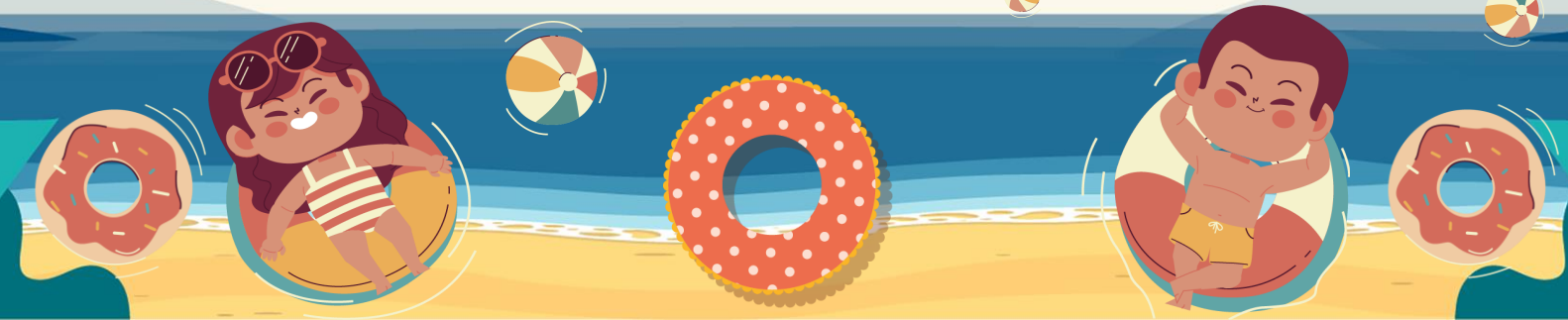
UIT
Trường Đại học Công nghệ Thông tin
Khoa Khoa học và Kỹ thuật Thông tin
BAN HỌC TẬP

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

TRAINER

Trịnh Đức HTCL2021
Võ Huy Hoàng MTCL2021
Bảo Trần CNTT2021



Nội dung:

I. Cây - Tree
II. Các thuật toán sắp xếp - Sort
III. Bảng băm – Hash Table

I. Cây - Tree

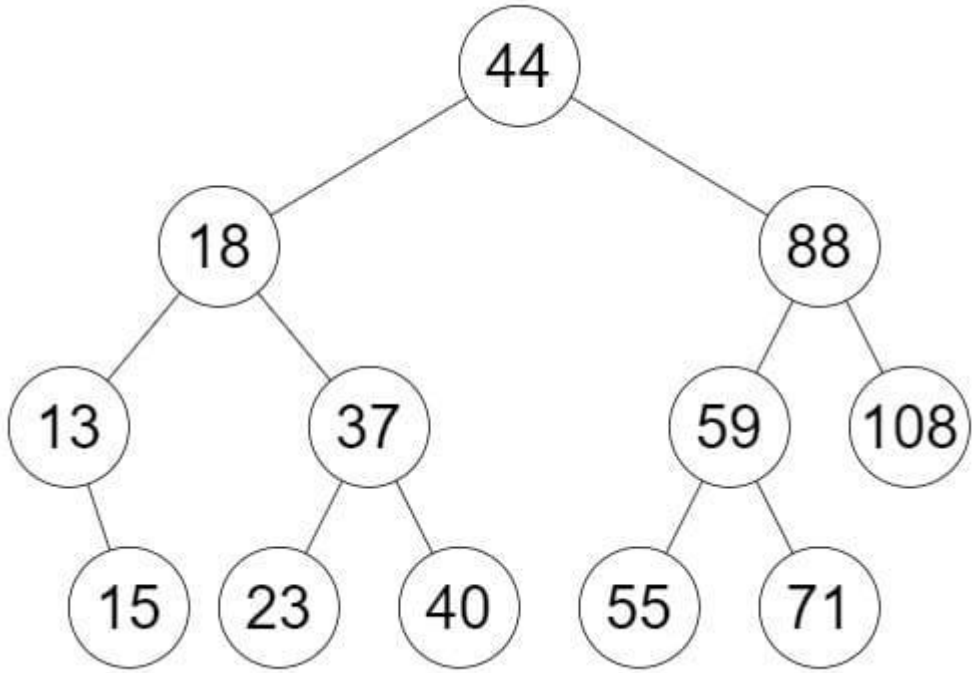
Nội dung:

1. Định nghĩa
2. Các khái niệm
3. Cây nhị phân
   - ➢ Cấu trúc dữ liệu của cây nhị phân
   - ➢ Duyệt cây nhị phân
   - ➢ Cây nhị phân tìm kiếm
4. Các thao tác trên cây nhị phân
5. Vận dụng

1. Định Nghĩa

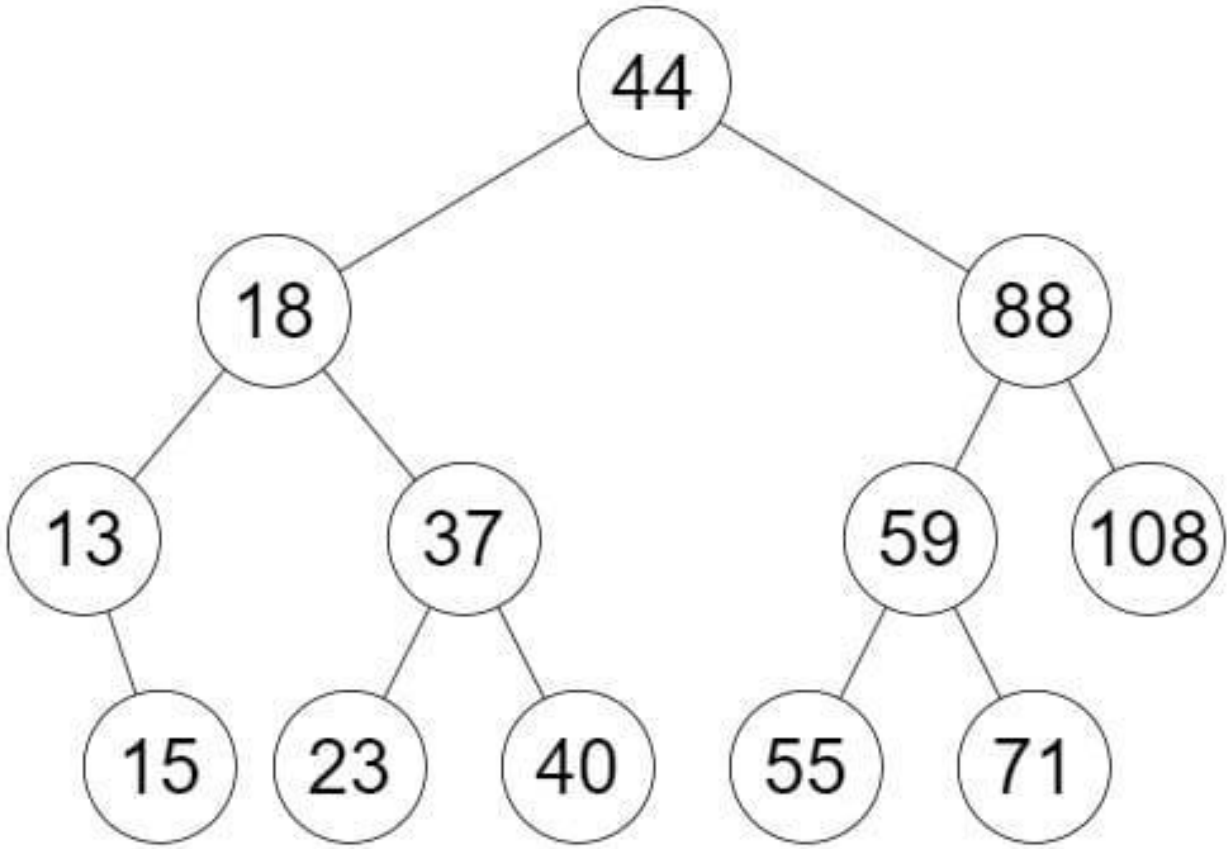
Cây là một tập hợp T các phần tử (gọi là nút của cây), trong đó có một nút đặc biệt gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau T_1 , T_2 , …,T_n theo quan hệ phân cấp, trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp i+1. Quan hệ này người ta gọi là quan hệ cha – con.

Ví dụ:



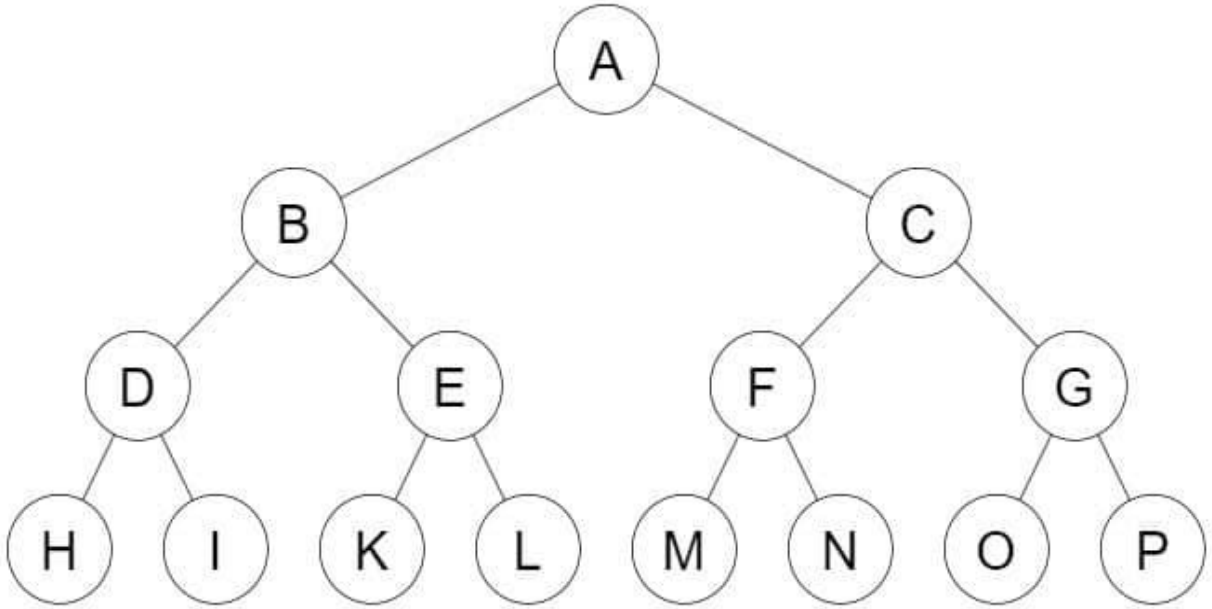
2. Các khái niệm

- Bậc của một nút: là số cây con của nút đó
- Bậc của một cây: là bậc lớn nhất của các nút trong cây
- Nút gốc: là Nút không có cha
- Nút là: là nút có bậc bằng 0
- Độ dài đường đi từ gốc đến nút x: là số nhánh cần đi qua kể từ gốc đến x



3. Cây Nhị Phân

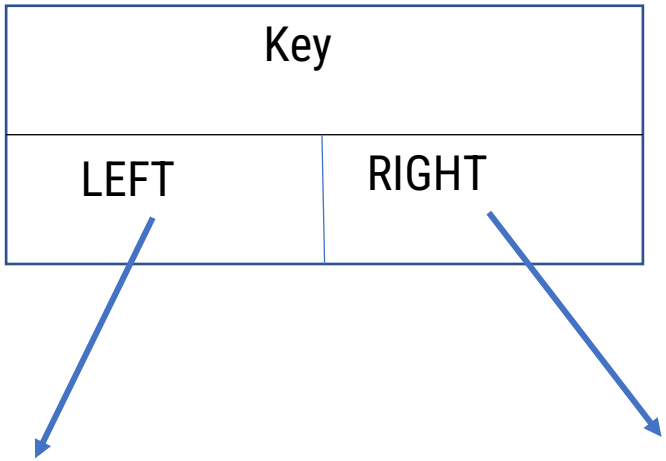
- Số nút nằm ở mức i là 2^i.
- Số nút lá <=2^{h-1}, với h là chiều cao của cây.
- Chiều cao của cây h >= $\log_2(N)$ - N = số nút trong cây
- Số nút trong cây <= 2^h-1



3. Cây Nhị Phân

➢ Cấu trúc dữ liệu của cây nhị phân

```
struct NODE
{
        int Key;
        NODE* pLeft;
        NODE* pRight;
}; typedef NODE* TREE;
```



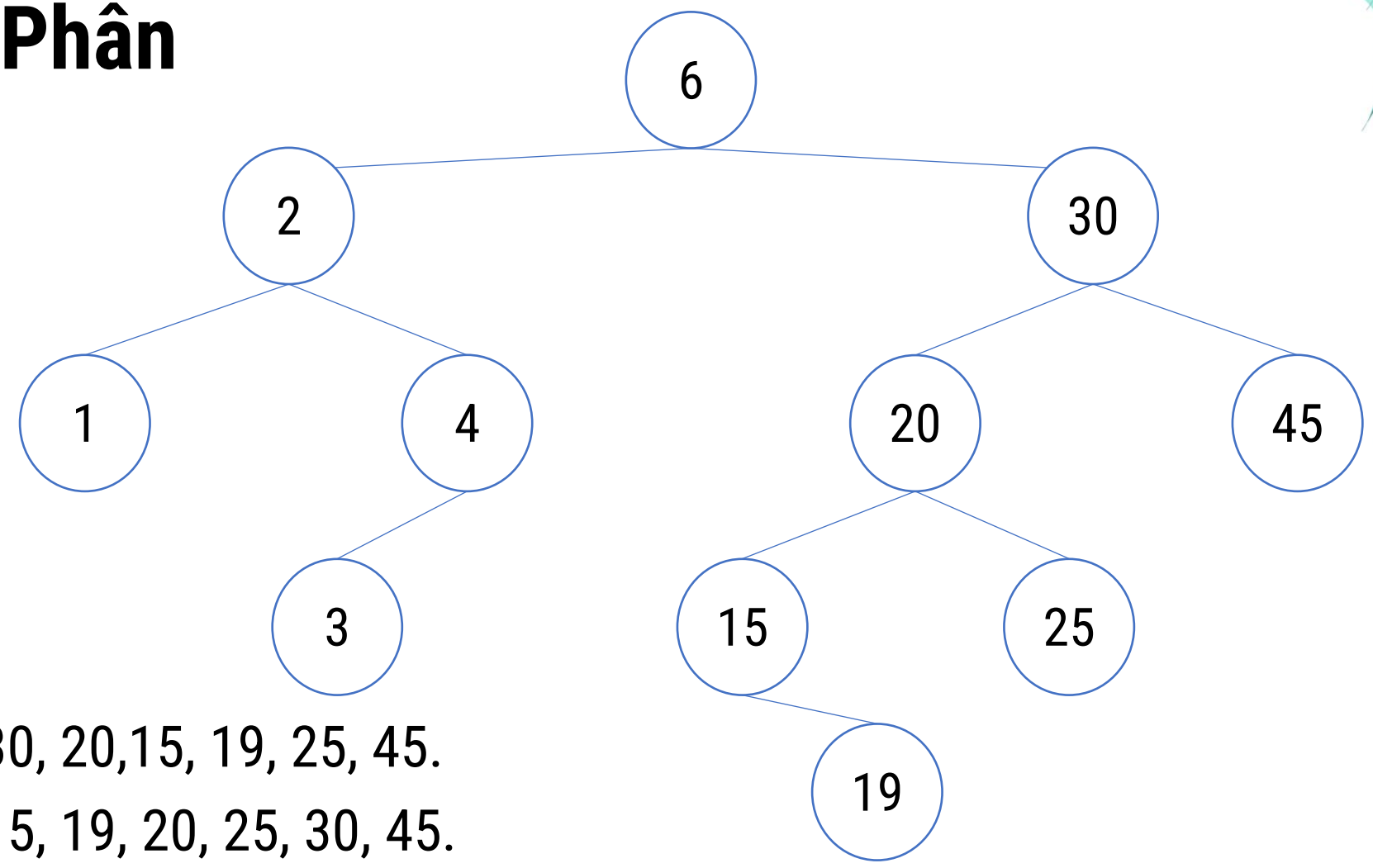
3. Cây Nhị Phân

- Duyệt cây nhị phân
  - Có 3 trình tự thăm gốc
    - Duyệt trước
    - Duyệt giữa
    - Duyệt sau
  - Độ phức tạp $O(\log_2(h))$

(h là chiều cao của cây)

3. Cây Nhị Phân

Ví dụ



Kết quả:

- NLR: 6, 2, 1, 4, 3, 30, 20,15, 19, 25, 45.
- LNR: 1, 2, 3, 4, 6, 15, 19, 20, 25, 30, 45.
- LRN:1, 3, 4, 2, 19, 15, 25, 20, 45, 30, 6.

3. Cây Nhị Phân

- Duyệt trước

```
void NLR(TREE root)
{
        if (!root)
        {
                NLR(root->pLeft);
                NLR(root->pRight);
        }
}
```

3. Cây Nhị Phân

- Duyệt giữa

```
void LNR(TREE Root)
{
        if (root != NULL)
        {
                LNR(Root->pLeft);
                LNR(Root->pRight);
        }
}
```

3. Cây Nhị Phân

- Duyệt sau:

```
void LRN(TREE Root)
{
        if (Root != NULL)
        {
                LRN(Root->pLeft);
                LRN(Root->pRight);
        }
}
```

3. Cây Nhị Phân

➢ Cây nhị phân tìm kiếm

- Nhờ trật tự bố trí khóa trên cây :
  - Định hướng được khi tìm kiếm
- Cây gồm N phần tử :
  - Các thao tác searchNode, insertNode, delNode có độ phức tạp trung bình O(h), với h là chiều cao của cây
  - Trường hợp tốt nhất, độ cao h = log2(N). Tương đương tìm kiếm nhị phân trên mảng có thứ tự.
  - Trường hợp xấu nhất, cây là 1 DSLK. Lúc đó các thao tác trên sẽ có độ phức tạp O(N).

4. Các thao tác trên cây nhị phân

- Tạo 1 cây rỗng
- Tạo 1 nút có key bằng x
- Thêm 1 nút vào cây nhị phân tìm kiếm
- Xóa 1 nút có key bằng x trên cây
- Tìm 1 nút có khóa bằng x trên cây

4. Các thao tác trên cây nhị phân

- Tạo cây rỗng

```
void createTree(TREE &T)
{
        T = NULL;
}
```

4. Các thao tác trên cây nhị phân

- Tạo 1 nút có key bằng x

```
NODE * CreateNode(int x)
{
        NODE *p = new NODE;
        p->Key = x;
        p->pLeft = NULL;
        p->pRight= NULL;
        return p;
}
```

4. Các thao tác trên cây nhị phân

➢ Thêm một nút x

```
void addNode(TREE &T , int x)
{
        if(T)
        {
                if(T->Key == x) return;
                if(T->Key > x)     return addNode(T->pLeft,x);
                if(T->Key < x)     return addNode(T->pRight,x);
        }
        T = new NODE;
        CreateNode(T,x);
}
```

4. Các thao tác trên cây nhị phân

- Tìm nút có khóa bằng x

```
NODE *SearchNode(TREE T, int x)
{
        if(T)
        {
                if(T->Key == x) return T;
                else if (x > T->Key)    return SearchNode(T->Right,x);
                else                    return SearchNode(T->Left,x);
        }
        return NULL;
}
```

4. Các thao tác trên cây nhị phân

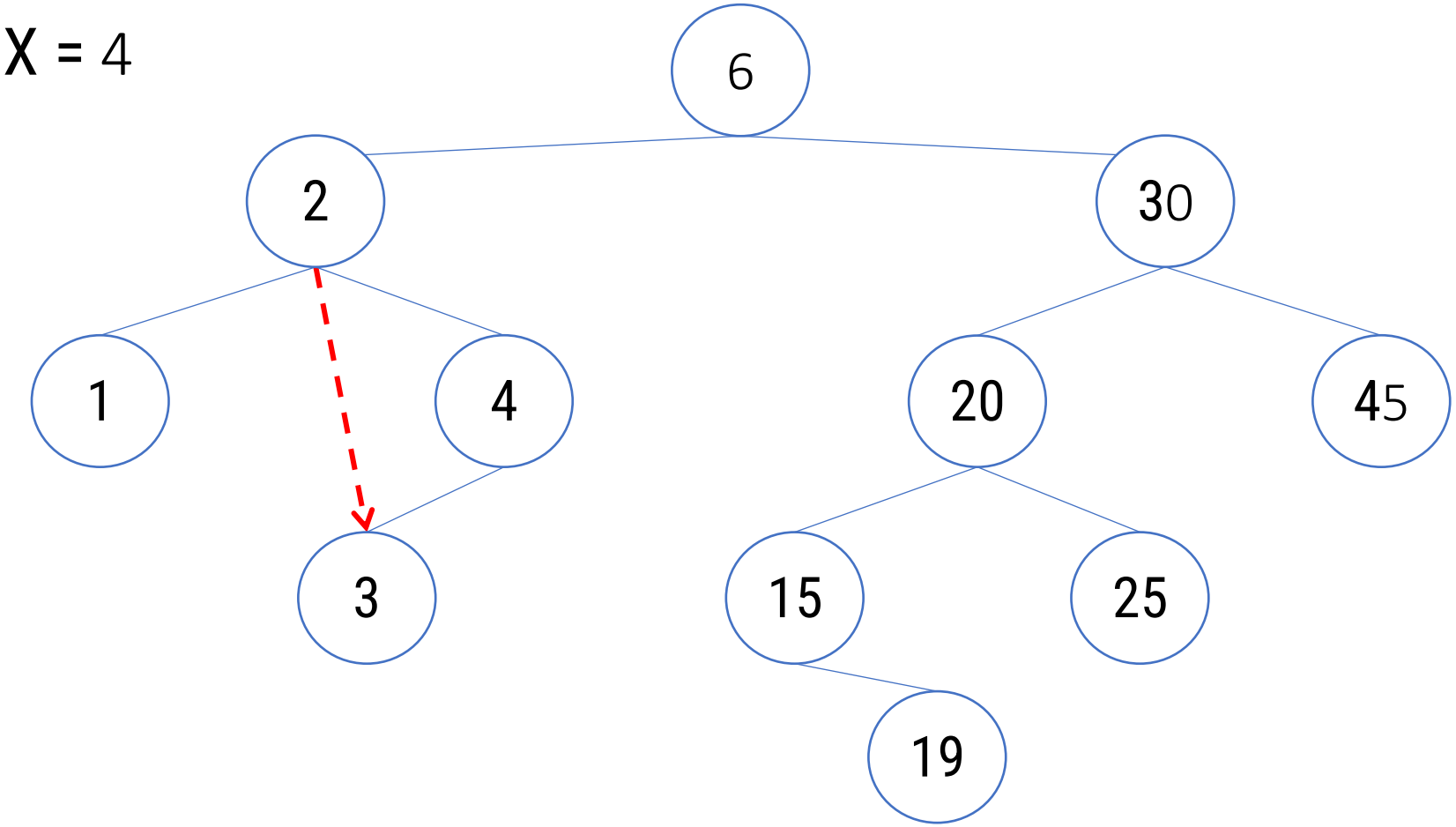
➢ Hủy 1 nút có khóa bằng x trên cây

- Hủy 1 phần tử trên cây phải đảm bảo điều kiện ràng buộc của cây nhị phân tìm kiếm.
- Có 3 trường hợp khi hủy 1 nút trên cây:
  - TH1: X là nút lá
  - TH2: X có 1 cây con (trái hoặc phải)
  - TH3: X có 2 cây con
- TH1: Ta xóa nút lá mà không ảnh hưởng đến các nút khác trên cây.
- TH2: Trước khi xóa x ta móc nối cha của X với con duy nhất của X.
- TH3: Xóa gián tiếp.

4. Các thao tác trên cây nhị phân

- Minh họa huỷ phần tử x có 1 cây con

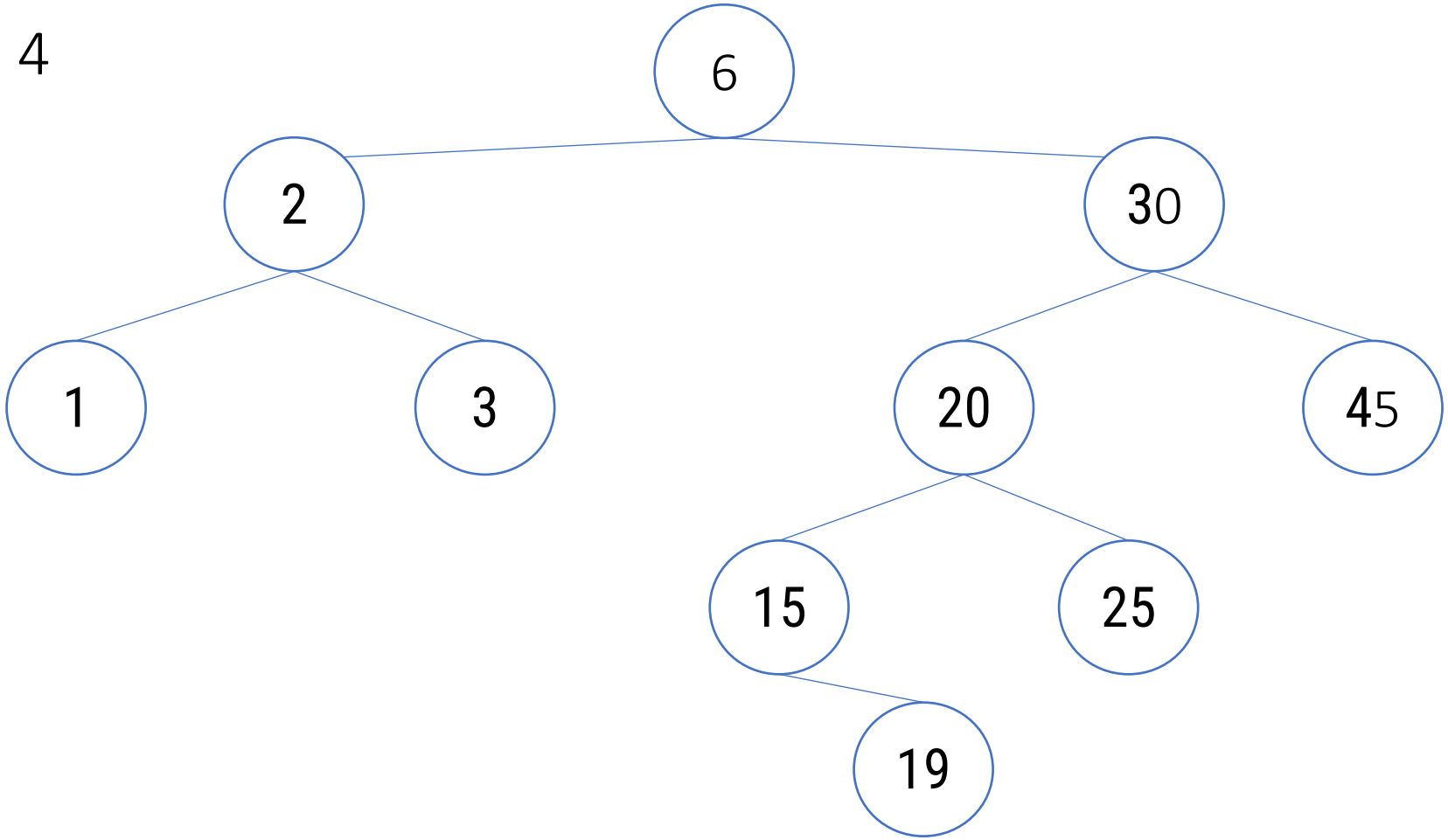
Hủy X = 4



4. Các thao tác trên cây nhị phân

- Minh họa huỷ phần tử x có 1 cây con

Hủy X = 4



4. Các thao tác trên cây nhị phân

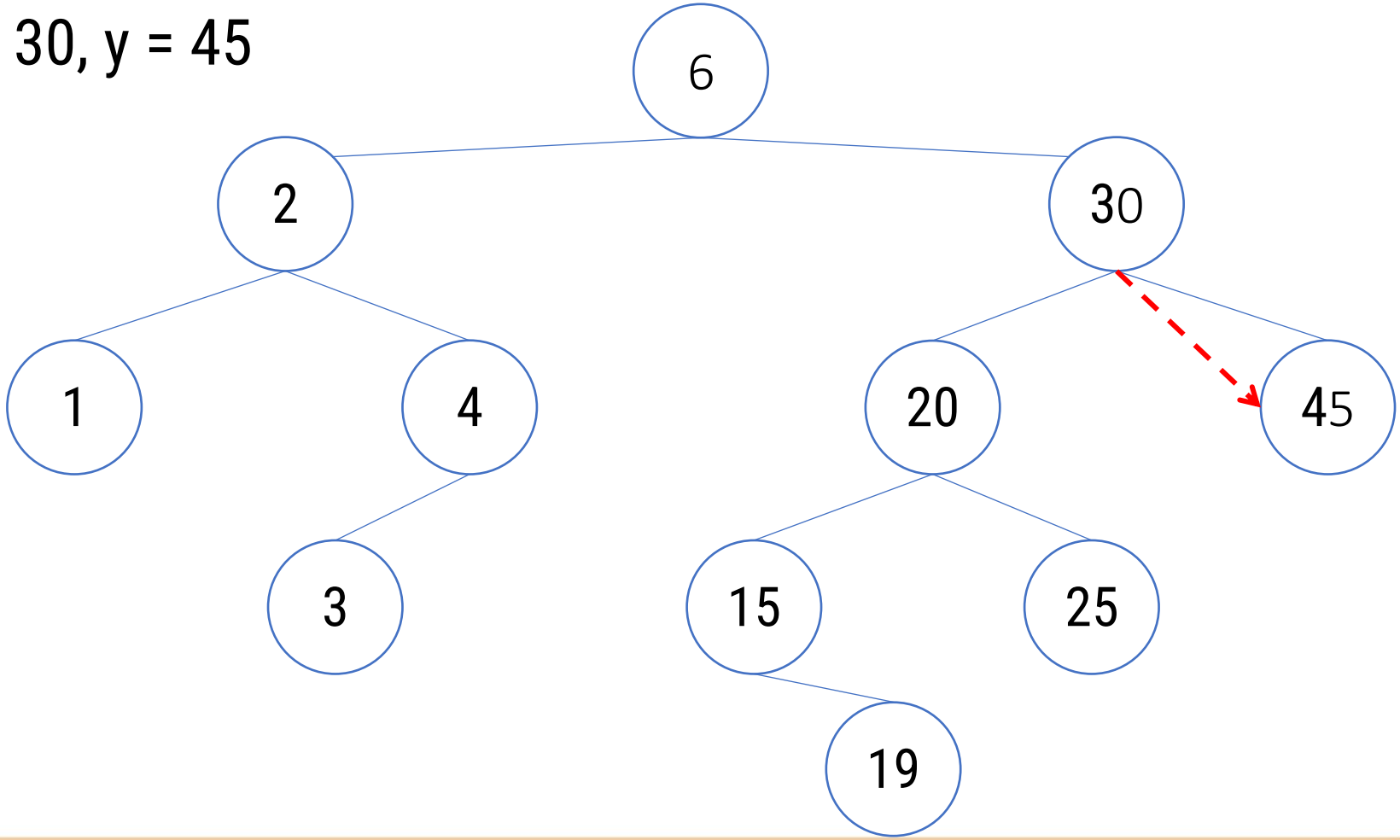
➢ Hủy 1 nút có 2 cây con

- Thay vì hủy X ta tìm phần tử thế mạng Y. Nút Y có tối đa 1 cây con.
- Thông tin lưu tại nút Y sẽ được chuyển lên lưu tại X.
- Ta tiến hành xóa hủy nút Y (xóa Y giống 2 trường hợp đầu).
- Cách tìm nút thế mạng Y cho X: có 2 cách:
  - C1: Nút Y là nút có khóa nhỏ nhất (trái nhất) bên cây con phải X.
  - C2: Nút Y là nút có khóa lớn nhất (phải nhất) bên cây con trái của X.

4. Các thao tác trên cây nhị phân

➢ Minh họa huỷ phần tử x có 2 cây con

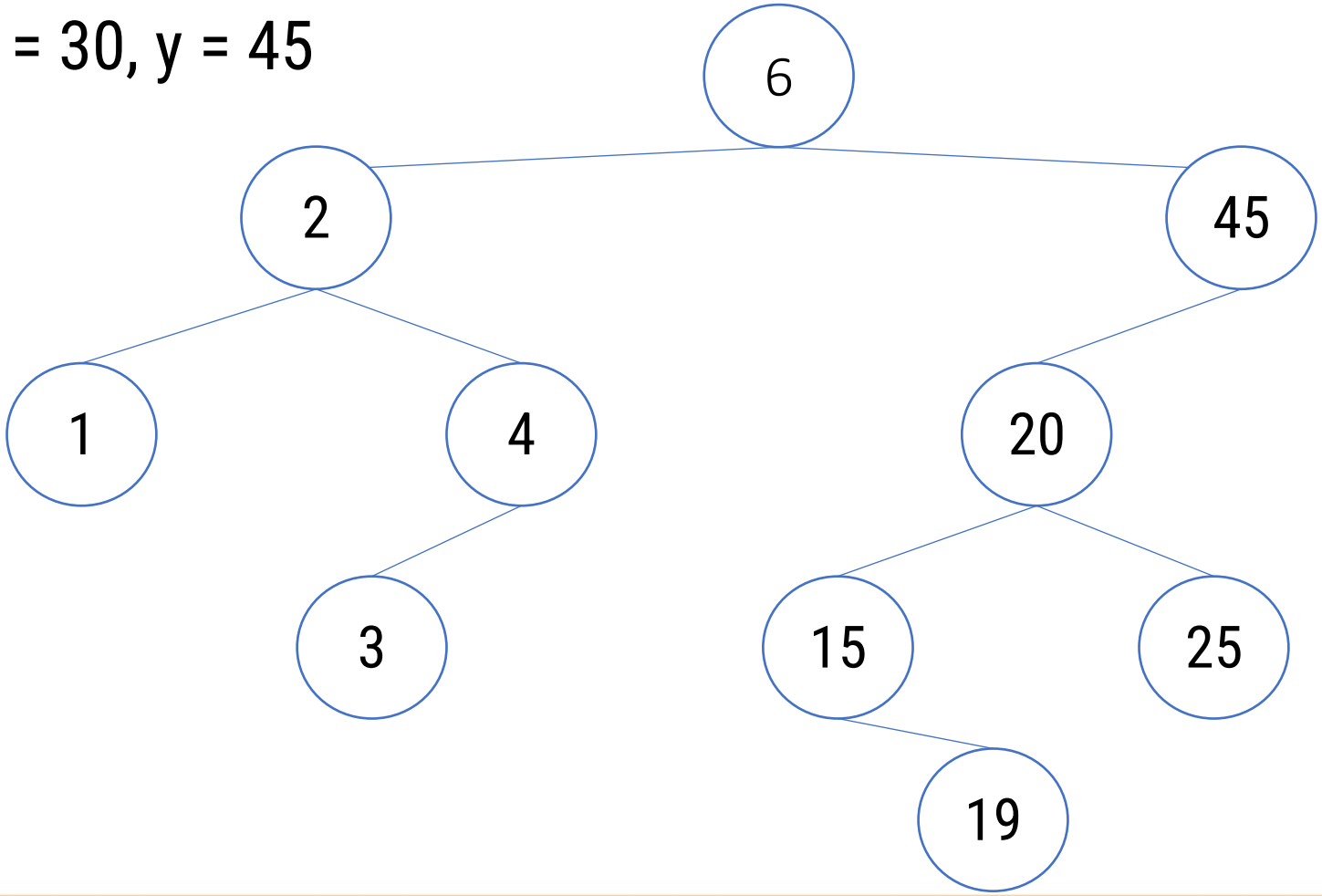
- Xóa nút x = 30, y = 45



4. Các thao tác trên cây nhị phân

➢ Minh họa huỷ phần tử x có 2 cây con

- Xóa nút x = 30, y = 45



4. Các thao tác trên cây nhị phân

- Minh họa huỷ phần tử x có 2 cây con

```
void DeleteNode(Tree& T,int x)
{
    if (T!=NULL)
    {
        if (T->Key > x)
        {
            DeleteNode(T->pLeft, x);
        }
        else if (T->Key < x)
        {
            DeleteNode(T->pRight, x);
        }
```

4. Các thao tác trên cây nhị phân

➢ Minh họa huỷ phần tử x có 2 cây con

```
        else
        {
            Node* X = T;
            if (T->pLeft == NULL)
            {
                T = T->pRight;
            }
            else if (T->pRight == NULL)
            {
                T = T->pLeft;
            }
            else
            {
                NodeTheMang(X, T->pRight);
            }
            delete X;
        }
    }
    else
    return;
}
```

4. Các thao tác trên cây nhị phân

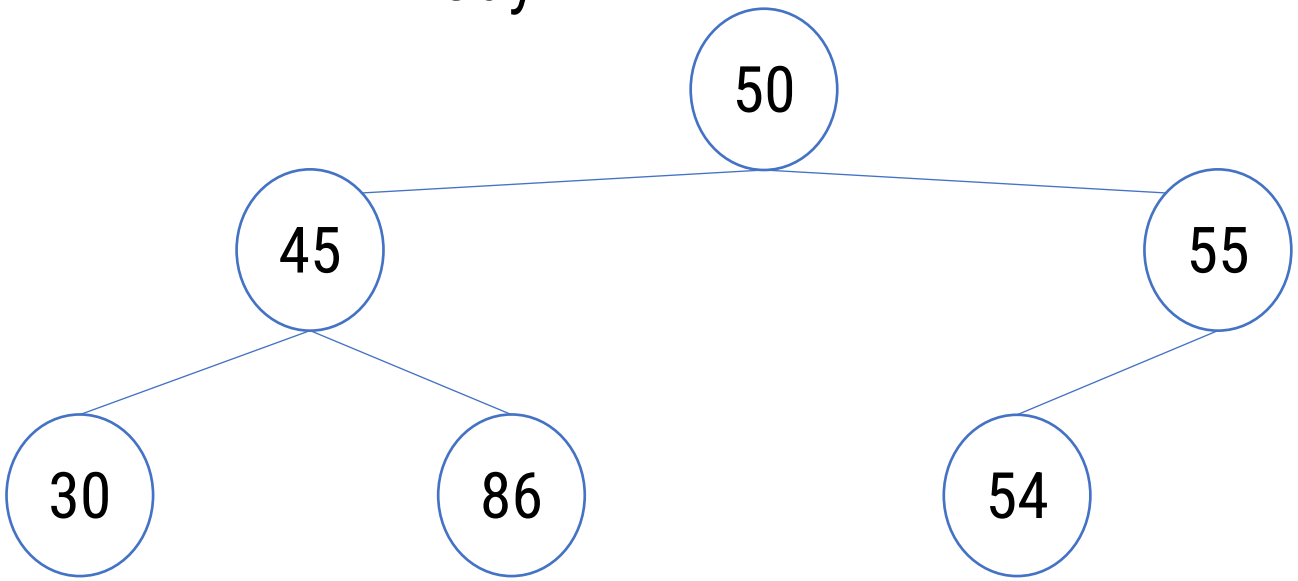
- Minh họa huỷ phần tử x có 2 cây con

```
void NodeTheMang(Tree& X, Tree &Y)
{
    if (Y->pLeft != NULL)
    {
        NodeTheMang(X, Y->pLeft);
    }
    else
    {
        X->Key = Y->Key;
        X = Y;
        Y = Y->pRight;
    }
}
```

5. Vận dụng

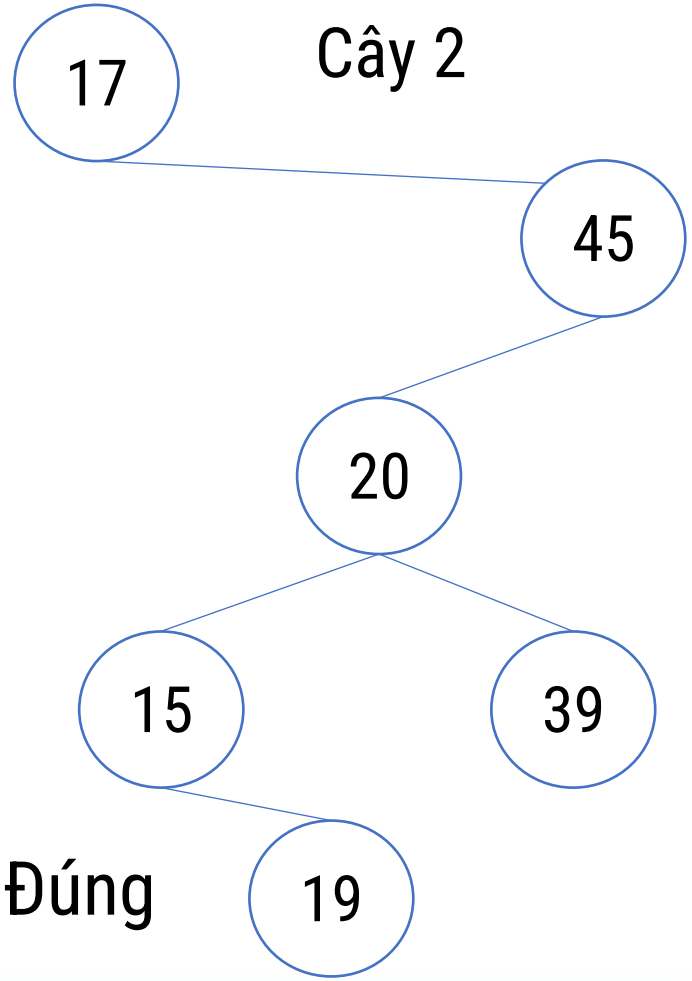
Trong các cây sau, cây nào là cây nhị phân tìm kiếm? Vì sao?

Cây 1



Sai

Cây 2



Đúng

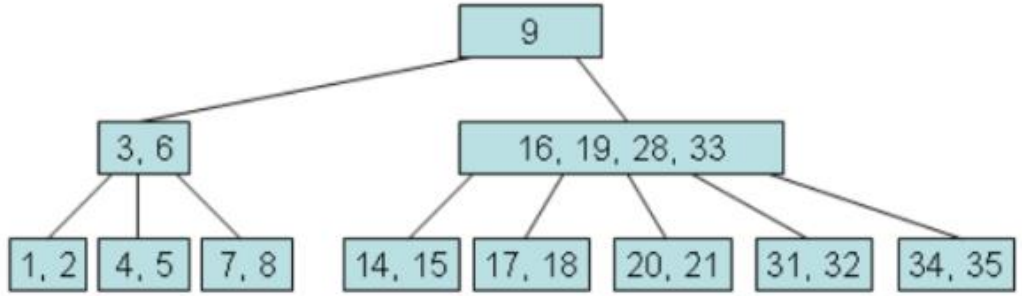
B TREE

1. Định Nghĩa

Một B-Tree bậc M là cây M-Phân tìm kiếm có các tính chất sau:

- Mỗi node (trừ node gốc) có tối thiểu M/2 khóa.
- Mỗi node có tối đa M-1 khóa.
- Mỗi node (ngoại trừ lá không có nhánh con) có tối đa M nhánh con
- Tại mỗi khóa X bất kỳ, các khóa ở nhánh trái < x < các khóa ở nhánh phải.
- Các node là nằm cùng mức.

• Ví dụ: B-tree bậc 5



2. Giới thiệu

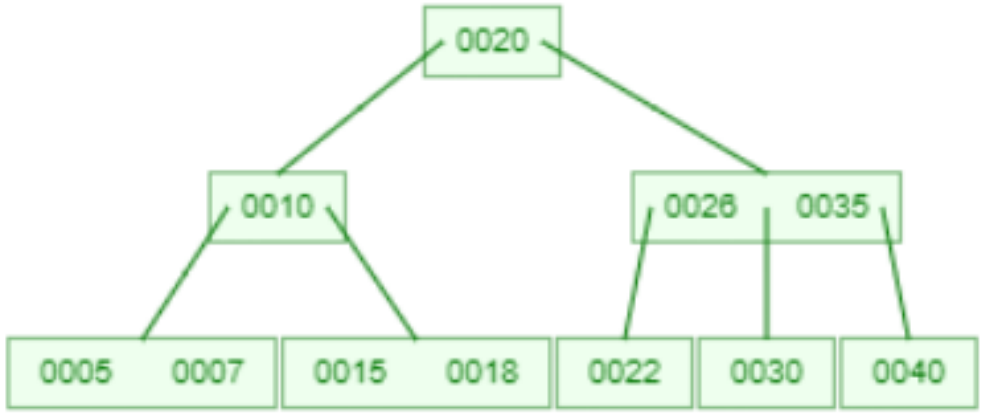
- Do R.Bayer và E.M.McCreight đưa ra năm 1972.
- B-Tree là cấu trúc dữ liệu phù hợp cho việc lưu trữ và truy xuất dữ liệu trên bộ nhớ ngoài (đĩa cứng)
- Hạn chế số thao tác đọc mỗi lần tìm kiếm cây:
  - Mỗi lần truy xuất đọc toàn bộ dữ liệu trong 1 node (mỗi node chứa M phần tử)
  - Sử dụng thuật toán tìm nhị phân để tìm phần tử x (giá trị cần tìm).
- Chiều cao cây $= \log_M N$, tăng M chiều cao cây giảm rất nhanh

3. Thêm

- Khóa mới sẽ được thêm vào node lá
- Nếu chưa đầy-> Thêm được
- Nếu đầy -> tách node:
  - Khóa giữa node được lan truyền ngược lên node cha
    - Trong những trường hợp đặc biệt lan truyền đến tận gốc của B-Tree
  - Phần còn lại chia thành 2 node cạnh nhau trong cùng 1 mức

3. Thêm

Tạo B-Tree bậc 3 từ dãy các khóa sau :
20,40,10,30,15,35,7,26,18,22,5



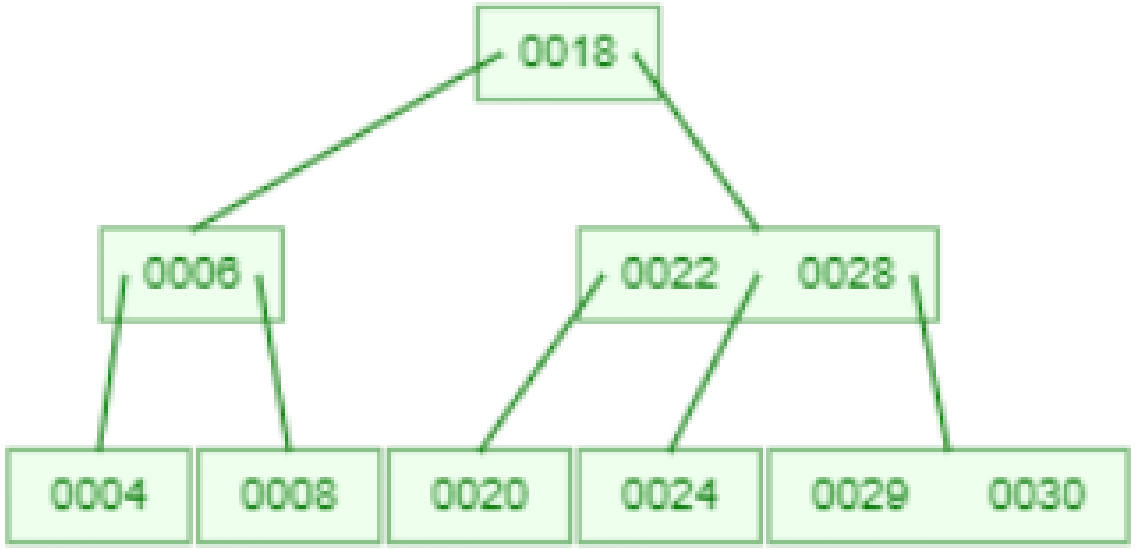
Link tham khảo: https://bom.so/QFULnq

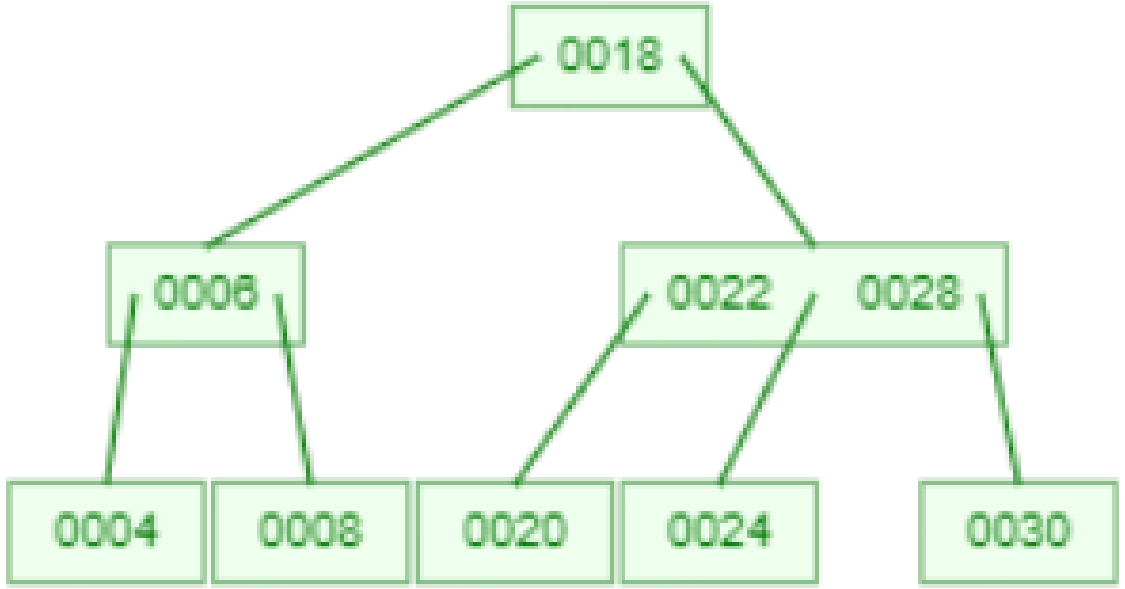
3. Xóa

- Khóa cần xóa nằm trên node lá -> Xóa bình thường
- Khóa cần xóa không trên node lá
- Tìm phần tử thay thế: trái nhất ở cây con bên phải hoặc phải nhất của cây con bên trái

3. Xóa

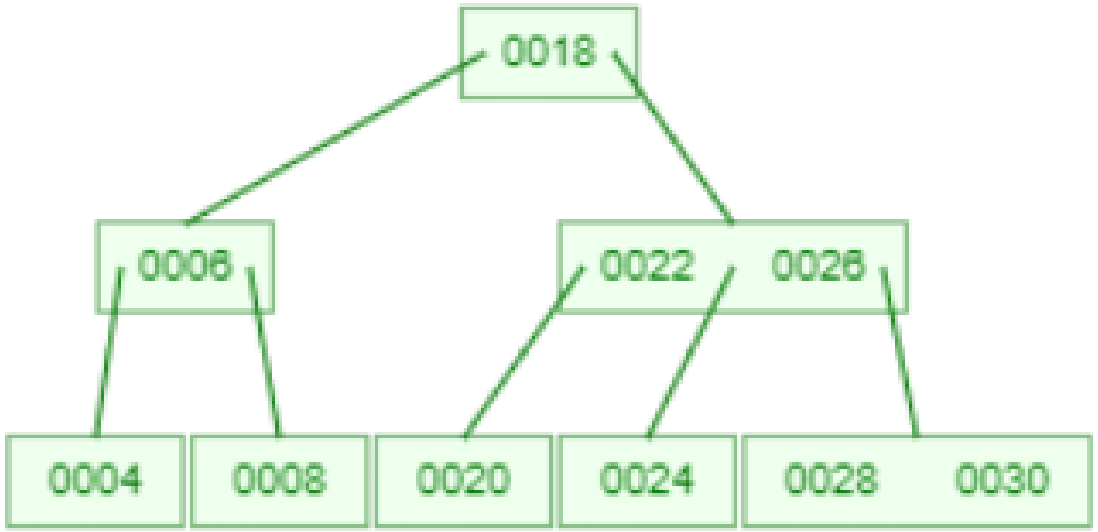
Ví dụ: Xóa 1 khóa trên trang lá (Xóa 29)

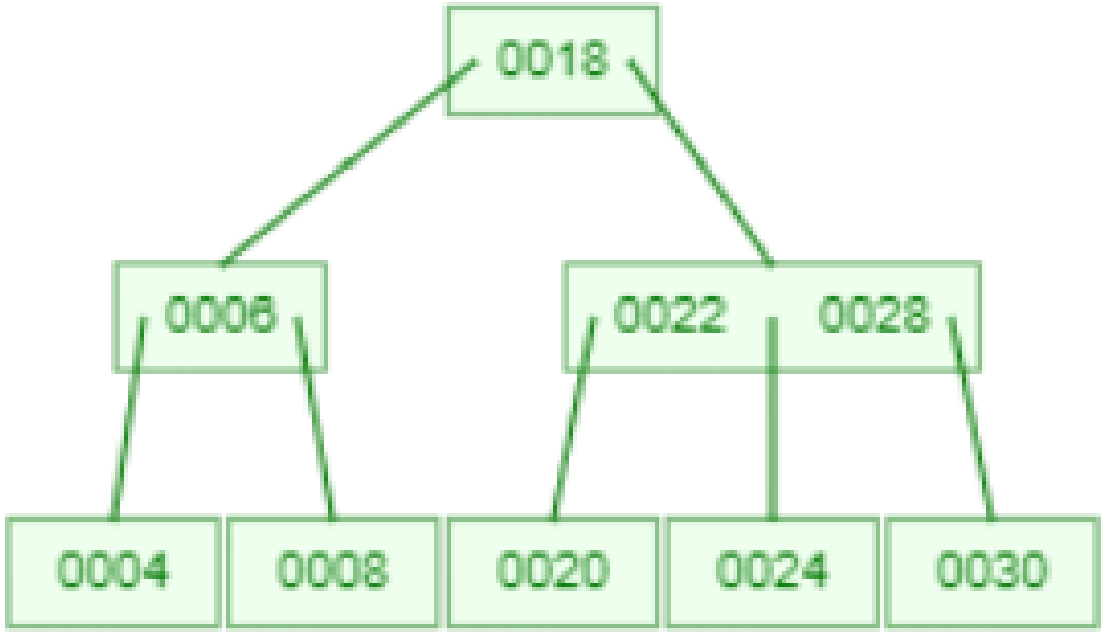




3. Xóa

Ví dụ: Xóa 26



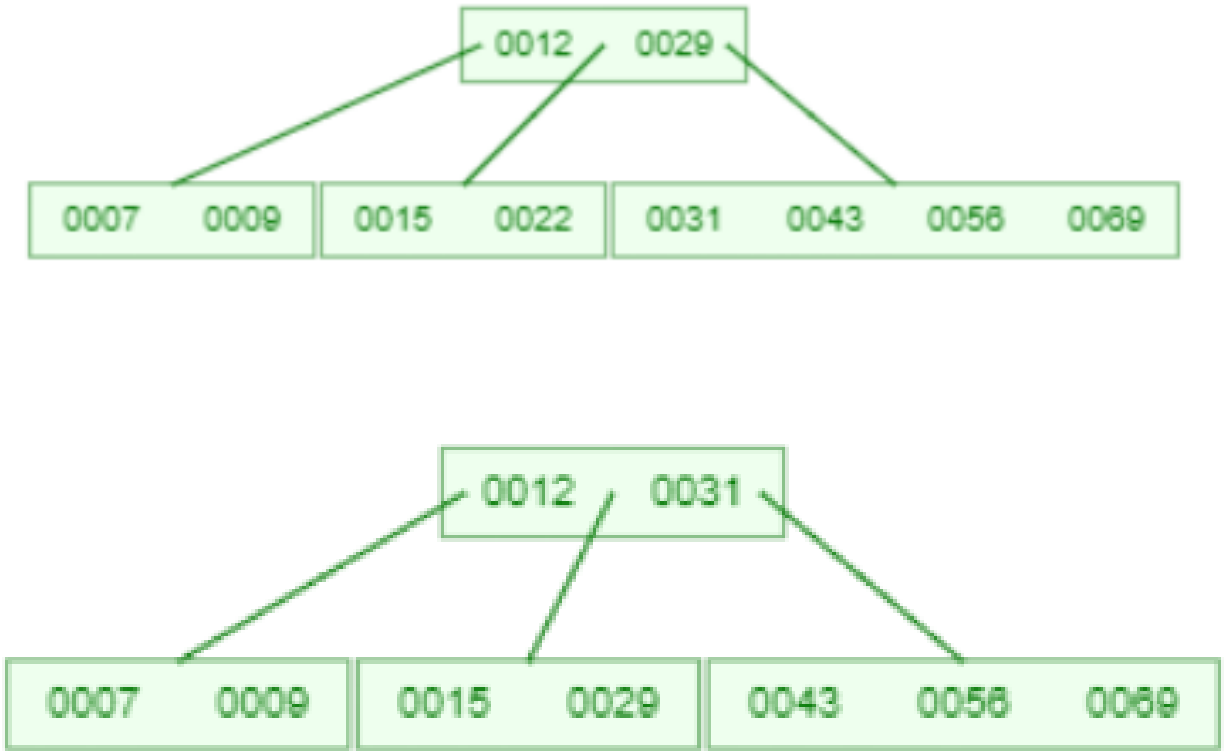


4. Cân bằng cây sau khi xóa

Sau khi xóa node bị thiếu vi phạm điều kiện của B-Tree

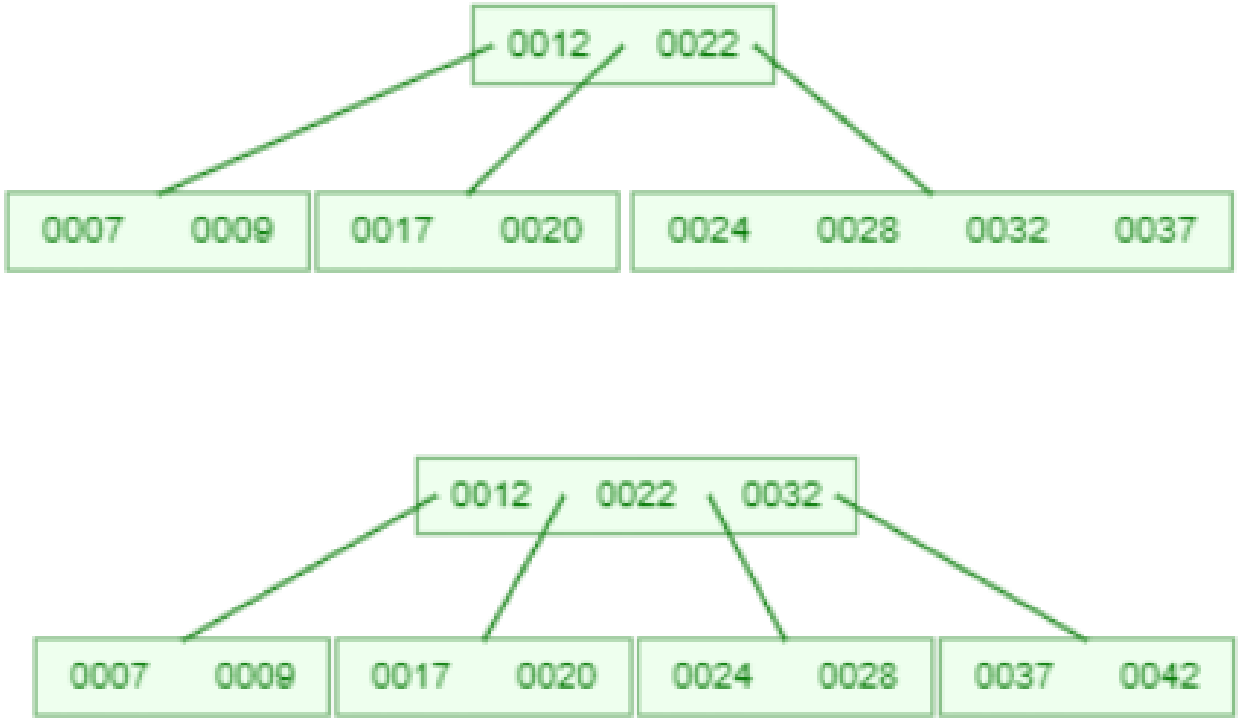
- Chuyển dời phần tử từ node thừa
- Ghép với node bên cạnh

4. Cân bằng cây sau khi xóa



- Nếu một trong các nút kế cận nút đang xét có số lượng khóa nhiều hơn số lượng tối thiệu
  - Đưa 1 khóa của nút kế cận lên nút cha
  - Đưa 1 khóa ở nút cha xuống nút đang xét

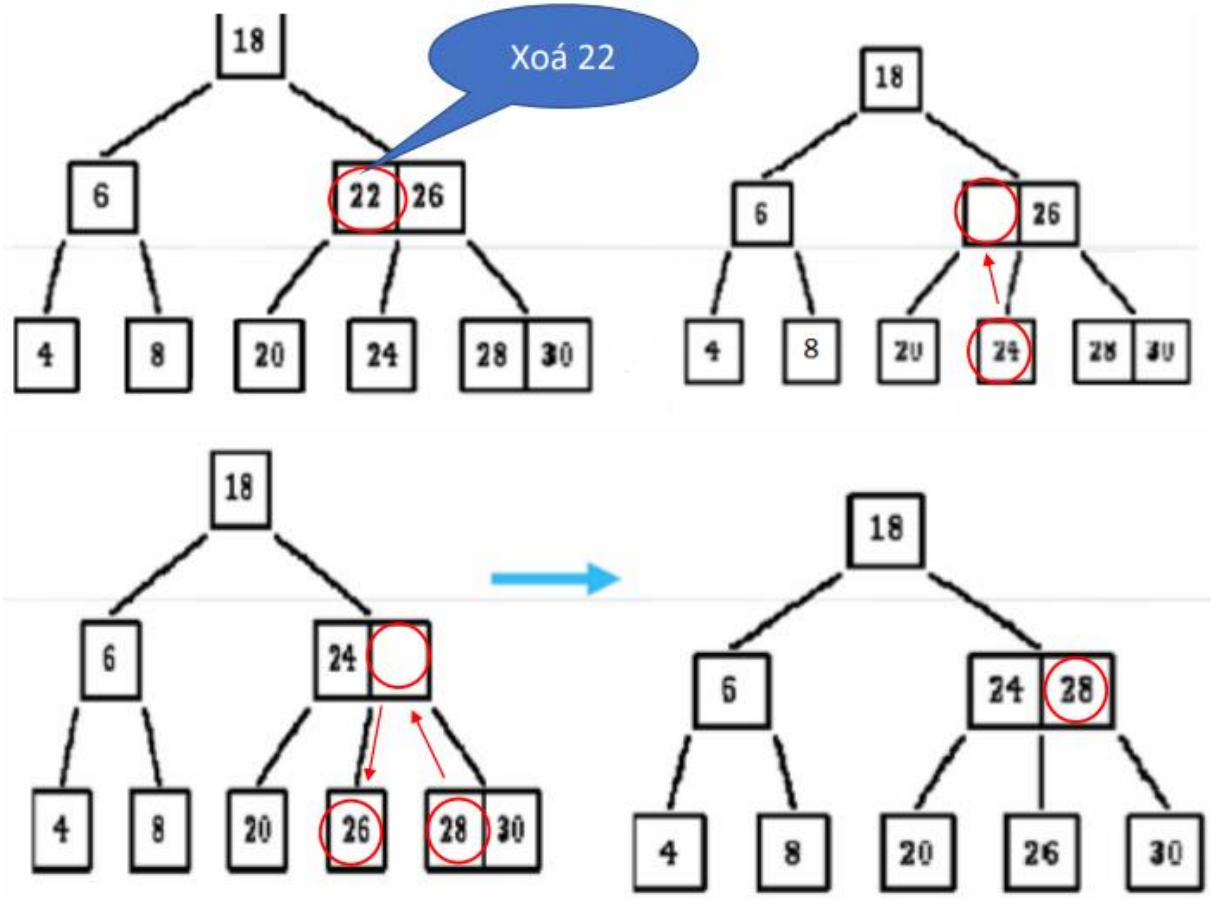
4. Cân bằng cây sau khi xóa



- Tất cả nút kế cận nút đang xét có số lượng khóa vừa đủ:
  - Chọn 1 nút kế cận để hợp nhất với nút đang xét và khóa tương ứng ở nút cha
  - Nếu nút cha thiếu khóa thì lặp lại quá trình này

4. Cân bằng cây sau khi xóa

- Xóa 22(Bậc 3)
  - Nút thay thế là 24 : trái nhất của nhành phải:
    - -> Thiếu lá
  - Đưa 1 khóa của nút kế cận lên nút cha
  - Đưa 1 khóa ở nút cha xuống nút thiếu



II. Các thuật toán sắp xếp SORT



Nội dung:

1. Đổi chỗ trực tiếp – Interchange Sort
2. Chọn trực tiếp – Selection Sort
3. Nổi bọt – Bubble Sort
4. Chèn trực tiếp – Insertion Sort
5. Chèn nhị phân – Binary Insertion Sort
6. Quick Sort

1. Đổi chỗ trực tiếp – Interchange Sort

➢**Ý tưởng:**

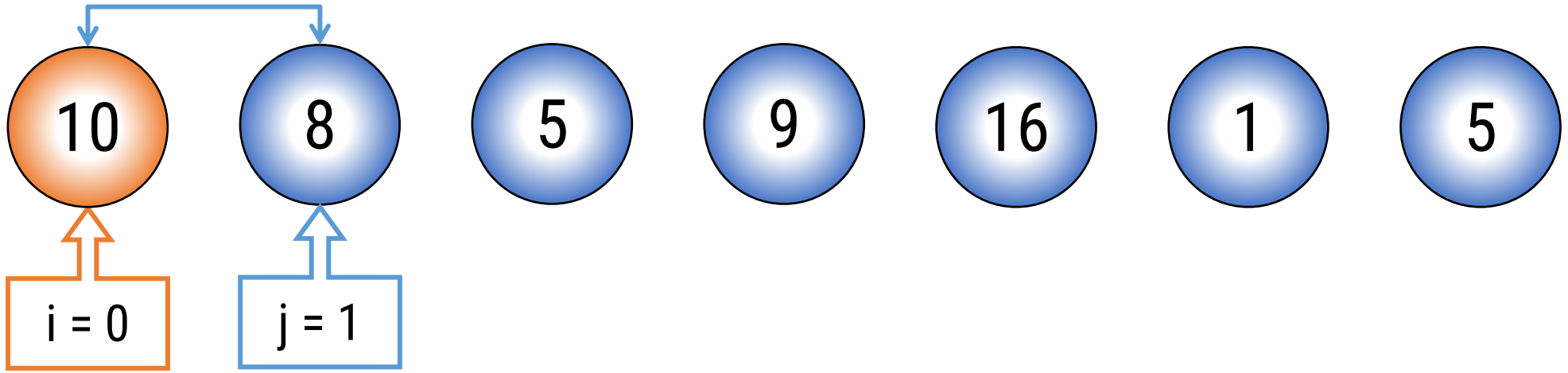
Duyệt từ đầu dãy, tìm tất cả các nghịch thế chứa phần tử này, triệt tiêu chúng bằng cách đổi chỗ 2 phần tử trong cặp nghịch thế. Lặp lại xử lý trên với phần tử kế trong dãy.

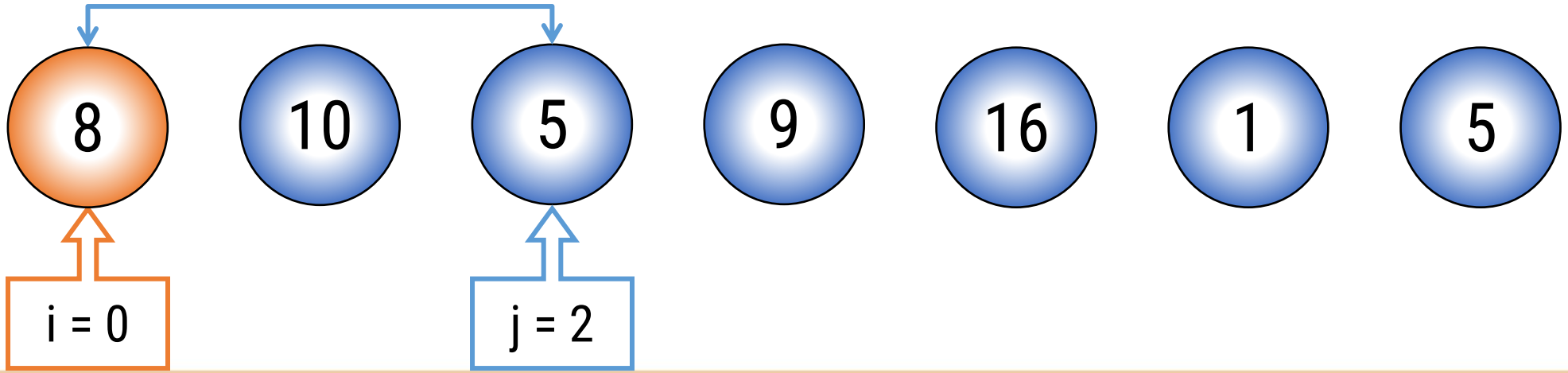
➢**Ví dụ:**

Cho dãy số a = {10,8,5,9,16,1,5}. Hãy sắp xếp dãy a theo tự tăng dần.

1. Đổi chỗ trực tiếp – Interchange Sort

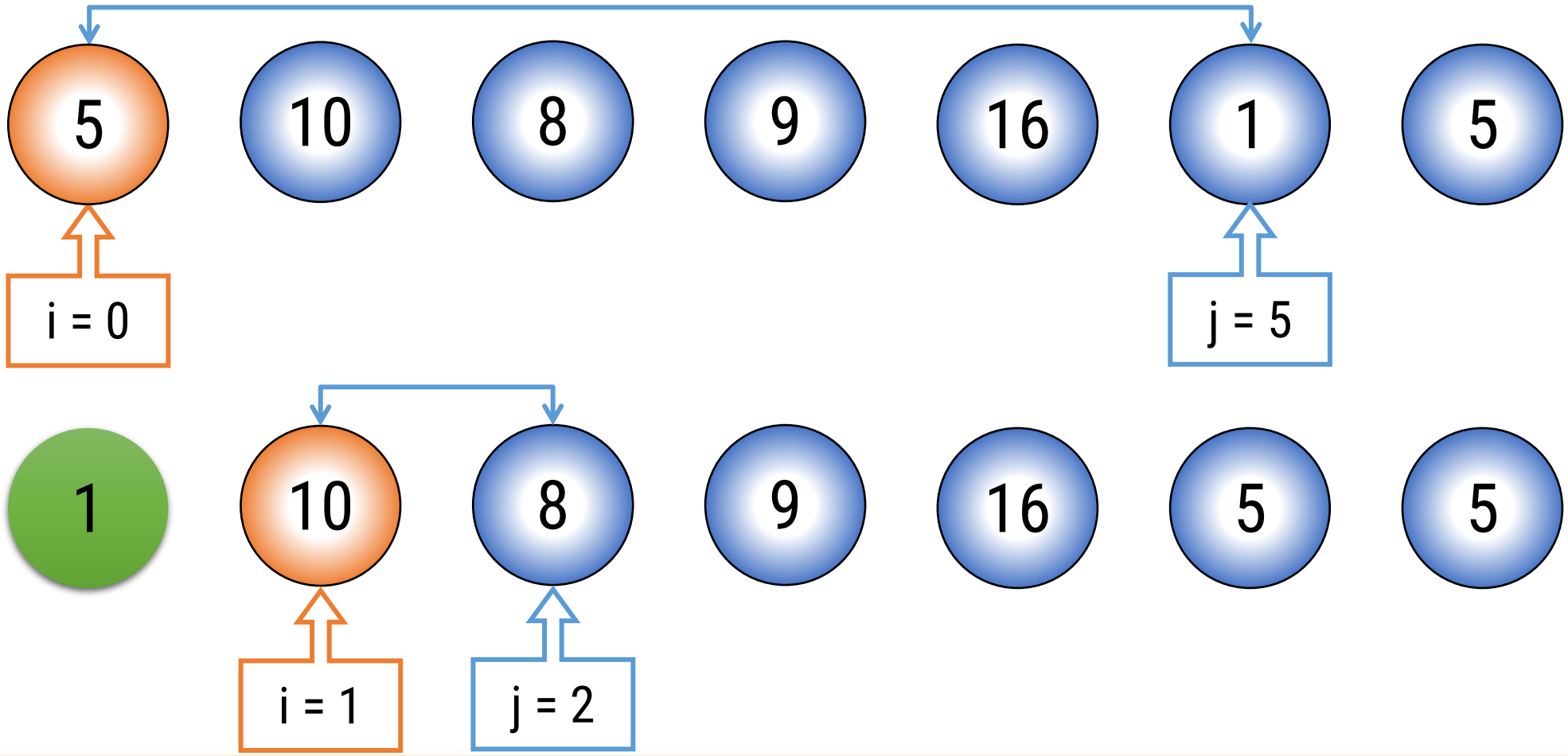
➢ Minh họa thuật toán:





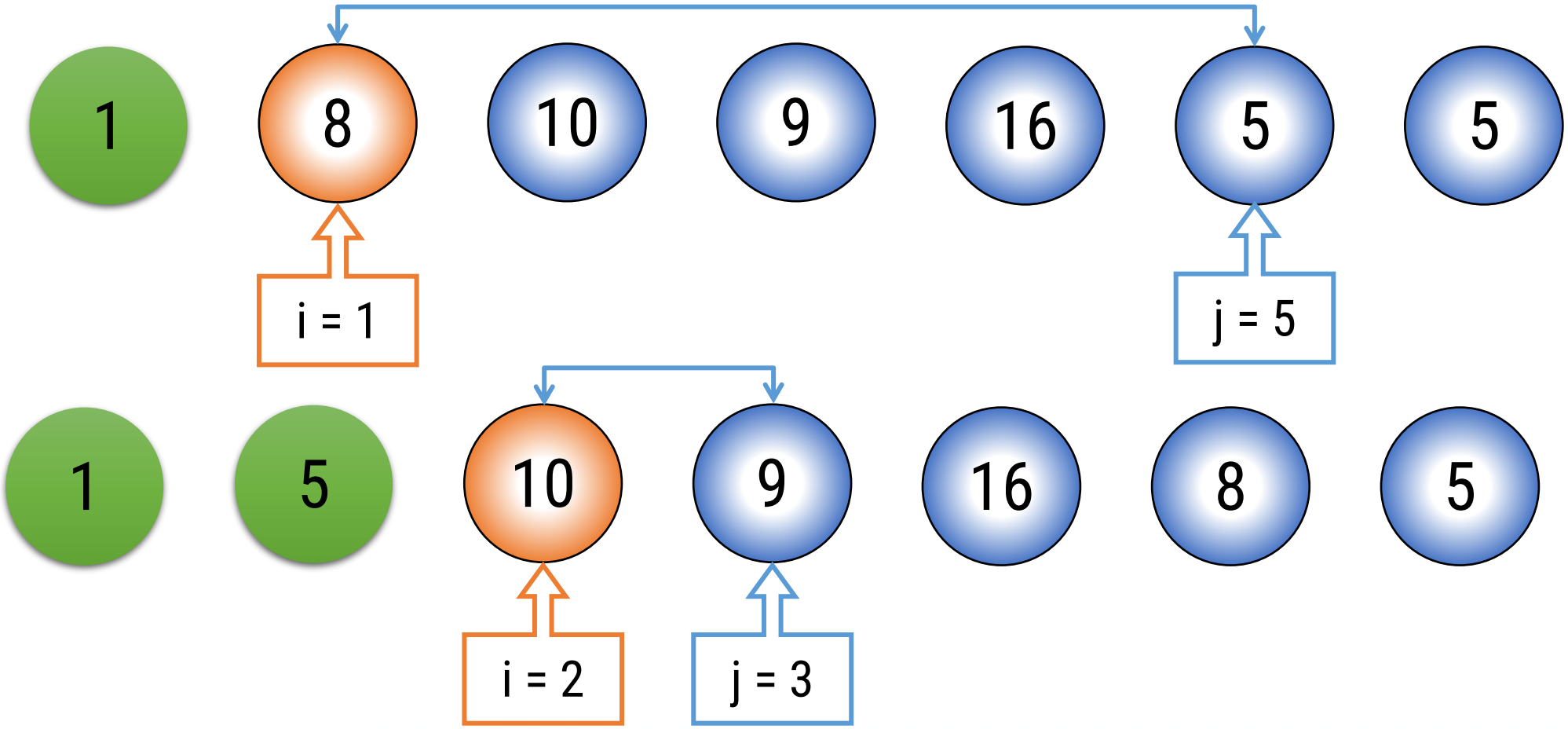
1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



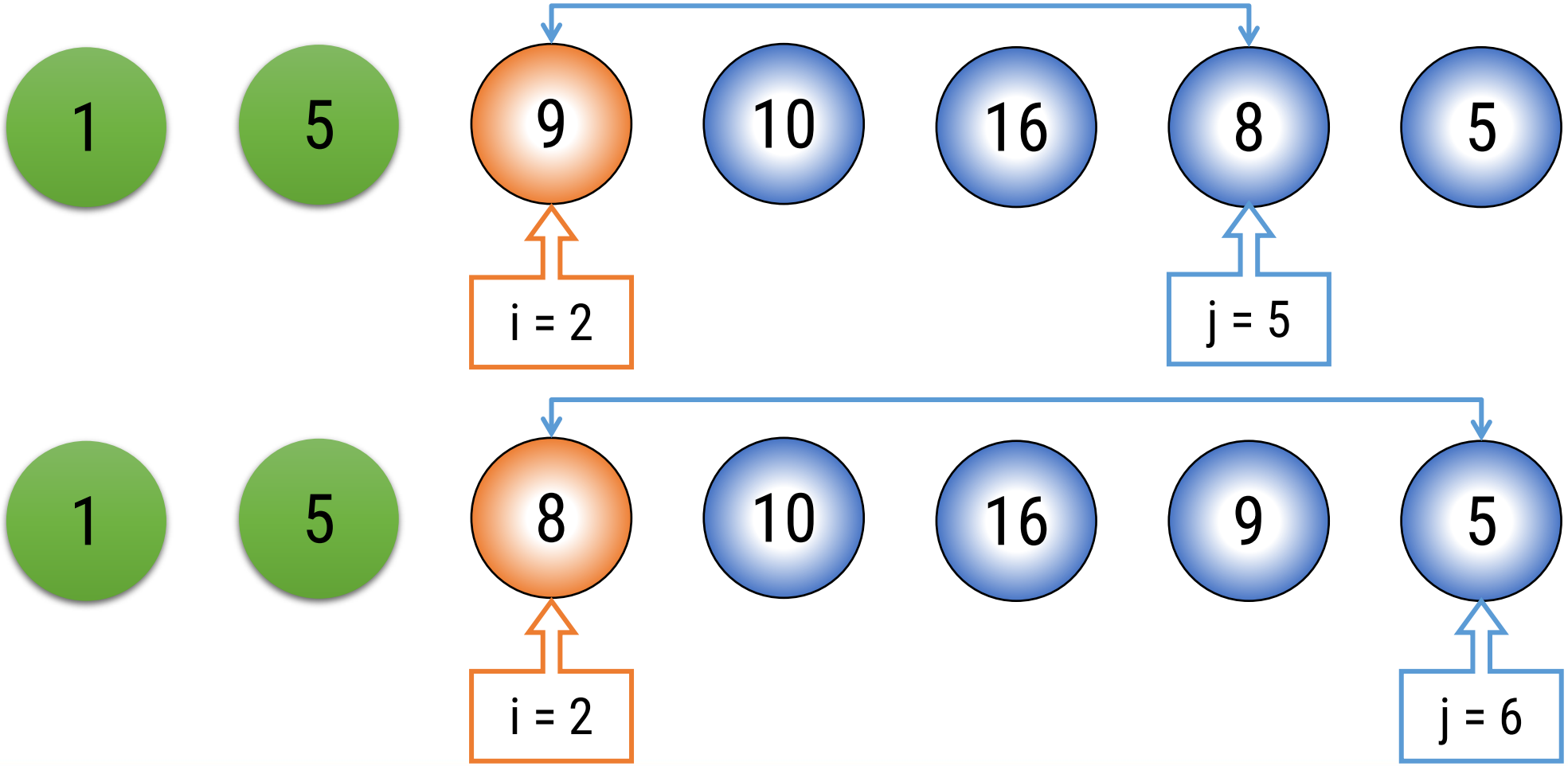
1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



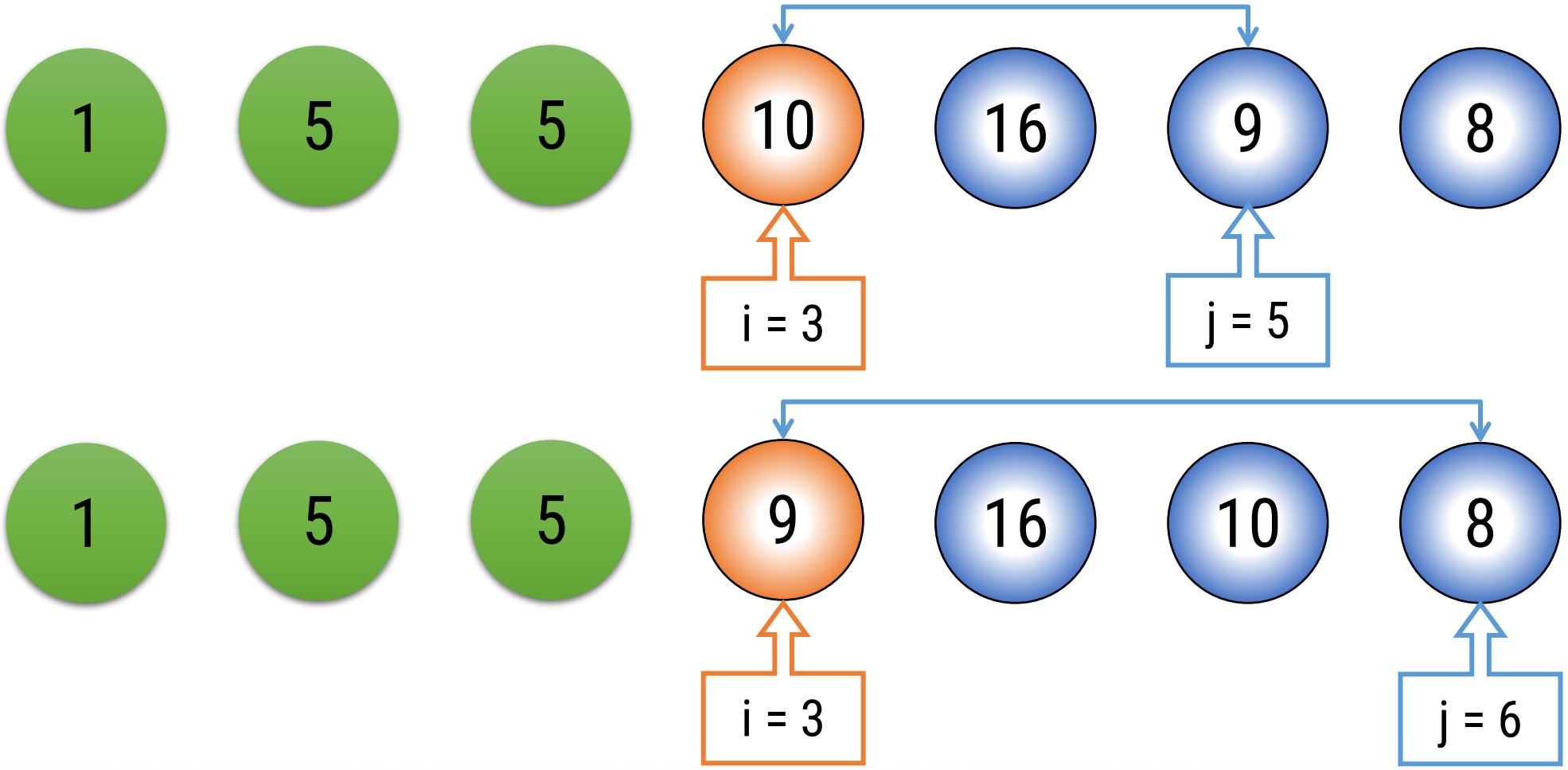
1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



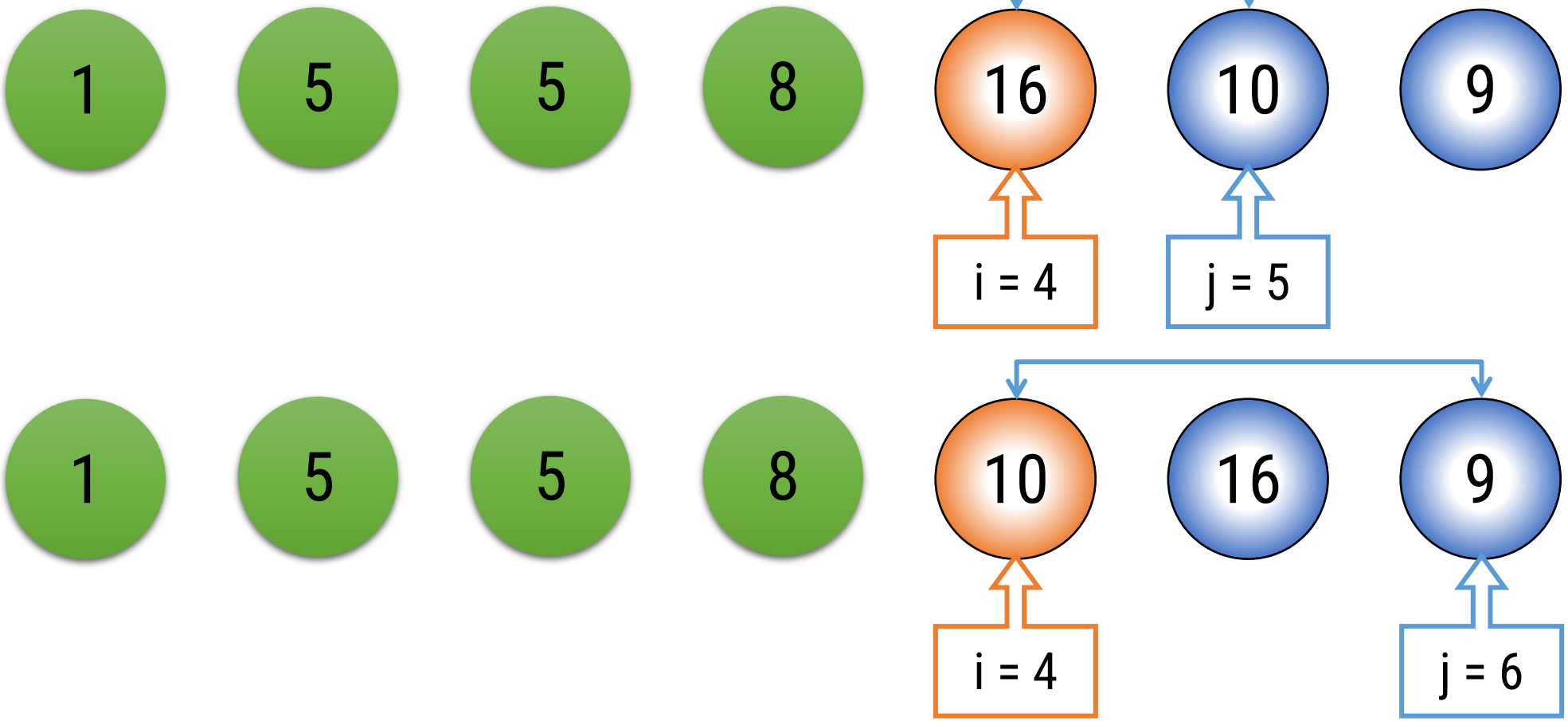
1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



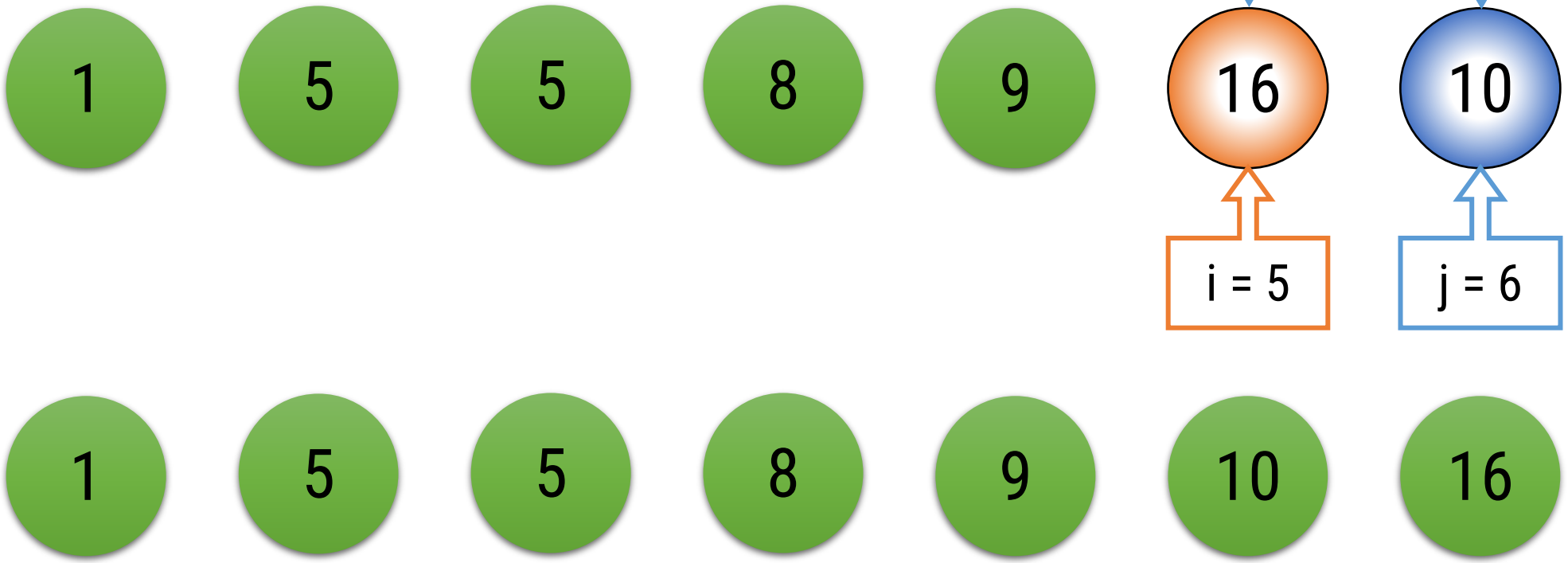
1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



1. Đổi chỗ trực tiếp – Interchange Sort

➢ Minh họa thuật toán:



1. Đổi chỗ trực tiếp – Interchange Sort

➢ Các bước tiến hành:

- Bước 1: i = 0; *// bắt đầu từ đầu dãy*
- Bước 2: j = i+1; *//tìm các nghịch thế với a[i]*
- Bước 3:

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

- Bước 4: i = i+1;

  Nếu i < N-1: Lặp lại Bước 2.

  Ngược lại: Dừng.

1. Đổi chỗ trực tiếp – Interchange Sort

➢ Thuật toán:

```
1   void InterchangeSort(int a[], int n)
2   {
3       for (int i = 0; i < n-1; i++)
4           for (int j = i + 1; j < n; j++)
5               if (a[i] > a[j]) swap(a[i], a[j]);
6   }
```

1. Đổi chỗ trực tiếp – Interchange Sort

- Độ phức tạp thuật toán: $O(n^2)$

2. Chọn trực tiếp – Selection Sort

➢ **Ý tưởng:**

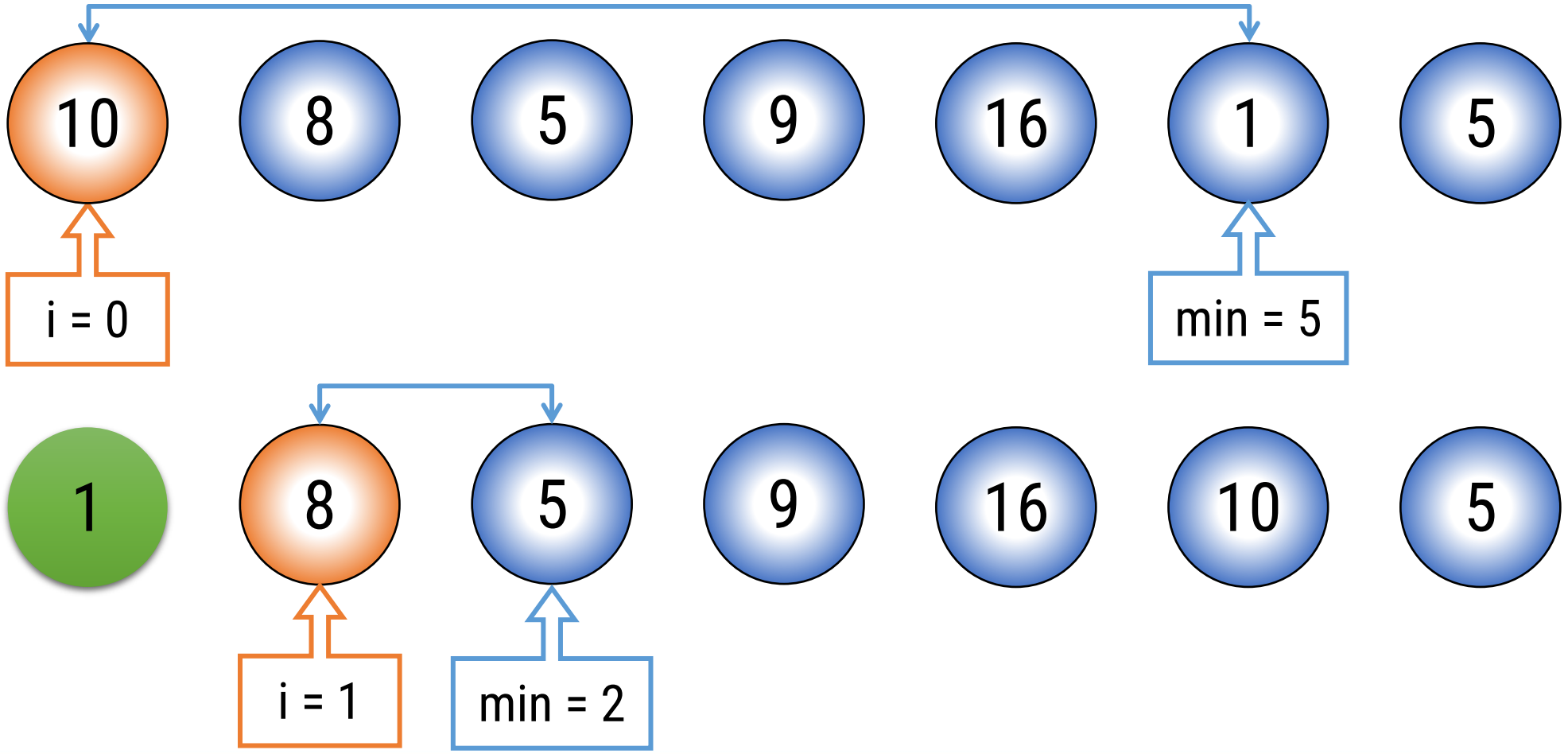
- Chọn phần tử nhỏ nhất trong N phần tử trong dãy ban đầu
- Đưa phần tử này về vị trí đầu dãy
- Xem dãy chỉ còn N-1 phần tử của dãy ban đầu
  - Bắt đầu từ vị trí thứ 2
  - Lặp lại quá trình trên cho đến khi dãy chỉ còn 1 phần tử.

➢ **Ví dụ:**

Cho dãy số a = {10,8,5,9,16,1,5}. Hãy sắp xếp dãy a theo tự tăng dần.

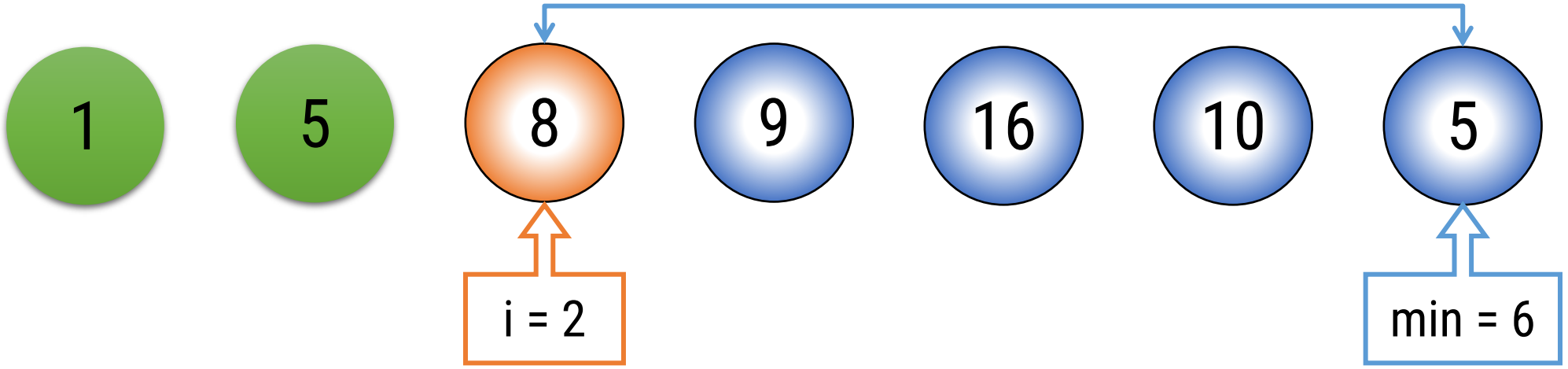
2. Chọn trực tiếp – Selection Sort

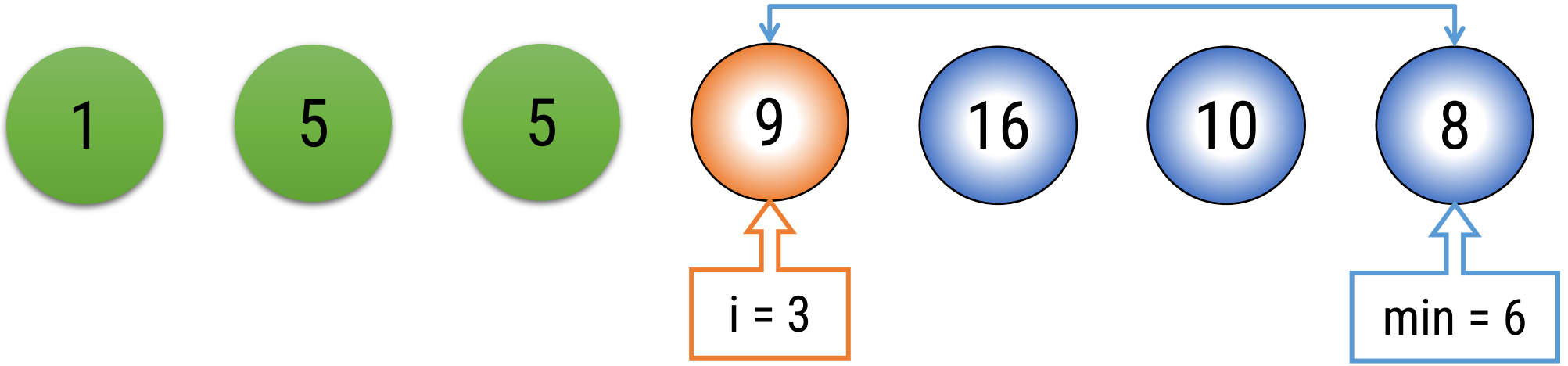
➢ Minh họa thuật toán:



2. Chọn trực tiếp – Selection Sort

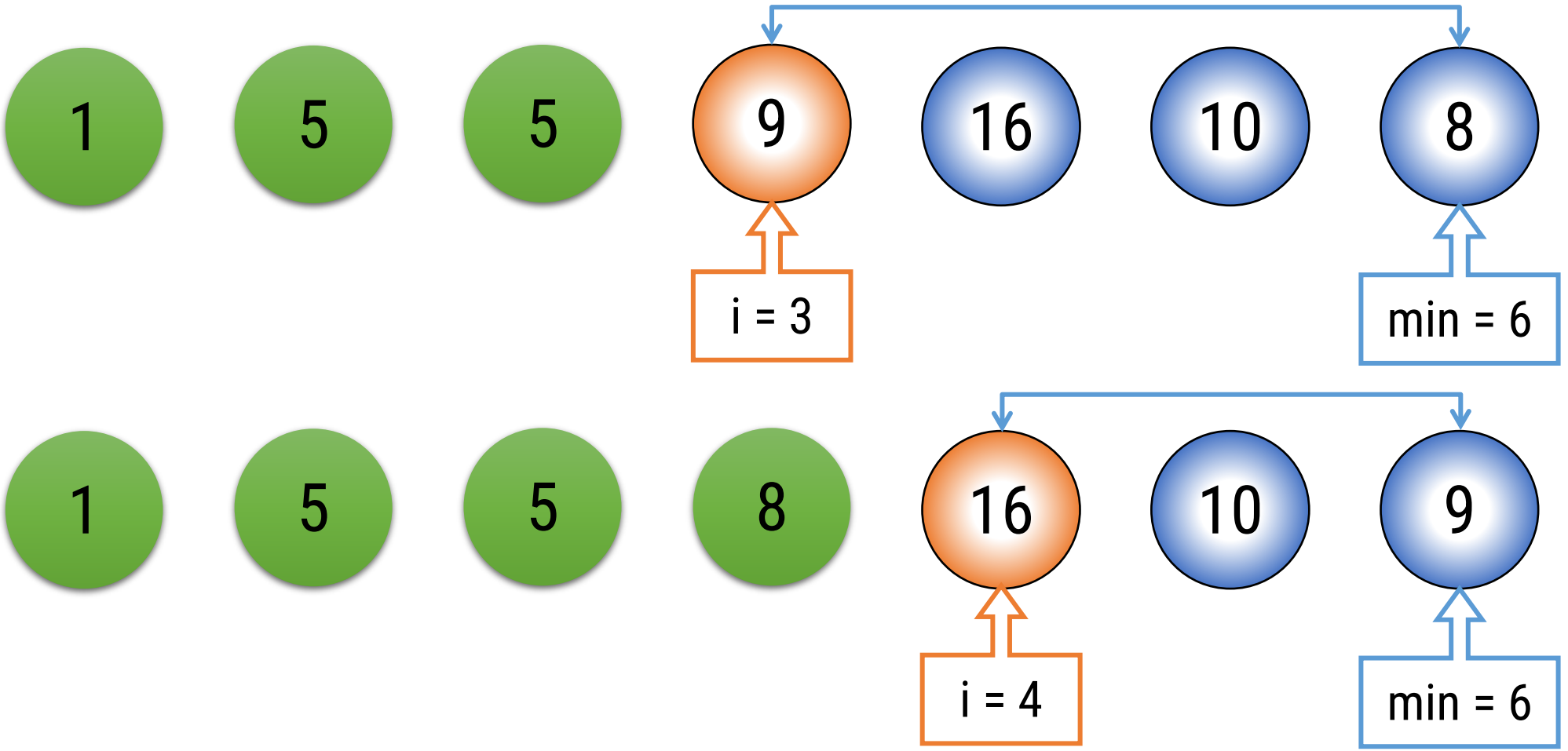
➢ Minh họa thuật toán:





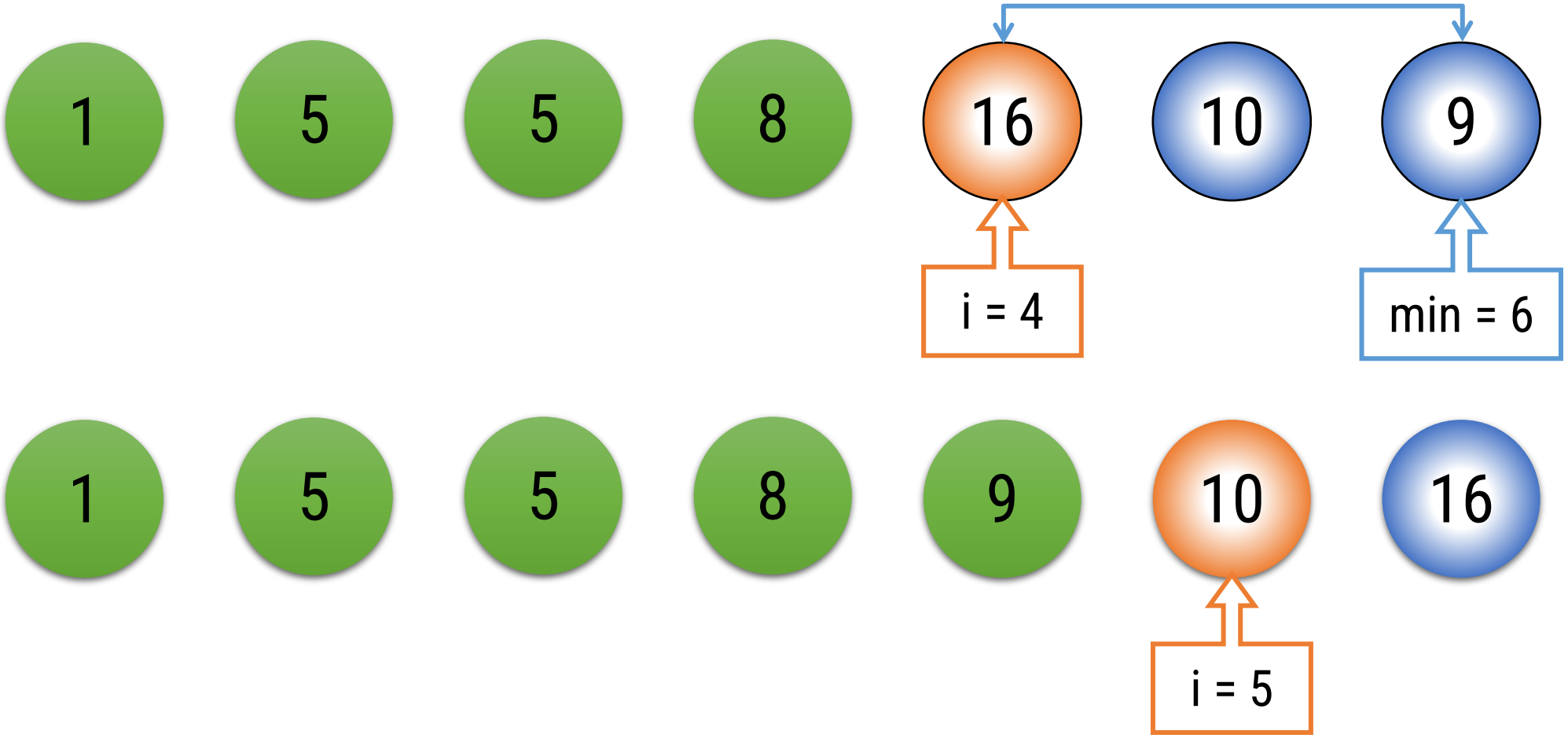
2. Chọn trực tiếp – Selection Sort

➢ Minh họa thuật toán:



2. Chọn trực tiếp – Selection Sort

➢ Minh họa thuật toán:



2. Chọn trực tiếp – Selection Sort

➢ Minh họa thuật toán:

2. Chọn trực tiếp – Selection Sort

➢ Các bước tiến hành:

- Bước 1: i = 0;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy 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.

2. Chọn trực tiếp – Selection Sort

➢ Thuật toán:

```
1   void SelectionSort(int a[], int n)
2   {
3       for (int i = 0; i < n-1; i++)
4       {
5           int min = i;
6           for (int j = i + 1; j < n; j++)
7               if (a[min] > a[j]) min = j;
8               //Lưu vị trí phần tử nhỏ nhất
9
10          swap(a[min], a[i]);
11      }
12  }
```

2. Chọn trực tiếp – Selection Sort

- Độ phức tạp thuật toán: $O(n^2)$

3. Nổi bọt – Bubble Sort

➢Ý tưởng:

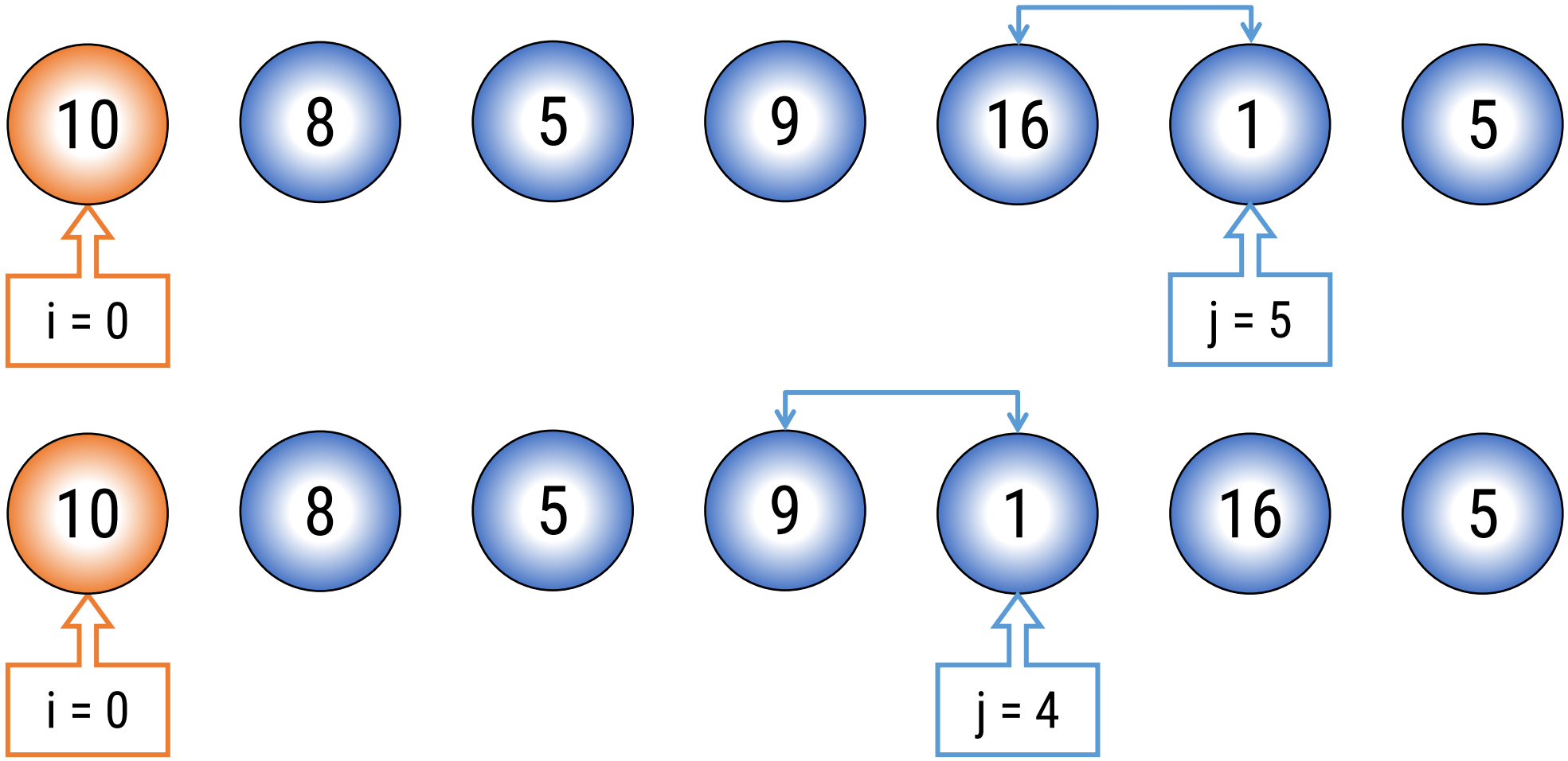
- Xuất phát từ cuối dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ hơn trong cặp phần tử đó về vị trí đứng đầu dãy, sau đó sẽ không xét đến nó ở bước tiếp theo.
- Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

➢Ví dụ:

Cho dãy số a = {10,8,5,9,16,1,5}. Hãy sắp xếp dãy a theo tự tăng dần.

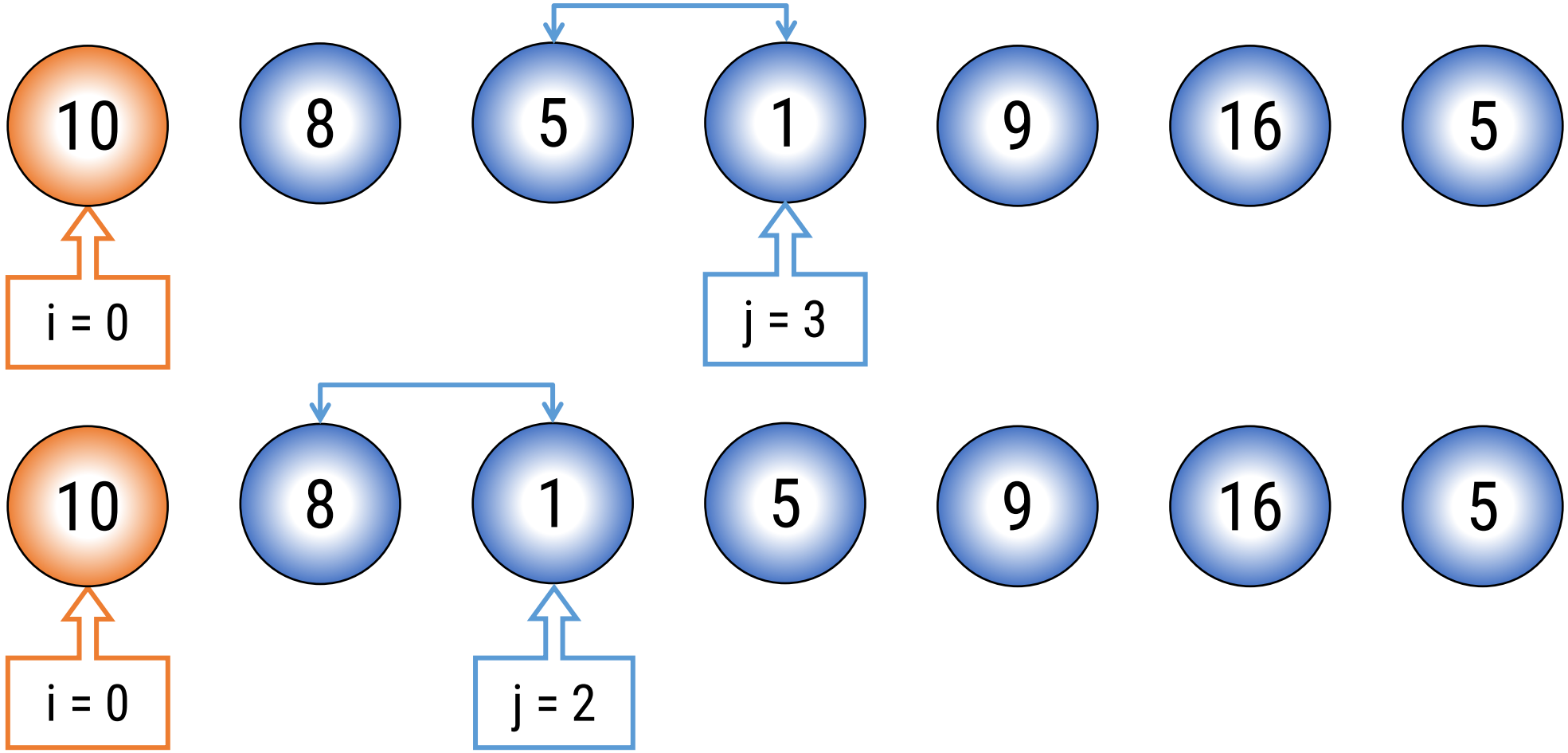
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



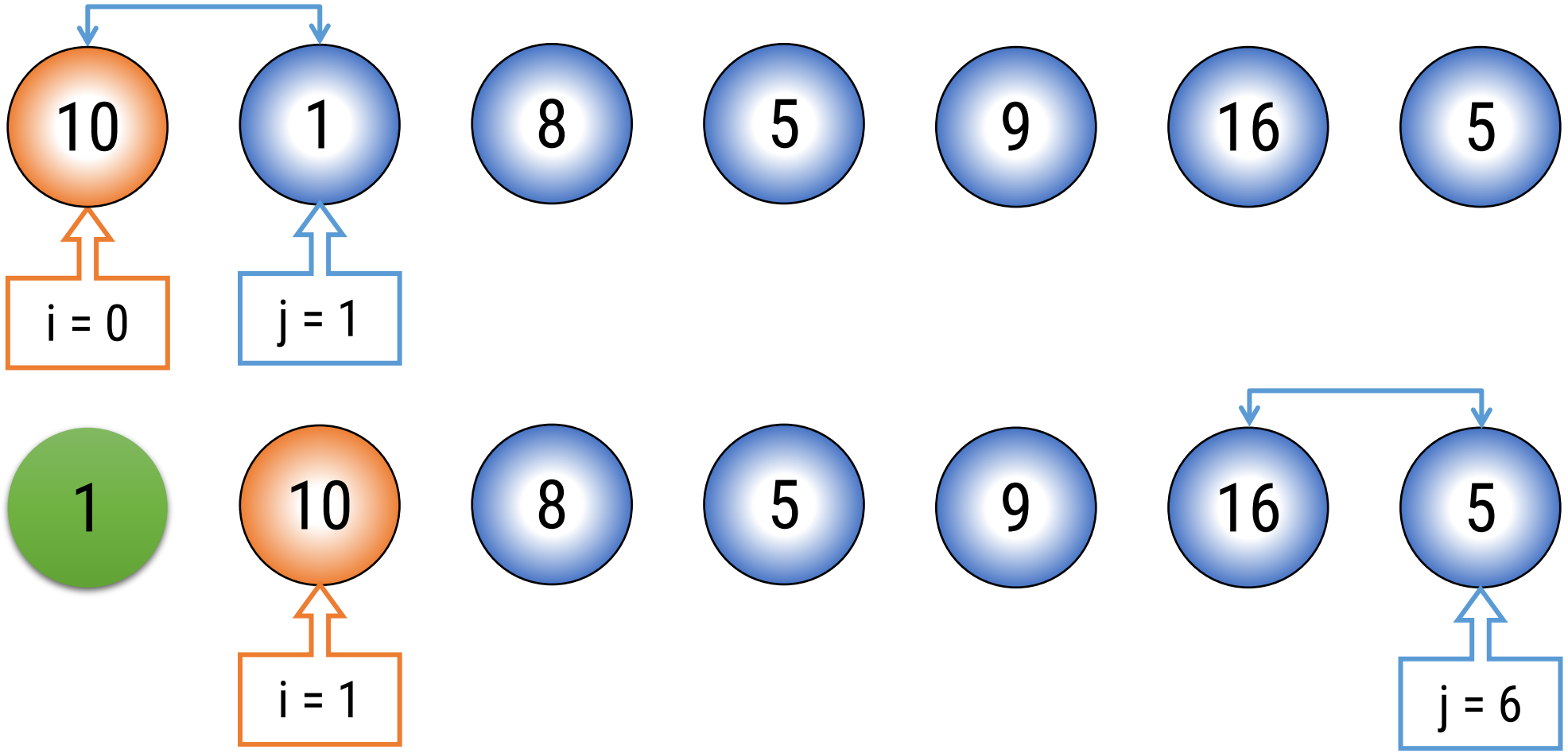
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



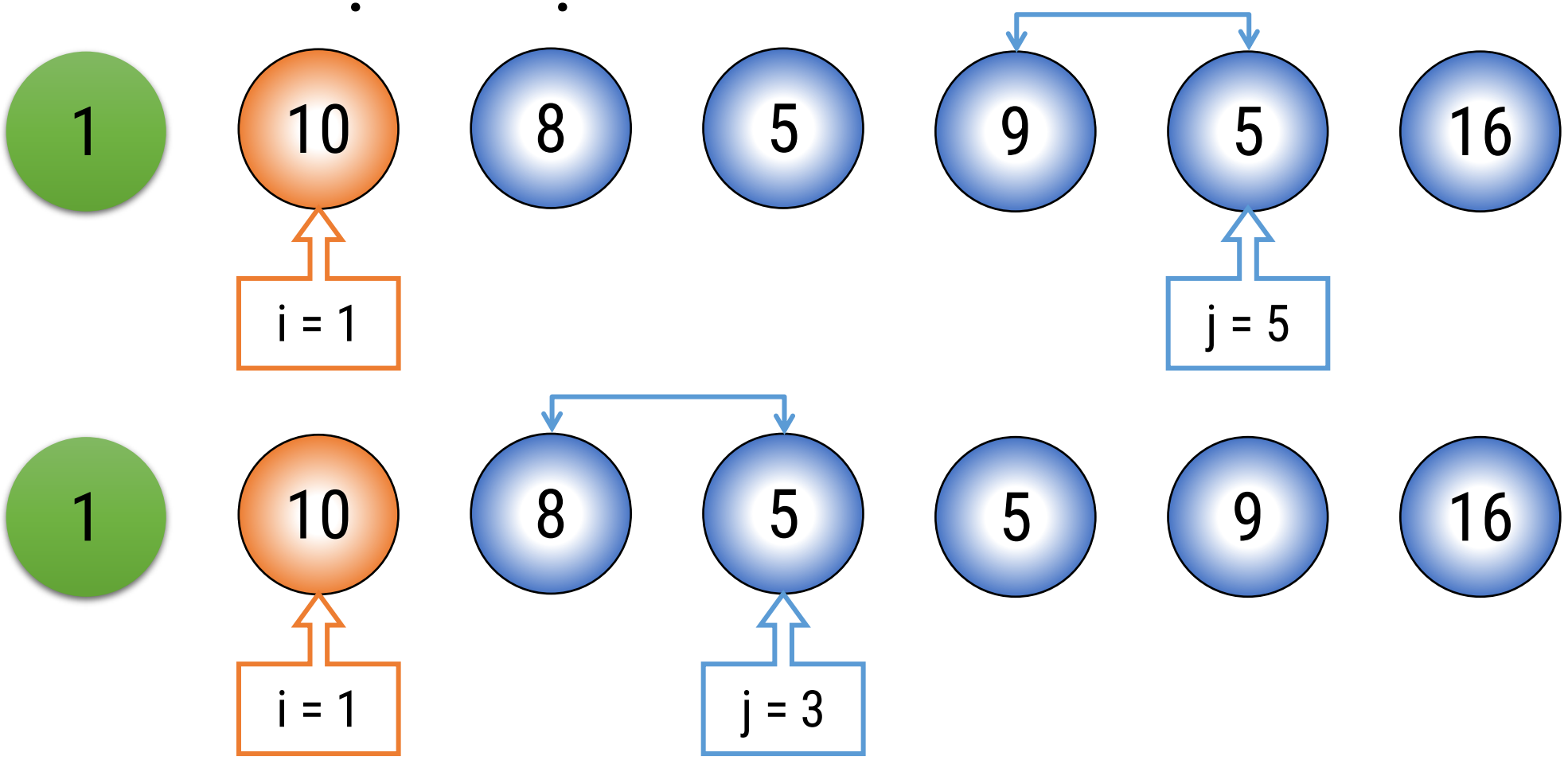
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



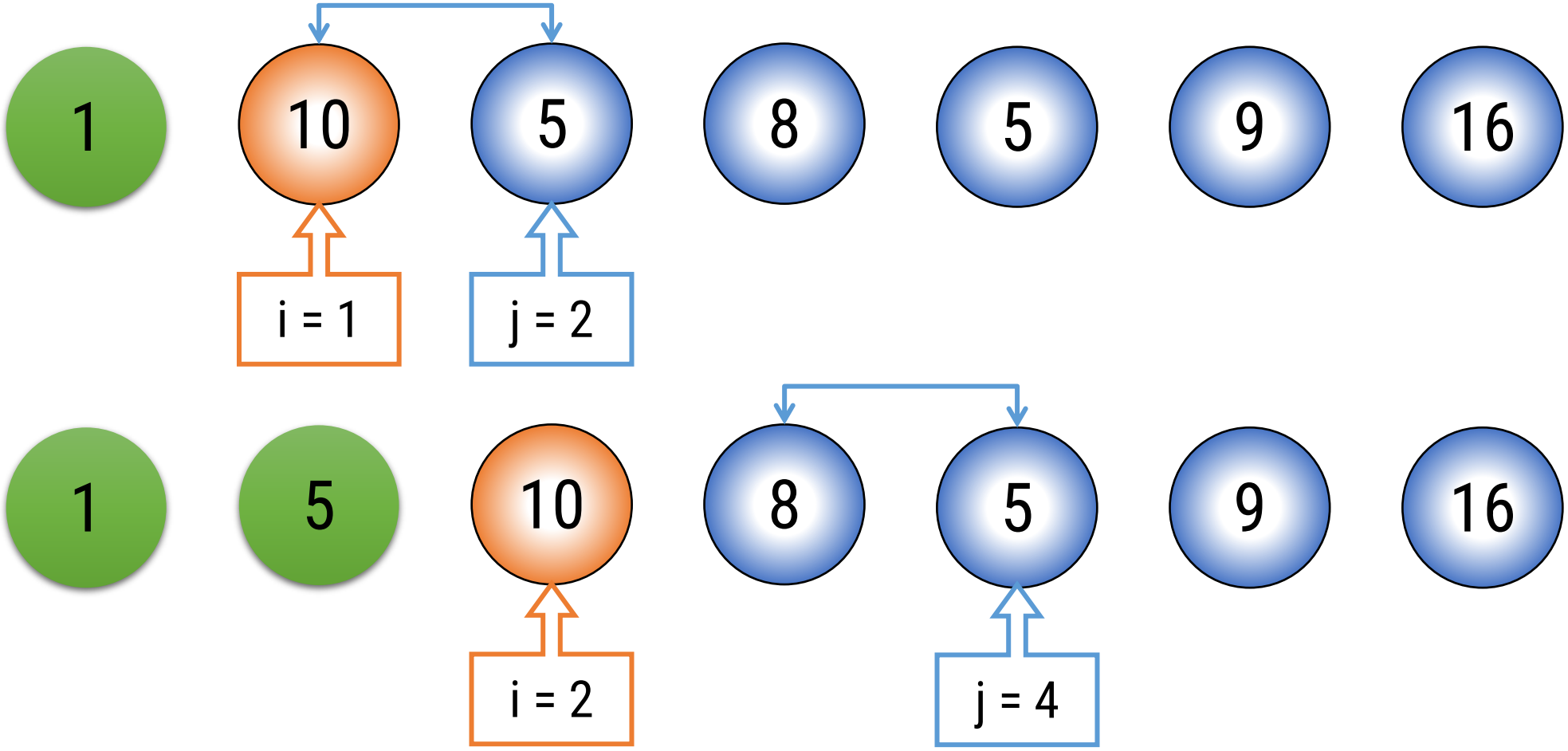
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



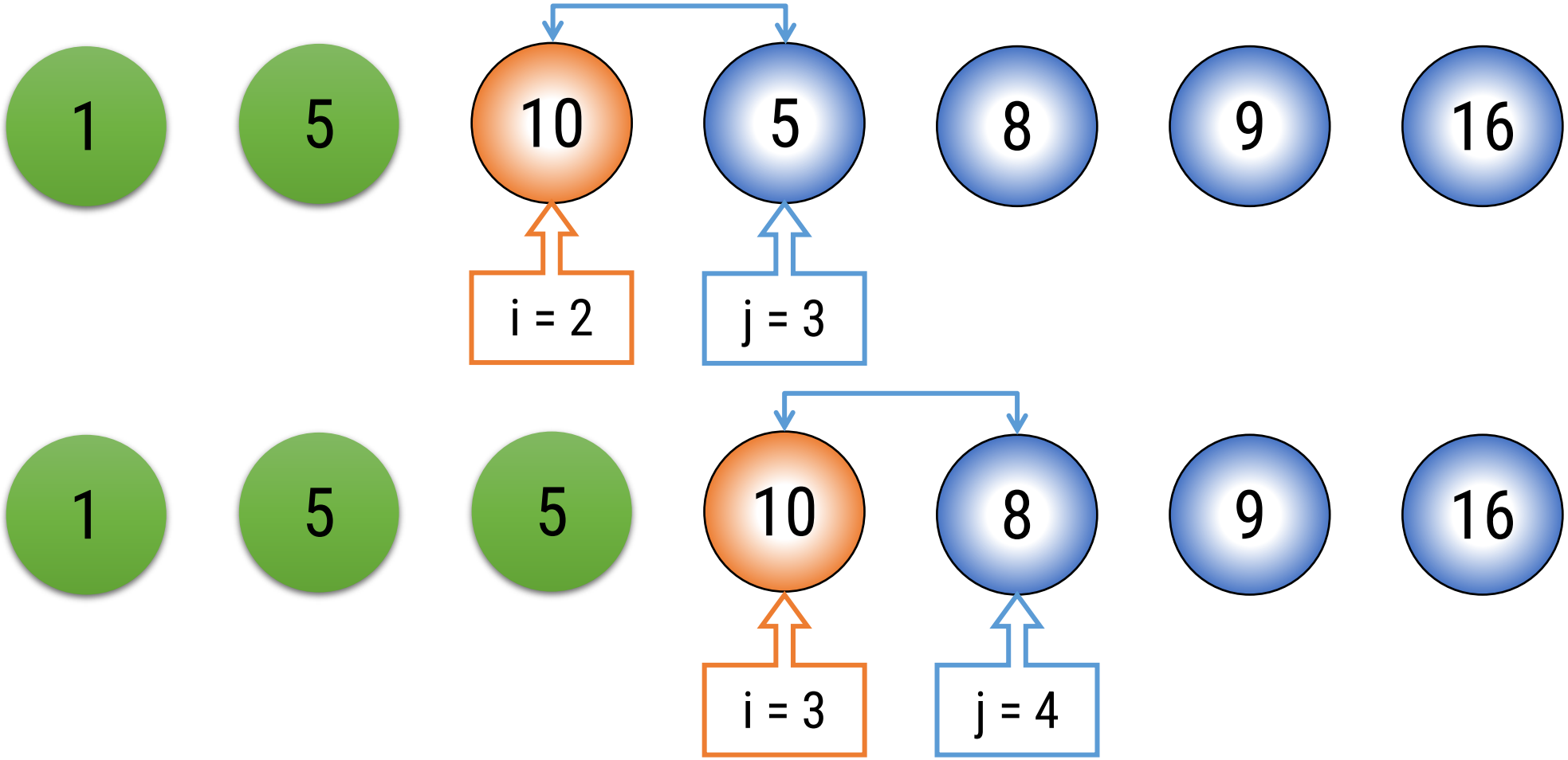
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



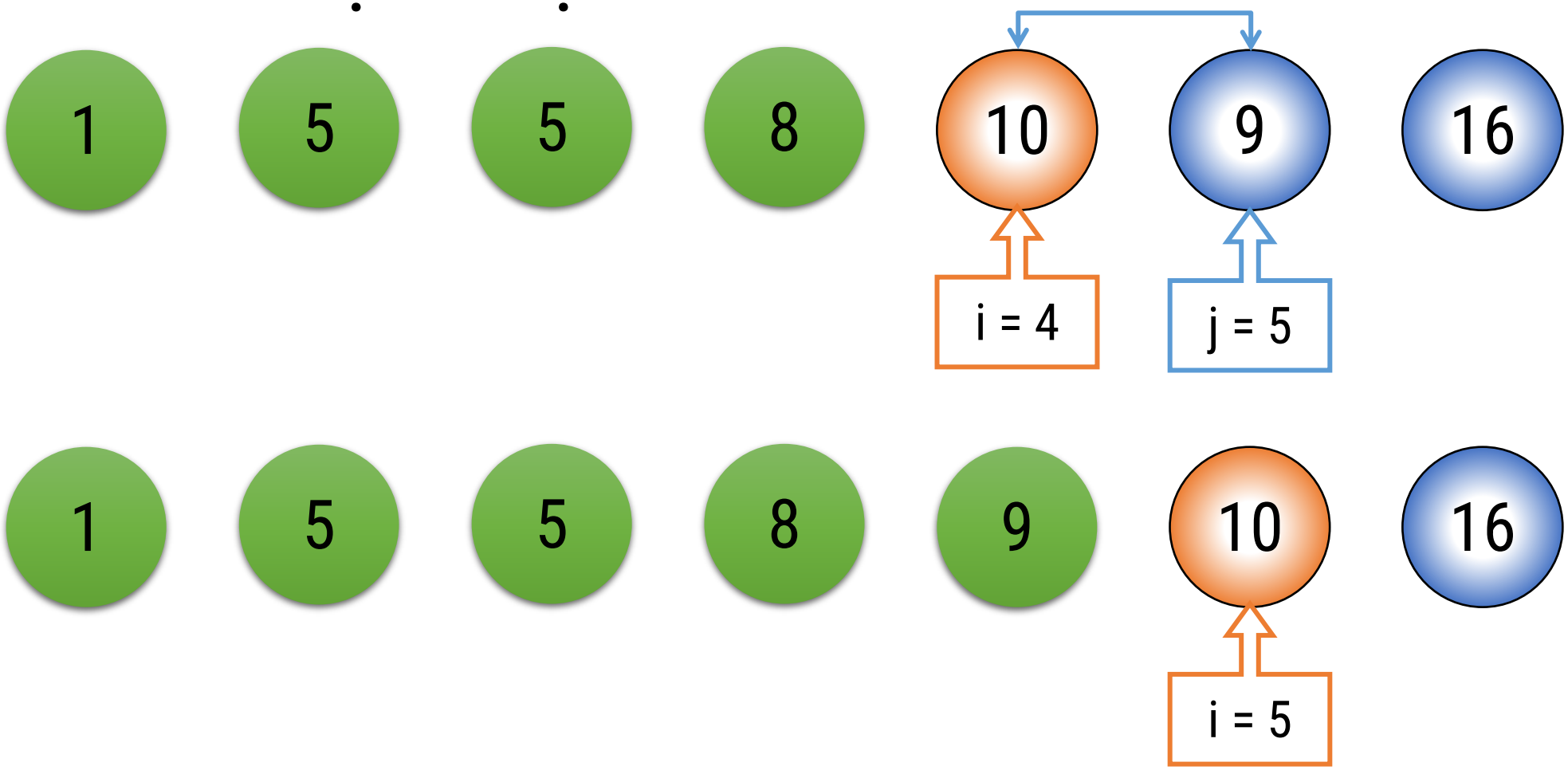
3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:



3. Nổi bọt – Bubble Sort

➢ Minh họa thuật toán:

3. Nổi bọt – Bubble Sort

➢ Các bước tiến hành:

- Bước 1 : i = 0; // lần xử lý đầu tiên
- Bước 2 : j = N-1; //Duyệt từ cuối dãy ngược về vị trí i

  Trong khi (j > i) thực hiện:

  Nếu a[j] < a[j-1]

  Doicho(a[j], a[j-1]);

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

  Nếu i >= N-1: Hết dãy. Dừng

  Ngược lại: Lặp lại Bước 2.

3. Nổi bọt – Bubble Sort

➢ Thuật toán:

```
1   void BubbleSort(int a[], int n)
2   {
3       for (int i = 0; i < n-1; i++)
4           for (int j = n-1; j >= i+1; j--)
5               if (a[j] < a[j-1]) swap(a[j], a[j-1]);
6               //Đổi chỗ các cặp nghịch thế
7   }
```

3. Nổi bọt – Bubble Sort

- Độ phức tạp thuật toán: $O(n^2)$

4. Chèn trực tiếp – Insertion Sort

➢ **Ý tưởng:**

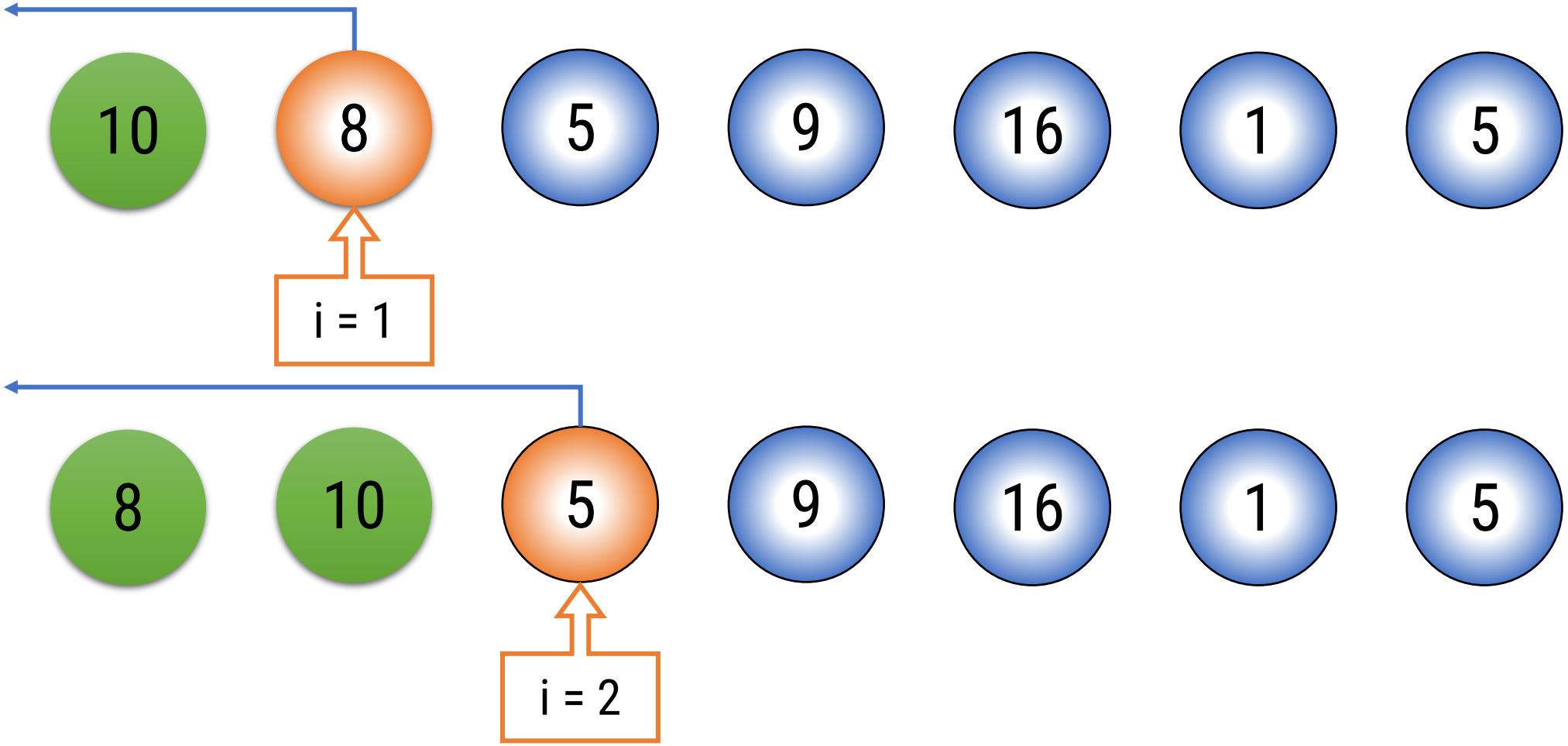
- Giả sử có một dãy $a_0 , a_1 , \ldots , a_{n-1}$ trong đó i phần tử đầu tiên $a_0 , a_1 , \ldots , a_{i-1}$ đã có thứ tự.
- Tìm cách chèn phần tử a_i vào vị trí thích hợp của đoạn đã được sắp để có dãy mới $a_0 , a_1, \ldots , a_i$ trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử a_{k-1}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

➢ **Ví dụ:**

Cho dãy số a = {10,8,5,9,16,1,5}. Hãy sắp xếp dãy a theo tự tăng dần.

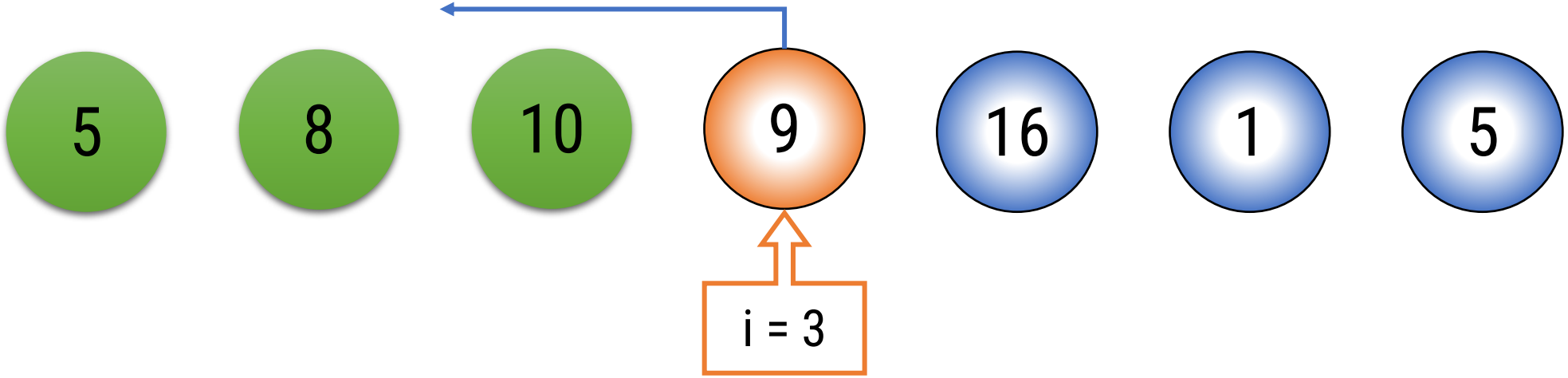
4. Chèn trực tiếp – Insertion Sort

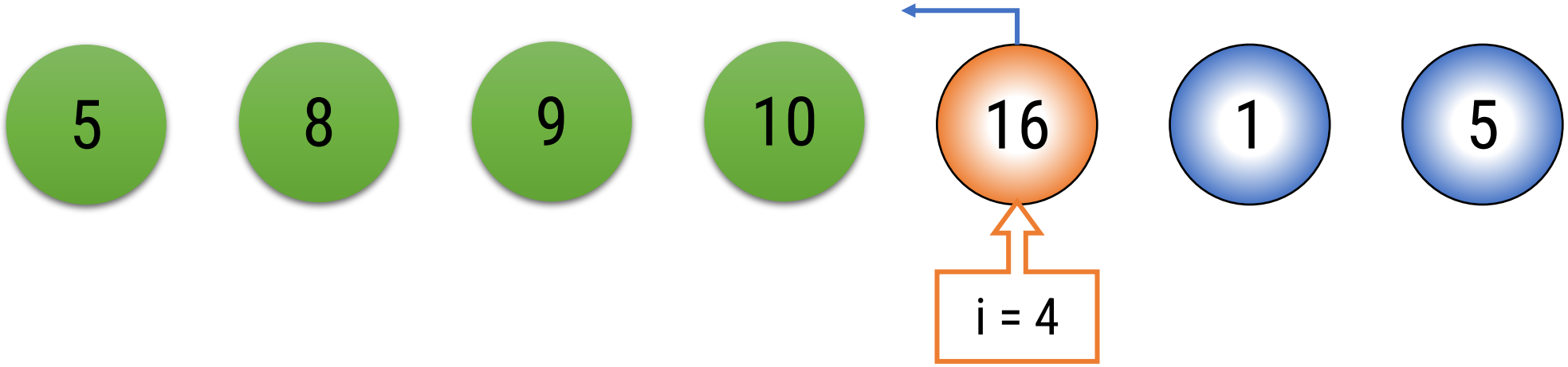
➢ Minh họa thuật toán:



4. Chèn trực tiếp – Insertion Sort

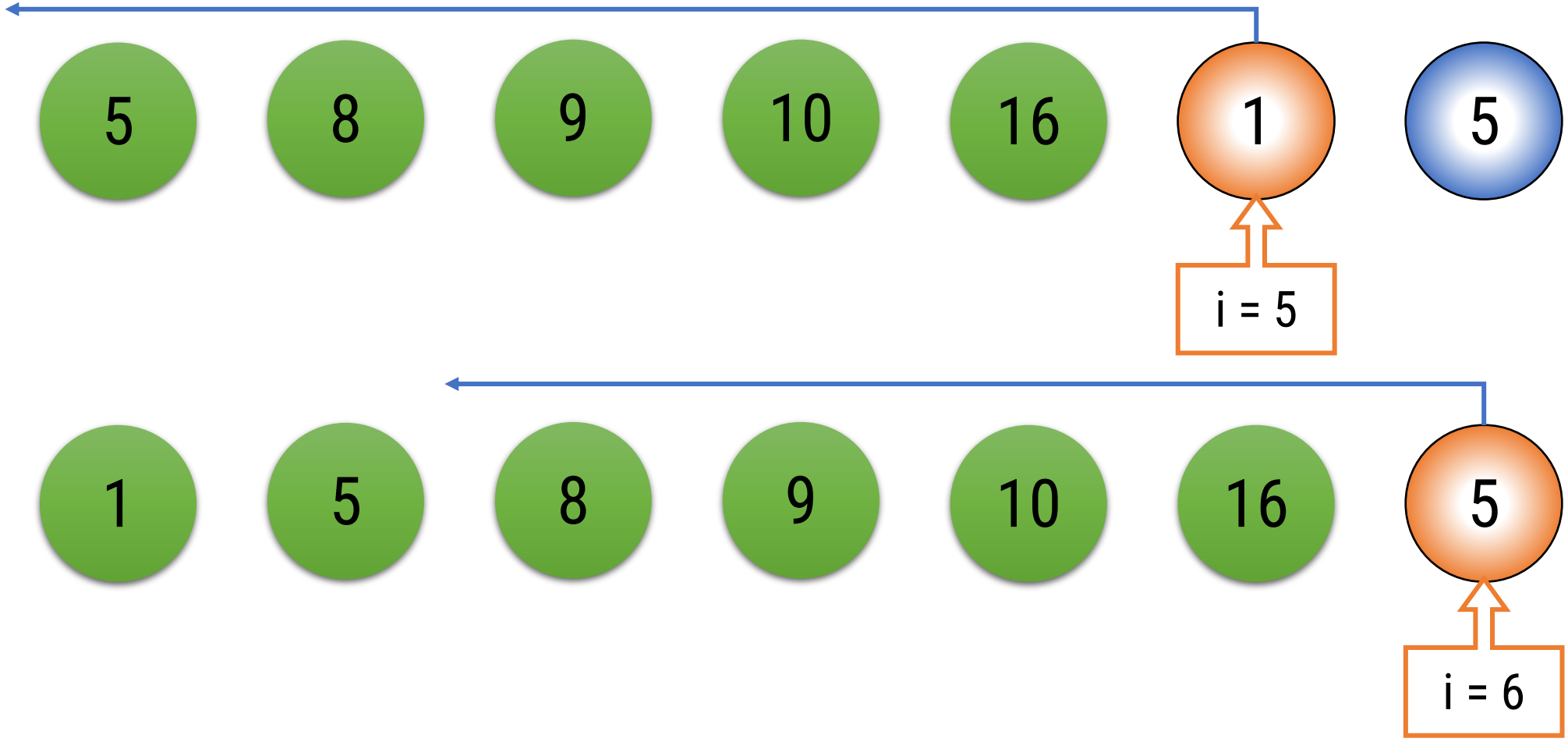
➢ Minh họa thuật toán:





4. Chèn trực tiếp – Insertion Sort

➢ Minh họa thuật toán:



4. Chèn trực tiếp – Insertion Sort

- Minh họa thuật toán:

4. Chèn trực tiếp – Insertion Sort

➢ Các bước tiến hành:

- Bước 1: i = 1; *//giả sử có đoạn a[1] đã được sắp*
- Bước 2: x = a[i];

  Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào
- Bước 3: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]
- Bước 4: a[pos] = x; *//có đoạn a[1]..a[i] đã được sắp*
- Bước 5: i = i+1;

  Nếu i < n : Lặp lại Bước 2

  Ngược lại : Dừng

4. Chèn trực tiếp – Insertion Sort

➢ Thuật toán:

```
1  void InsertionSort(int a[], int n)
2  {
3      for (int i = 1; i < n; i++)
4      {
5          //Lưu giá trị của a[i] tránh việc bị ghi đè dữ liệu
6          int x = a[i];
7
8          //vị trí chèn
9          int j = i-1;
10
11         //Tìm vị trí chèn và dời chỗ các phần tử
12         while (j >= 0 && a[j] > x)
13         {
14             a[j+1] = a[j];
15             j--;
16         }
17         a[j+1] = x;
18     }
19 }
```

4. Chèn trực tiếp – Insertion Sort

- Độ phức tạp thuật toán: $O(n^2)$

5. Chèn nhị phân – Binary Insertion Sort

➢ **Ý tưởng:**

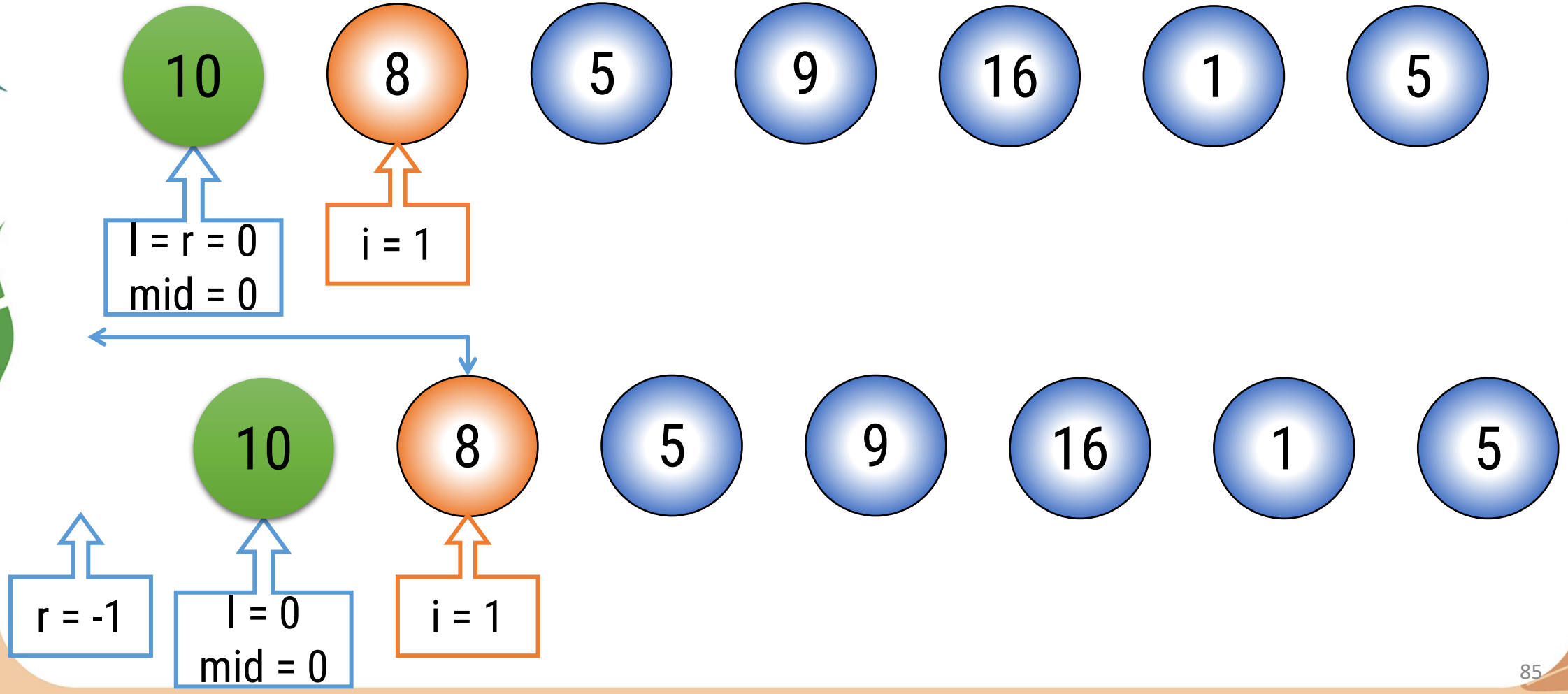
Tương tự chèn trực tiếp nhưng áp dụng công thức tìm kiếm nhị phân.

➢ **Ví dụ:**

Cho dãy số a = {10,8,5,9,16,1,5}. Hãy sắp xếp dãy a theo tự tăng dần.

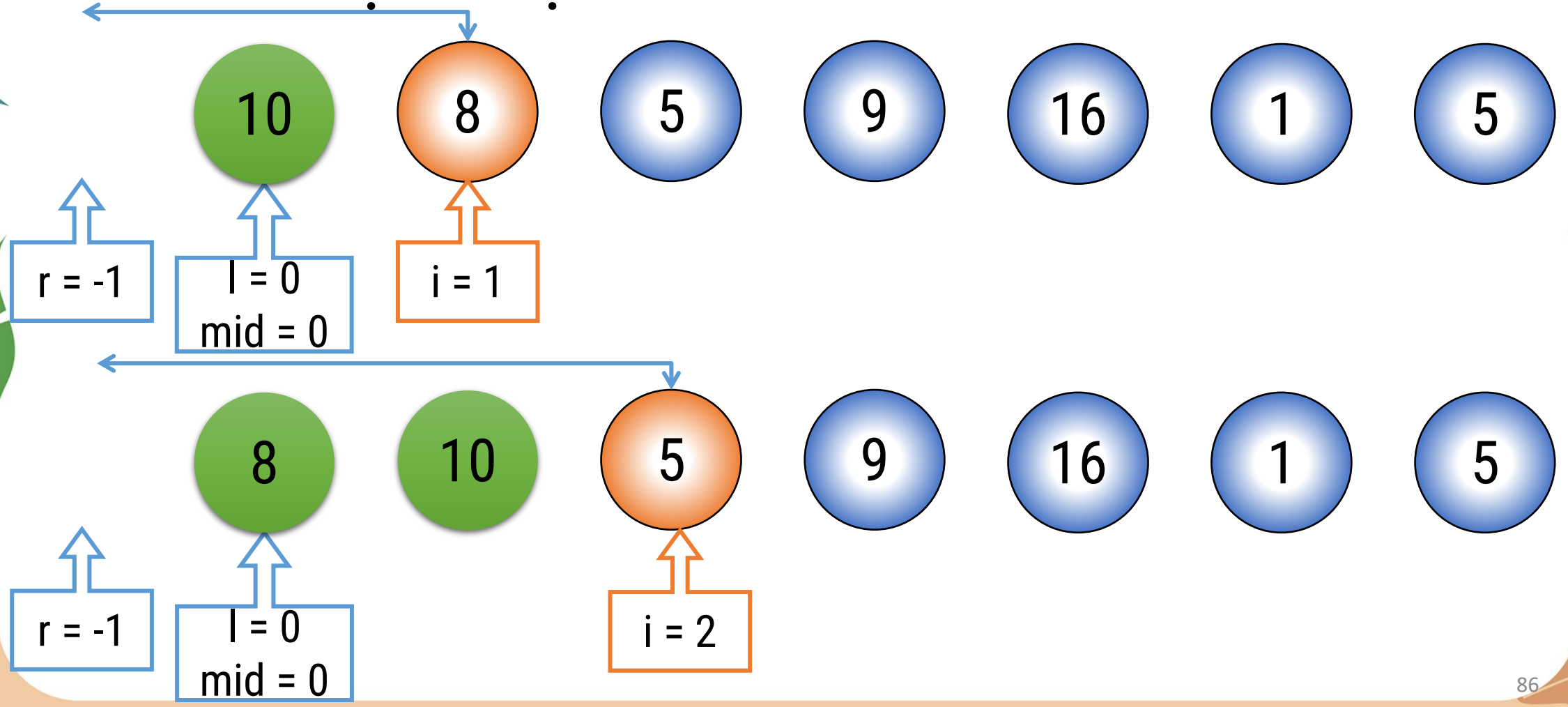
5. Chèn nhị phân – Binary Insertion Sort

➢ Minh họa thuật toán:



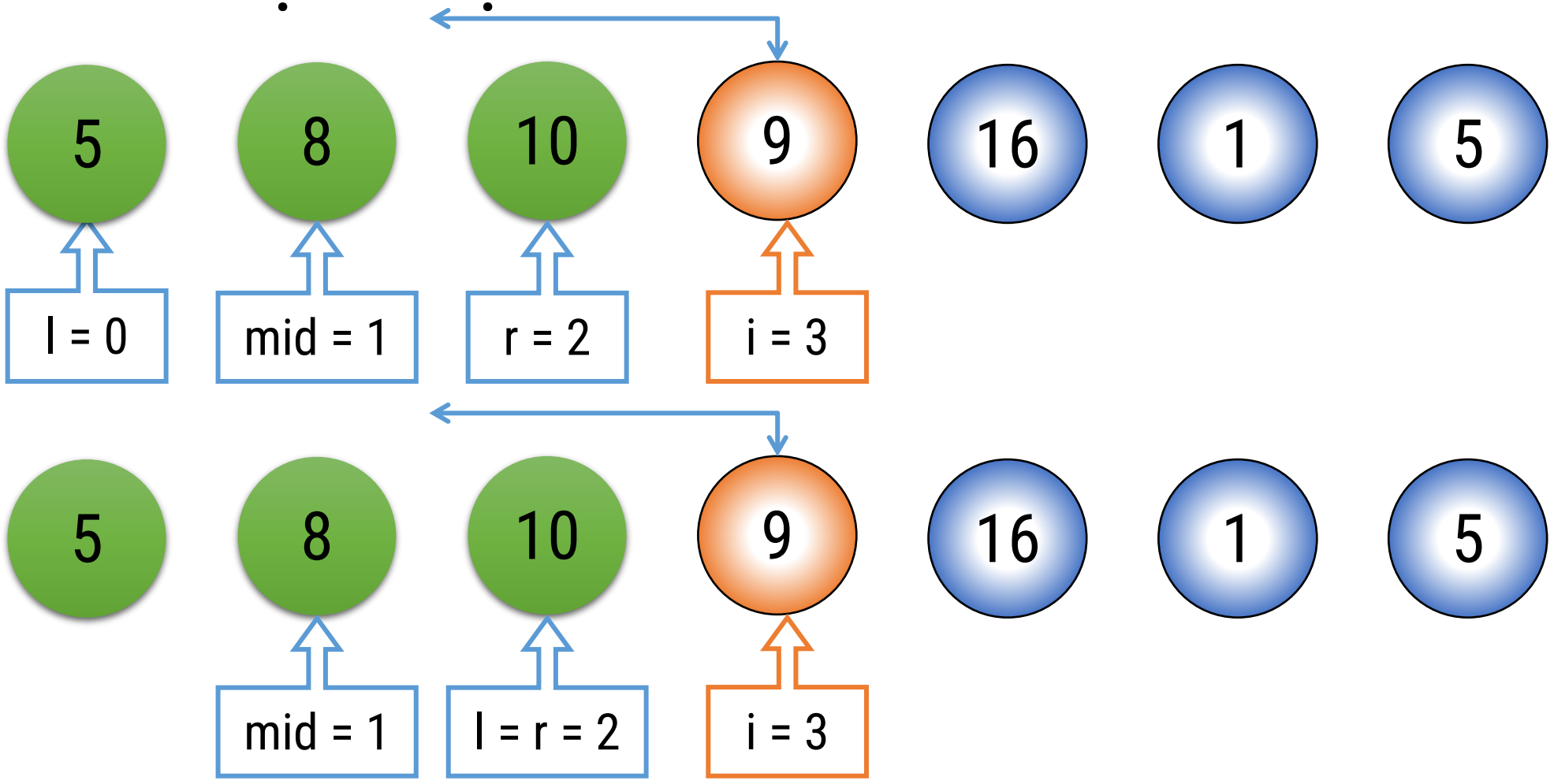
5. Chèn nhị phân – Binary Insertion Sort

➢ Minh họa thuật toán:



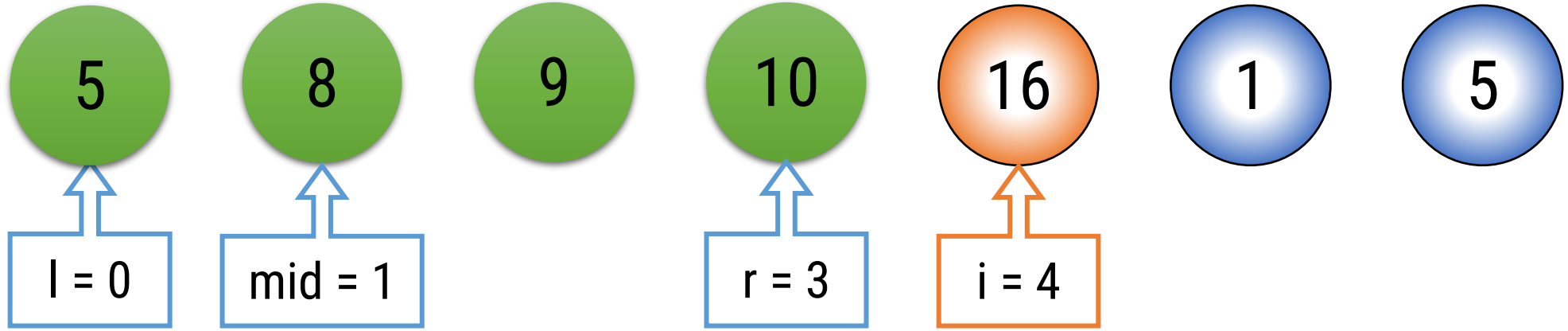
5. Chèn nhị phân – Binary Insertion Sort

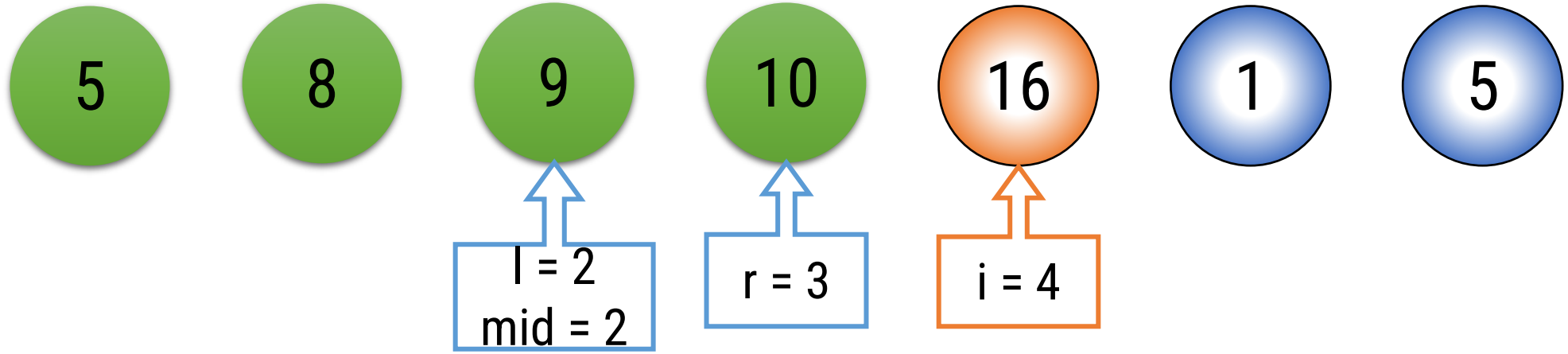
➢ Minh họa thuật toán:



5. Chèn nhị phân – Binary Insertion Sort

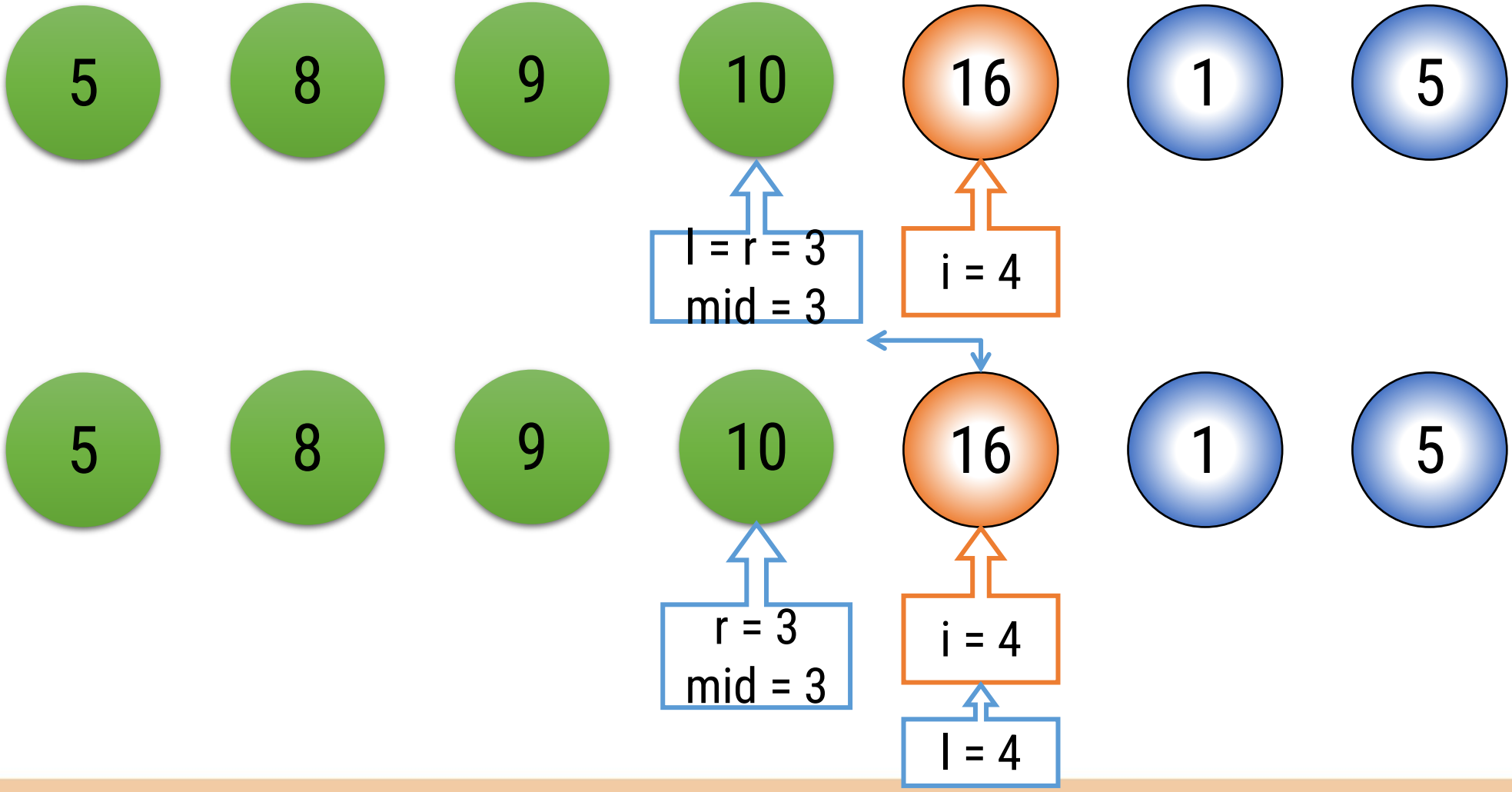
➢ Minh họa thuật toán:





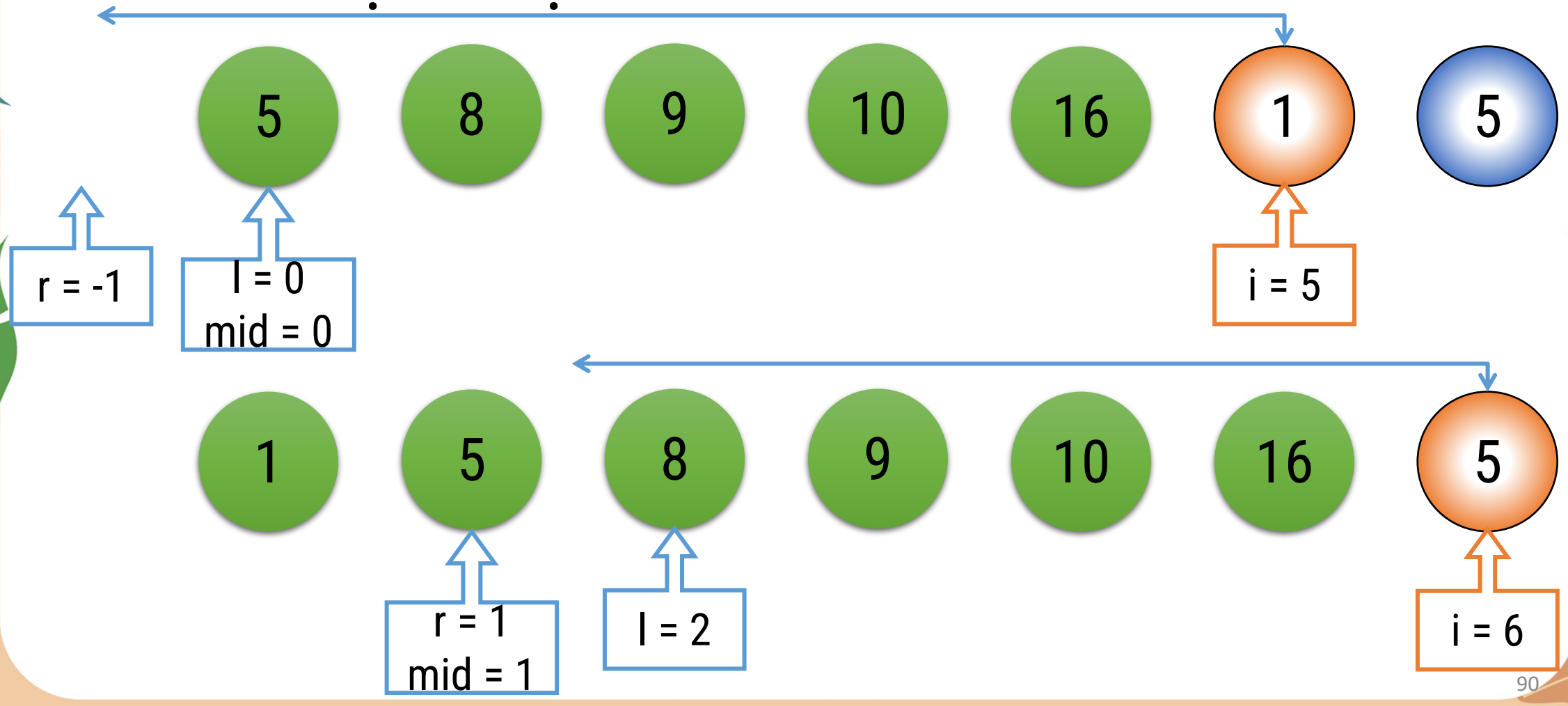
5. Chèn nhị phân – Binary Insertion Sort

➢ Minh họa thuật toán:



5. Chèn nhị phân – Binary Insertion Sort

➢ Minh họa thuật toán:



5. Chèn nhị phân – Binary Insertion Sort

- Minh họa thuật toán:

5. Chèn nhị phân – Binary Insertion Sort

➢ Các bước tiến hành:

- Bước 1: i = 1; *//giả sử có đoạn a[1] đã được sắp*
- Bước 2: x = a[i], left = 0, right = i-1;

  Trong khi (left <= right) thực hiện*//Tìm kiếm nhị phân*

  mid = (left + right) / 2;

  Nếu (x < a[mid]) thì right = mid – 1;

  Ngược lại: left = mid + 1;
- Bước 3: Dời chỗ các phần tử từ a[left] đến a[i-1] sang phải 1 vị trí để dành chổ cho a[i]
- Bước 4: a[l] = 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

5. Chèn nhị phân – Binary Insertion Sort

➢ Thuật toán:

```
1   void BInsertionSort(int a[], int n)
2   {
3       for (int i = 1; i < n; i++)
4       {
5           //Lưu giá trị của a[i] tránh việc bị ghi đè dữ liệu
6           int x = a[i];
7           //Tìm vị trí chèn
8           int l = 0, r = i-1;
9           while (l <= r)
10          {
11              int mid = (l + r) / 2;
12              if (x < a[mid]) r = mid - 1;
13              else            l = mid + 1;
14          }
15          //dời chỗ các phần tử và chèn dữ liệu sau khi dời
16          for (int j = i-1; j >= l; j--)
17              a[j+1] = a[j];
18          a[l] = x;
19      }
20  }
```

5. Chèn nhị phân – Binary Insertion Sort

- Độ phức tạp thuật toán: $O(n^2)$

6. Quick Sort

➢ Ý tưởng:

- Phân hoạch dãy thành 3 phần:
  - Phần 1: gồm các phần tử có giá trị bé hơn x.
  - Phần 2: gồm các phần tử có giá trị bằng x.
  - Phần 3: gồm các phần tử có giá trị lớn hơn x.

  (x: một phần tử bất kỳ trong dãy)
- Sau khi phân hoạch, dãy được chia làm 3 đoạn:

k = 1 .. j	k = j+1 .. i-1	k = i .. N
$a_k < x$	$a_k = x$	$a_k > x$

6. Quick Sort

➢ Ý tưởng:

k = 1 .. j	k = j+1 .. i-1	k = i .. N
$a_k < x$	$a_k = x$	$a_k > x$

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

→ Khi đó dãy con đã được sắp

6. Quick Sort

➢ Ý tưởng:

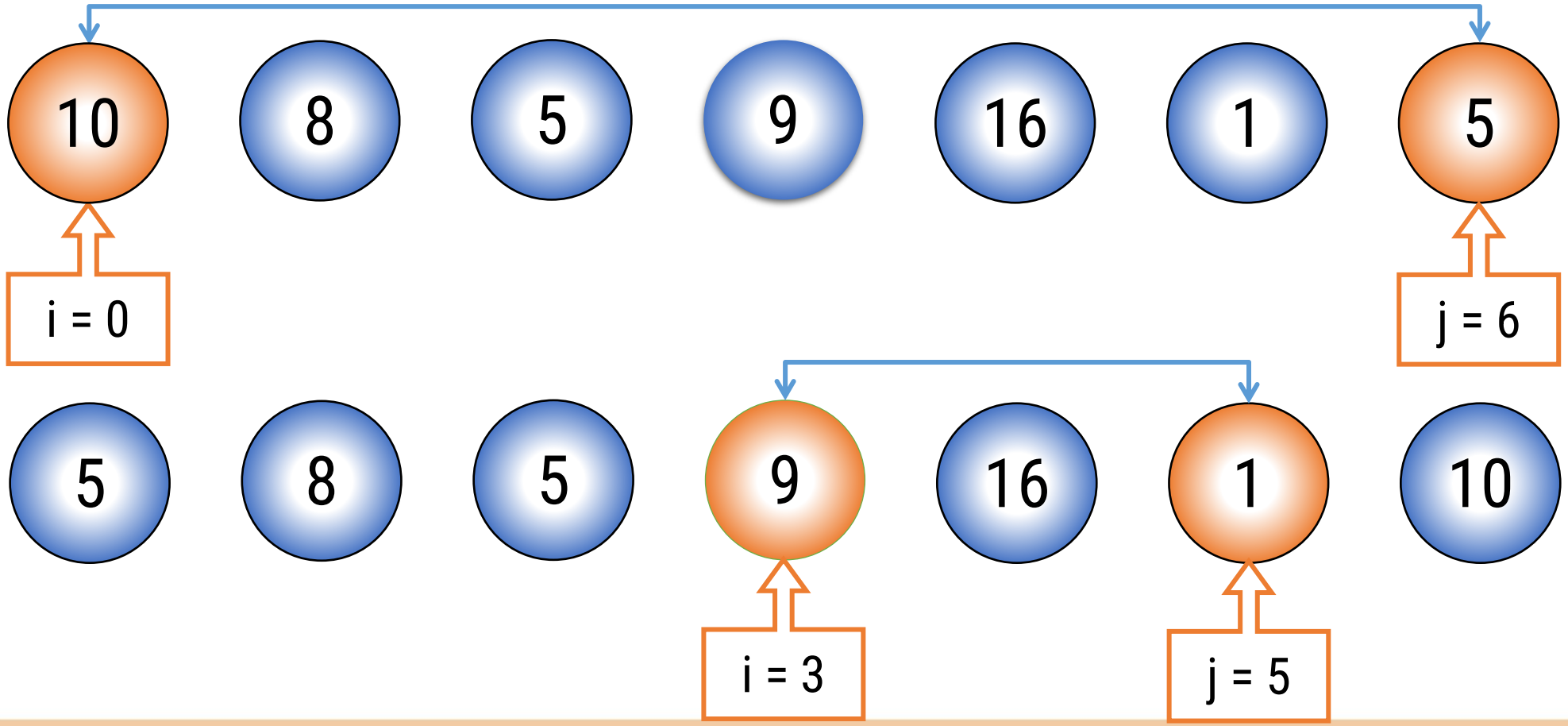
k = 1 .. j	k = j+1 .. i-1	k = i .. N
$a_k < x$	$a_k = x$	$a_k > x$

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 có nhiều hơn 1 phần tử thì dãy chỉ có thứ tự khi các đoạn 1 và 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 vừa trình bày (gọi đệ quy).

6. Quick Sort

➢ Minh họa thuật toán:

- Phân hoạch đoạn l = 0, r = 6:

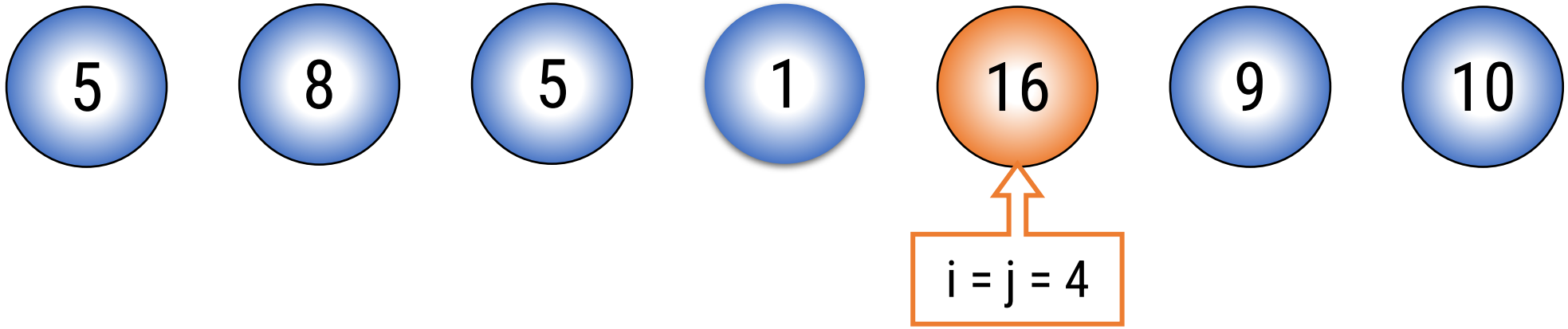


6. Quick Sort

➢ Minh họa thuật toán:

x = 9

- Phân hoạch đoạn l = 0, r = 6:

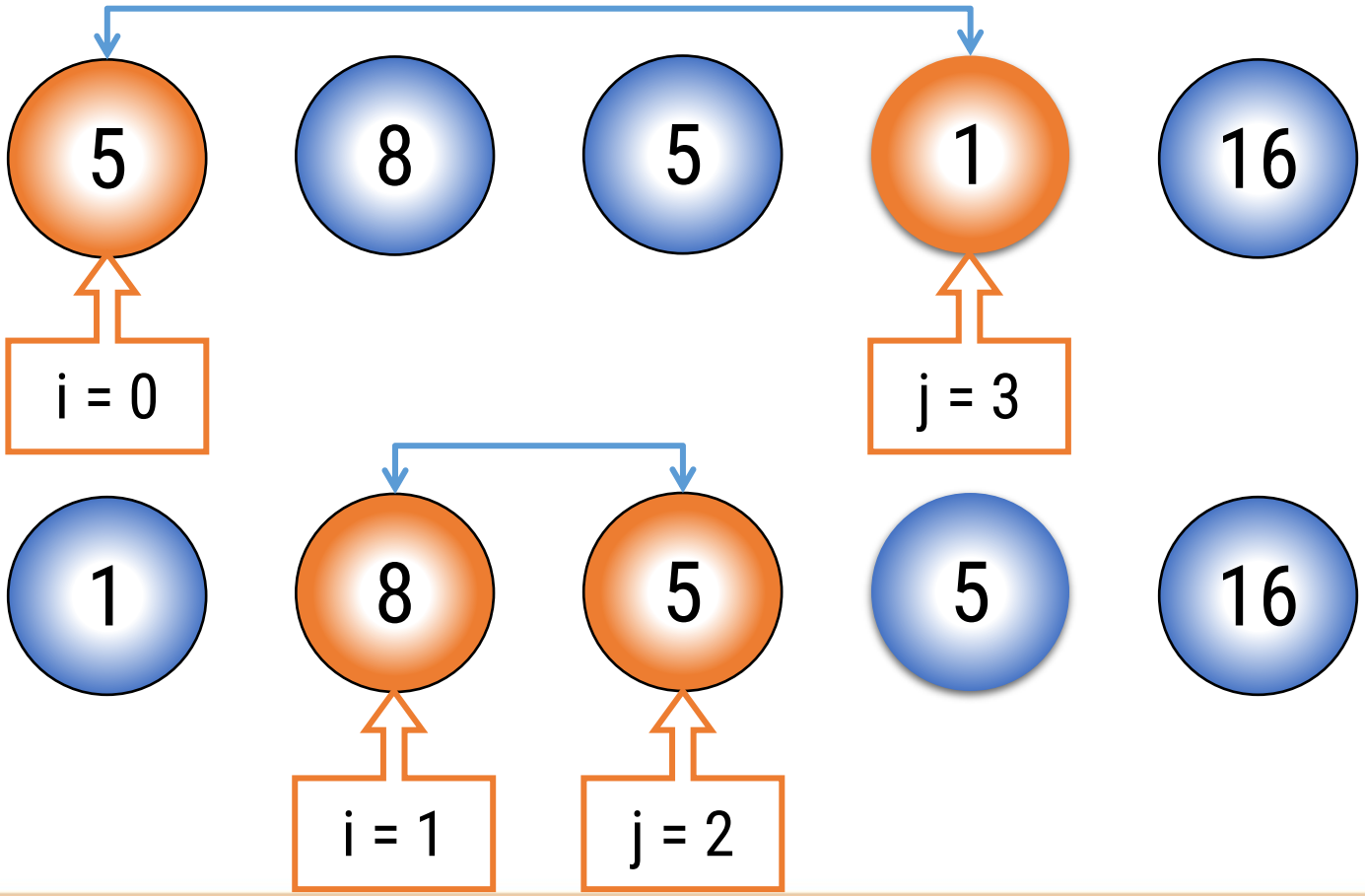


6. Quick Sort

➢ Minh họa thuật toán:

x = 5

- Phân hoạch đoạn l = 0, r = 4:

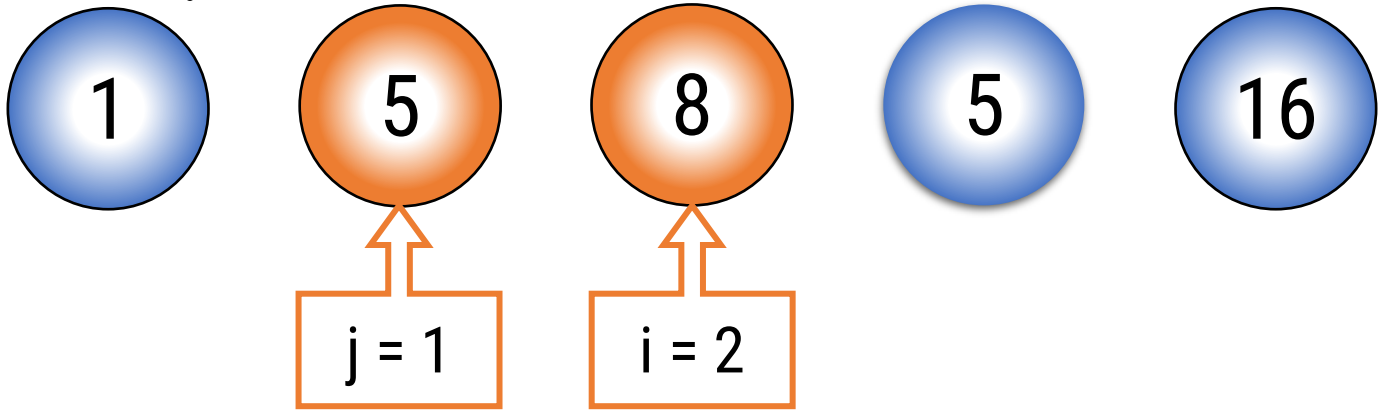


6. Quick Sort

➢ Minh họa thuật toán:

- Phân hoạch đoạn l = 0, r = 4:

x = 5



- Phân hoạch đoạn l = 0, r = 1:

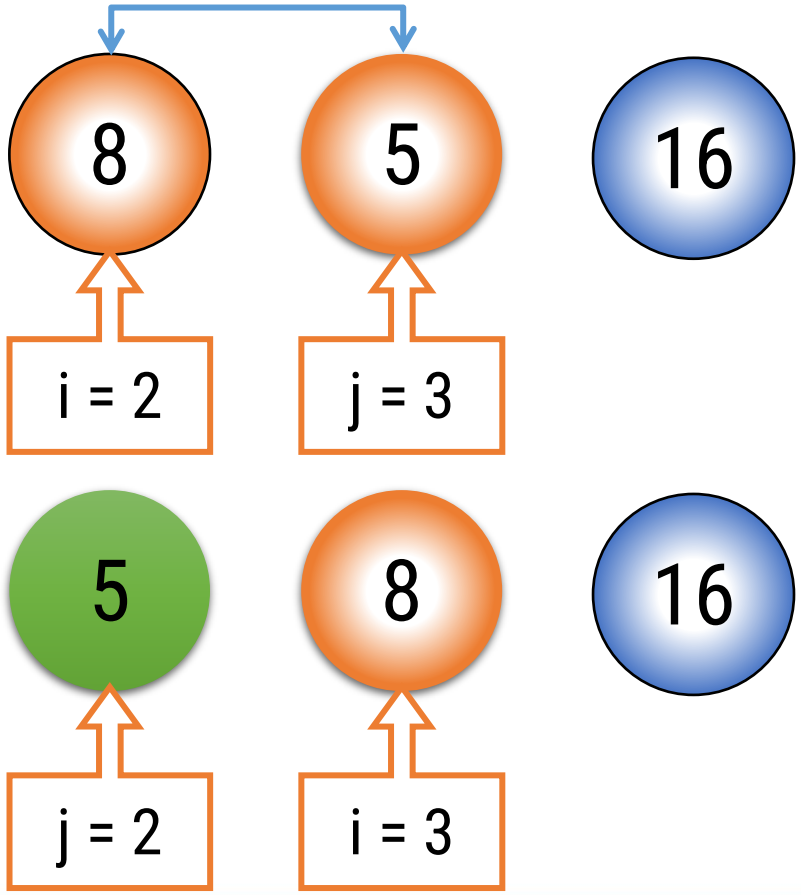
x = 1

6. Quick Sort

➢ Minh họa thuật toán:

- Phân hoạch đoạn l = 2, r = 4:

x = 5

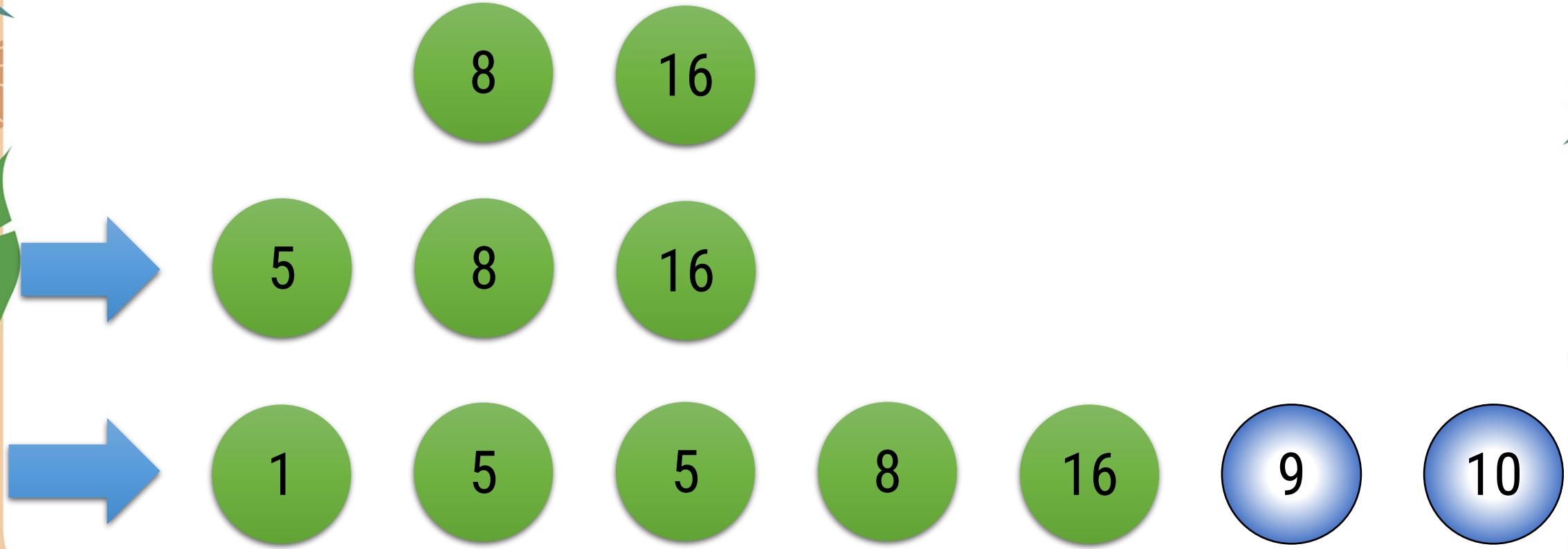


6. Quick Sort

➢ Minh họa thuật toán:

- Phân hoạch đoạn l = 3, r = 4:

x = 5

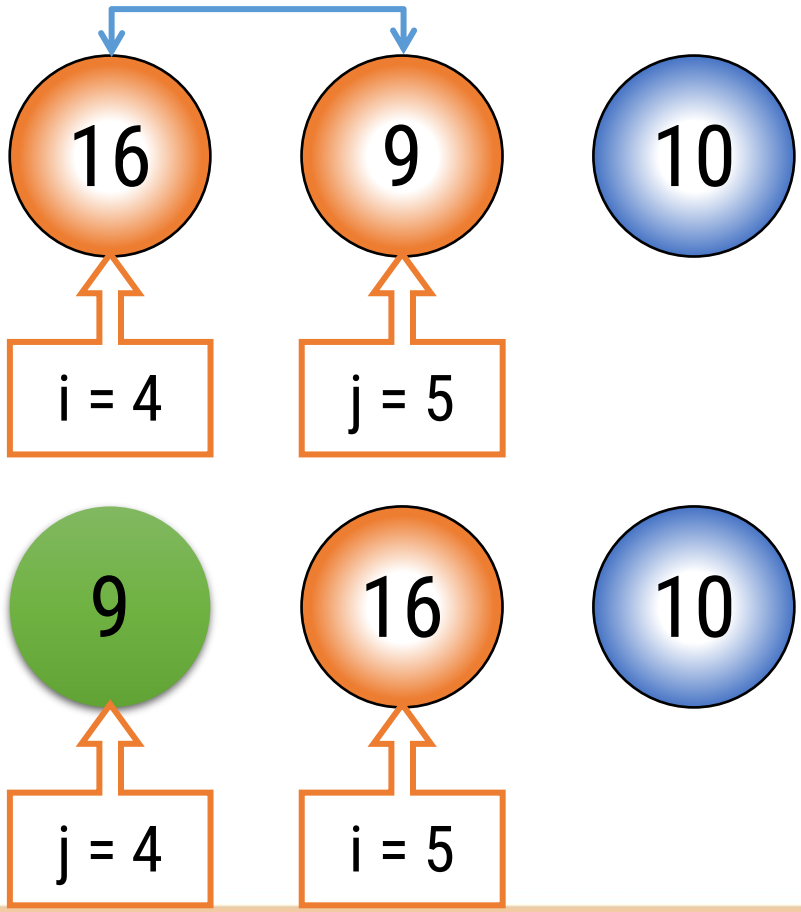


6. Quick Sort

➢ Minh họa thuật toán:

- Phân hoạch đoạn l = 4, r = 6:

x = 9

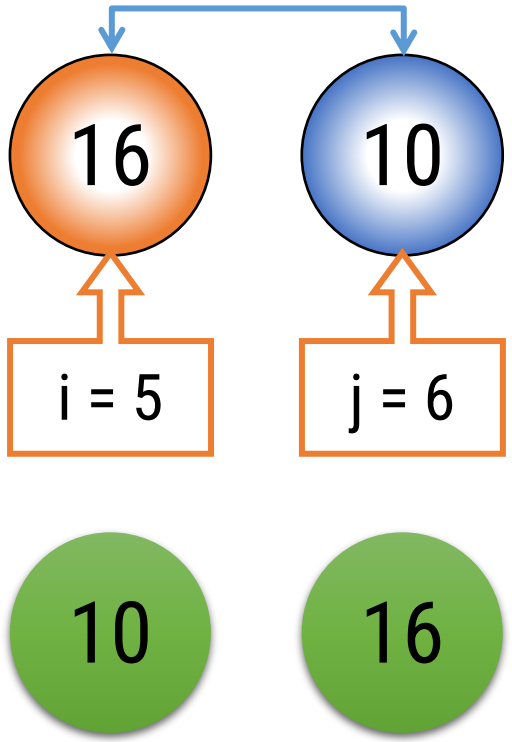


6. Quick Sort

➢ Minh họa thuật toán:

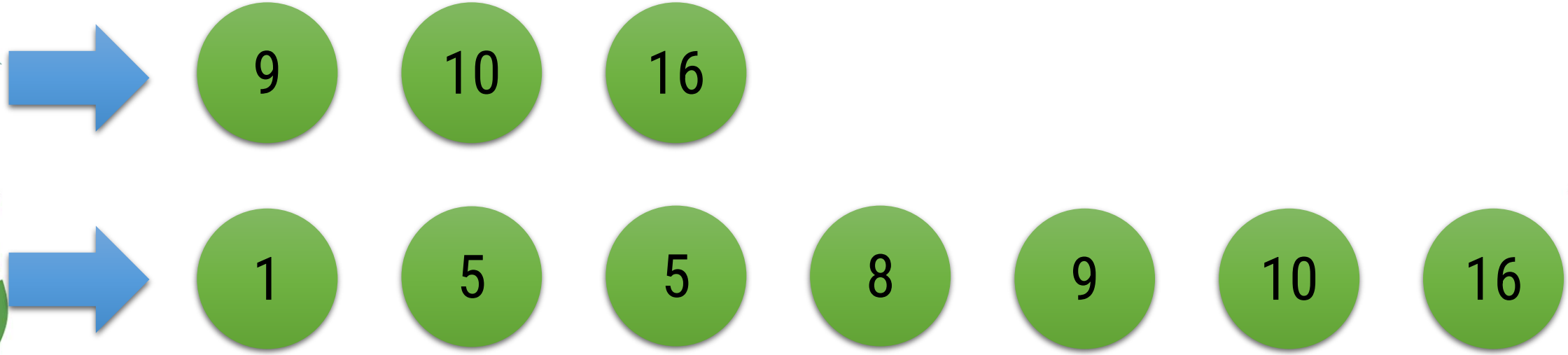
x = 16

- Phân hoạch đoạn l = 5, r = 6:



6. Quick Sort

- Minh họa thuật toán:



6. Quick Sort

➢ Các bước tiến hành:

- 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: Doicho(a[i],a[j]);
- Bước 3 : Nếu i < j: Lặp lại Bước 2.

  Ngược lại: Dừng

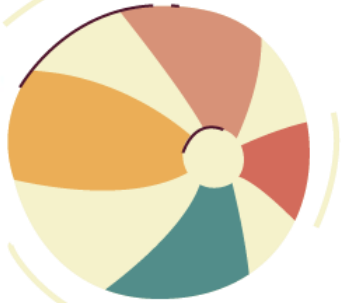
6. Quick Sort

➢ Thuật toán:

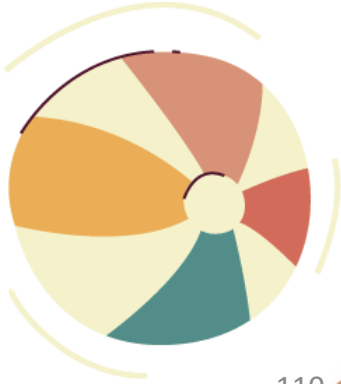
```
1   void QuickSort(int l, int r)
2   {
3       int i = l, j = r, x = a[(l + r) / 2];
4       //Phân hoạch dãy thành 3 phần
5       while (i < j)
6       {
7           while (a[i] < x) i++;   //Phần 1
8           while (a[j] > x) j--;   //Phần 3
9
10          //Tồn tại 1 giá trị a[i], a[j] hoặc bằng x hoặc chưa nằm đúng dãy
11          if (i <= j)
12          {
13              swap(a[i], a[j]);
14              i++; j--;
15          }
16      }
17      //Gọi đệ quy để tiếp tục phân hoạch các dãy con
18      if (j > l) QuickSort(l, j);
19      if (i < r) QuickSort(i, r);
20  }
```

6. Quick Sort

- Độ phức tạp thuật toán: O(n.logn)



CÂU HỎI VUI



Tên thuật toán	Độ phức tạp thuật toán
Interchange Sort	n^2
Selection Sort	n^2
Bubble Sort	n^2
Insertion Sort	n^2
Binary Insertion Sort	n^2
Quick Sort	n.logn

III. Bảng băm HASH TABLE

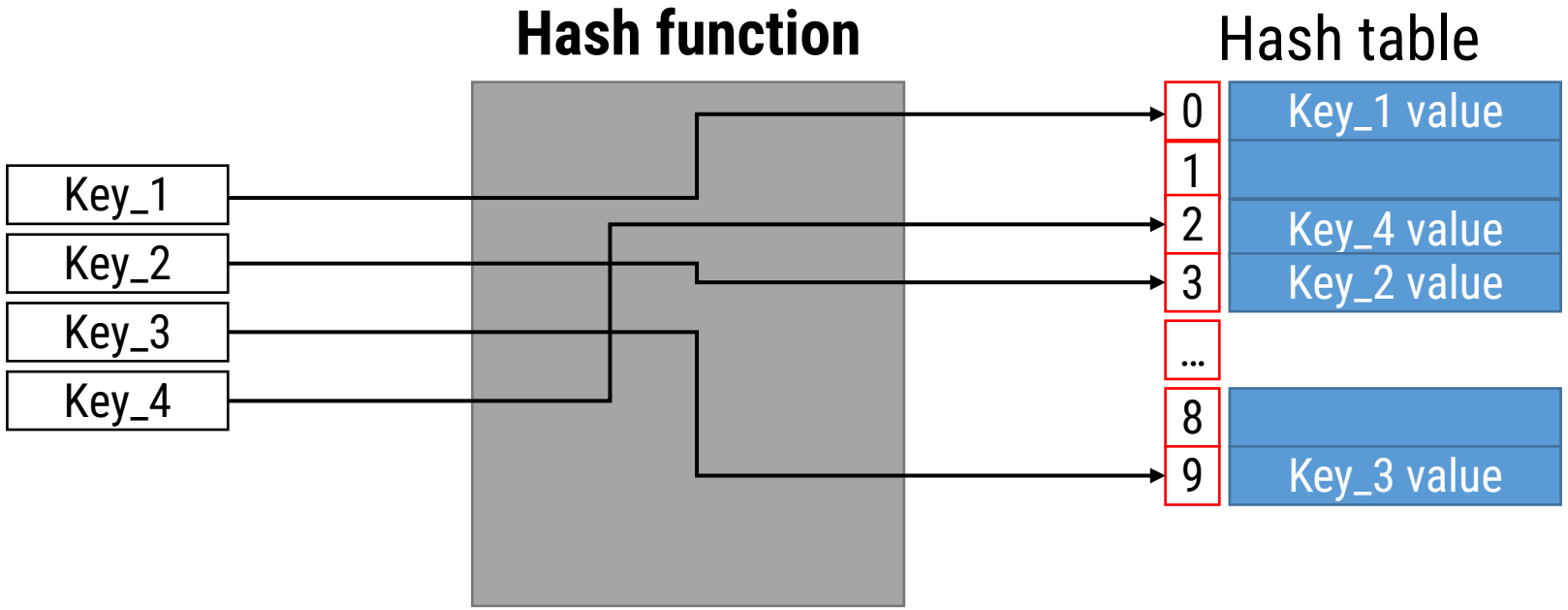


Nội dung:

1. Ý tưởng bảng băm
2. Các khái niệm
3. Lưu trữ bảng băm
   - ➢ Xác định hàm băm
   - ➢ Xử lý các xung đột
4. Ví dụ
5. Vận dụng

1. Ý tưởng bảng băm

Bảng Băm (Hash Table) là một **cấu trúc dữ liệu** dạng từ điển (Dictionary) gồm 2 phần chính là khoá tìm kiếm (hash key) và giá trị (value).

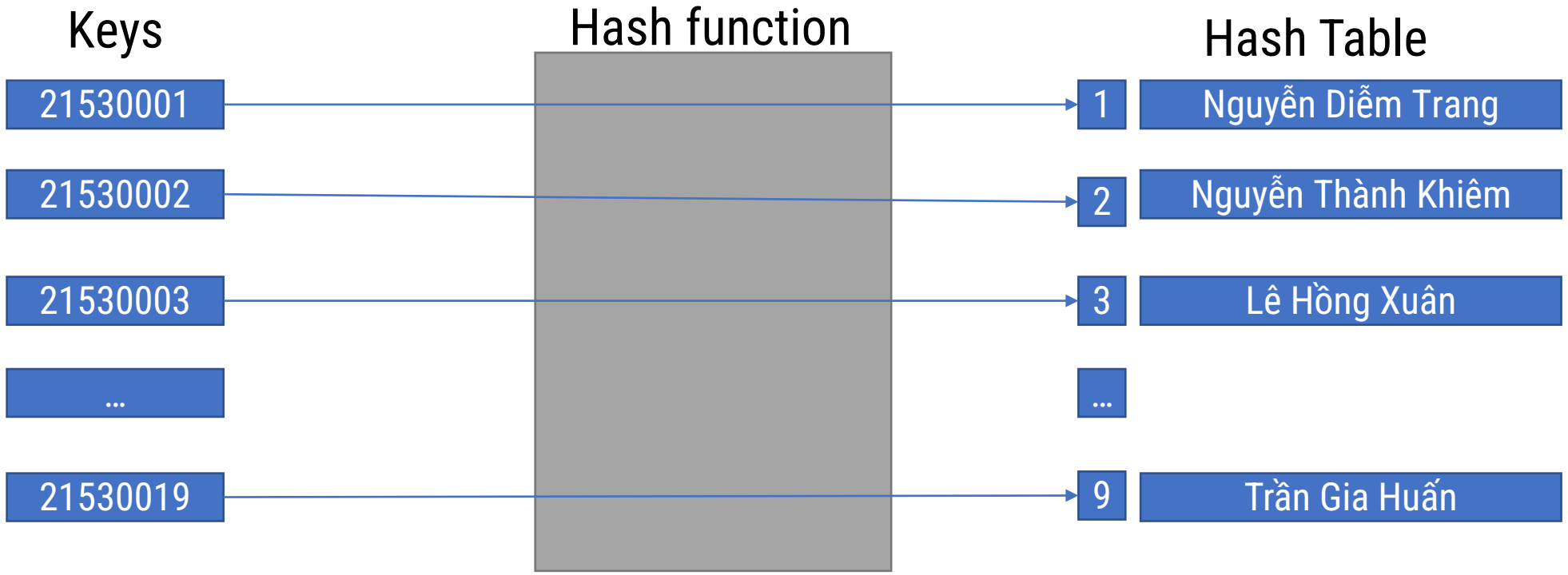


1. Ý tưởng bảng băm

21530001	Nguyễn Diễm Trang
21530002	Nguyễn Thành Khiêm
21530003	Lê Hồng Xuân
...	
21530019	Trần Gia Huấn

Tìm kiếm một tên sinh viên dựa trên MSSV.
Tìm kiếm tuần tự sẽ tốn O(n).

1. Ý tưởng bảng băm



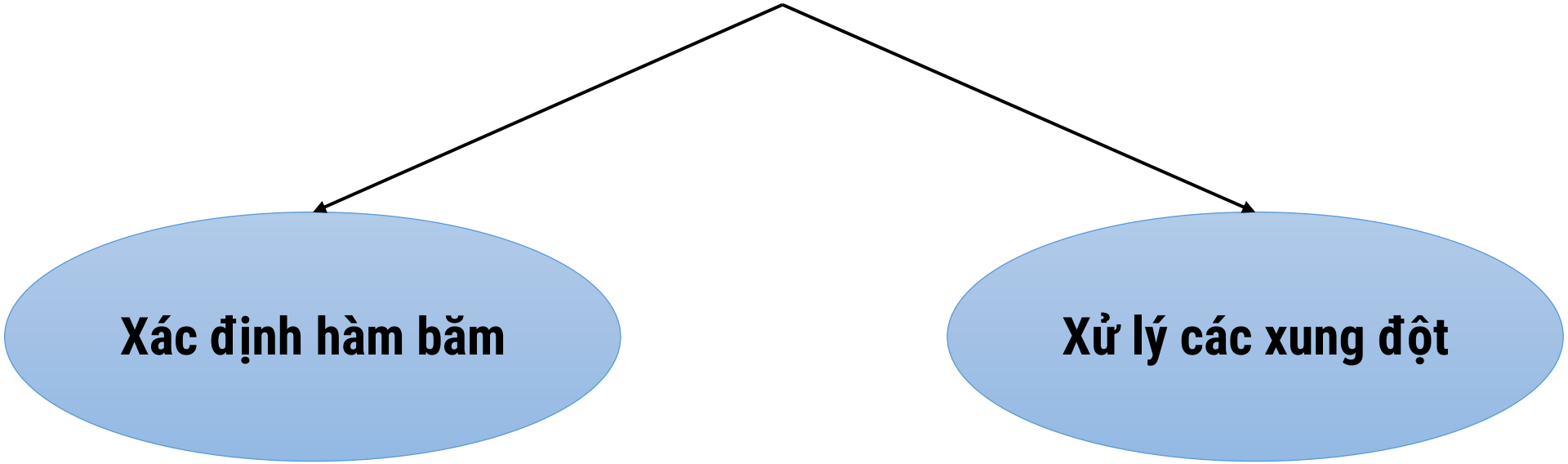
2. Các khái niệm

- Bảng băm (Hash Table) là mảng lưu trữ các record, ký hiệu: HT.
- HT có M vị trí được đánh chỉ mục từ 0 đến M-1, M là kích thước của bảng băm.

- Phép Băm (Hashing): Là quá trình ánh xạ một giá trị khóa vào một vị trí trong bảng bằng hàm băm. Một hàm băm ánh xạ các giá trị khóa đến các vị trí ký hiệu: h(key).

- Bảng băm thích hợp cho các ứng dụng được cài đặt trên đĩa và bộ nhớ, là một cấu trúc dung hòa giữa thời gian truy xuất và không gian lưu trữ.

3. Lưu trữ bằng bảng băm



Xác định hàm băm

Hàm băm thông dụng sử dụng phương pháp chia lấy dư:

F(key) = key % N

trong đó, key là khoa, N là kích thước của bảng.

Chọn được hàm băm tốt sẽ giúp hạn chế xung đột.

Lưu ý khi chọn N để có được hàm băm tốt:

- N không nên là lũy thừa của 2.
- N nên là số nguyên tố không gần với 2.

➢ Xử lý xung đột

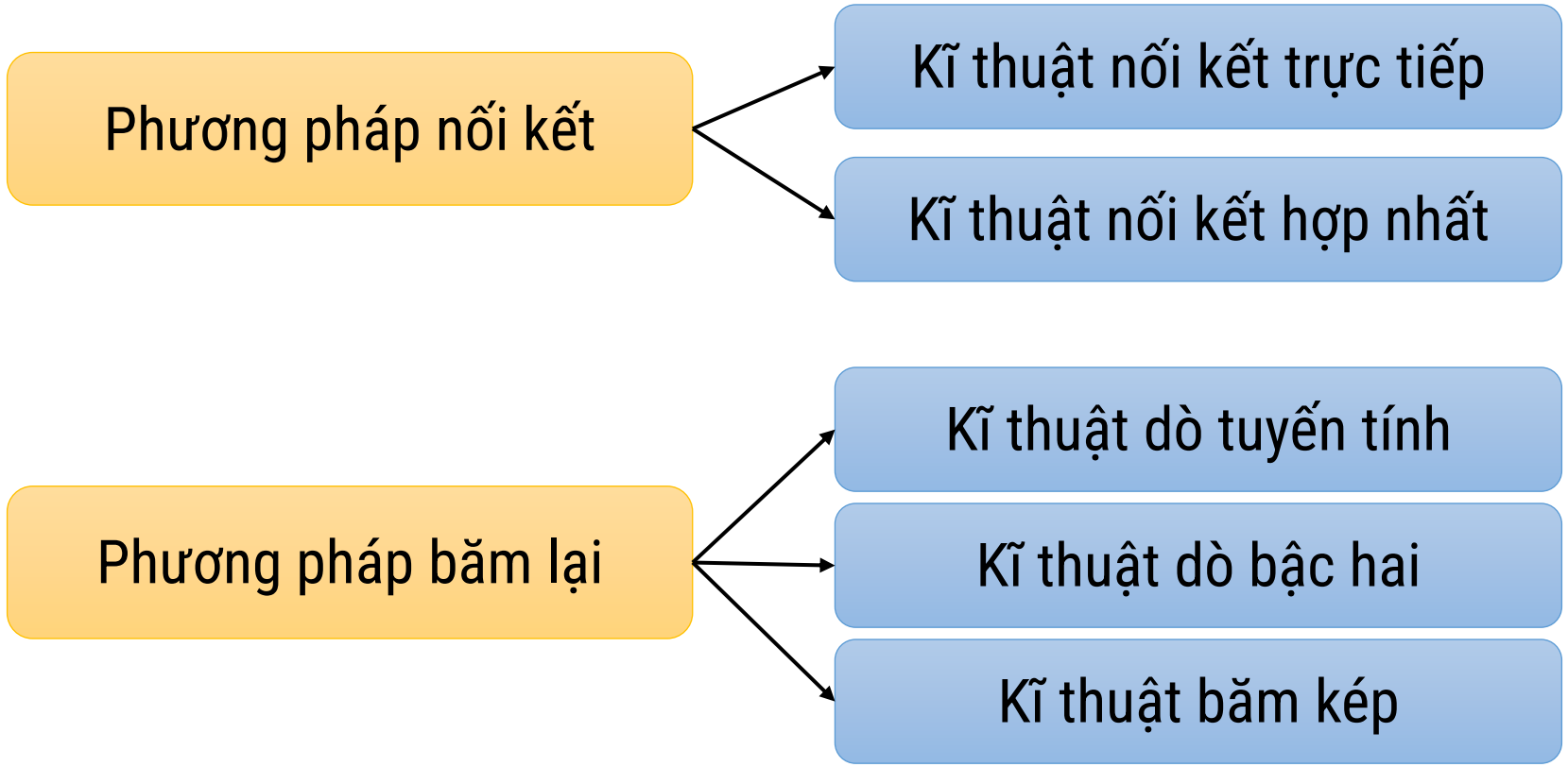
Phương pháp nối kết

Ý tưởng: Áp dụng danh sách liên kết đối với các khóa bị đụng độ.

Phương pháp băm lại

Ý tưởng: Nếu vị trí hiện tại đã bị khóa khác chiếm, ta sẽ thử tìm đến vị trí nào đó kế tiếp trong mảng.

➢ Xử lý xung đột



➢ Xử lý xung đột

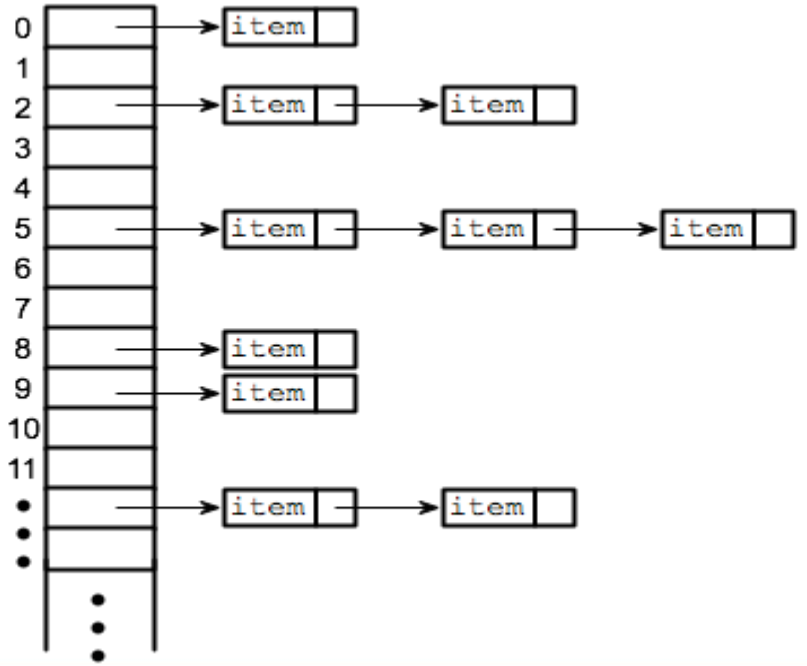
Phương pháp nối kết

Kĩ thuật nối kết trực tiếp

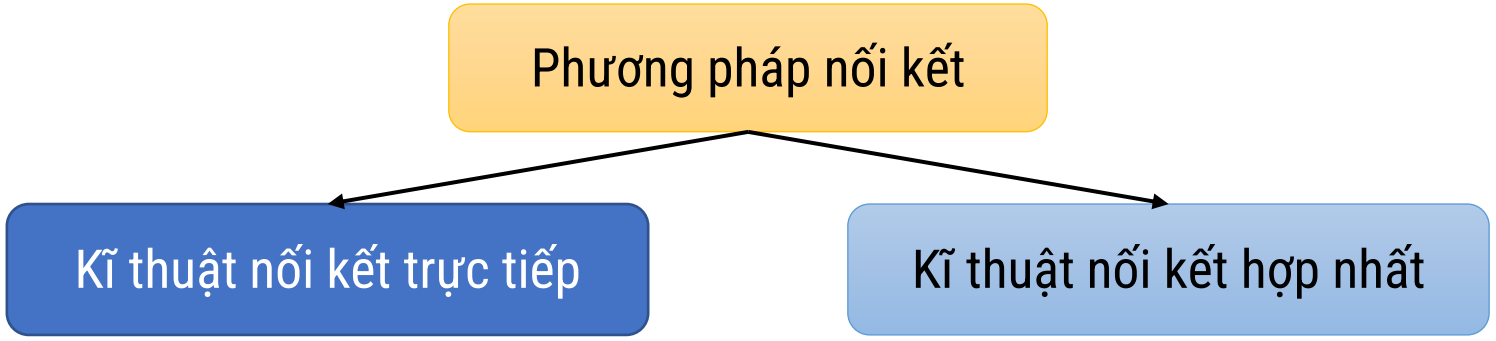
Kĩ thuật nối kết hợp nhất

Các nút bị băm cùng địa chỉ (các nút bị xung đột) được gom thành một danh sách liên kết.

Các nút bị xung đột tại địa chỉ i được nối kết trực tiếp với nhau qua danh sach liên kết i.



➢ Xử lý xung đột



- Bảng băm sẽ lưu giá trị của một nút gồm 2 thành phần là khóa và địa chỉ nút kế tiếp.
- Nếu xảy ra xung đột thì lưu nút vào vị trí còn trống cuối mảng, cập nhập địa chỉ nút kế tiếp sao cho các nút bị xung đột hình thành một danh sách liên kết.

➢ Xử lý xung đột

Phương pháp nối kết

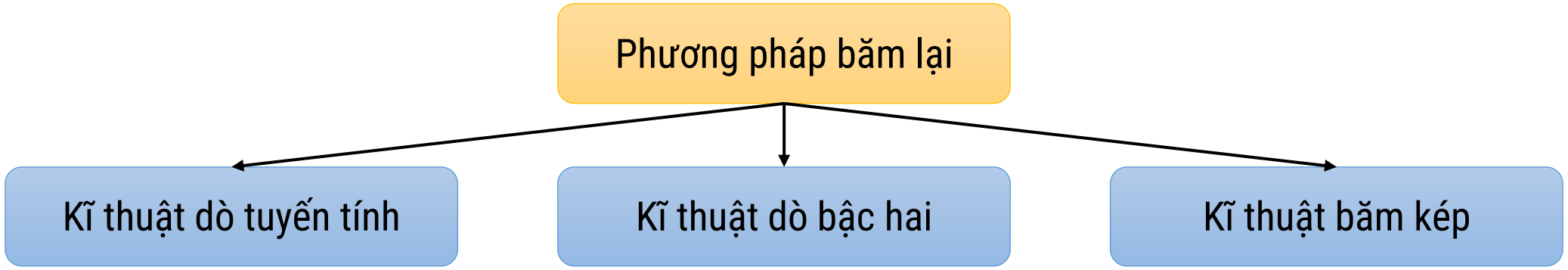
Kĩ thuật nối kết trực tiếp

Kĩ thuật nối kết hợp nhất

VD: K = {12, 5, 23, 43, 57}
H(key) = key % 7

	KEY	NEXT
0	NULL	-1
1	43	4
2	23	-1
3	NULL	-1
4	57	-1
5	12	6
6	5	-1

➢ Xử lý xung đột



Ý tưởng: Nếu vị trí hiện tại đã bị khóa khác chiếm, ta sẽ thử tìm đến vị trí nào đó kế tiếp trong mảng mà chưa bị chiếm.

Hàm tìm vị trí mới thường có dạng:

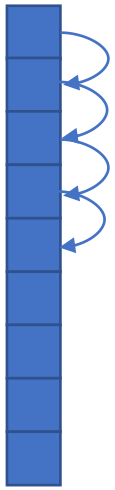
H(key, i) = (h(key) + f(i)) % N

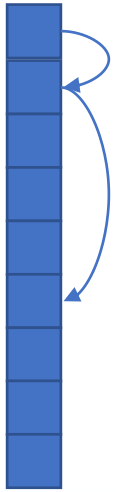
(trong đó hàm f(i) sẽ tùy thuộc vào kỹ thuật sử dụng)

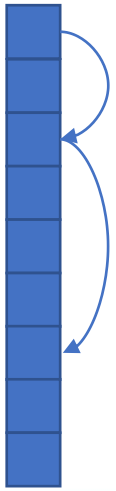
➢ Xử lý xung đột

Phương pháp băm lại

Kĩ thuật dò tuyến tính	Kĩ thuật dò bậc hai	Kĩ thuật băm kép
f(i) = i H(key, i) = (h(key) + i) % N	f(i) = i^2 H(key, i) = (h(key) + i^2) % N	f(i) = i * h2(key) H(key, i) = (h(key) + i * h2(key)) % N







4. Ví dụ

Đề bài:

a) Trình bày ưu điểm và hạn chế của cấu trúc bảng băm và cho ví dụ minh họa cụ thể ?

b) Cho bảng A kích thước 11 ô và tập khóa K = {7, 20, 16, 24, 12, 40, 15, 19, 5}, ta cần nạp các giá trị khóa K vào bảng A sử dụng hàm băm h1(k) = k % 11.

Hãy vẽ bảng A sau khi tất cả các giá trị khóa trong tập K được lưu trữ vào bảng A, sử dụng kỹ thuật băm kép để xử lý xung đột, với hàm băm kép thứ 2 h2(k) = 7 – k%7, hàm băm lại có dạng

H(k, i) = (h1(k) + i * (h2(k)))%11 với k là giá trị khóa, i là số lần xảy ra xung đột.

4. Ví dụ

a) Trình bày ưu điểm và hạn chế của cấu trúc bảng băm và cho ví dụ minh họa cụ thể ?

Ví dụ với tập khóa

K = {12, 33, 4, 20, 45, 10, 14}

và hàm băm

h(k) = k mod 9.

Khi đó các giá trị được lưu như bảng bên cạnh:

	Giá trị lưu
0	45
1	10
2	20
3	12
4	4
5	14
6	33
7	NULL
8	NULL

Kiểm tra 1 số có nằm trong tập khóa K không có độ phức tạp là O(1)

=> **Tốc độ truy xuất nhanh.**

4. Ví dụ

a) Trình bày ưu điểm và hạn chế của cấu trúc bảng băm và cho ví dụ minh họa cụ thể ?

- **Ưu điểm:**

Bảng băm là một cấu trúc dung hòa giữa thời gian truy xuất và dung lượng bộ nhớ: Nếu **không có sự giới hạn về bộ nhớ** thì chúng ta có thể xây dựng bảng băm với mỗi khóa ứng với một địa chỉ với mong muốn thời gian truy xuất tức thời: O(1).

Nếu xảy ra ít xung đột, bảng băm có tốc tộ truy xuất tốt hơn so với các cấu trúc dữ liệu khác như mảng hay cây.

4. Ví dụ

a) Trình bày ưu điểm và hạn chế của cấu trúc bảng băm và cho ví dụ minh họa cụ thể ?

- **Nhược điểm:**

  Nếu tập khóa lớn, rất khó để không xảy ra xung đột.

  Bảng băm sẽ trở nên kém hiệu quả nếu xảy ra nhiều xung đột.

4. Ví dụ

a) Trình bày ưu điểm và hạn chế của cấu trúc bảng băm và cho ví dụ minh họa cụ thể ?

Ví dụ với tập khóa

K = {12, 33, 4, 20, 45, 10, 14, 18}

và hàm băm

h(k) = k mod 9.

Xử lý xung đột bằng phương pháp dò tuyến tính.

Xác định vị trí khóa key = 18 trong bảng tốn gần O(n) do xảy ra nhiều xung đột

=> **Truy xuất kém hiệu quả.**

	Giá trị lưu	
0	45	18 mod 9 = 0 (xung đột)
1	10	
2	20	
3	12	
4	4	
5	14	
6	33	
7	18	
8	NULL	

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

	Giá trị lưu
0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL
9	NULL
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(7) = 7→ không xảy ra xung đột

	Giá trị lưu
0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	7
8	NULL
9	NULL
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(20) = 9 → không xảy ra xung đột

	Giá trị lưu
0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(16) = 5 ⟶ không xảy ra xung đột

	Giá trị lưu
0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	16
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(24) = 2 → không xảy ra xung đột

	Giá trị lưu
0	NULL
1	NULL
2	24
3	NULL
4	NULL
5	16
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(12) = 1 → không xảy ra xung đột

	Giá trị lưu
0	NULL
1	12
2	24
3	NULL
4	NULL
5	16
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(40) = 7 → xung đột, băm lại

H(40, 1) = 9 → xung đột, băm lại

H(40, 2) = 0 → không xảy ra xung đột

	Giá trị lưu
0	40
1	12
2	24
3	NULL
4	NULL
5	16
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k% 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(15) = 4 ⟶ không xảy ra xung đột

	Giá trị lưu
0	40
1	12
2	24
3	NULL
4	15
5	16
6	NULL
7	7
8	NULL
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1:

h1(k) = k % 11

Hàm băm 2:

h2(k) = 7 – k % 7

Hàm băm lại:

H(k, i) = (h1(k) + i * h2(k)) % 11

h1(19) = 8 ⟶ không xảy ra xung đột

	Giá trị lưu
0	40
1	12
2	24
3	NULL
4	15
5	16
6	NULL
7	7
8	19
9	20
10	NULL

4. Ví dụ

Cho bảng A kích thước 11 ô

Tập khóa **K = {7, 20, 16, 24, 12, 40, 15, 19, 5}**

Hàm băm 1: **h1(k) = k % 11**

Hàm băm 2: **h2(k) = 7 – k % 7**

Hàm băm lại: **H(k, i) = (h1(k) + i * h2(k)) % 11**

h1(5) = 5 → xung đột, băm lại

H(5,1) = 7 → xung đột, băm lại

H(5,2) = 9 → xung đột, băm lại

H(5,3) = 0 → xung đột, băm lại

H(5,4) = 2 → xung đột, băm lại

H(5,5) = 4 → xung đột, băm lại

H(5,6) = 6 → không xảy ra xung đột

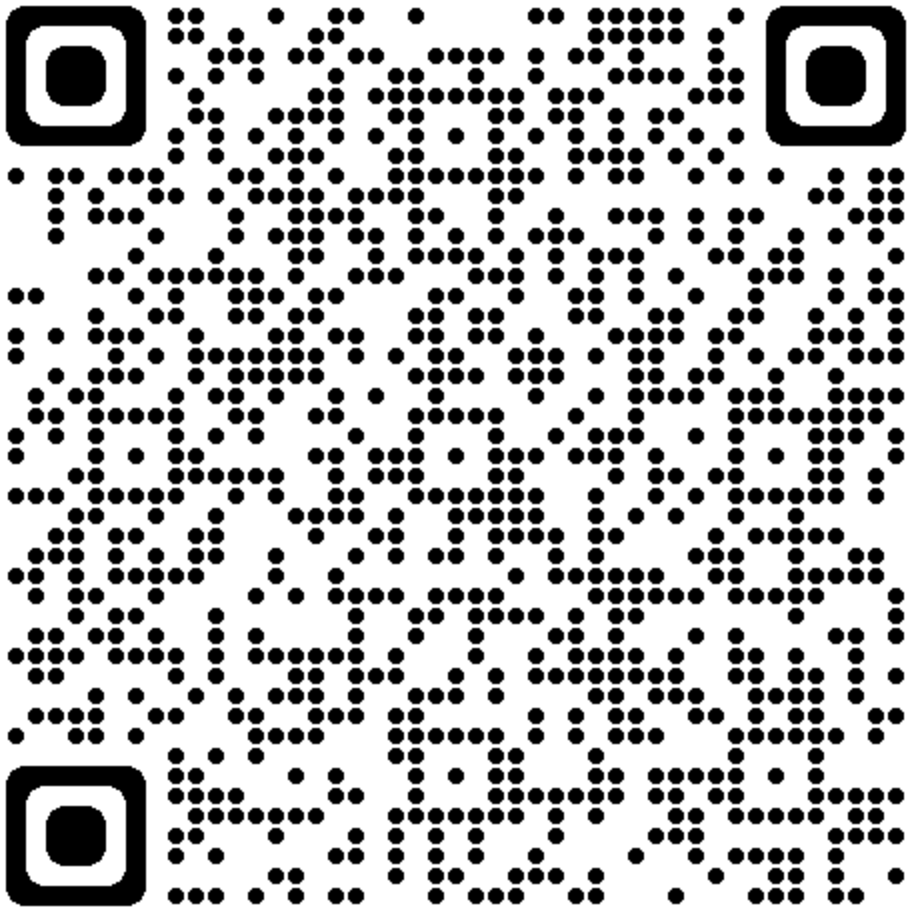
	Giá trị lưu
0	40
1	12
2	24
3	NULL
4	15
5	16
6	5
7	7
8	19
9	20
10	NULL

5. Vận dụng

Cho bảng A kích thước 10 ô và tập khóa K = {23,12,65,27,8,42,50,58}, nạp giá trị lần lược vào bảng A sử dụng hàm băm H(k) = k % 10. Khi nạp đến khóa nào thì sẽ xảy ra đụng độ?

A. 27
B. 8
C. 42
D. 50

XIN CẢM ƠN CÁC BẠN ĐÃ LẮNG NGHE



https://bit.ly/CK_CTDLGT