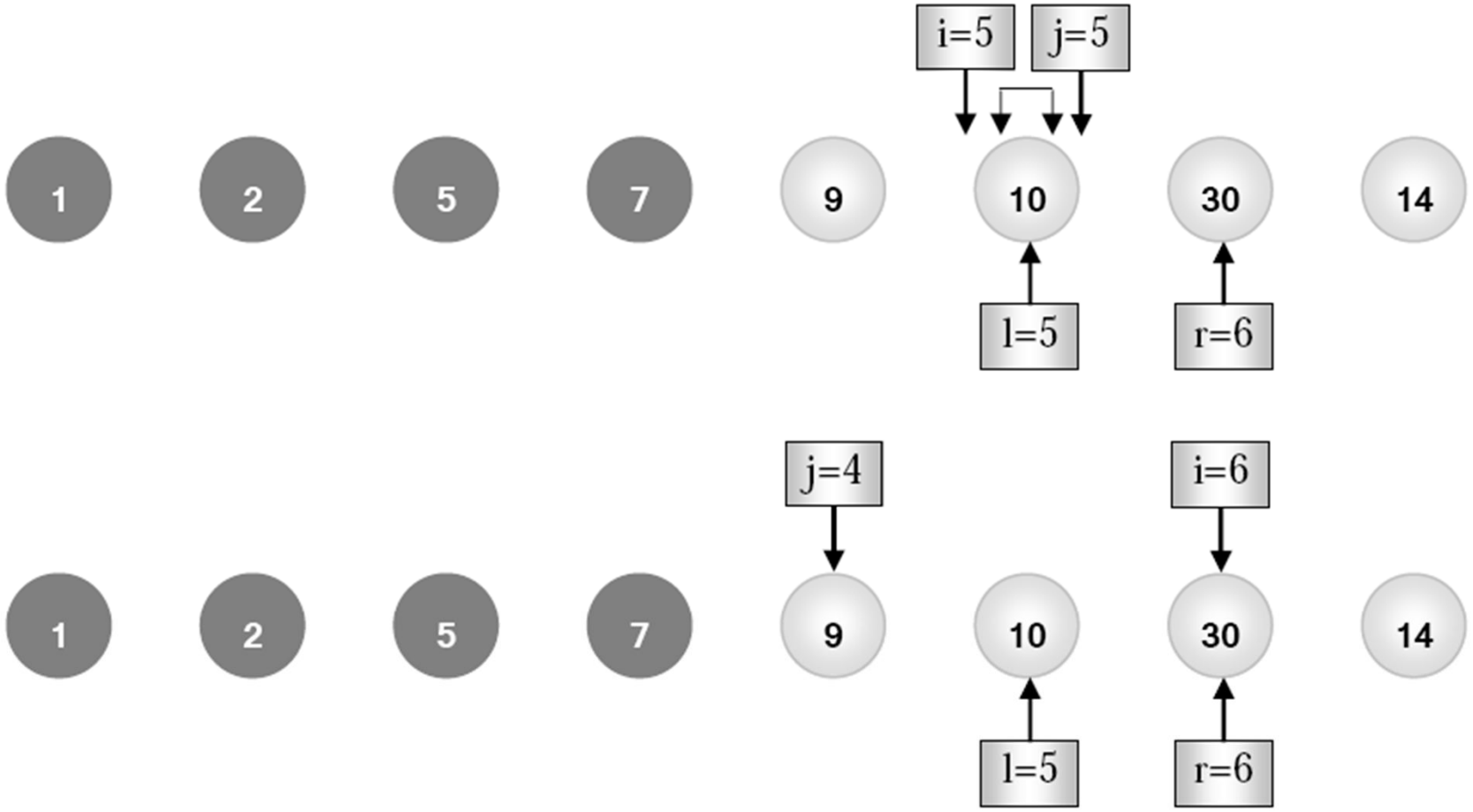
Gọi lại phân hoạch đoạn l=i=4,r=6,mid=5,x=9.



Dừng lặp.

Minh họa ví dụ cho TT Quick Sort (tt)

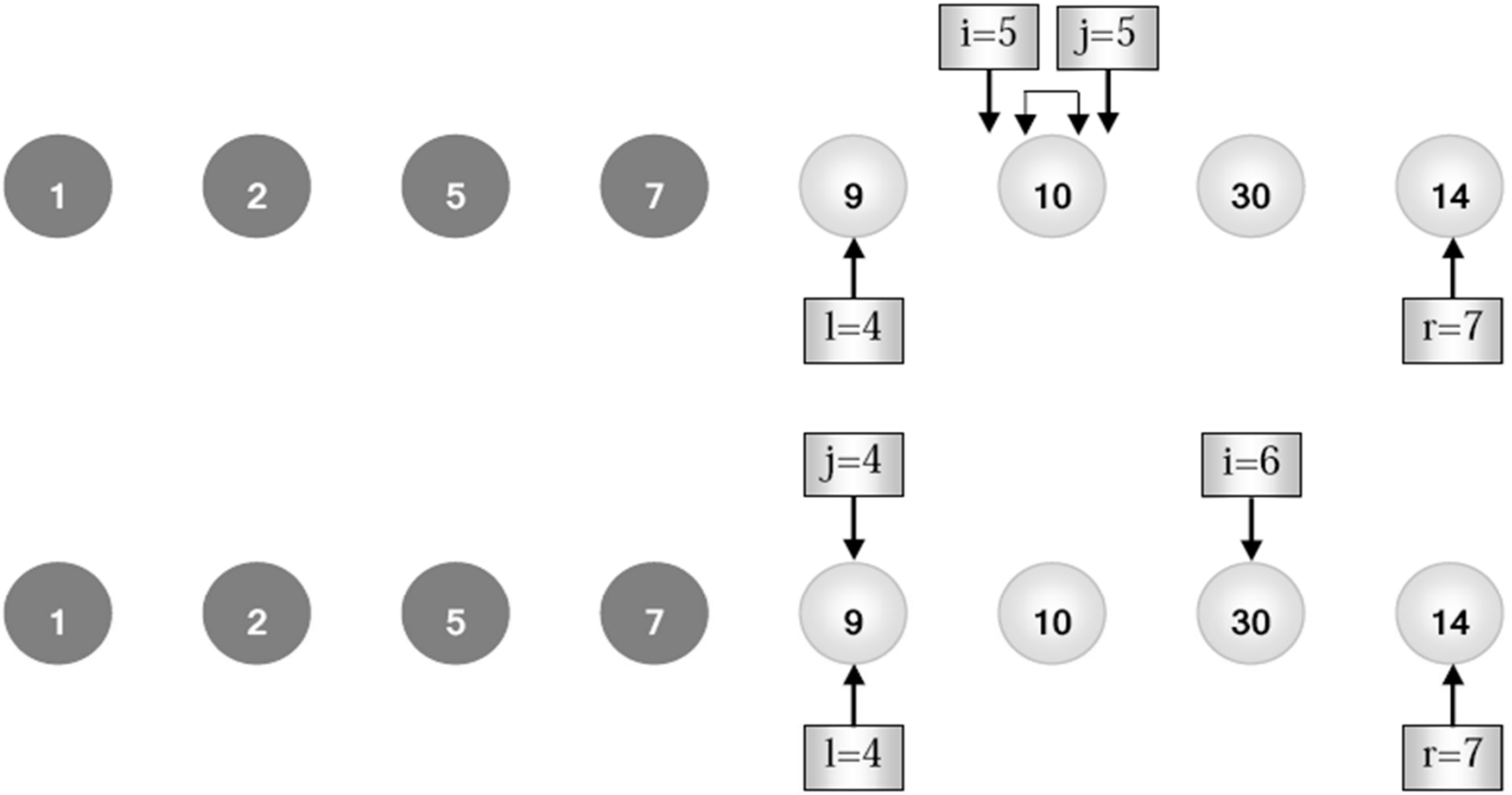
Phân hoạch đoạn l=i=5, r=6, mid=5, x=10.



Dừng lặp.

Minh họa ví dụ cho TT Quick Sort (tt)

Gọi lại phân hoạch đoạn l=i=4, r=7, mid=5, x=10.



Dừng lặp.

Minh họa ví dụ cho TT Quick Sort (tt)

Phân hoạch đoạn l=i=6, r=7, mid=6,x=30.

Dừng lặp.

Dừng.

Vậy dãy đã được sắp xếp.

Cài đặt TT Quick Sort

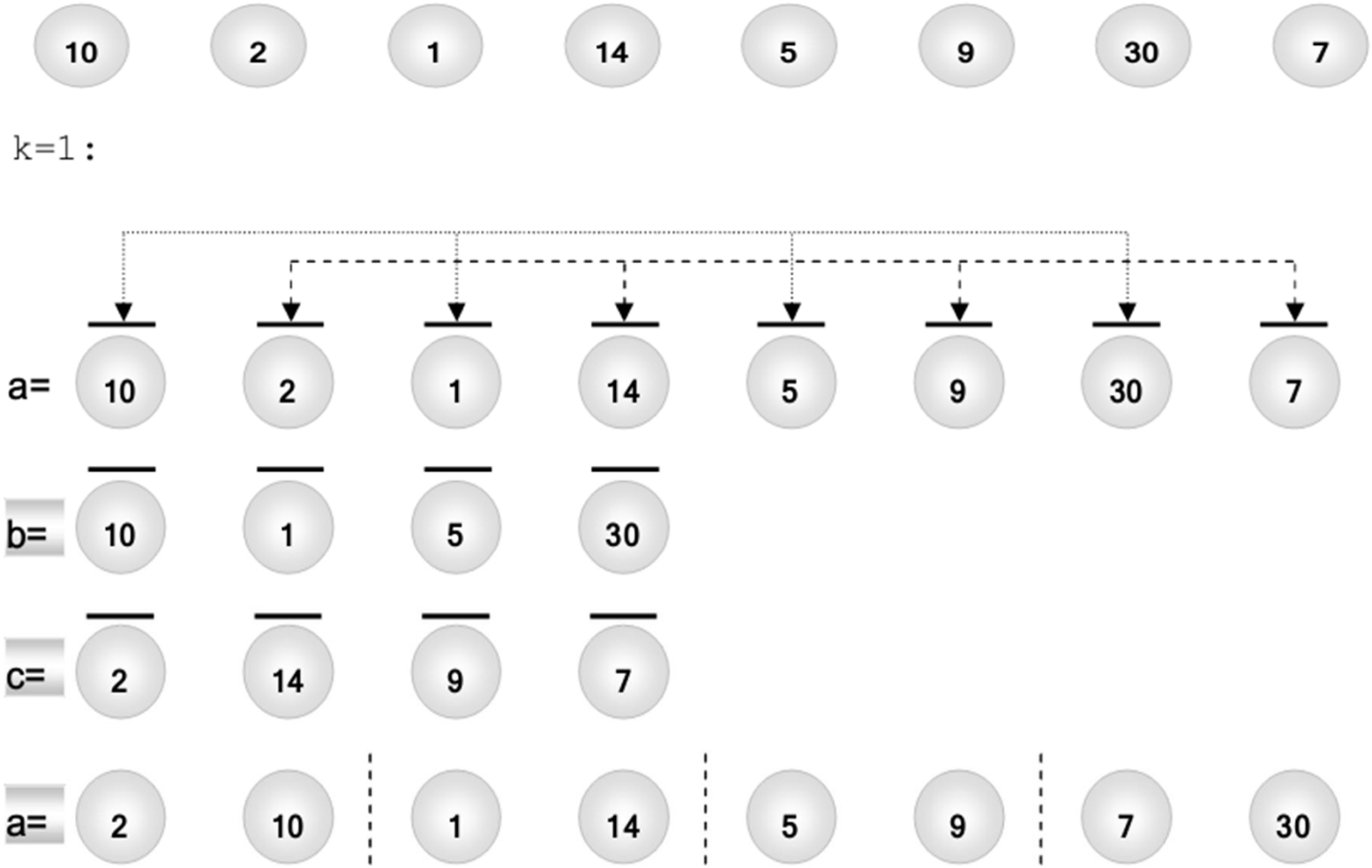
```
void QuickSort(int a[],int l,int r)
{
     int i,j,mid,x,flag;
     mid=(l+r)/2;
    x=a[mid];
    i=l;
    j=r;
    do{
         flag=1;
         while(a[i]<x)  i++;
         while(a[j]>x)  j--;
         if((i>=mid)||(j<=mid))
              flag=0;
         if(i<=j)
         {
              Swap(a[i],a[j]);
              i++;      j--;
         }
    }while((i<=j)&&(flag!=0));
    if(l<j)   QuickSort(a,l,j);
    if(i<r)   QuickSort(a,i,r);
}
```

2.6. Sắp xếp theo giải thuật Merge Sort

- Ý tưởng
  - Phân hoạch dãy ban đầu thành các dãy con. Sau khi phân hoạch xong, dãy ban đầu sẽ được tách thành 2 dãy phụ theo nguyên tắc phân phối đều luân phiên.
  - Trộn từng cặp dãy con của 2 dãy phụ thành 1 dãy với nguyên tắc thứ tự tăng dần. Lặp lại qui trình trên sau 1 số bước, ta sẽ nhận được 1 dãy chỉ gồm 1 dãy con không giảm.

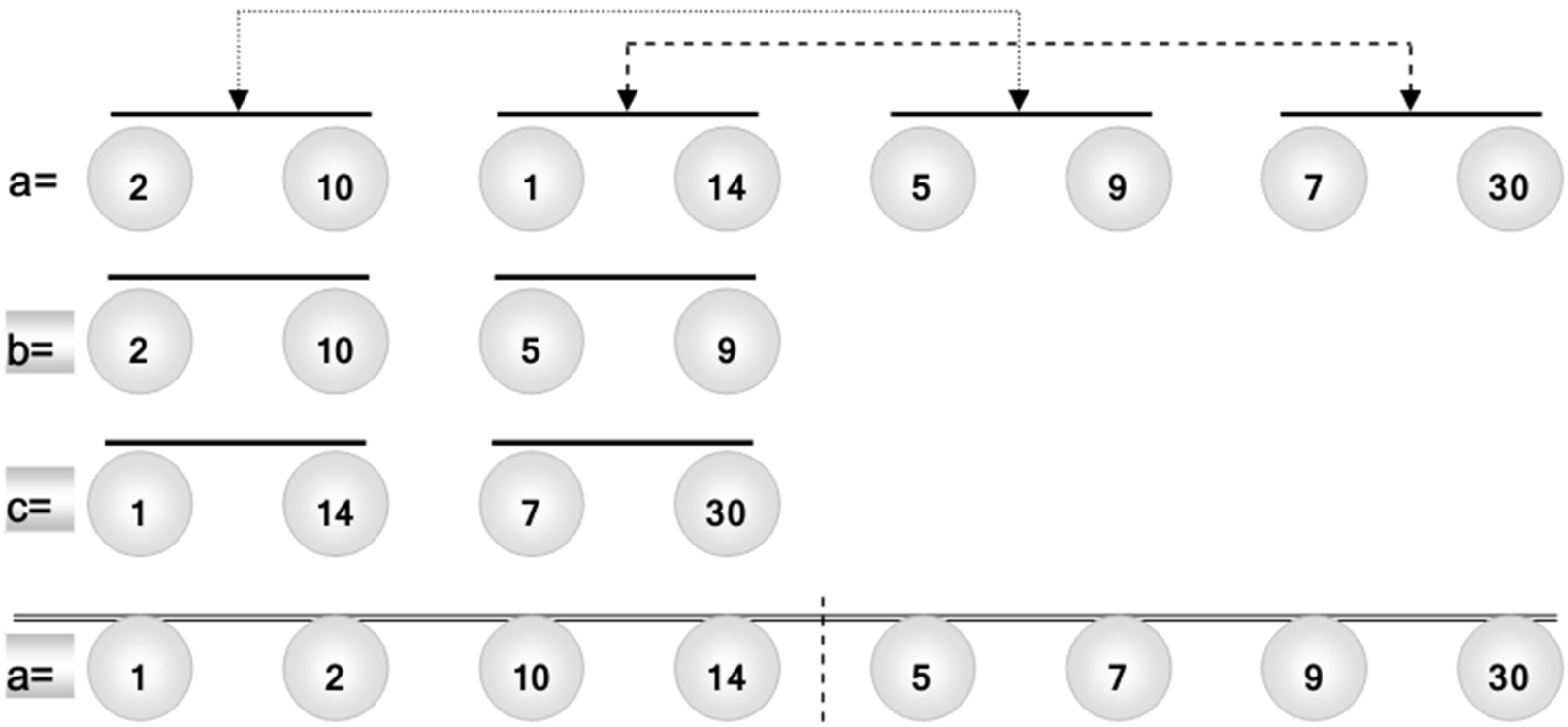
Minh họa ví dụ Merge Sort

Cho dãy số a sau:



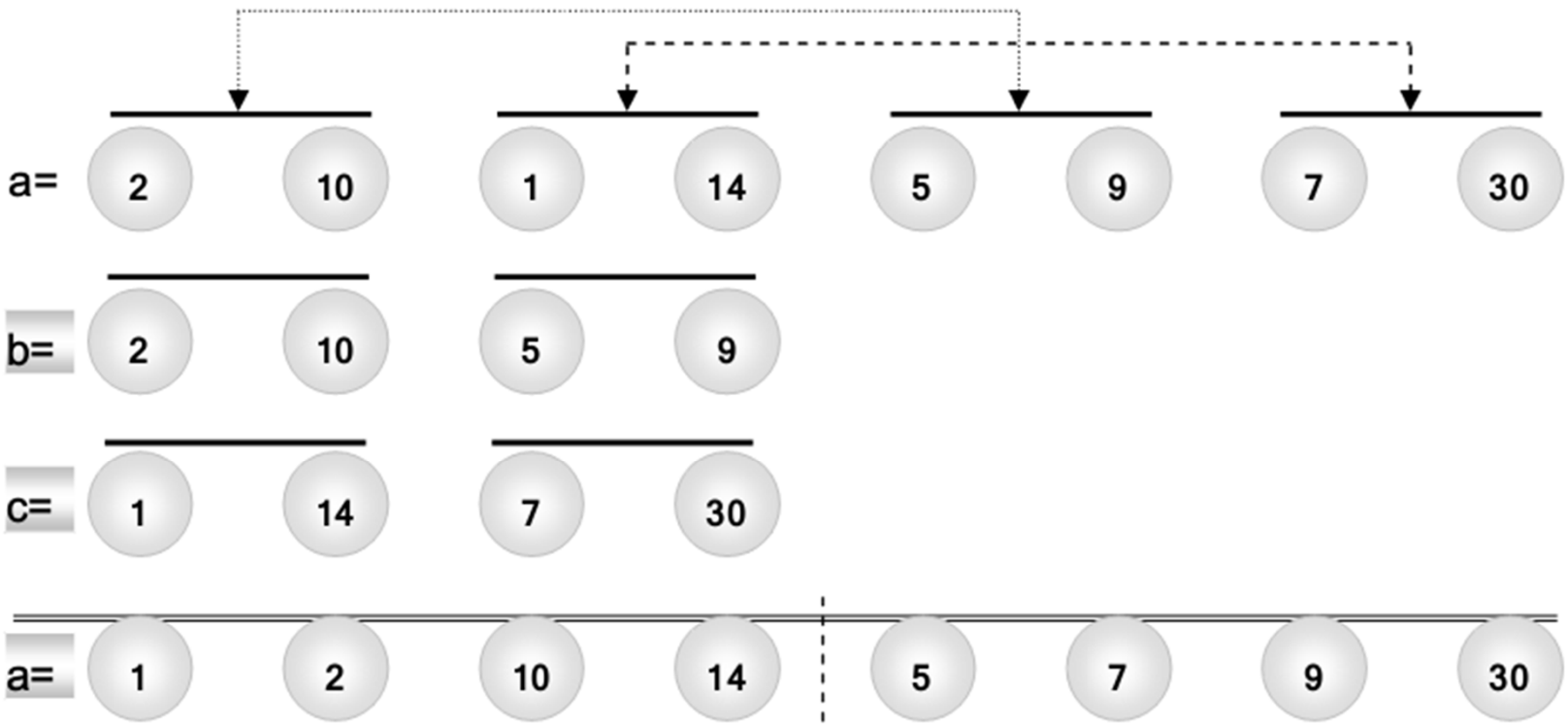
Minh họa ví dụ Merge Sort (tt)

k=2:



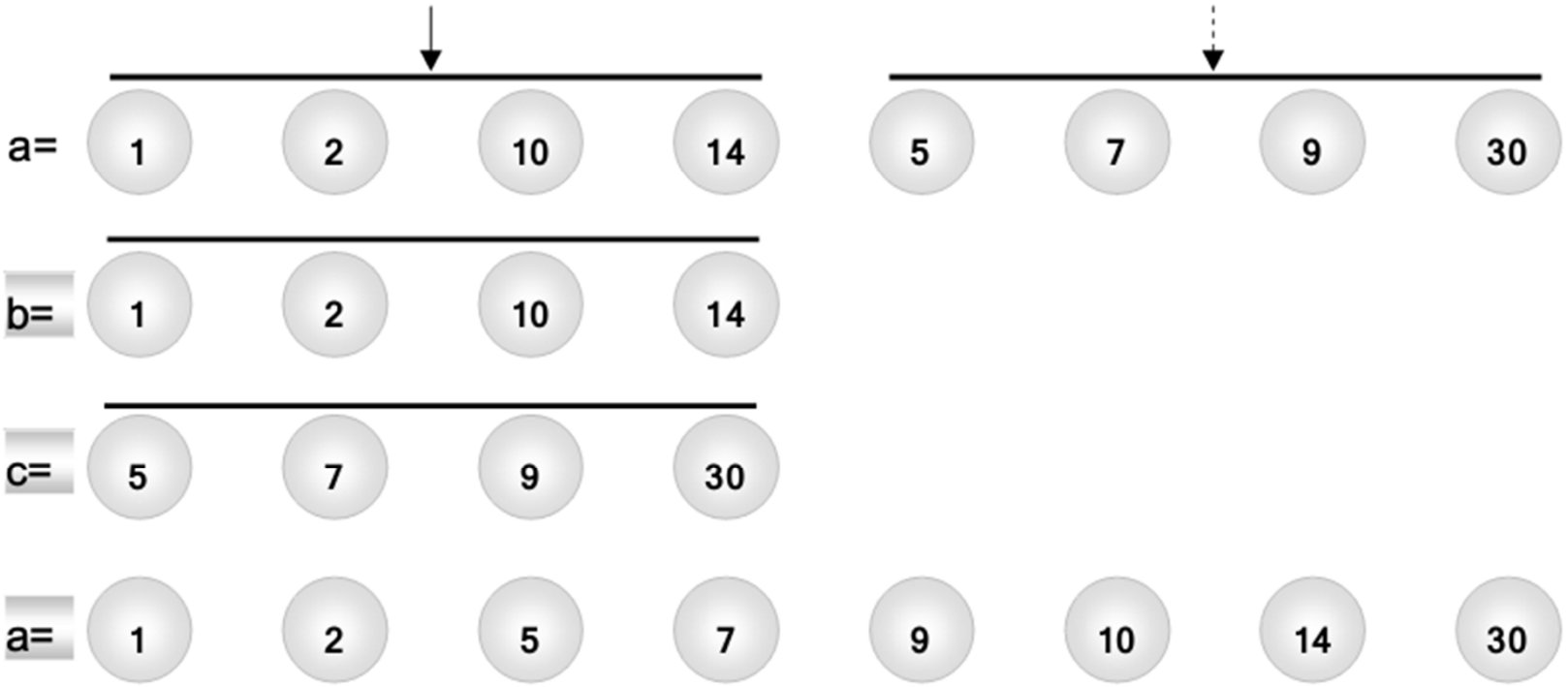
Minh họa ví dụ Merge Sort (tt)

k=2:



Minh họa ví dụ Merge Sort (tt)

k=4:



Minh họa ví dụ Merge Sort (tt)

`k=8`. Dừng.

Vậy dãy đã được sắp xếp.

Cài đặt

```
void MergeSort(int a[],int n)
{
	int p,pb,pc;
	int k=1;
	int b[MAX],c[MAX];
	do{
		p=pb=pc=0;
		while(p<n)
		{
			for(int i=0;i<k;i++)
			if(p<n)
			{
				b[pb]=a[p];	pb++;	p++;
			}
			for(int i=0;i<k;i++)
			if(p<n)
			{
				c[pc]=a[p];	pc++;	p++;
			}
		}
	Merge(a,b,pb,c,pc,k);
	k*=2;
	}while(k<n);
}
```

Cài đặt (tt)

```
void Merge(int a[],int b[],int nb,int c[],int nc,int k)

{
     int p,pb,pc,ib,ic,kb,kc;
     p=0;
     pb=0;
     pc=0;
     ib=0;
     ic=0;
     kb=min(k,nb);
     kc=min(k,nc);
     while((0<nb)&&(0<nc))
     {
          if(b[pb+ib]<=c[pc+ic])
          {
               a[p]=b[pb+ib];
               p++;
               ib++;
               if(ib==kb)
               {
                    while(ic<kc)
                    {
                         a[p]=c[pc+ic];
                         p++;
                         ic++;
                    }
                    pb+=kb;
                    pc+=kc;
                    ib=ic=0;
                    nb-=kb;
                    nc-=kc;
                    kb=min(k,nb);
                    kc=min(k,nc);
               }
          }
          else
          {
               a[p]=c[pc+ic];
               p++;
               ic++;
               if(ic==kc)
                    {
                         while(ib<kb)
                         {
                              a[p]=b[pb+ib];
                              p++;
                              ib++;
                         }
                         pb+=kb;
                         pc+=kc;
                         ib=ic=0;
                         nb-=kb;
                         nc-=kc;
                         kb=min(k,nb);
                         kc=min(k,nc);
                    }
               }
          }
          if(nb>0)
          {
               for(int i=1;i<=nb;i++)
               {
                    a[p]=b[pb+ib];
                    p++;
                    ib++;
               }
          }
}
```

Bài tập

- Minh họa thao tác sắp xếp dữ liệu theo phương pháp Quick Sort và Merge Sort cho các dãy dữ liệu sau:
  - Sắp xếp tăng:
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**
  - Sắp xếp giảm
    - **13 8 12 6 9 10 12 7**
    - **A H K R E C Z G**

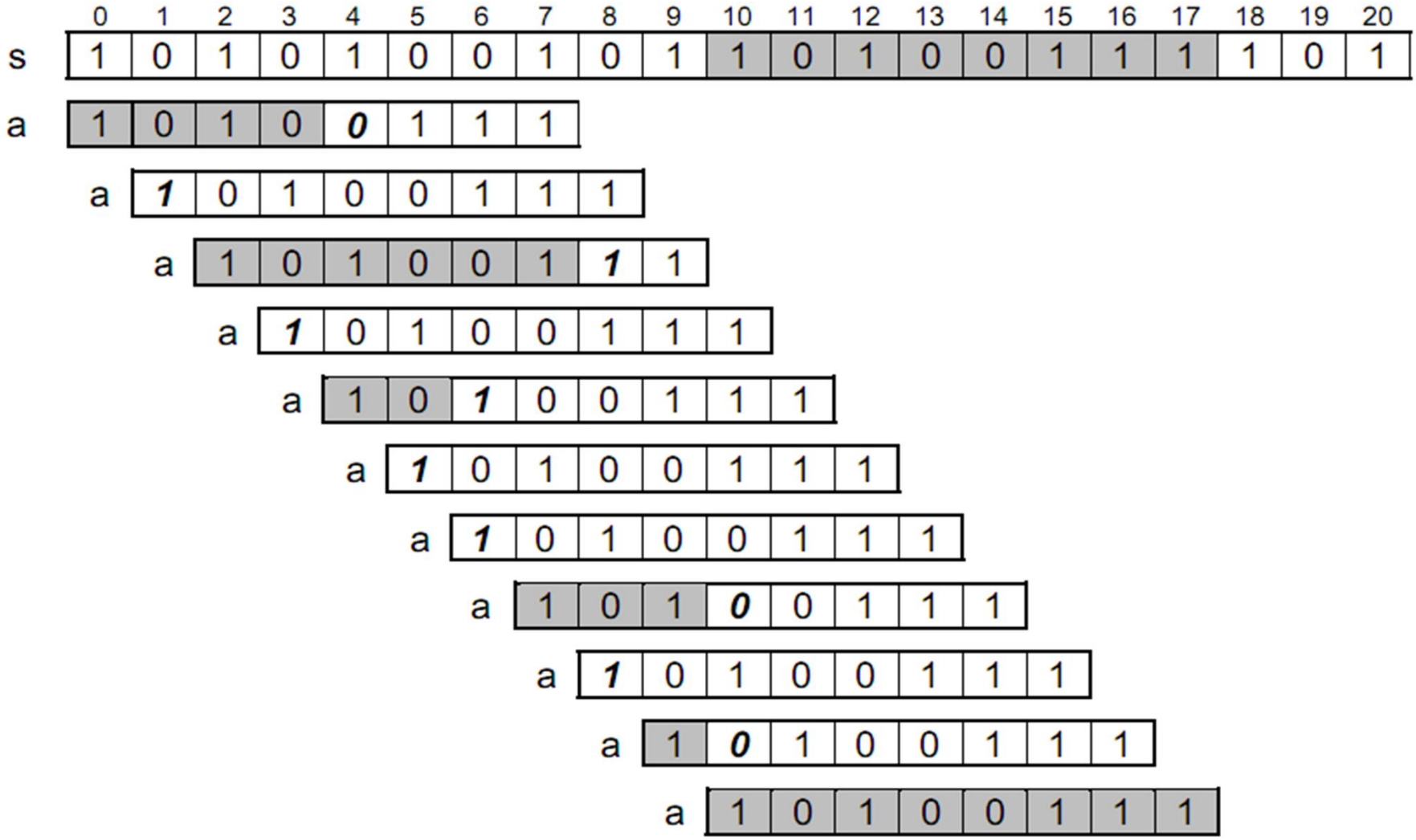
3. Các giải thuật tìm kiếm trên chuỗi

3.1. Tìm kiếm tuần tự (Giải thuật Brute-Force)

- **Ý tưởng:**
  - Giả sử có chuỗi *s* và chuỗi *a*, cần tìm a trong s.
  - Giải thuật thử so chuỗi a tại mọi vị trí con trong chuỗi s, bắt đầu từ vị trí 0, 1, 2… trong s.
  - Mỗi lần so trùng, lần lượt so từng cặp ký tự của a và s từ trái qua phải. Khi gặp 2 ký tự khác, việc so sánh lại bắt đầu từ đầu chuỗi a với vị trí mới trong s.

3.1. Tìm kiếm tuần tự (Giải thuật Brute-Force)

- **Ví dụ minh họa**



Cài đặt

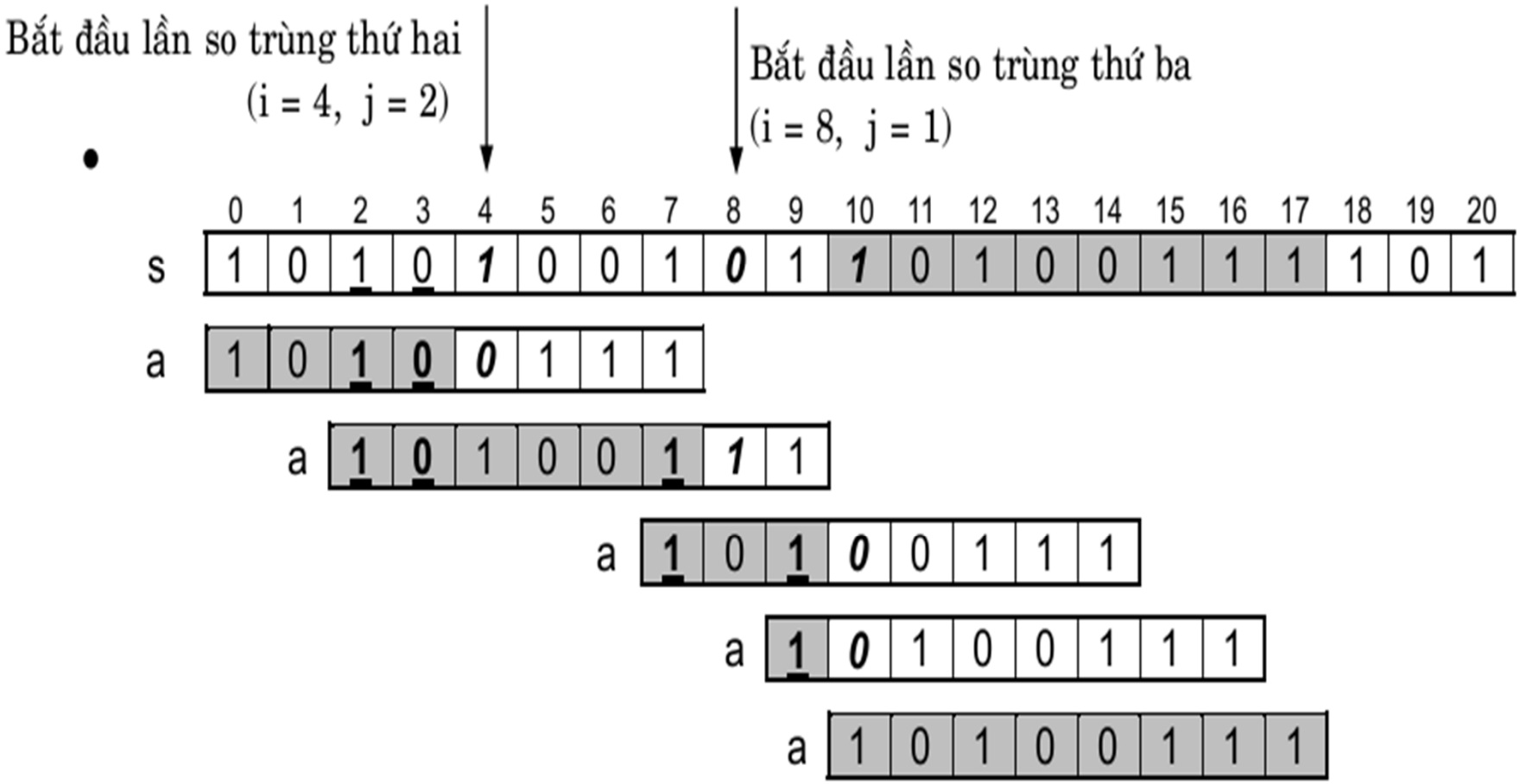
```
//Tim chuoi a trong chuoi s
int Brute_Force (const String &s, const String &a)
{
        int     i = 0,                  //chi so chay tren s
                j = 0;                  //chi so chay tren a
        int     ls = s.strlen();        // chieu dai cua s
        int     ls = a.strlen();        //chieu dai cua s
        const char* pa = a.c_str();      //ky tu dau tien cua chuoi a
        const char* ps = s.c_str();      //ky tu dau tien cua chuoi s
        do {
                if (pa[j] == ps[i])
                {
                        i++;
                        j++;
                };
                else
                {
                        i = (i – j) + 1; // Lui i ve vi tri so sanh moi
                        j = 0;
                }
        } while ((j<la) && (i<ls));
        if (j>=la)
                return i – la; //tra ve vi tri cua a nam trong s
        else
                return –1; //khong tim duoc a trong s
}
```

3.2. Giải thuật tìm kiếm Knuth-Morris-Pratt

- **Ý tưởng (Giải thuật còn được gọi là KMP-Search)**
  - Giả sử có chuỗi *s* và chuỗi *a*, cần tìm *a* trong *s*.
  - Gọi *i*, *j* lần lượt là biến chạy trên *s* và *a*. *la*, *ls* là độ dài chuỗi *a* và *s*.
  - Ý tưởng chính:
    - Trong lần so sánh trùng a và s thứ 1, khi tại vị trí thứ i: có aj ≠si.
    - Khi đó ta dịch chuyển về phía phải sao cho đoạn đầu của a trùng khớp với đoạn cuối của a trong phần đã duyệt qua.
    - Lần so trùng kế tiếp của s và a bắt đầu từ sau vị trí trùng khớp đầu cuối trên a này. Khi đó i không cần lùi lại (tiết kiệm số lần so sánh).
    - Quá trình lặp lại cho đến khi $j > la$ (có s trong a), hoặc $i > ls$ (không có a trong s.

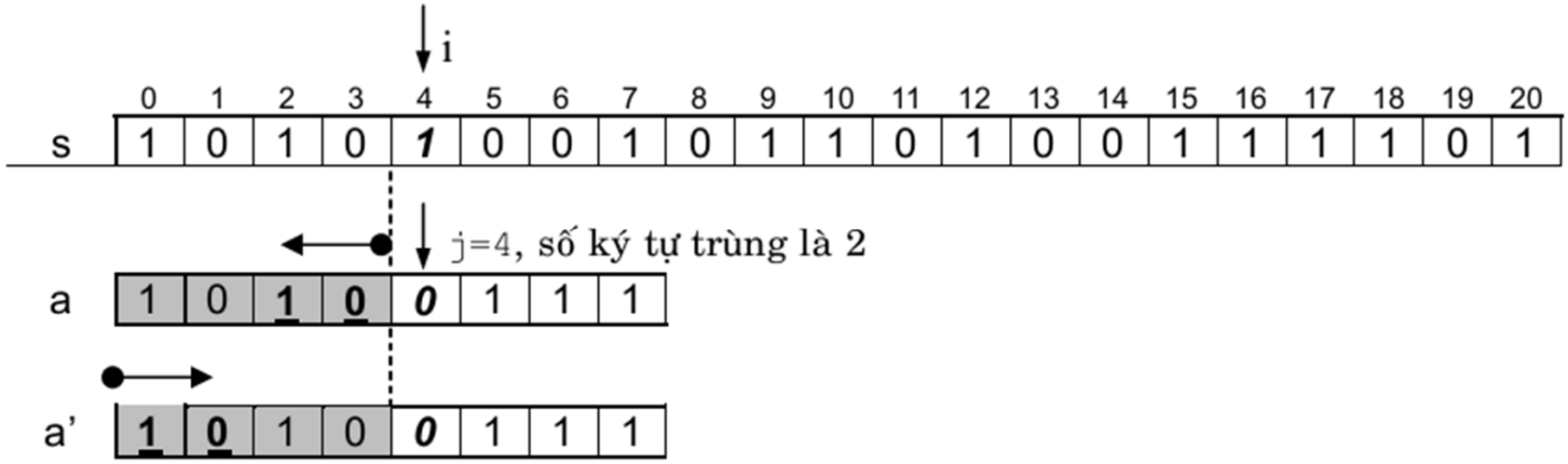
3.2. Giải thuật tìm kiếm Knuth-Morris-Pratt

- **Ví dụ minh họa**



3.2. Giải thuật tìm kiếm Knuth-Morris-Pratt

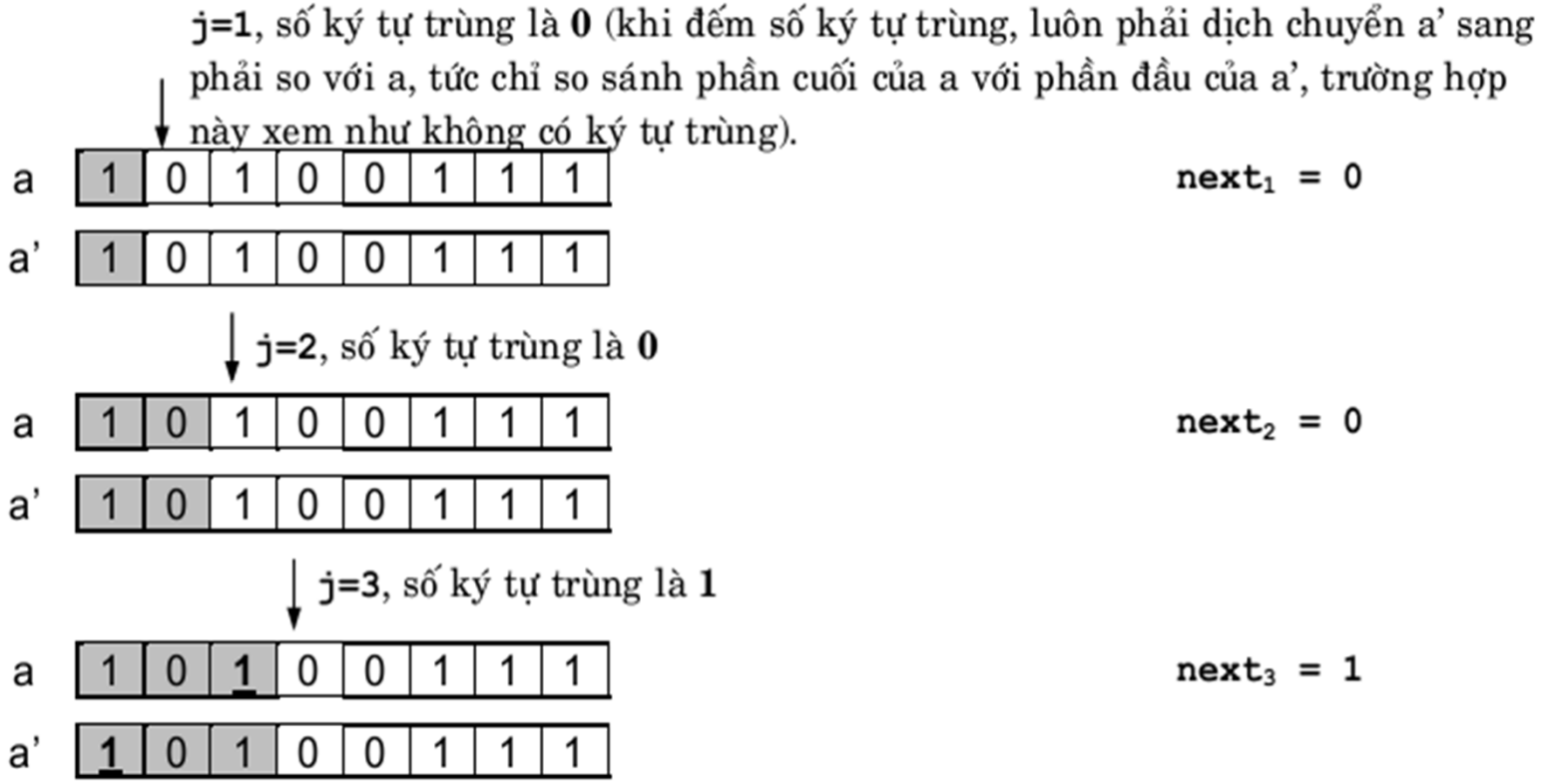
- **Chúng ta chỉ tìm sự trùng khớp phần đầu chuỗi a với phần cuối của dãy màu xám (dãy đã trùng khớp giữa s và a).**



- **Cách tìm kiếm này được thực hiện như slide tiếp theo.**

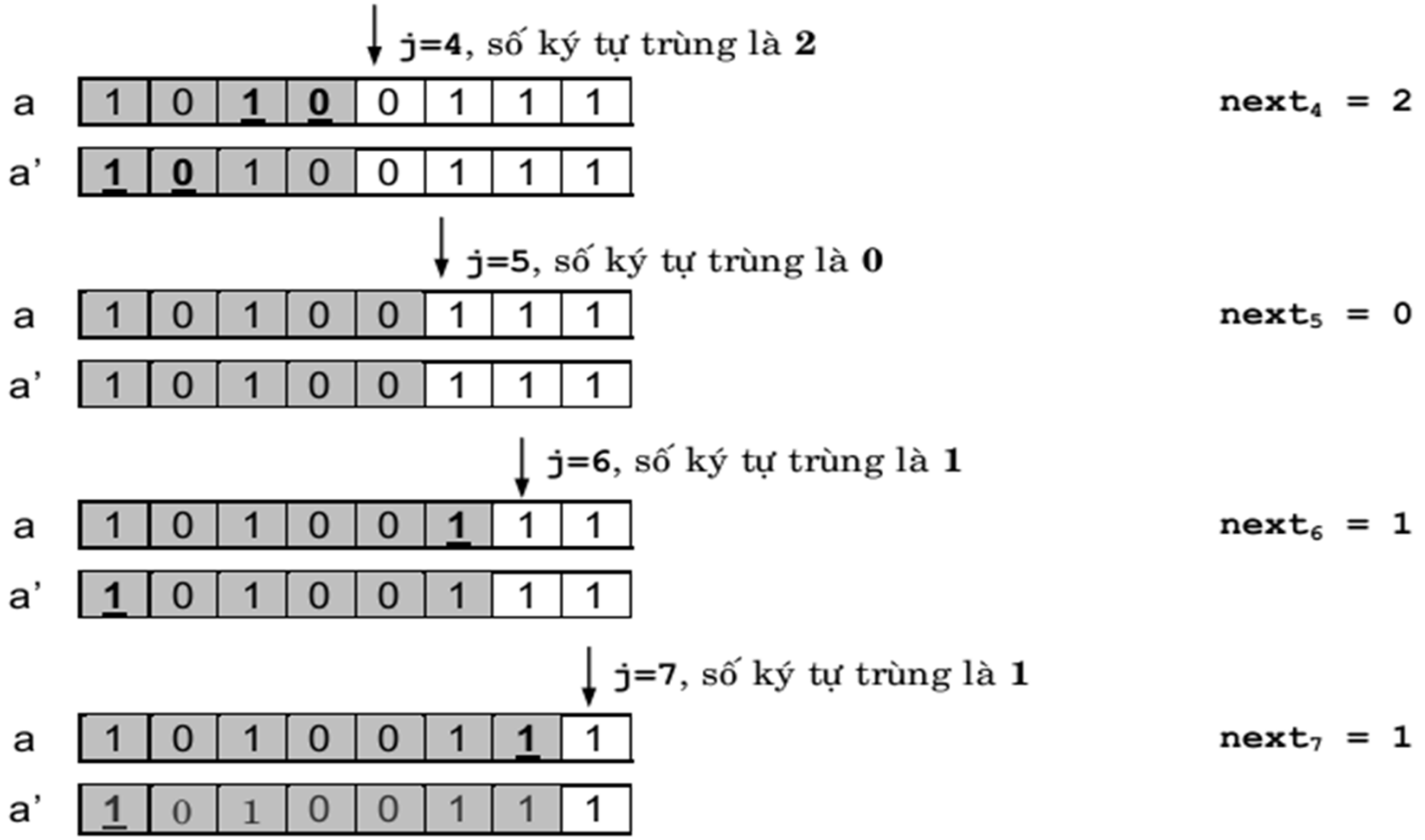
3.2. Giải thuật tìm kiếm Knuth-Morris-Pratt

- **Ví dụ minh họa: cách tìm các ký tự đầu/cuối trùng khớp trên 1 chuỗi a**



3.2. Giải thuật tìm kiếm Knuth-Morris-Pratt

- **Ví dụ minh họa: cách tìm các ký tự đầu/cuối trùng khớp trên a trong phần so khớp với s (tt)**



Cài đặt

```
// Giải thuật Knuth- Morris – Pratt

int strstr(const String &s, const String &a);
/*
post: Nếu a là chuỗi con của s, hàm trả về vị trí xuất hiện đầu tiên của a trong s;
      ngược lại, hàm trả về -1.
*/
{
   List<int> next;
   int    i = 0, // Chỉ số chạy trên s.
          j = 0, // Chỉ số chạy trên a.
          ls = s.strlen(); // Số ký tự của s.
          la = a.strlen(), // Số ký tự của a.
   const char* pa = a.c_str(); //Địa chỉ ký tự đầu tiên của a.
   const char* ps = s.c_str(); //Địa chỉ ký tự đầu tiên của s.
   InitNext(a, next);     // Khởi gán các phần tử next1, next2,..,nextla-1.
                          // Không sử dụng next0.
   do {
      if (pa[j]==ps[i]){// Vẫn còn ký tự trùng trong một lần so trùng
            i++;        // nào đó, i và j được quyền nhích tới.
            j++;
      }
      else
         if (j == 0)    // Đây là trường hợp đặc biệt, phải dịch a sang phải
            i++;        // một bước, có nghĩa là cho i nhích tới.
         else
            next.retrieve(j, j); // Cho j lùi về trị đã chứa trong nextj.
   } while ((j<la) && (i<ls));
   if (j>=la) return i - la;
   else return -1;
}
```

Cài đặt (tt)

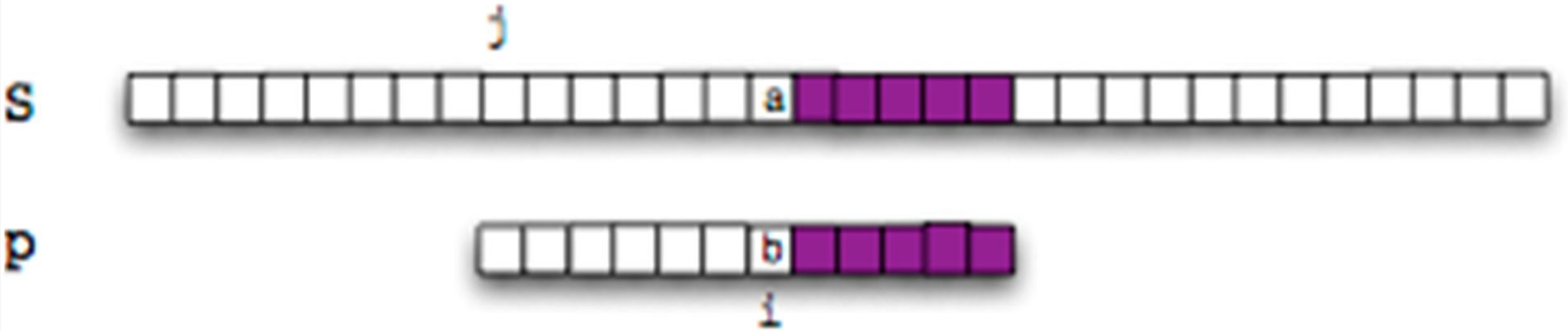
```
// Hàm phụ trợ gán các phần tử cho danh sách next.

void InitNext(const String &a, List<int> &next);
/*
post: Gán các trị cho các phần tử của next dựa trên chuỗi ký tự a.
*/
{
   int    i = 1, // Chỉ số chạy trên a.
          j = 0, // Chỉ số chạy trên a'.
          la = a.strlen(), // Số ký tự của a (cũng là của a').
   const char* pa = a.c_str(); //Địa chỉ ký tự đầu tiên của a (cũng là của a').
   next.clear();
   next.insert(1, 0); // Luôn đúng với mọi a.
   do {
      if (pa[j]==pa[i]){    // Vẫn còn ký tự trùng trong một lần so trùng
      i++;                  // nào đó, i và j được quyền nhích tới.
         j++;               // Từ vị trí i trên a trở về trước, j xem như đã
         next.insert(i, j);// quét được số phần tử trùng của a' so với a.
      }
      else
         if (j == 0){    // Trường hợp đặc biệt, phải dịch a sang phải
            i++;         // một bước, có nghĩa là cho i nhích tới.
            next.insert(i, j);
         };
         else
            next.retrieve(j, j); // Cho j lùi về trị đã chứa trong next_j.
   } while (i<la); // i=la là đã gán xong la phần tử của next,
                   // không sử dụng next_0.
}
```

3.3. Giải thuật Boyer - Moore

- **Ý tưởng:**
  - Giả sử có chuỗi *s* và chuỗi *p*, cần tìm p trong s.
  - Kiểm tra các ký tự của p và s từ phải sang trái và khi phát hiện sự khác nhau đầu tiên thuật toán sẽ tiến hành dịch p qua phải để thực hiện so sánh tiếp.
  - Trong thuật toán này có hai cách dịch chuyển p qua phải để so sánh với s.

3.3. Giải thuật Boyer - Moore

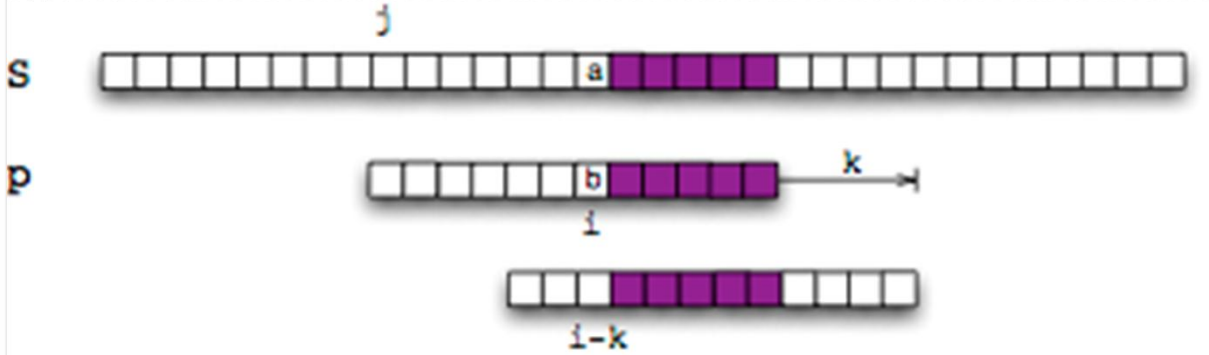
- **Ví dụ minh họa**
  - Giả sử ta đang khớp mẫu p ở vị trí j của chuỗi nhập s. Ta sẽ bắt đầu ngược bằng cách so sánh p[i] với s[j+i] với i đi từ m-1 xuống 0 (so ngược từ phải sang trái). Giả sử đến một giá trị i nào đó ta gặp hai ký tự khác nhau như hình dưới đây. (đoạn màu tím là giống nhau)



  - Cần tìm cách để dịch chuyển p qua phải để so sánh tiếp với s.

3.3. Giải thuật Boyer - Moore

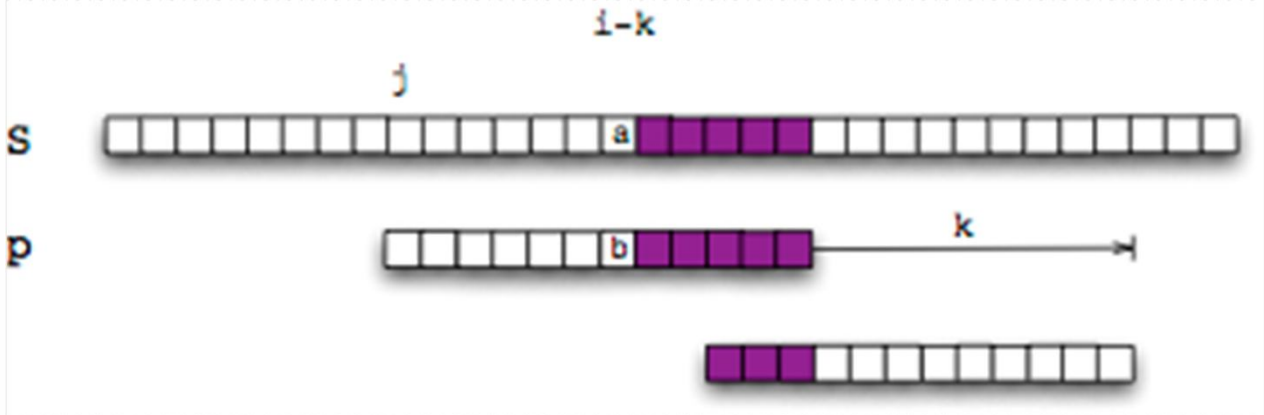
- **Luật matching shift:** Bây giờ ta sẽ phải dịch chuỗi p đi một đoạn. Có hai trường hợp:
  - **TH1:** nếu đoạn màu tím cũng xuất hiện ở một chỗ khác trong a thì ta dời p về bên phải một đoạn *ngắn nhất* cho đến một tái xuất hiện của đoạn màu tím.



  - Giả sử ta dịch p đi một đoạn có độ dài k. Nếu p[i-k] == b thì ta lại có một mis-match mới. Vì p[i-k] != a (ký tự bên trái đoạn màu tím của s).
  - Quá trình dịch chuyển p tiếp tục được lặp lại.

3.3. Giải thuật Boyer - Moore

- **Luật matching shift (tt)**
  - **<u>TH2:</u>** không tồn tại đoạn màu tím thoả mãn điều kiện trên. Trong trường hợp này ta phải dời p đi xa hơn nữa, cho đến khi một tiền tố dài nhất của p trùng với một hậu tố của đoạn màu tím. (Tiền tố dài nhất để cho độ xê dịch ngắn nhất có thể có.)



3.3. Giải thuật Boyer - Moore

- **Luật occurrence shift:**
  - Ta phải dịch p đến vị trí nào đó sao cho ký tự bên trái đoạn màu tím *giống* như ký tự bên trái đoạn màu tím của s.
  - Chiều dài phép dịch này, ta tính dãy os[a] (vị trí lớn nhất của ký tự a trong p**):**

    os[a] = max({-1} U { x | p[x] == a })

  - Cụ thể hơn, nếu tồn tại 0 ≤ x ≤ m-1 sao chop[x] == a thì ta chọn x lớn nhất như vậy, và gán os[a] = x. Nếu không có ký tự a trong chuỗi p thì ta đặt os[a] = -1
  - Giả sử ta đang quét s đến ký tự a, và ở vị trí i của mẫu p, thì theo luật occurrence shift ta nên dịch p đi một đại lượng bằng i - os['a']

3.3. Giải thuật Boyer - Moore

- Cài đặt

```
int BoyerMoore(char *source, char *find)
{
  int skip[MAX], i = 0, len, j=-1, lensource;

  len = strlen(find);
  lensource = strlen(source);
  for (i=0; i<MAX; i++)
    skip[i] = len-1;
  for (i=0; i<len; i++)
    if (skip[find[i]] == len-1)
        skip[find[i]] = len-i-1;

  i = j = len-1;
  do {
    if (source[i] == find[j])
    {
      i--;
      j--;
    }
    else
    {
      if (len-j+1 > skip[source[i]])
        i += len-j+1;
      else
        i += skip[source[i]];
      j = len-1;
    }
  } while (j>0 && i<lensource);

  if (j<=0)
    return i;
  else
    return -1;
}
```

Bài tập

- Cài đặt code thực thi các giải thuật tìm kiếm chuỗi


STOP