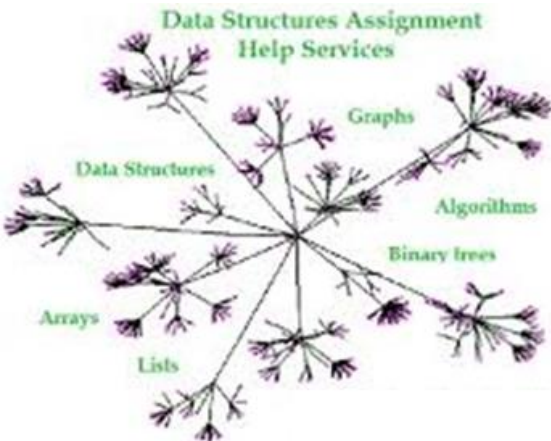
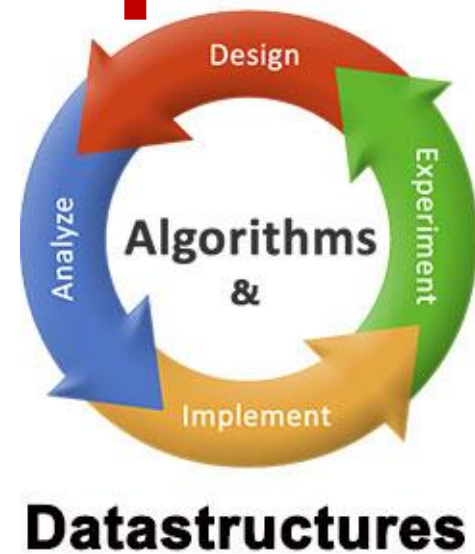


CẤU TRÚC DỮ LIỆU & GIẢI THUẬT



Lê Văn Hạnh

levanhanhvn@gmail.com

NỘI DUNG MÔN HỌC

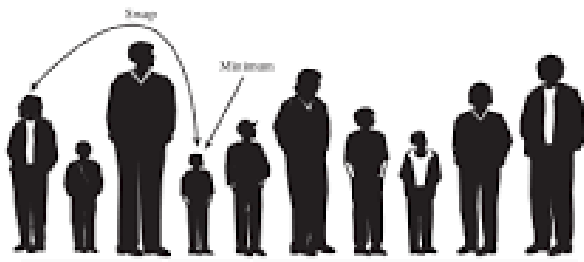
- Chương 1: Ôn tập ngôn ngữ lập trình C
- Chương 2: Kiểu dữ liệu con trỏ
- Chương 3: Tổng quan về cấu trúc dữ liệu và giải thuật
- Chương 4: Danh sách kê (Danh sách tuyến tính)
- Chương 5: Các giải thuật tìm kiếm trên danh sách kê
- **Chương 6: Các giải thuật sắp xếp trên danh sách kê**
- Chương 7: Danh sách liên kết động (*Linked List*)
- Chương 8: Ngăn xếp (*Stack*)
- Chương 9: Hàng đợi (*Queue*)
- Chương 10: Cây nhị phân tìm kiếm (*Binary Search Tree*)
- Chương 11: Cây cân bằng (*Binary Search Tree*)
- Chương 12: Bảng băm (*Hash Table*)



Chương 6



CÁC GIẢI THUẬT SẮP XẾP TRÊN DANH SÁCH KÈ



MỤC TIÊU

- i. Hiểu và giải thích được các giải thuật sắp xếp trên mảng 1 chiều
- ii. Nắm vững, minh họa và tính toán được các phép gán (hoán vị) các giải thuật sắp xếp cơ bản trên mảng một chiều
- iii. Cài đặt thành công các giải thuật sắp xếp trên mảng 1 chiều

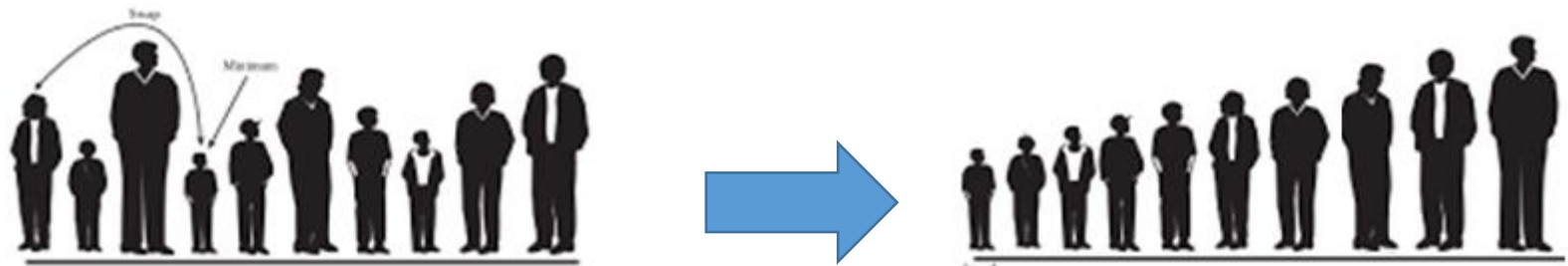
NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ **vi-xii**

GIỚI THIỆU BÀI TOÁN SẮP XẾP

- Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng theo một thứ tự nhất định.




- Trường tham gia quá trình tìm kiếm gọi là **khóa** (key)
- Việc sắp xếp sẽ được tiến hành dựa vào giá trị **khóa** này

Giới thiệu bài toán sắp xếp

- Mô tả bài toán: Cho tập n phần tử, mỗi phần tử lại có m thuộc tính, được biểu diễn dưới dạng record.

Mssv	Họ tên	Nsinh	Điểm
32	Lê Minh Tý	1996	7.50
56	Nguyễn Thị Dần	1998	8.25
12	Huỳnh Văn Sửu	1997	9.00



32	56	12
Lê Minh Tý	Nguyễn Thị Dần	Huỳnh Văn Sửu
1996	1998	1997
7.50	8.25	9.00

- Dựa vào một (hoặc vài) thuộc tính để sắp xếp các phần tử theo trật tự mới.

□ Sắp theo năm sinh

32	12	56
Lê Minh Tý	Huỳnh Văn Sửu	Nguyễn Thị Dần
1996	1997	1998
7.50	9.00	8.25

□ Sắp theo điểm

32	56	12
Lê Minh Tý	Nguyễn Thị Dần	Huỳnh Văn Sửu
1996	1998	1997
7.50	8.25	9.00

Giới thiệu bài toán sắp xếp

- Hai thao tác cơ bản:
 - So sánh
 - Hoán vị

32	56	12
Lê Minh Tý	Nguyễn Thị Dần	Huỳnh Văn Sửu
1996	1998	1997
7.50	8.25	9.00

32	12	56
Lê Minh Tý	Huỳnh Văn Sửu	Nguyễn Thị Dần
1996	1997	1998
7.50	9.00	8.25

32	56	12
Lê Minh Tý	Nguyễn Thị Dần	Huỳnh Văn Sửu
1996	1998	1997
7.50	8.25	9.00

Giới thiệu bài toán sắp xếp

Khái niệm “nghịch thế”

- Xét một mảng các số a_0, a_1, \dots, 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.

2	3	5	9	7
---	---	---	---	---

- Mảng đã có thứ tự sẽ không chứa nghịch thế. Khi đó a_0 sẽ là phần tử nhỏ nhất rồi đến a_1, a_2, \dots, a_n với $a_0 \leq a_1 \leq \dots \leq a_n$

2	3	5	7	9
---	---	---	---	---

NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ **vi-xii**

1. ĐỔI CHỖ TRỰC TIẾP (*Interchange Sort*)

- **Ý tưởng:** Xuất phát 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.

1. Đổi Chỗ Trục 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($a[i], a[j]$);

$j = j+1$;

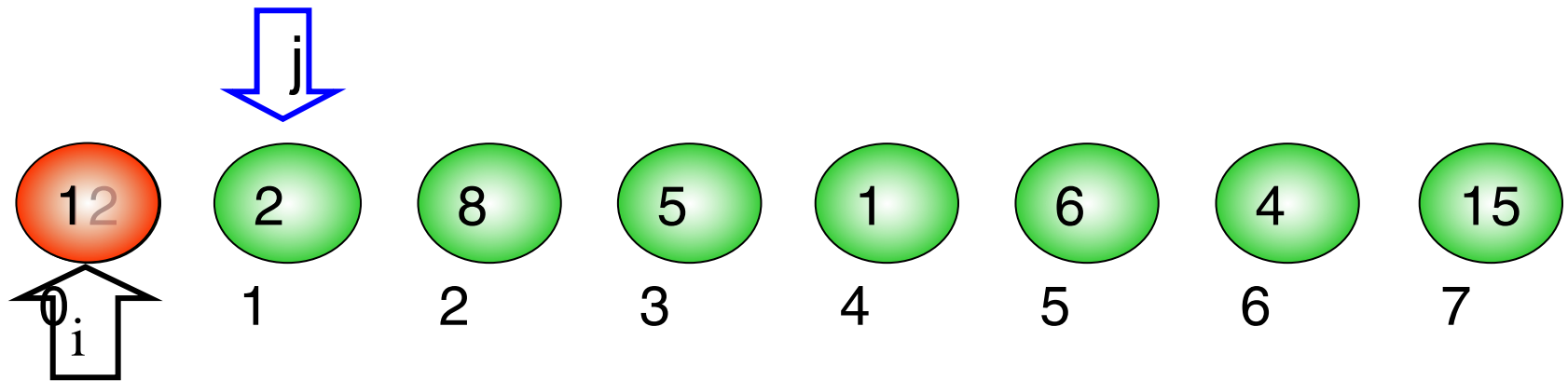
- Bước 4: $i = i+1$;

Nếu $i < N-1$: Lặp lại Bước 2.

Ngược lại: Dừng.

1. Đổi Chỗ Trực Tiếp – *Interchange Sort*

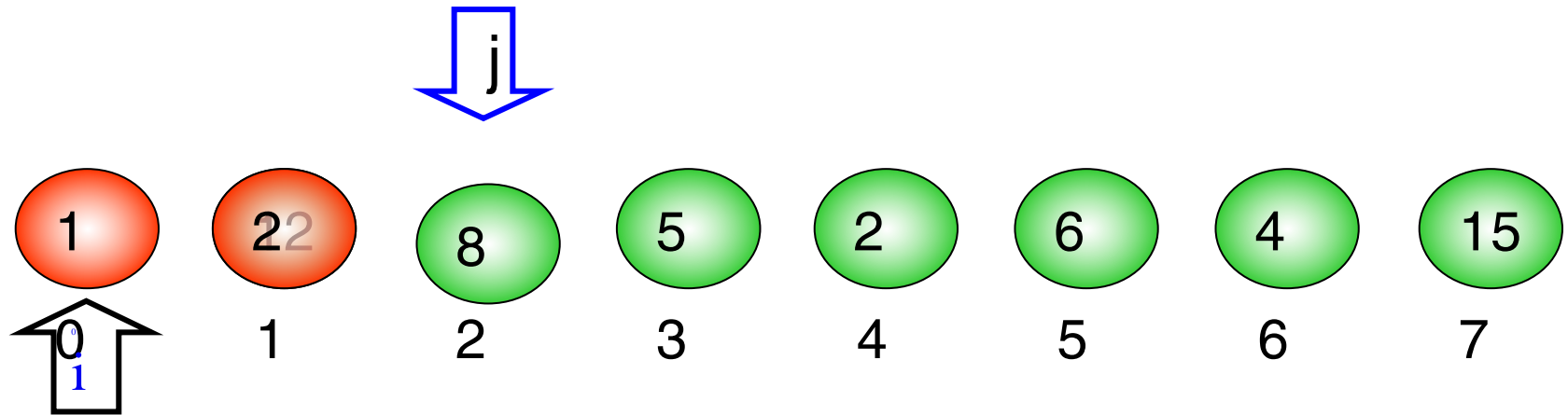
- **Minh họa lại giải thuật**



```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j] < a[i])
                Swap(a[i], a[j]);
}
```

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

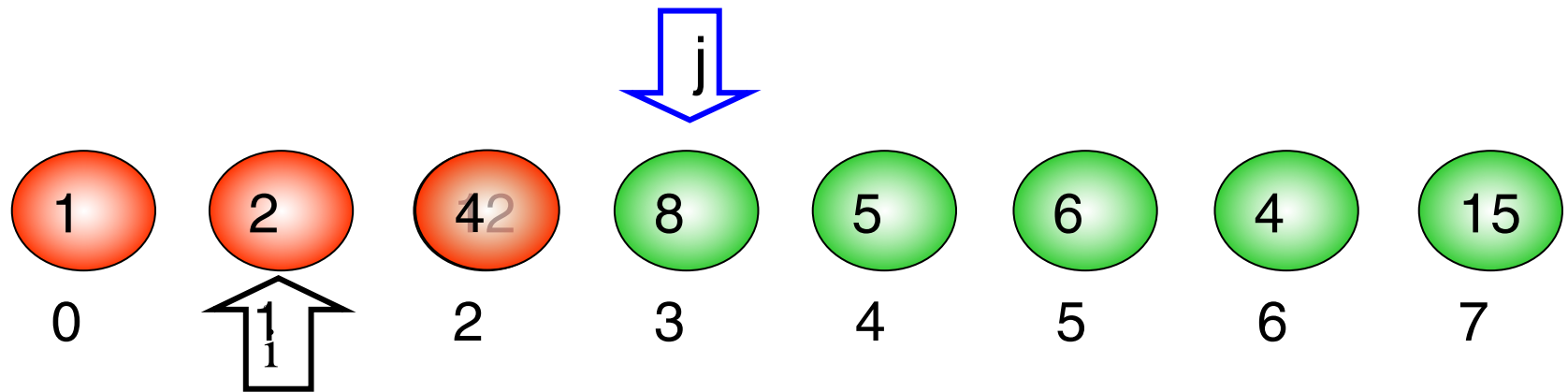
- Minh họa lại giải thuật



```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j] < a[i])
                Swap(a[i], a[j]);
}
```

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

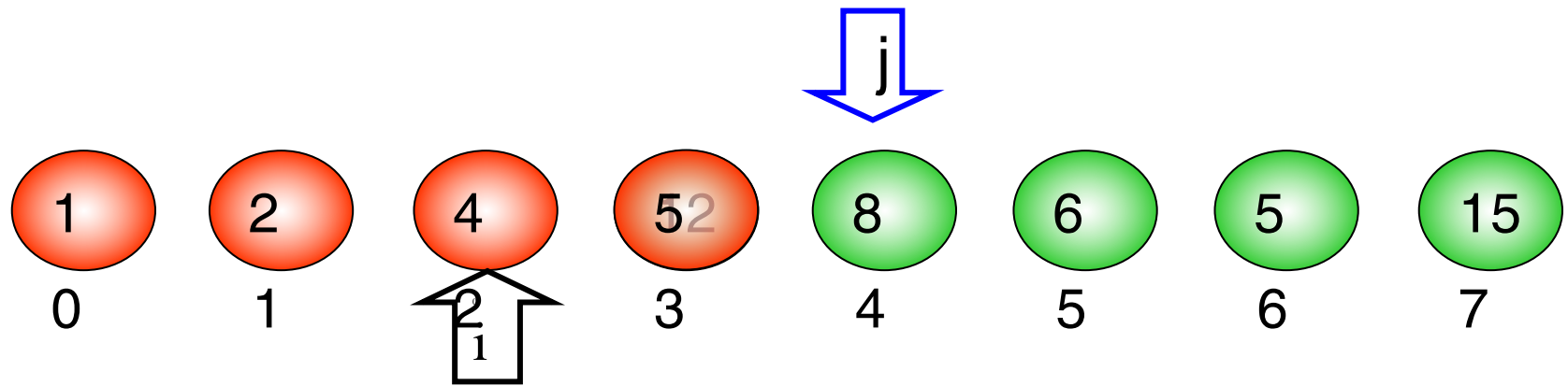
- Minh họa lại giải thuật



```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j] < a[i])
                Swap(a[i], a[j]);
}
```

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

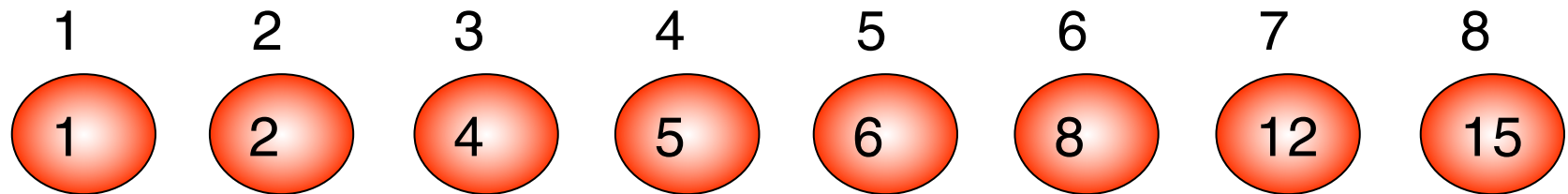
- Minh họa lại giải thuật



```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j] < a[i])
                Swap(a[i], a[j]);
}
```


1. Đổi Chỗ Trực Tiếp – *Interchange Sort*

- Minh họa lại giải thuật



```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j] < a[i])
                Swap(a[i], a[j]);
}
```

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

- Cài Đặt

```
void Swap(int &x, int &y)
{
    int tam = x;
    x = y;
    y = tam;
}

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]) //Tìm thấy 1 nghịch thế
                Swap(a[i], a[j]);
}
```

1. Đổi Chỗ Trục Tiếp – Interchange Sort

- Trình bày dạng bảng

```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j ]< a[i])
                Swap(a[i], a[j]);
}
```

i	j	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
		12	2	8	5	1	6	4	15
0	1	<u>12/2</u>	<u>2/12</u>	8	5	1	6	4	15
0	2	2	12	8	5	1	6	4	15
0	3	2	12	8	5	1	6	4	15
0	4	<u>2/1</u>	12	8	5	<u>1/2</u>	6	4	15
0	5	1	12	8	5	2	6	4	15
0	6	1	12	8	5	2	6	4	15
0	7	1	12	8	5	2	6	4	15
1	2	1	<u>12/8</u>	<u>8/12</u>	5	2	6	4	15
1	3	1	<u>8/5</u>	12	<u>5/8</u>	2	6	4	15
1	4	1	<u>5/2</u>	12	8	<u>2/5</u>	6	4	15
1	5	1	2	12	8	5	6	4	15
1	6	1	2	12	8	5	6	4	15
1	7	1	2	12	8	5	6	4	15

1. Đổi Chỗ Trục Tiếp – Interchange Sort

- Trình bày dạng bảng

```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j ]< a[i])
                Swap(a[i] , a[j]);
}
```

i	j	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
1	7	1	2	12	8	5	6	4	15
2	3	1	2	<u>12/8</u>	<u>8/12</u>	5	6	4	15
2	4	1	2	<u>8/5</u>	12	<u>5/8</u>	6	4	15
2	5	1	2	5	12	8	6	4	15
2	6	1	2	<u>5/4</u>	12	8	6	<u>4/5</u>	15
2	7	1	2	4	12	8	6	5	15
3	4	1	2	4	<u>12/8</u>	<u>8/12</u>	6	5	15
3	5	1	2	4	<u>8/6</u>	12	<u>6/8</u>	5	15
3	6	1	2	4	<u>6/5</u>	12	8	<u>5/6</u>	15
3	7	1	2	4	5	12	8	6	15

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

- Trình bày dạng bảng

```
void InterchangeSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j = i+1; j < n ; j++)
            //Tìm thấy 1 nghịch thế
            if(a[j ]< a[i])
                Swap(a[i], a[j]);
}
```

i	j	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
4	5	1	2	4	5	<u>12/8</u>	<u>8/12</u>	6	15
4	6	1	2	4	5	<u>8/6</u>	12	<u>6/8</u>	15
4	7	1	2	4	5	6	12	8	15
5	6	1	2	4	5	6	<u>12/8</u>	<u>8/12</u>	15
5	7	1	2	4	5	6	8	12	15
6	7	1	2	4	5	6	8	12	15

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

- Đánh giá giải thuật

Số lượng các phép so sánh xảy ra không phụ thuộc vào tình trạng của dãy số ban đầu, nhưng số lượng phép hoán vị thực hiện tùy thuộc vào kết quả so sánh, có thể ước lượng trong từng trường hợp như sau:

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

1. Đổi Chỗ Trục Tiếp – *Interchange Sort*

- **Bài tập áp dụng:** Trình bày quá trình sắp xếp mảng sau bằng giải thuật Interchange Sort

i. $n=6$

9 8 7 4 5 6

ii. $n=7$

9 8 7 6 5 4 3

NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ **vi-xii**

2. CHỌN TRỰC TIẾP (*Selection Sort*)

2.1. Ý tưởng

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

2.2. Các bước của giải thuật chọn trực tiếp

- Bước 1: $i = 0$;
- Bước 2: Tìm phần tử **a[min]** nhỏ nhất trong dãy hiện hành từ $a[i]$ đến $a[n-1]$
- Bước 3 : Đổi chỗ $a[\text{min}]$ 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

2.3. Cài Đặt giải thuật Chọn Trực Tiếp

```
void SelectionSort(int a[],int n )
{
    int min,i,j;
    for(i=0;i<n-1;i++)
    {
        min = i;
        for(j = i+1; j <n ; j++)
            if (a[j ] < a[min])
                min=j; //lưu vtrí phần tử hiện nhỏ nhất
        Swap(a[min],a[i]);
    }
}
```

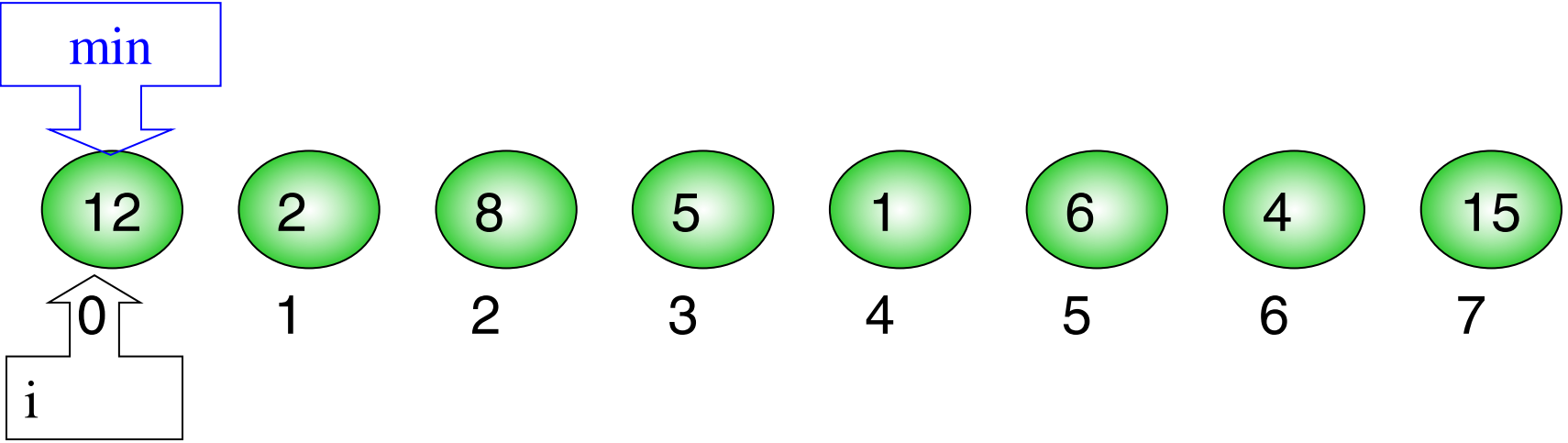
2. Chọn Trực Tiếp – Selection Sort

2.4. Minh Họa giải thuật Chọn Trực Tiếp

```
void SelectionSort(int a[],int n )
{
    int min,i,j;
    for(i=0;i<n-1;i++)
    {
        min = i;
        for(j = i+1; j <n ; j++)
            if (a[j ] < a[min])
                min=j;
        Swap(a[min] ,a[i]);
    }
}
```

$i = 0; min=1; j=4;$
 $\Rightarrow A[min] = 2 > A[j] = 1$
 \Rightarrow cập nhật lại $min = j = 4$

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



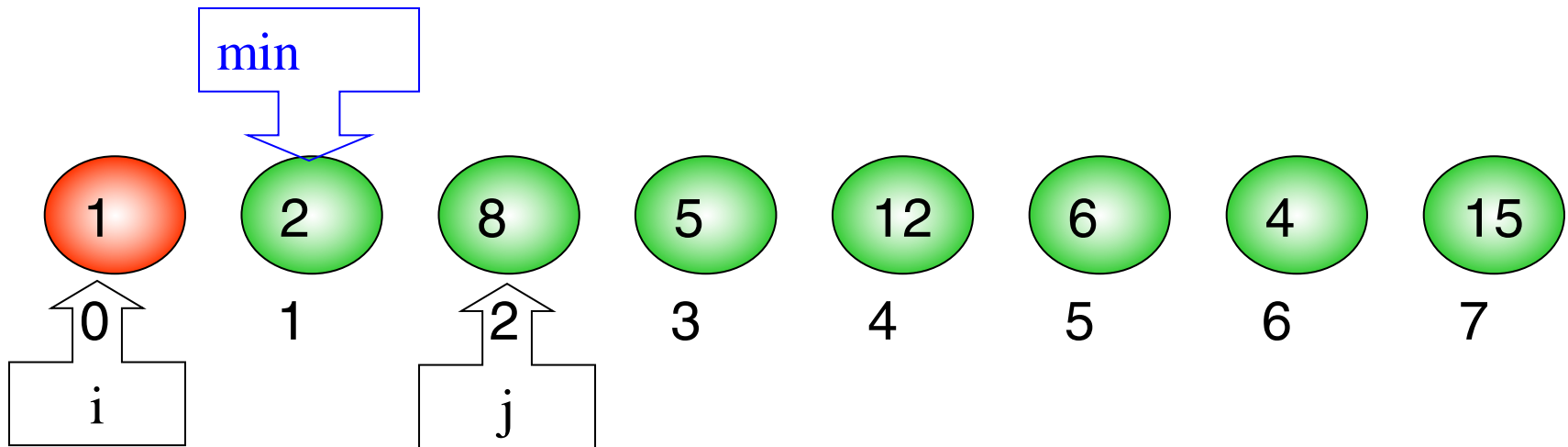
2. Chọn Trực Tiếp – Selection Sort

2.4. Minh Họa giải thuật Chọn Trực Tiếp

```
void SelectionSort(int a[],int n )
{
    int min,i,j;
    for(i=0;i<n-1;i++)
    {
        min = i;
        for(j = i+1; j <n ; j++)
            if (a[j] < a[min])
                min=j;
        Swap(a[min],a[i]);
    }
}
```

$i = 1; min=1; j=i+1=2;$
 $\Rightarrow A[min] = 2$

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



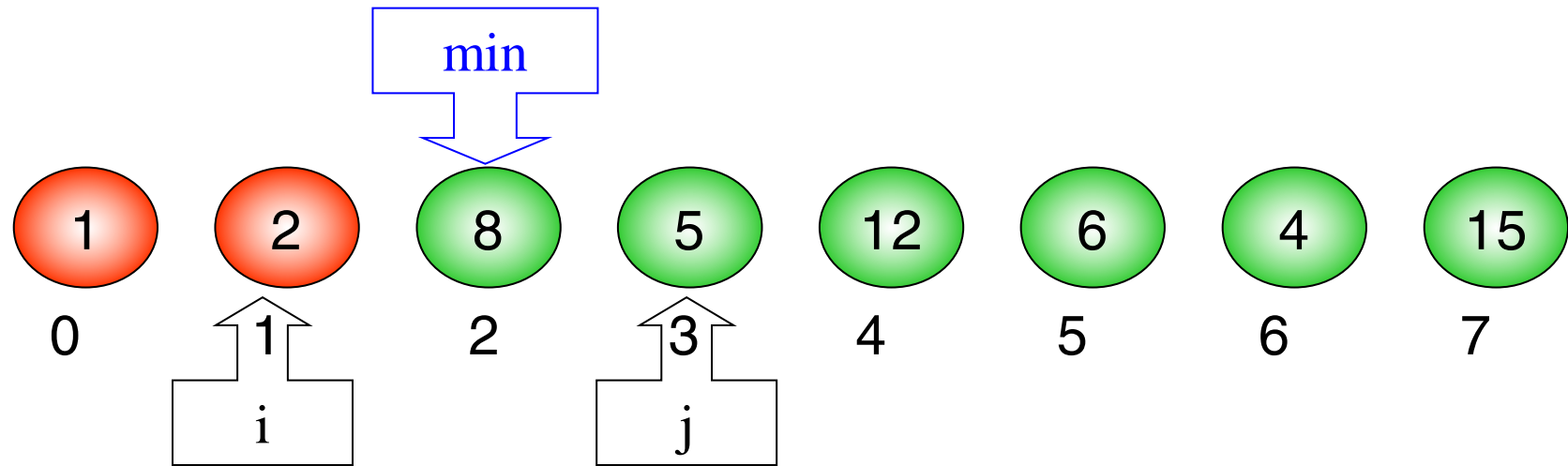
2. Chọn Trực Tiếp – Selection Sort

2.4. Minh Họa giải thuật Chọn Trực Tiếp

```
void SelectionSort(int a[],int n )
{
    int min,i,j;
    for(i=0;i<n-1;i++)
    {
        min = i;
        for(j = i+1; j <n ; j++)
            if (a[j ] < a[min])
                min=j;
        Swap(a[min],a[i]);
    }
}
```

$i = 2; min=3; j=6;$
 $\Rightarrow A[min] = 5 > A[j]= 4$
 $\Rightarrow min = 6$
 $\Rightarrow A[min] = 4$

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



2. Chọn Trực Tiếp – Selection Sort

2.5. Minh Họa giải thuật Chọn Trực Tiếp dưới dạng bảng

i	min	j	A[min]	A[j]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
0	0		12		12	2	8	5	1	6	4	15
0	<u>0 1</u>	1	<u>12 2</u>	2	12	2	8	5	1	6	4	15
0	1	2	2	8	12	2	8	5	1	6	4	15
0	1	3	2	5	12	2	8	5	1	6	4	15
0	<u>1 4</u>	4	<u>2 1</u>	1	12	2	8	5	1	6	4	15
0	4	5	1	6	12	2	8	5	1	6	4	15
0	4	6	1	4	12	2	8	5	1	6	4	15
0	4	7	1	15	<u>12 1</u>	2	8	5	<u>1 12</u>	6	4	15
1	1	2									4	15
1	1	3									4	15
1	1	4									4	15
1	1	5									4	15
1	1	6									4	15
1	1	7									4	15

```
void SelectionSort(int a[],int n )
{
    int min,i,j;
    for(i=0;i<n-1;i++)
    {
        min = i;
        for(j = i+1; j <n ; j++)
            if (a[j ] < a[min])
                min=j;
        Swap(a [min] ,a [i]);
    }
}
```

2. Chọn Trục Tiếp – Selection Sort

2.5. Minh Họa giải thuật Chọn Trục Tiếp dưới dạng bảng

i	min	j	A[min]	A[j]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
1		7	2		1	2	8	5	12	6	4	15
2	2 3	3	8 5	5	1	2	8	5	12	6	4	15
2	3	4	5	12	1	2	8	5	12	6	4	15
2	3	5	5	6	1	2	8	5	12	6	4	15
2	6	6	4	4	1	2	8	5	12	6	4	15
2	6	7	4	15	1	2	4	5	12	6	8	15
3	3	4	5	12	1	2	4	5	12	6	8	15
3	3	5	5	6	1	2	4	5	12	6	8	15
3	3	6	5	8	1	2	4	5	12	6	8	15
3	3	7	5	15	1	2	4	5	12	6	8	15
4	4 5	5	12 6	6	1	2	4	5	12	6	8	15
4	5	6	6	8	1	2	4	5	12	6	8	15
4	5	7	6	15	1	2	4	5	12 6	6 12	8	15
5	5 6	6	6 8	8	1	2	4	5	6	12	8	15
5	6	7	8	15	1	2	4	5	6	12 8	8 12	15
6	6	7	12	15	1	2	4	5	6	8	12	15

2. Chọn Trực Tiếp – Selection Sort

2.6. Đánh giá giải thuật Chọn Trực Tiếp

- Đối với giải thuật chọn trực tiếp, có thể thấy rằng ở lượt thứ i , bao giờ cũng cần $(n-i)$ lần so sánh để xác định phần tử nhỏ nhất hiện hành.
- Số lượng phép so sánh này không phụ thuộc vào tình trạng của dãy số ban đầu, do vậy trong mọi trường hợp có thể kết luận:

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	$n(n-1)/2$	0
Xấu nhất	$n(n-1)/2$	$3n(n-1)/2$

NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ vi-xii

3. CHÈN TRỰC TIẾP (*Insertion Sort*)

3.1. Ý tưởng

- Giả sử có một dãy a_0, a_1, \dots, a_{n-1} trong đó i phần tử đầu tiên a_0, a_1, \dots, 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, \dots, 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 $a_{k-1} < a_i < a_k$ ($1 \leq k \leq i$).

3. Chèn Trực Tiếp – Insertion Sort

3.2. Giải thuật

- Bước 1: $i = 1$; //giả sử có đoạn $a[1]$ đã được sắp
- Bước 2: Tìm vị trí pos thích hợp trong đoạn $a[1]$ đến $a[i-1]$ để chèn $a[i]$ vào
$$x = a[i];$$
- Bước 3: Dời chỗ các phần tử từ $a[pos]$ đến $a[i-1]$ sang phải 1 vị trí để dành chỗ cho $a[i]$
- Bước 4: $a[pos] = x$; //có đoạn $a[1]..a[i]$ đã được sắp
- Bước 5: $i = i+1$;
 Nếu $i < n$: Lặp lại Bước 2
 Ngược lại : Dừng

3. Chèn Trực Tiếp – Insertion Sort

3.3. Cài đặt

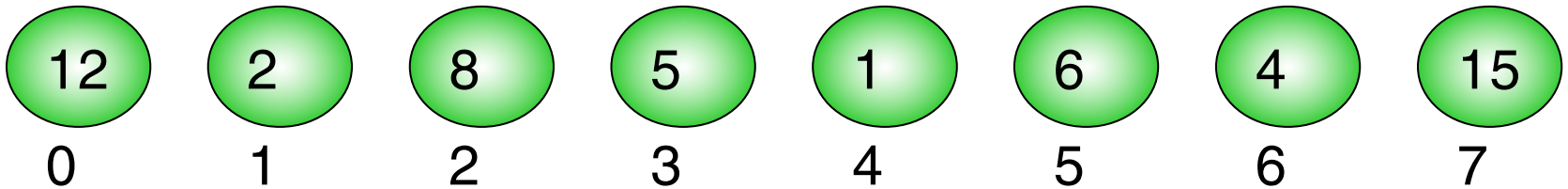
```
void InsertionSort(int a[], int n )
{
    int pos, i;
    int x; /* lưu giá trị a[i] tránh bị ghi đè khi dời chỗ
                                                    các phần tử. */

    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
    }
}
```

3. Chèn Trực Tiếp – Insertion Sort

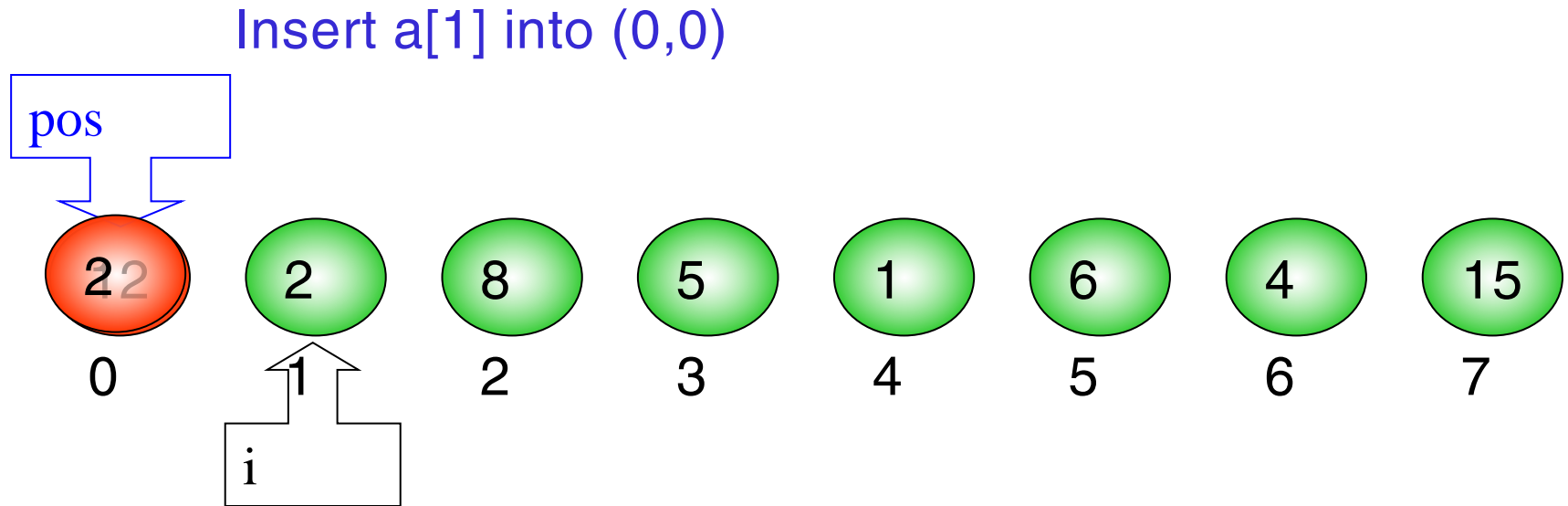
3.4. Minh Họa giải thuật Insertion Sort



```
void InsertionSort(int a[], int n )
{
    int pos, i;
    int x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        }
        a[pos+1] = x;
    }
}
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh Họa giải thuật Insertion Sort

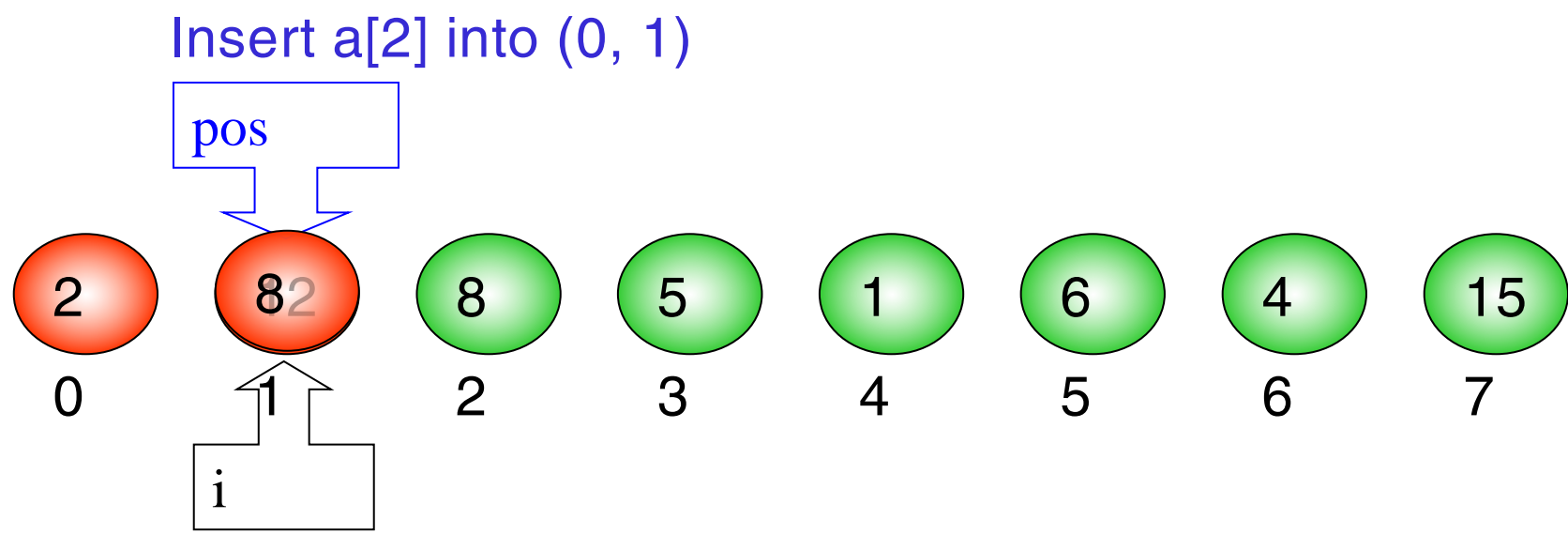


X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort

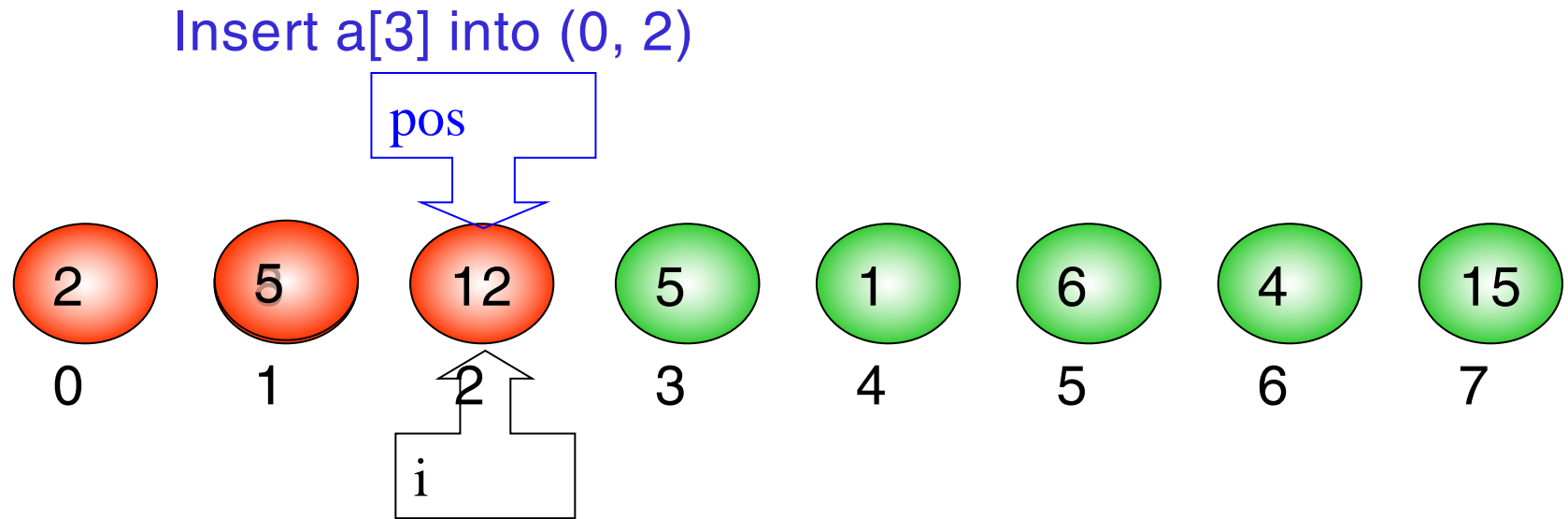


X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```


3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort

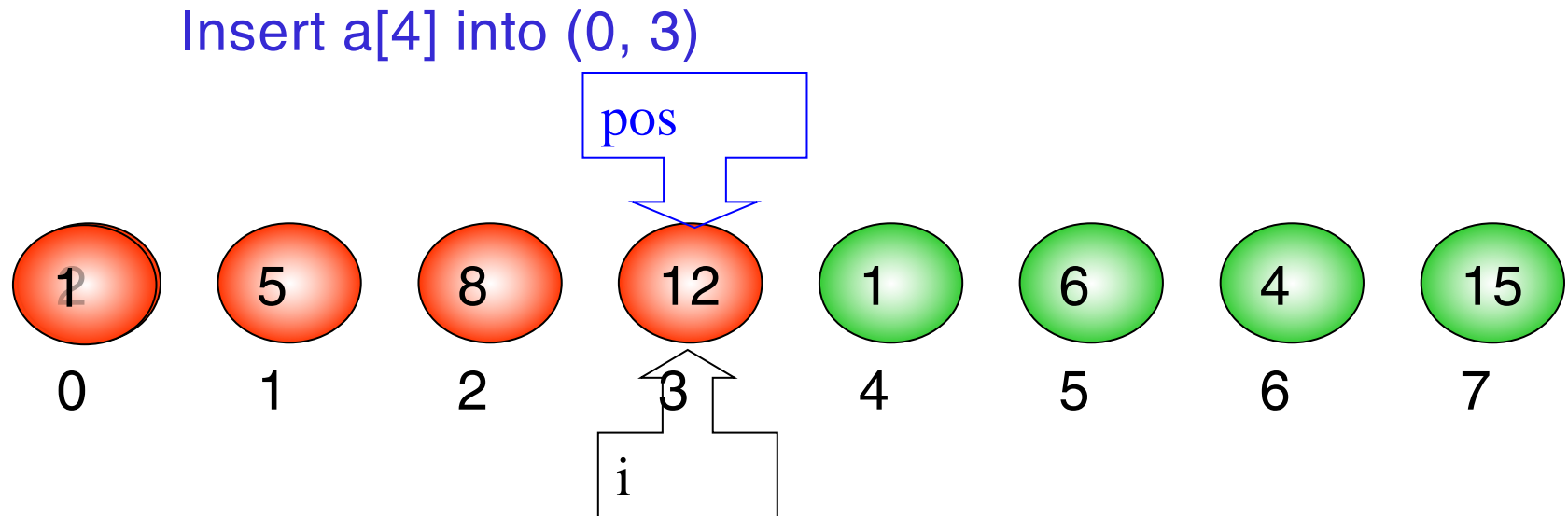


X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort



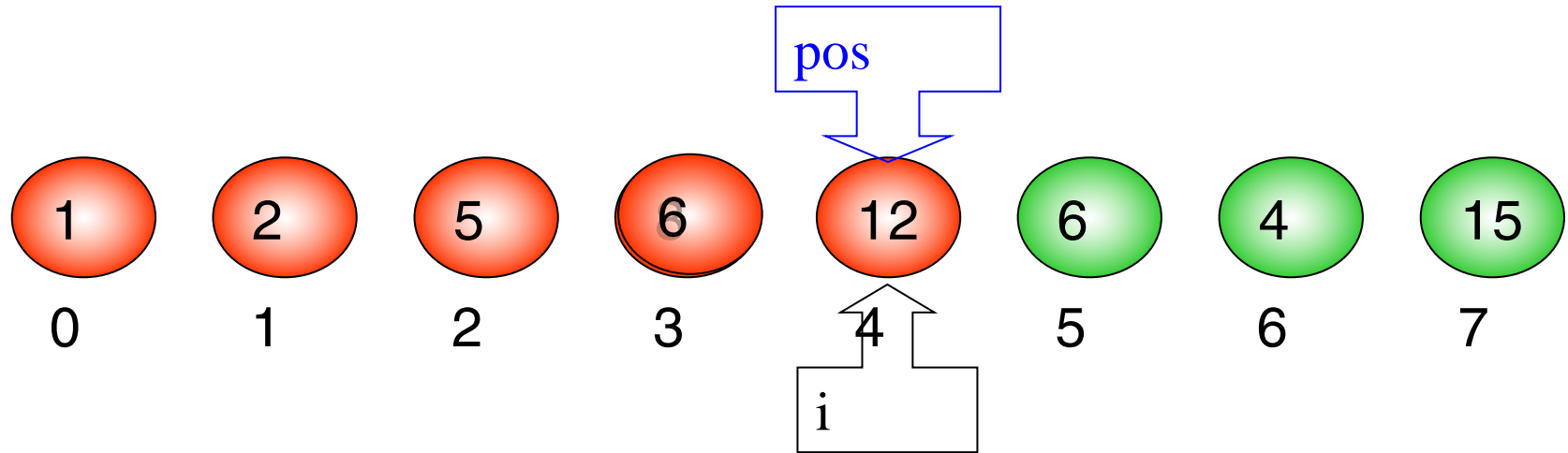
X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort

Insert a[5] into (0, 4)



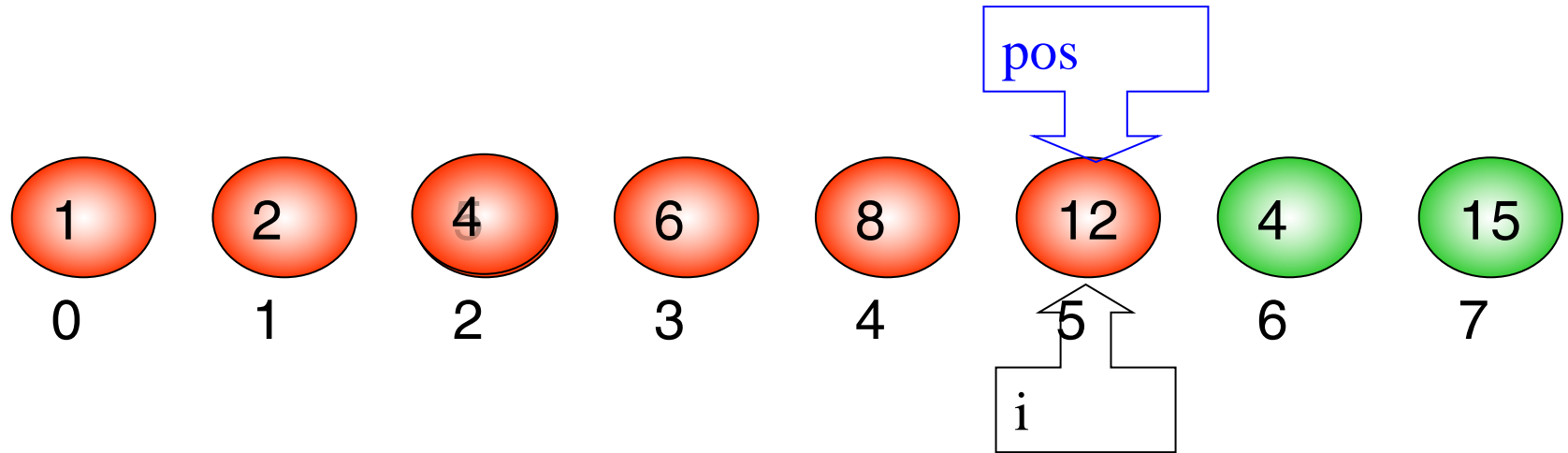
X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort

Insert a[6] into (0, 5)



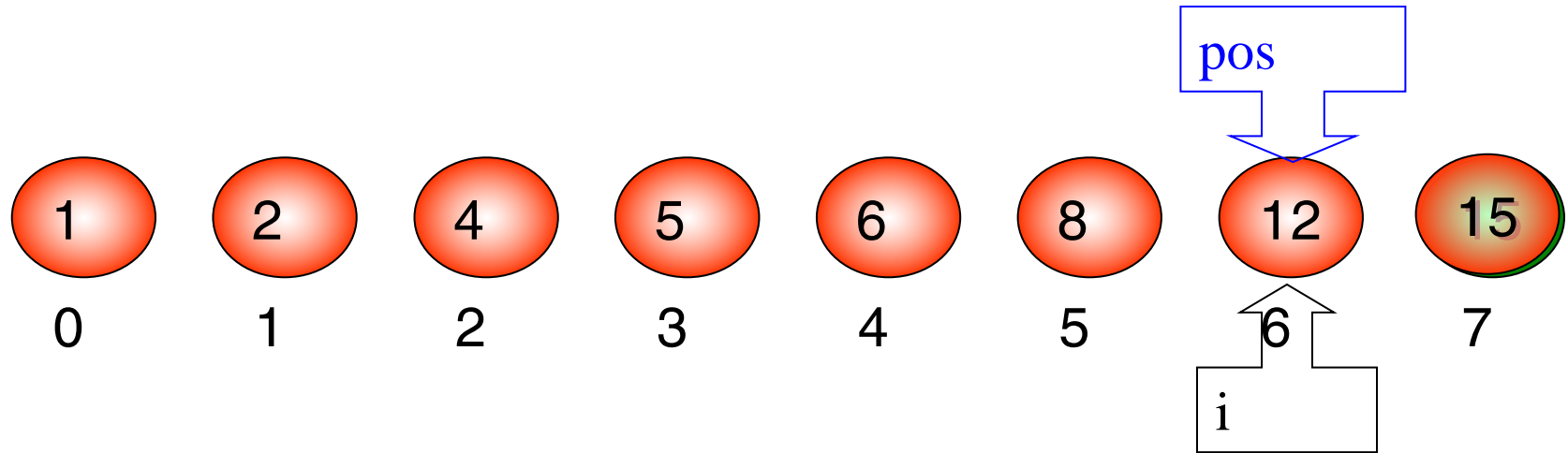
X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh họa giải thuật Insertion Sort

Insert a[8] into (0, 6)

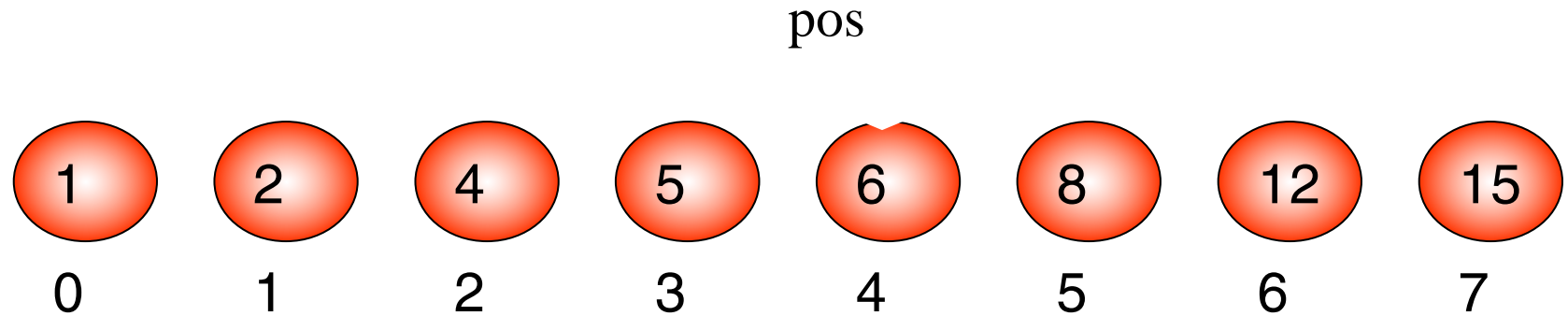


X

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.4. Minh Họa Giải thuật Insertion Sort



```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0) && (a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;
        } //end while
        a[pos+1] = x;
    } //end for
} //end function
```

3. Chèn Trực Tiếp – Insertion Sort

3.5. Minh họa giải thuật Insertion Sort dưới dạng bảng

i	Pos	a[Pos]	x=a[i]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
				12	2	8	5	1	6	4	15
1	0/-1	12/2	2	12	<u>2/12</u>	8	5	1	6	4	15
				Kết thúc vòng lặp while do Pos = -1 < 0							
				<u>12/2</u>	12	8	5	1	6	4	15
2	1	12	8	2	12	<u>8/12</u>	5	1	6	4	15
2	0	2	8	Kết thúc vòng lặp while do a[Pos]=2 !=> x=8							
				2	<u>12/8</u>	12	5	1	6	4	15
3	2	12	5	2	8	12	<u>5/12</u>	1	6	4	15
3	1	8	5	2	8	<u>12/8</u>	12	1	6	4	15
3	0	2	5	Kết thúc vòng lặp while do a[Pos]=2 !=> x=5							
				2	<u>8/5</u>	8	12	1	6	4	15
4	3	12	1								
4	2	8	1								
4	1	5	1								
4	0	2	1								
4	-1										

```
void InsertionSort(int a[], int n )
{
    int pos, i, x;
    for(i=1; i<n; i++)
    {
        x = a[i]; pos = i-1;
        while((pos >= 0)&&(a[pos] > x))
        {
            a[pos+1] = a[pos];
            pos--;           //end while
        }
        a[pos+1] = x;       //end for
    } //end function
}
```

3. Chèn Trực Tiếp – Insertion Sort

3.5. Minh họa giải thuật Insertion Sort dưới dạng bảng

i	Pos	a[Pos]	x=a[i]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
				1	2	5	8	12	6	4	15
5	4	12	6	1	2	5	8	12	<u>6/12</u>	4	15
5	3	8	6	1	2	5	8	<u>12/8</u>	12	4	15
5	2	5	6	Kết thúc vòng lặp while do a[Pos]=5 !> x=6							
				1	2	5	<u>8/6</u>	8	12	4	15
6	5	12	4	1	2	5	6	8	12	<u>4/12</u>	15
	4	8	4	1	2	5	6	8	<u>12/8</u>	12	15
	3	6	4	1	2	5	6	<u>8/6</u>	8	12	15
	2	5	4	1	2	5	<u>6/5</u>	6	8	12	15
	1	2	4	Kết thúc vòng lặp while do a[Pos]=2 !> x=4							
				1	2	<u>5/4</u>	5	6	8	12	15
7	6	12	15	Kết thúc vòng lặp while do a[Pos]=12 !> x=15							
8	Kết thúc vòng lặp for do i (=8) !< n (=8)										
Mảng sau sắp xếp				1	2	4	5	6	8	12	15

3.6. Đánh giá giải thuật

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	$\sum_{i=1}^{n-1} (i - 1) = \frac{n(n - 1)}{2}$	$\sum_{i=1}^{n-1} (i + 2) = \frac{(n + 1)(n + 2)}{2} - 3$

NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ **vi-xii**

4. NỔI BỌT (*Bubble Sort*)

4.1. Ý tưởng

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

4.2. Các bước của giải thuật Bubble Sort

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

Trong khi $(j > i)$ thực hiện:

Nếu $a[j] < a[j - 1]$

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

$j = j - 1$;

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

Nếu $i = N - 1$: Hết dãy. Dừng

Ngược lại : Lặp lại Bước 2.

4. Nổi Bật – Bubble Sort

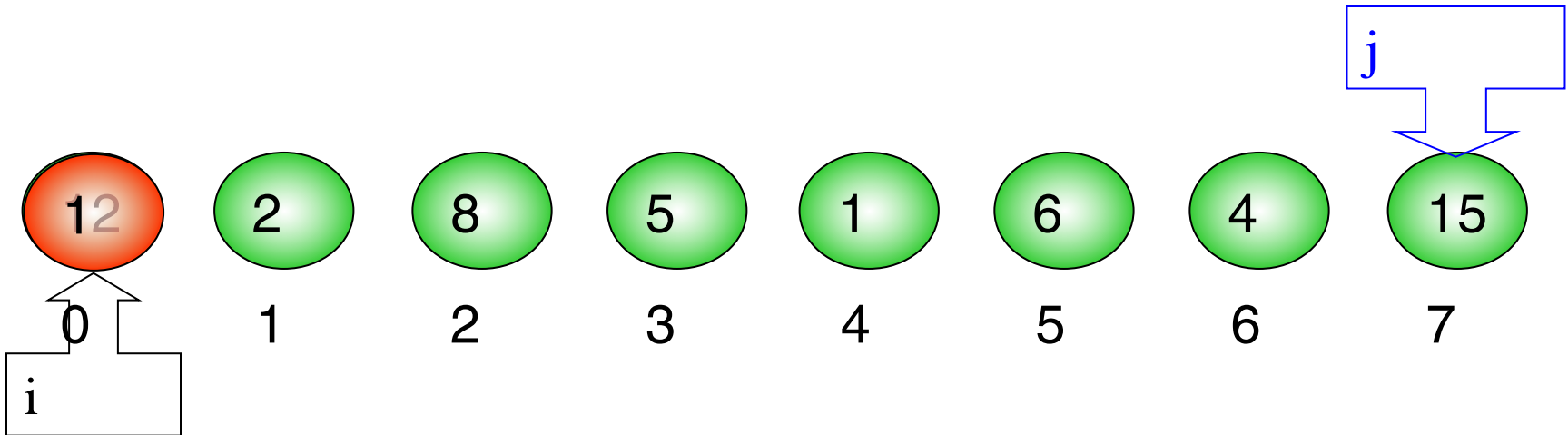
4.3. Cài Đặt giải thuật Buble Sort

```
void BubbleSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = n-1; j > i ; j --)
            /* phát hiện nghịch thế giữa 2
               phần tử liền kề  $\Rightarrow$  đổi chỗ */
            if (a[j] < a[j-1])
                Swap(a[j], a[j-1]);
}
```

4. Nỗ Bọt – Bubble Sort

4.4. Minh Họa giải thuật

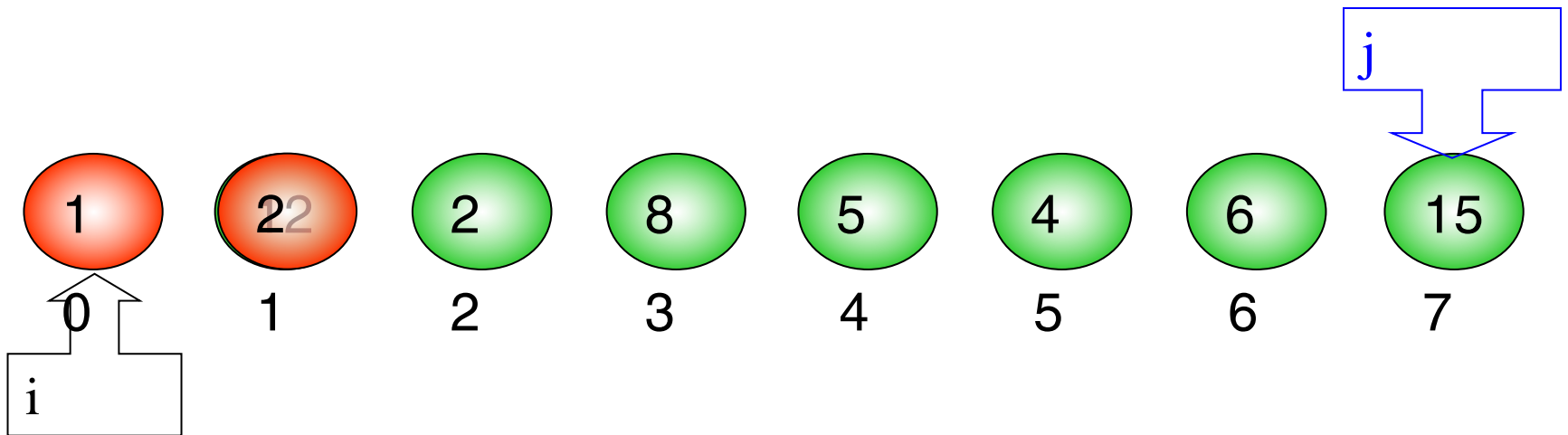
```
void BubbleSort(int a[],int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j =n-1; j >i ; j --)
            if(a[j]< a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bật – Bubble Sort

4.4. Minh Họa giải thuật

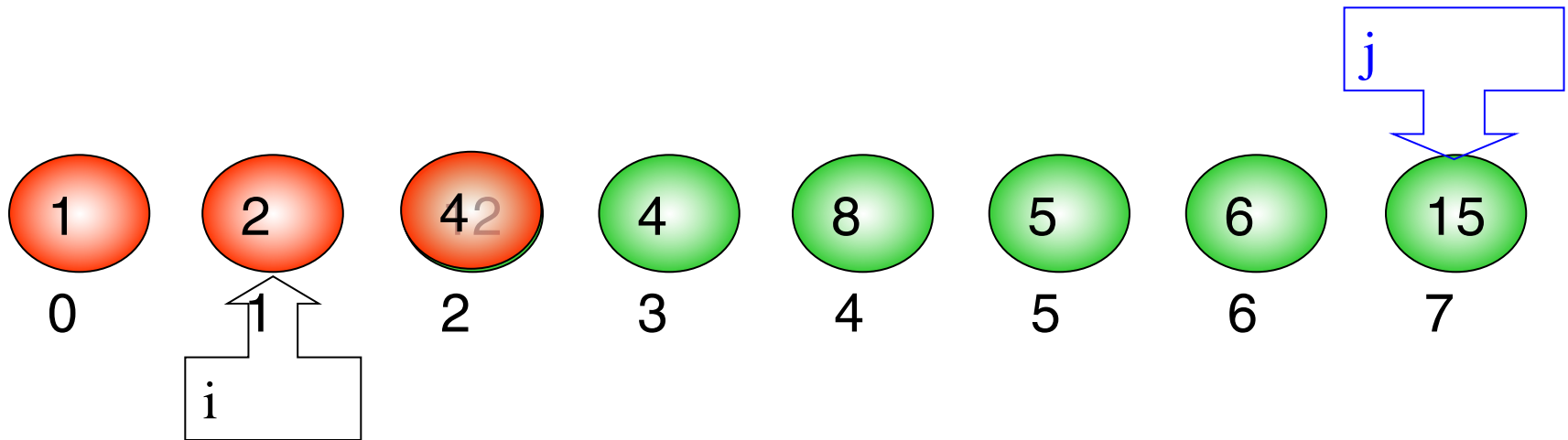
```
void BubbleSort(int a[],int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j =n-1; j >i ; j --)
            if(a[j]< a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bật – Bubble Sort

4.4. Minh Họa giải thuật

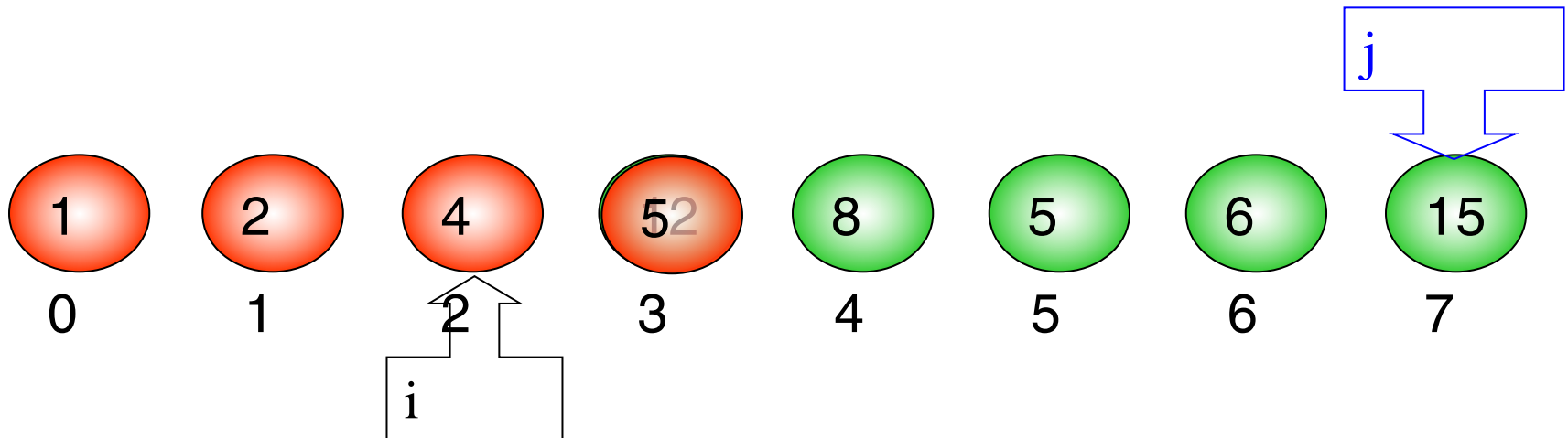
```
void BubbleSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = n-1; j > i ; j --)
            if (a[j] < a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bật – Bubble Sort

4.4. Minh Họa giải thuật

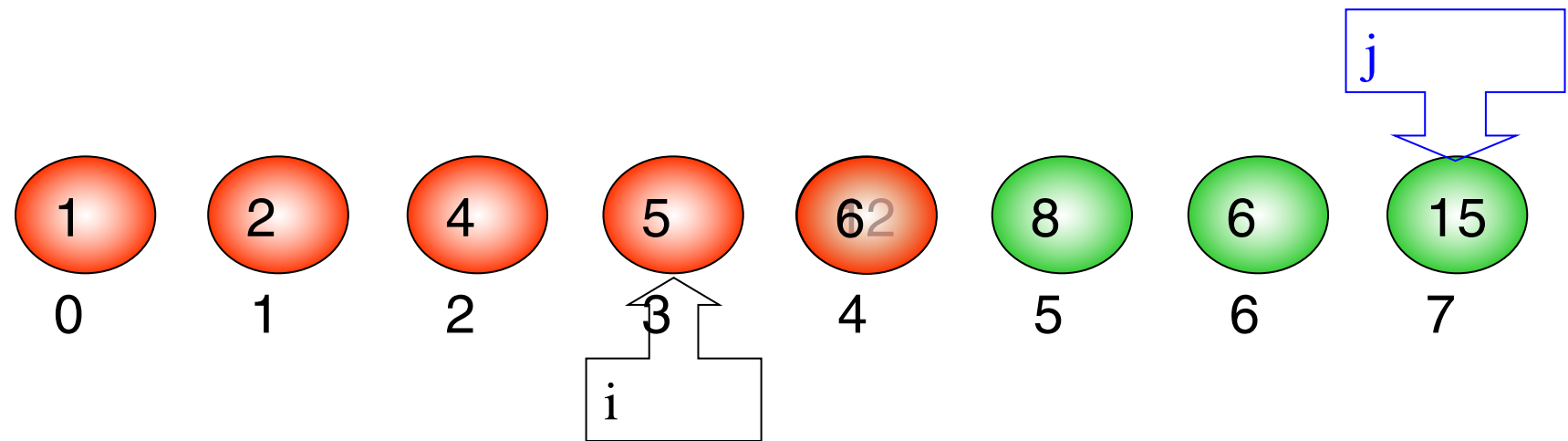
```
void BubbleSort(int a[], int n)
{
    int i, j;
    for (i = 0 ; i < n-1 ; i++)
        for (j = n-1; j > i ; j --)
            if (a[j] < a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bọt – Bubble Sort

4.4. Minh Họa giải thuật

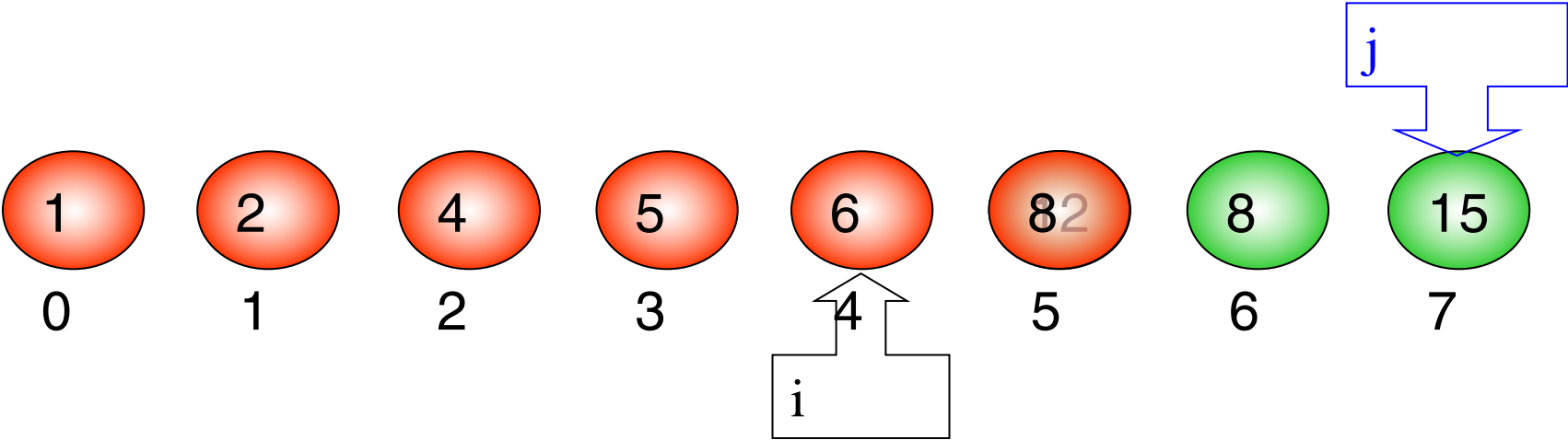
```
void BubbleSort(int a[],int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j =n-1; j >i ; j --)
            if(a[j]< a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bọt – Bubble Sort

4.4. Minh Họa giải thuật

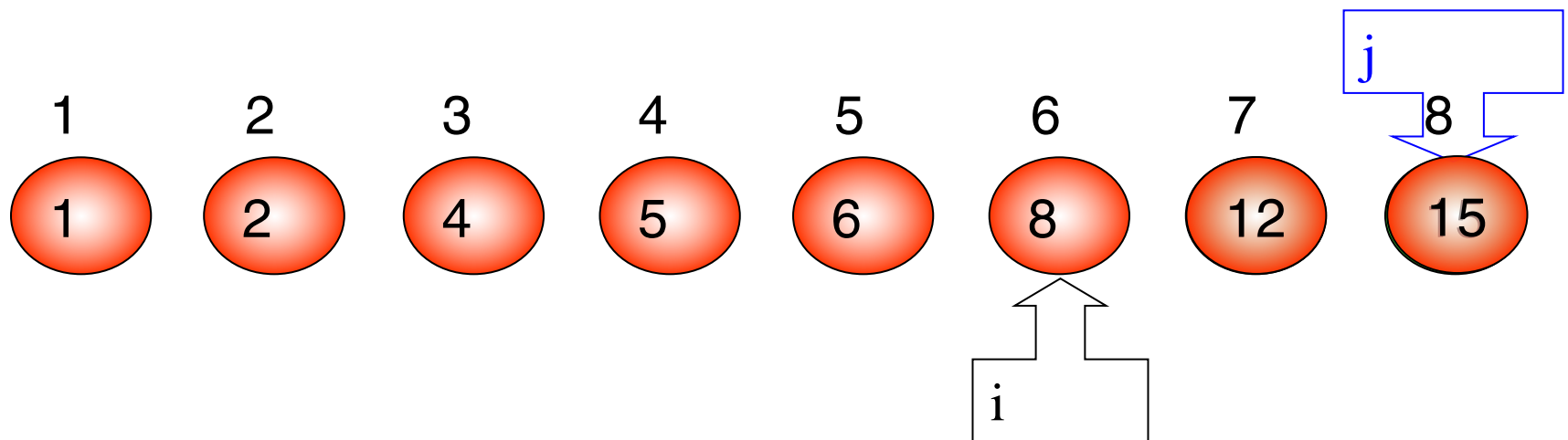
```
void BubbleSort(int a[],int n)
{
    int i, j;
    for (i = 0 ; i<n-1 ; i++)
        for (j =n-1; j >i ; j --)
            if(a[j]< a[j-1])
                Swap(a[j], a[j-1]);
}
```



4. Nổi Bọt – Bubble Sort

4.4. Minh Họa giải thuật

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



4. Nổi Bọt – Bubble Sort

4.5. Minh Họa giải thuật Buble Sort dưới dạng bảng

i	j	A[j-1]	A[j]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
				12	2	8	5	1	6	4	15
0	7	4	15	12	2	8	5	1	6	4	15
0	6	6/4	4/6	12	2	8	5	1	6/4	4/6	15
0	5	1	4	12	2	8	5	1	4	6	15
0	4	5/1	1/5	12	2	8	5/1	1/5	4	6	15
0	3	8/1	1/8	12	2	8/1	1/8	5	4	6	15
0	2	2/1	1/2	12	2/1	1/2	8	5	4	6	15
0	1	12	1	12/1	1/12	2	8	5	4	6	15
1	7	6	15	1	12	2	8	5	4	6	15
1	6	4	6	1	12	2	8	5	4	6	15
1	5	5/4	4/5	1	12	2	8	5/4	4/5	6	15
1	4	8/4	4/8	1	<pre>void BubbleSort(int a[],int n) { int i,j; for(i = 0 ; i<n-1 ; i++) for(j =n-1; j >i ; j --)</pre>						
1	3	2	4	1							
1	2	12/2	2/12	1							

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

4. Nội Bọt – Bubble Sort

4.5. Minh Họa giải thuật Buble Sort dưới dạng bảng

i	j	A[j-1]	A[j]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
				1	2	12	4	8	5	6	15
2	7	6	15	1	2	12	4	8	5	6	15
2	6	5	6	1	2	12	4	8	5	6	15
2	5	8/5	5/8	1	2	12	4	8/5	5/8	6	15
2	4	4	5	1	2	12	4	5	8	6	15
2	3	12/4	4/12	1	2	12/4	4/12	5	8	6	15
3	7	6	15	1	2	4	12	5	8	6	15
3	6	8/6	6/8	1	2	4	12	5	8/6	6/8	15
3	5	5	6	1	2	4	12	5	6	8	15
3	4	12/5	5/12	1	2	4	12/5	5/12	6	8	15
4	7	8	15	1	2	4	5	12	6	8	15
4	6	6	8	1	2	4	5	12	6	8	15
4	5	12/6	6/12	1	2	4	5	12/6	6/12	8	15
5	7	8	15	1	2	4	5	6	12	8	15
5	6	12/8	8/12	1	2	4	5	6	12/8	8/12	15
6	7	12	15	1	2	4	5	6	8	12	15

4.6. *Đánh giá giải thuật Buble Sort*

- Trong mọi trường hợp, số phép so sánh là:

$$(n-1) + (n-2) + \dots + 1 = n(n-1)/2 = \mathbf{O(n^2)}$$

- Số phép hoán vị:

- Trường hợp xấu nhất : $\mathbf{n(n-1)/2}$

- Trường hợp tốt nhất : $\mathbf{0}$

NỘI DUNG

- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ **vi-xii**

5. Shaker Sort

5.1. Ý tưởng

- Giải thuật *Shaker Sort* là cải tiến của *Bubble Sort* bằng cách thực hiện 2 lượt đi và về cùng lúc. Lượt đi sẽ đẩy các phần tử nhỏ về đầu dãy, lượt về sẽ đẩy các phần tử lớn về cuối dãy.

5. Shaker Sort

5.2. Giải thuật

- **Bước 1:**

Khởi tạo: $L=0$; $k = n-1$; //từ L đến R là đoạn cần được sắp xếp
 $R=n-1$;

- **Bước 2:**

$j = R$; //đẩy phần tử nhỏ nhất về đầu mảng

Trong khi ($j > L$)

 Nếu $a[j] < a[j-1]$:

 hoán vị $a[j], a[j-1]$;

$k=j$; //lưu lại nơi xảy ra hoán vị

$j = j-1$;

$L=k$; /*ghi nhận lại vị trí L xảy ra hoán vị sau cùng để làm cơ sở thu hẹp đoạn 1 đến r */

- **Bước 3:**

$j=L$; // đẩy phần tử lớn về cuối mảng

 Trong khi ($j < r$)

 Nếu $a[j] > a[j+1]$:

 hoán vị $a[j], a[j+1]$;

$k=j$; //lưu lại nơi xảy ra hoán vị

$j=j+1$;

$R=k$; //loại bớt phần tử đã có thứ tự ở cuối dãy

- **Bước 4:**

 Nếu $L < R$: lặp lại bước 2.

5. Shaker Sort

5.3. Cài đặt

```
void ShakerSort(int a[], int n)
{
    int i, k, left, right;
    k = 0;
    left = 0;
    right = n - 1;
    while (left < right)
    {
        for (i = right; i > left; i--)
            if (a[i-1] > a[i])
            {
                Swap(a[i-1], a[i]); // Hoan vi a[i], a[i - 1]
                k = i; //Biến k đánh dấu để bỏ qua đoạn đã có thứ tự
            }
        left = k;
        for (i = left; i < right; i++)
            if (a[i] > a[i + 1])
            {
                Swap(a[i], a[i + 1]);
                k = i;
            }
        right = k;
    }
}
```

5. Shaker Sort

5.4. Minh Họa giải thuật Shaker Sort dưới dạng bảng

L	R	k	i	A[i-1]	A[i]	A[i+1]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
0	7	0					12	2	8	5	1	6	4	15
0	7	0	7	4	15		12	2	8	5	1	6	4	15
0	7	<u>6</u>	6	6	4		12	2	8	5	1	<u>4</u>	<u>6</u>	15
0	7	6	5	1	4		12	2	8	5	1	4	6	15
0	7	<u>4</u>	4	5	1		12	2	8	<u>1</u>	<u>5</u>	4	6	15
0	7	<u>3</u>	3	8	1		12	2	<u>1</u>	<u>8</u>	5	4	6	15
0	7	<u>2</u>	2	2	1		12	<u>1</u>	<u>2</u>	8	5	4	6	15
<u>1</u>	7	<u>1</u>	1	12	1		<u>1</u>	<u>12</u>	2	8	5	4	6	15
1	7	1	1		12	2	1	<u>2</u>	<u>12</u>	8	5	4	6	15
1	7	<u>2</u>	2		12	8	1	2	<u>8</u>	<u>12</u>	5	4	6	15
1	7	<u>3</u>	3		12	5	1	2	8	<u>5</u>	<u>12</u>	4	6	15
1	7	<u>4</u>	4		12	4	1	2	8	5	<u>4</u>	<u>12</u>	6	15
1	7	<u>5</u>	5		12	6	1	2	8	5	4	<u>6</u>	<u>12</u>	15
1	<u>5</u>	5	6		12	15	1	2	8	5	4	6	12	15

5. Shaker Sort

5.4. Minh Họa giải thuật Shaker Sort dưới dạng bảng

L	R	k	i	A[i-1]	A[i]	A[i+1]	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
1	5	5					1	2	8	5	4	6	12	15
1	5	5	5	6	12		1	2	8	5	4	6	12	15
1	5	<u>4</u>	4	5	4		1	2	8	<u>4</u>	<u>5</u>	6	12	15
1	5	<u>3</u>	3	8	4		1	2	<u>4</u>	<u>8</u>	5	6	12	15
<u>3</u>	5	3	2				1	2	4	8	5	6	12	15
3	5	<u>3</u>	3		8	5	1	2	4	<u>5</u>	<u>8</u>	6	12	15
4	<u>4</u>	<u>4</u>	4		8	6	1	2	4	5	<u>6</u>	<u>8</u>	12	15

NỘI DUNG

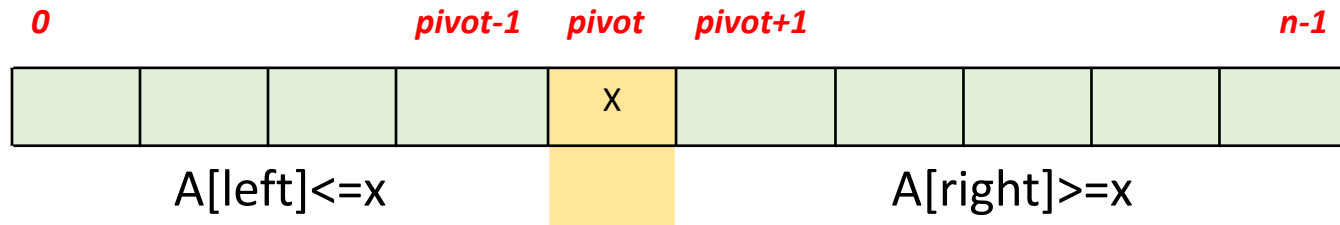
- i. Đổi chỗ trực tiếp – *Interchange Sort*
- ii. Chọn trực tiếp – *Selection Sort*
- iii. Chèn trực tiếp – *Insertion Sort*
- iv. Nổi bọt – *Bubble Sort*
- v. Sắp xếp nhanh - *Quick Sort*
- vi. Giải thuật lắc - *Shaker Sort* (*)
- vii. Sắp xếp trộn trực tiếp - *Merge Sort* (*)
- viii. Sắp xếp với độ dài bước giảm dần - *Shell Sort* (*)
- ix. Sắp xếp vun đống - *Heap Sort* (*)
- x. Sắp xếp theo cơ số - *Radix Sort* (*)
- xi. Sắp xếp bằng đếm phân khối - *Distribution Counting* (*)
- xii. Sắp xếp tuyến tính – *FlashSort* (*)

(*) Sinh viên tự nghiên cứu các giải thuật từ vi-xii

6. Giải thuật Quick Sort

6.1. Ý tưởng

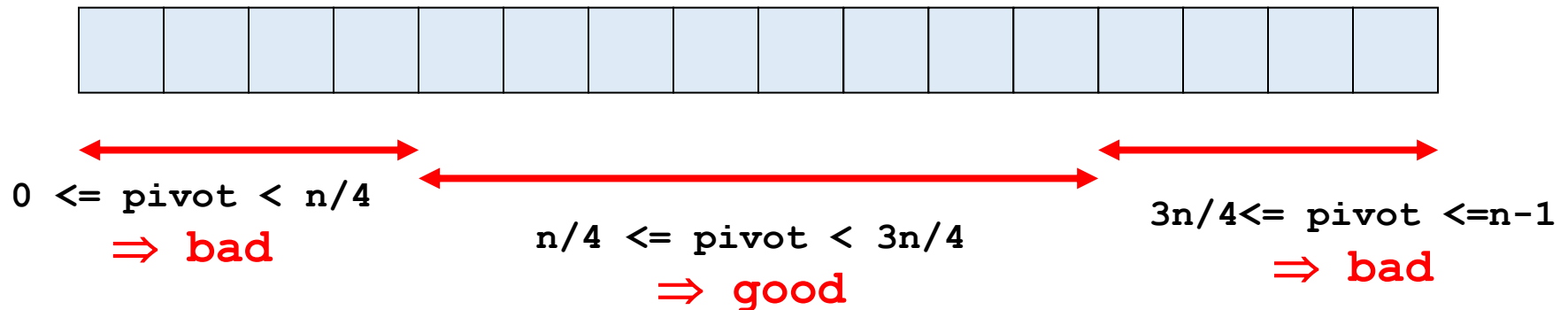
- Chọn một phần tử bất kỳ trong danh sách (giả sử là phần tử giữa) gọi là phần tử làm mốc (**pivot element**), xác định vị trí của phần tử này trong danh sách gọi là vị trí mốc.
- Tiếp theo phân hoạch các phần tử còn lại trong danh sách cần sắp xếp sao cho các phần tử từ vị trí **0** đến vị trí **pivot-1** đều có nội dung nhỏ hơn hoặc bằng phần tử mốc, các phần tử từ vị trí **pivot+1** đến **n-1** đều có nội dung lớn hơn hoặc bằng phần tử mốc.



- Quá trình lại tiếp tục như thế với hai danh sách con từ vị trí **0** đến vị trí **pivot-1** và từ vị trí **pivot+1** đến vị trí **n-1**.
- Kết quả của quá trình thực hiện sẽ được danh sách đã có thứ tự.

6.2. Cách chọn phần tử làm mốc (pivot element)

- Có thể chọn phần tử làm mốc bằng 1 trong các cách sau:
 - Chọn phần tử đứng **đầu** hoặc đứng **cuối**.
 - Chọn phần tử đứng **giữa** danh sách.
 - Chọn phần tử **ngẫu nhiên** làm phần tử mốc (\Rightarrow ưu điểm tránh được các trường hợp đặc biệt)
- Theo thống kê:



6.3. Hiệu quả của thuật toán phụ thuộc vào việc chọn giá trị của phần tử mốc (pivot)

- Trường hợp tốt nhất:

- Mỗi lần phân hoạch đều chọn được phần tử làm mốc có giá trị lớn hơn hay bằng nửa số phần tử và nhỏ hơn hay bằng nửa số phần tử còn lại. Khi đó dãy được phân hoạch thành hai phần bằng nhau
- Cần $\log_2(n)$ lần phân hoạch thì sắp xếp xong.
- Mỗi lần phân hoạch cần duyệt qua n phần tử. Vậy độ phức tạp trong trường hợp tốt nhất thuộc **$O(n\log_2(n))$** .

6.3. Hiệu quả của thuật toán phụ thuộc vào việc chọn giá trị của phần tử mốc (pivot)

- Trường hợp xấu nhất:

- Mỗi lần phân hoạch đều chọn phải phần tử có giá trị cực đại hoặc cực tiểu làm mốc. Khi đó dãy bị phân hoạch thành hai phần không đều: một phần chỉ có một phần tử, phần còn lại có $n-1$ phần tử.
- Cần n lần phân hoạch mới sắp xếp xong. Vậy độ phức tạp trong trường hợp xấu nhất thuộc **$O(n^2)$** .

6. Giải thuật Quick Sort

6.4. Giải thuật

- Bước 0: //chọn giá trị pivot tùy ý sao cho $0 \leq \text{pivot} \leq n-1$
 $\text{pivot} = 0$; // chọn pivot là vị trí đầu của mảng
 $\text{left} = \text{vị trí đầu mảng}$
 $\text{right} = \text{vị trí cuối mảng}$
Kết thúc; //mảng đã được sắp xếp
- Bước 1: Nếu $\text{left} \geq \text{right}$ // mảng có ít hơn 2 phần tử
Kết thúc; // mảng đã được sắp xếp
- Bước 2: Phân hoạch dãy $a_{\text{left}} \dots a_{\text{right}}$ thành các đoạn:
 $a_{\text{left}} \dots a_j, a_{j+1} \dots a_{i-1}, a_i \dots a_{\text{right}}$
Đoạn 1 $\leq x$
Đoạn 2: $a_{j+1} \dots a_{i-1} = x$
Đoạn 3: $a_i \dots a_{\text{right}} \geq x$
- Bước 3: Sắp xếp đoạn 1: $a_{\text{left}} \dots a_j$
- Bước 4: Sắp xếp đoạn 3: $a_i \dots a_{\text{right}}$

6. Giải Thuật Quick Sort

6.4. Giải thuật

- Bước 1: /* tham số của hàm là mảng 1 chiều (a), vị trí đầu tiên của mảng (left), vị trí cuối của mảng (right) */
 - $L = \text{left}; R = \text{right};$
 - $p = 0;$ /*Trong minh họa này, chọn pivot là vị trí đầu của mảng đang xét*/
 - $x = a[p];$
- Bước 2: Phát hiện và đổi chỗ cặp phần tử $a[L], a[R]$ nằm sai chỗ:
 - Bước 2a: Trong khi $(a[L] < x)$ $L++;$
 - Bước 2b: Trong khi $(a[R] > x)$ $R--;$
 - Bước 2c: Nếu $L < R \Rightarrow \text{Đổi chỗ}(a[L], a[R]);$
Nếu $L \leq R \Rightarrow L++ ; R--;$
- Bước 3: Nếu $L < R$: Lặp lại Bước 2. Ngược lại: Dừng
- Bước 4: $\text{Đổi chỗ}(a[L], a[R]);$
- Bước 5: Sắp xếp đoạn 1: $a_{\text{left}} \dots A_R$
- Bước 6: Sắp xếp đoạn 3: $a_L \dots a_{\text{right}}$

6. Giải Thuật Quick Sort

6.5. Cài đặt

```
void QuickSort(int a[], int left, int right)
{   int L, R, x, pivot=left; /* chọn mốc là phần tử đầu
    tiên trong mảng hiện tại*/
    x = a[pivot]; L = left; R = right;
    while(L <= R)
    {   while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {   if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    //nếu mảng bên TRÁI của pivot có nhiều hơn 1 phần tử
    if(left<R) QuickSort(a, left, R);
    //nếu mảng bên PHẢI của pivot có nhiều hơn 1 phần tử
    if(L<right) QuickSort(a, L, right);
}
```

6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort

- Cho mảng 1 chiều chứa các số nguyên, gồm 9 phần tử

40	20	10	80	60	50	7	30	100
----	----	----	----	----	----	---	----	-----

- Chọn phần tử mốc (pivot element) cho việc phân hoạch
Có nhiều cách chọn phần tử mốc. Trong ví dụ này lấy phần tử đầu tiên của mảng làm phần tử mốc

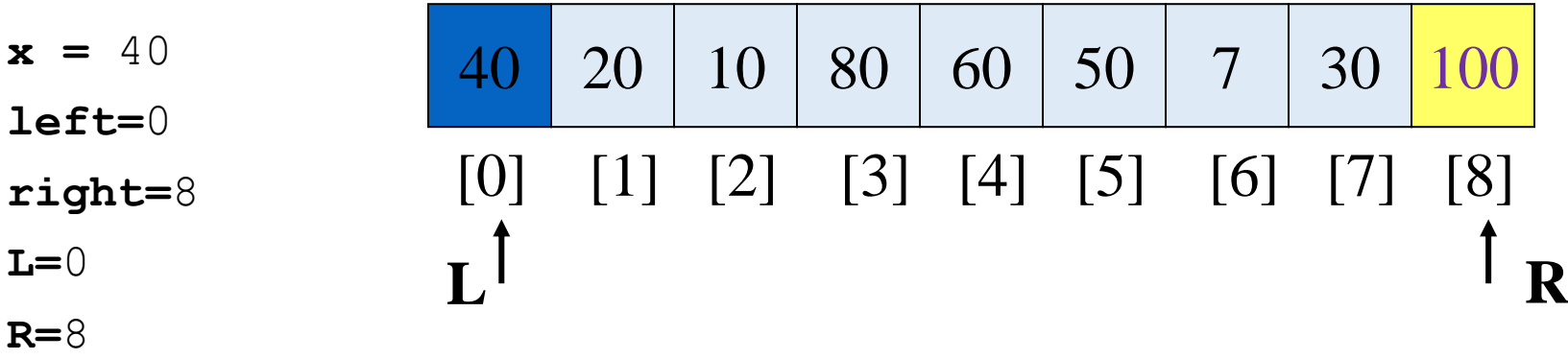
40	20	10	80	60	50	7	30	100
----	----	----	----	----	----	---	----	-----

6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort

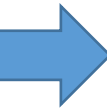
➡

```
void QuickSort(int a[], int left, int right)
{
    int pivot=left, x = a[left]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```



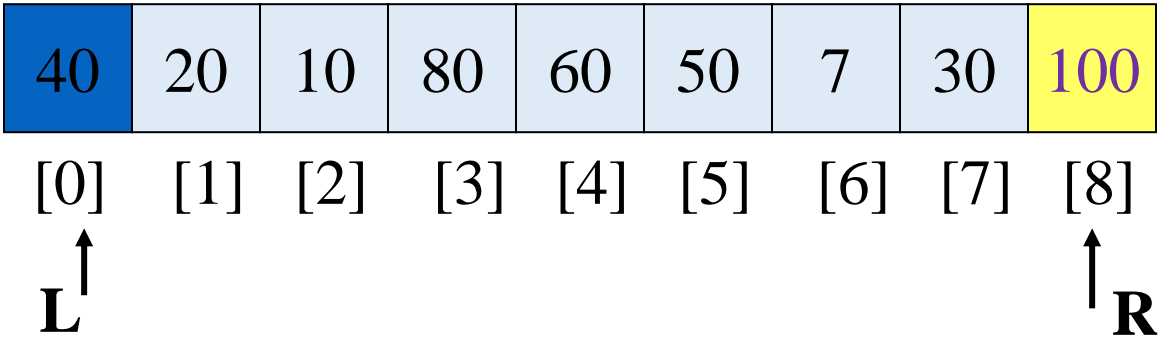
6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort



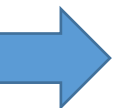
```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```

x = 40
left=0
right=8
L=0
R=8



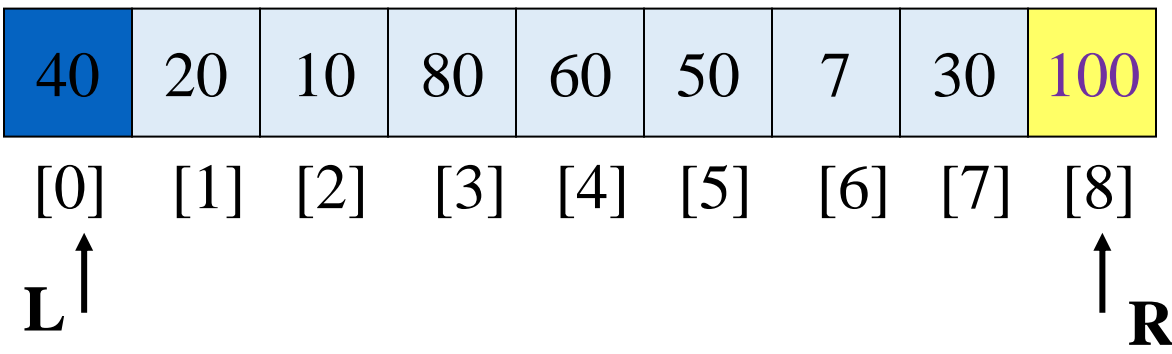
6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort



```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```

x = 40
left=0
right=8
L=0
R=7

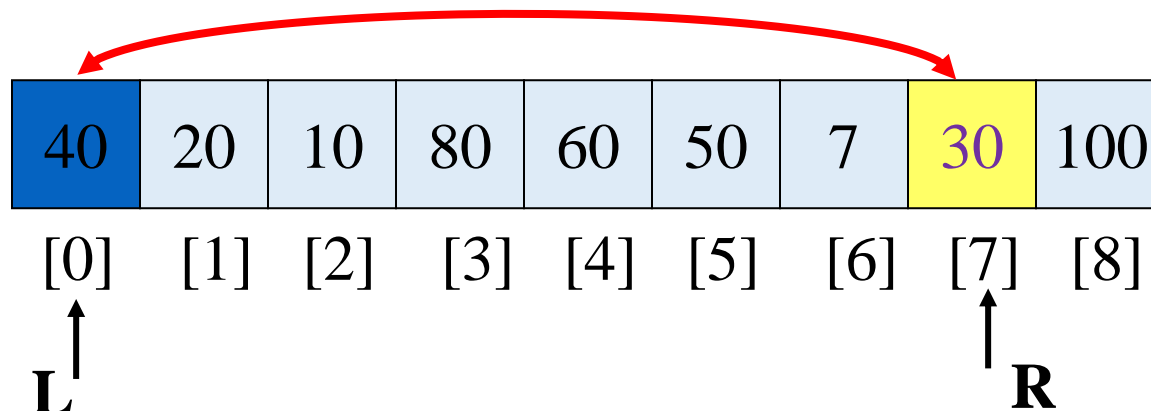


6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort

```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while(L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if(left<R) QuickSort(a, left, R);
    if(L<right) QuickSort(a, L, right);
}
```

```
x = 40
left=0
right=8
L=0
R=7
```




6.6. Minh họa chương trình cài đặt của Quick Sort



30	20	10	80	60	50	7	40	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
L ↑							↑ R	

6.6. Minh họa chương trình cài đặt của Quick Sort

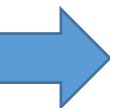


30	20	10	80	60	50	7	40	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	L ↑					↑ R		

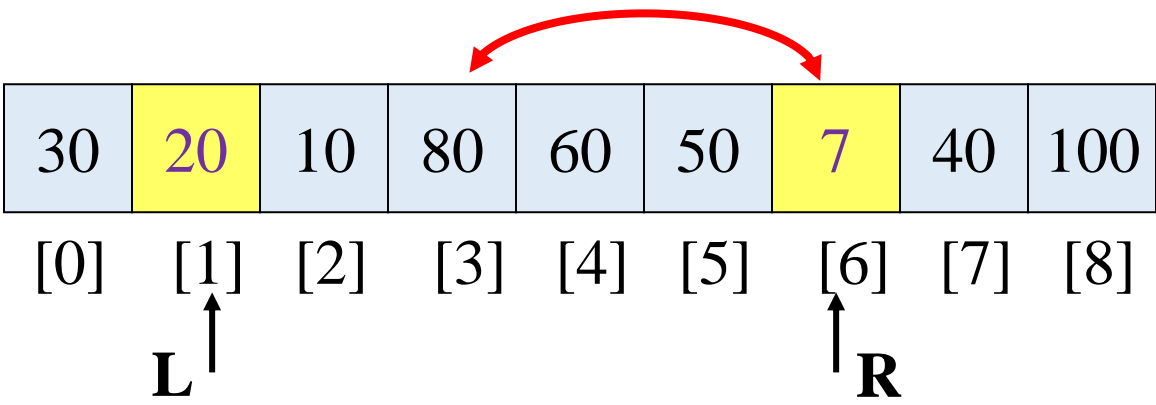
6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort


```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```



x = 40
left=0
right=8
L=3
R=6



6.6. Minh họa chương trình cài đặt của Quick Sort



30	20	10	7	60	50	80	40	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
			↑ L			↑ R		

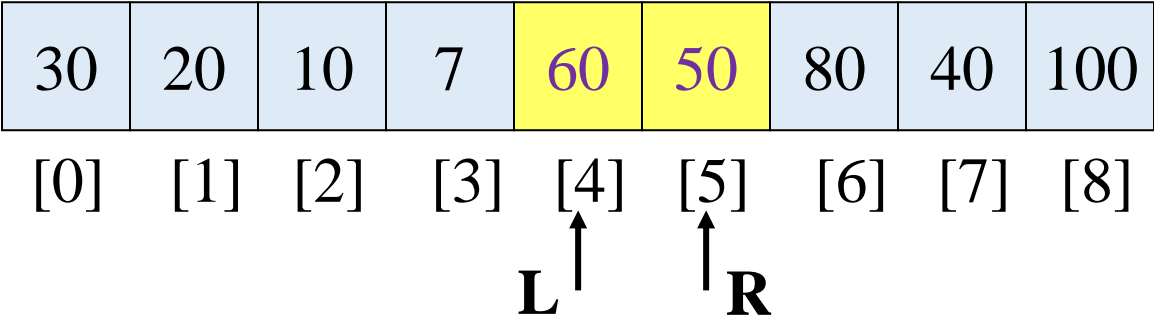
6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort

➡

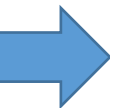
```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```

x = 40
left=0
right=8
L=4
R=3



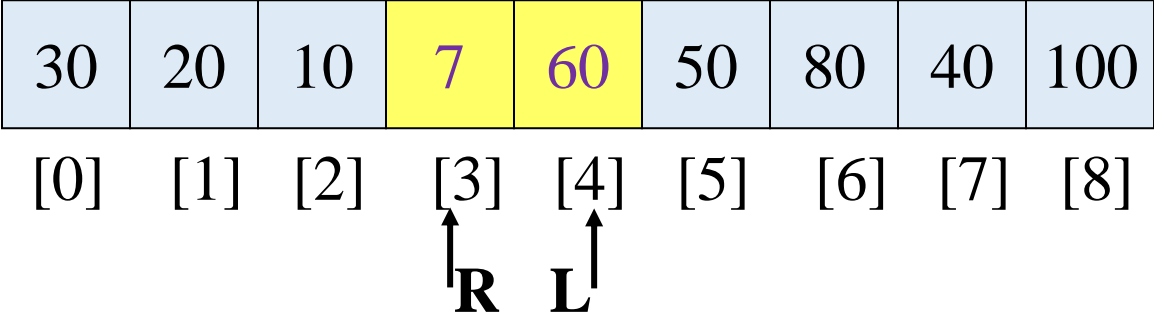
6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort



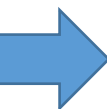
```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```

x = 40
left=0
right=8
L=4
R=3



6. Giải thuật Quick Sort

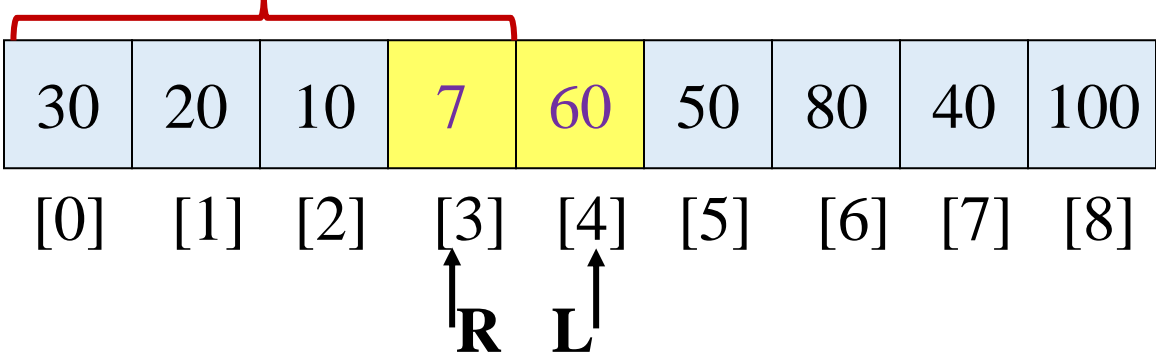
6.6. Minh họa chương trình cài đặt của Quick Sort



```
void QuickSort(int a[], int left, int right)
{
    int p=left, x = a[p]; L = left; R = right;
    while (L <= R)
    {
        while ((a[L] < x) && (L<right)) L++;
        while ((a[R] > x) && (R>left)) R--;
        if (L <= R)
        {
            if (L < R) Swap(a[L], a[R]);
            L++; R--;
        }
    }
    if (left<R) QuickSort(a, left, R);
    if (L<right) QuickSort(a, L, right);
}
```

QuickSort(a, left=0, R=3);

x = 40
left=0
right=8
L=4
R=3



6.6. Minh họa chương trình cài đặt của Quick Sort



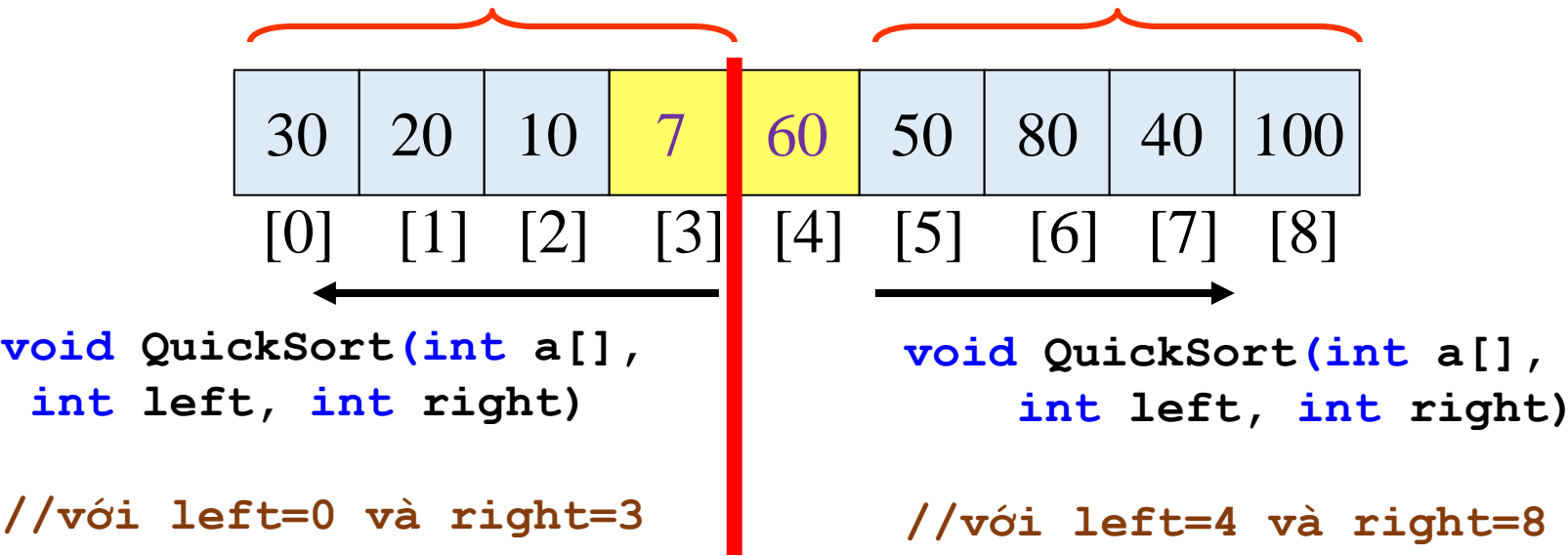
30	20	10	7	60	50	80	40	100
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]

R
L

6. Giải thuật Quick Sort

6.6. Minh họa chương trình cài đặt của Quick Sort

- Thực hiện đệ quy quá trình phân hoạch 2 mảng con vừa có



6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

$left=0 \ \& \ right=7 \Rightarrow x=a[pivot]=a[0]=12$

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu CT	0		7		12	2	8	5	1	6	4	15
<code>while (L <= R) //Lặp lần 1</code>	0	12	7	15								
<code>while ((a[L] < x) && (L<right)) L++</code>	0				12	2	8	5	1	6	4	15
<code>while ((a[R] > x) && (R>left)) R--;</code>	0	12	<u>7/6</u>	15/4								
<code>if (L<= R)</code>												
<code>if(L<=R) Swap(a[L],a[R])</code>	0	12/4	6	4/12	<u>12/4</u>	2	8	5	1	6	<u>4/12</u>	15
<code>L++; R--</code>	<u>0/1</u>	2	<u>6/5</u>	6	4	2	8	5	1	6	12	15
<code>while (L <= R) //Lặp lần 2</code>	<u>1</u>		5		4	2	8	5	1	6	12	15
<code>while ((a[L] < x) && (L<right)) L++</code>	<u>1/6</u>	2/12	5	6	4	2	8	5	1	6	12	15
<code>while ((a[R] > x) && (R>left)) R--;</code>	6	12	5	6	4	2	8	5	1	6	12	15
<code>if (L<= R)</code>												
<code>if(L<=R) Swap(a[L],a[R])</code>												
<code>L++; R--</code>												
					left					R	L	right
Gọi đệ quy do ! (L<=R)	<u>6</u>	12	5	6	4	2	8	5	1	6	12	15
					Mảng 1						Mảng 2	

6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

Gọi đệ quy mảng 1 với left=0 & right=5 ⇒ x=a[pivot]=a[0]=4

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu hàm	0		5		4	2	8	5	1	6	12	15
while (L <= R) //Lặp lần 1	0	4	5	6								
while ((a[L] < x) && (L<right)) L++	0	4	5	6	4	2	8	5	1	6	12	15
while ((a[R] > x) && (R>left)) R--;	0	4	5/4	6/1								
if (L<= R)	0		4									
if(L<=R) Swap(a[L],a[R])	0	4/1	4	1/4	4/1	2	8	5	1/4	6		
L++; R--	0/1	1/2	4/3	5	1	2	8	5	4	6	12	15
while (L <= R) //Lặp lần 2	1	2	3	5	1	2	8	5	4	6	12	15
while ((a[L] < x) && (L<right)) L++	1/2	2/8	3	5								
while ((a[R] > x) && (R>left)) R--;	2	8	3/1	5/2								
if (L<= R)	2		1									
if(L<=R) Swap(a[L],a[R])												
L++; R--												
					left	R	L			right		
Gọi đệ quy do !(L<=R)	3		2		1	2	5	8	4	6	12	15
					M 1.1		M1.2			M2		

6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

Gọi đệ quy mảng 1.1 với left=0 & right=1 $\Rightarrow x=a[pivot]=a[0]=1$

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu hàm	0		1		1	2	5	8	4	6	12	15
<code>while (L <= R) //Lặp lần 1</code>	0	1	1	2								
<code>while ((a[L] < x) && (L<right)) L++</code>	0		1									
<code>while ((a[R] > x) && (R>left)) R--;</code>	0		<u>1/-1</u>									
<code>if (L<= R)</code>												
<code>if (L<=R) Swap(a[L],a[R])</code>												
<code>L++; R--</code>												
Kết thúc hàm đệ quy trên mảng 1.1 do !L<R \Rightarrow M1.1 đã được sắp tang dần					1	2	5	8	4	6	12	15
					M 1.1		M1.2				M2	

6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

Gọi đệ quy mảng 1.2 với left=2 & right=5 ⇒ x=a[pivot]=a[2]=5

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu hàm	2	5	5	6	1	2	5	8	4	6	12	15
while (L <= R) //Lặp lần 1												
while ((a[L] < x) && (L<right)) L++	2	5										
while ((a[R] > x) && (R>left)) R--;			<u>5/4</u>	<u>6/4</u>								
if (L<= R)												
if (L<=R) Swap(a[L],a[R])							<u>5/4</u>	8	<u>4/5</u>	6		
L++; R--	<u>2/3</u>		<u>4/3</u>				4	8	5	6		
while (L <= R) //Lặp lần 1	3	8	3	8			4	8	5	6		
while ((a[L] < x) && (L<right)) L++												
while ((a[R] > x) && (R>left)) R--;			<u>3/2</u>	<u>4</u>								
if (L<= R)												
if (L<=R) Swap(a[L],a[R])												
L++; R--												
							left/R	L		right		
Gọi đệ quy do !(L>R)	3		2		1	2	4	8	5	6	12	15
					M 1.1		1.2.1	1.2.2			M2	

6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

Gọi đệ quy mảng 1.2.2 với left=3 & right=5 ⇒ x=a[pivot]=a[3]=8

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu hàm	3	8	5	6	1	2	4	8	5	6	12	15
while (L <= R) //Lặp lần 1												
while ((a[L] < x) && (L<right)) L++	3	8										
while ((a[R] > x) && (R>left)) R--;			5	6								
if (L<= R)												
if(L<=R) Swap(a[L],a[R])	3	8/6	5	6/8	1	2	4	8/6	5	6/8	12	15
L++; R--	4	6/5	4	5	1	2	4	6	5	8	12	15
while (L <= R) //Lặp lần 1					1	2	4	6	5	8	12	15
while ((a[L] < x) && (L<right)) L++	4/5	5/8										
while ((a[R] > x) && (R>left)) R--;			4	5								
if (L<= R)												
if(L<=R) Swap(a[L],a[R])												
L++; R--												
Gọi đệ quy do !(L>R)								left	R	L/ right		

Gọi đệ quy mảng 1.2.2.1. Gọi đệ quy mảng 1.2.2.2 chỉ gồm 1 phần tử ⇒ đã xếp xong.

M1.1

M2.1

M2.2

M2.3

M2

6. Giải thuật Quick Sort

6.7. Minh họa giải thuật Quick Sort dưới dạng bảng với pivot=0

Gọi đệ quy mảng 1.2.2.1 với left=3 & right=4 ⇒ x=a[pivot]=a[3]=6

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Khởi đầu hàm	3	6	4	5	1	2	4	6	5	8	12	15
while (L <= R) //Lặp lần 1												
while ((a[L] < x) && (L<right)) L++	3											
while ((a[R] > x) && (R>left)) R--;			4									
if (L<= R)												
if(L<=R) Swap(a[L],a[R])		6/5		5/6	1	2	4	<u>6/5</u>	<u>5/6</u>	8	12	15
L++; R--	<u>3/4</u>		<u>4/3</u>		1	2	4	5	6	8	12	15
Kết thúc hàm !(L>R)												
					M 1.1		1.2.1	1.2.2.1		1.2.2.2	M2	

6. Giải thuật Quick Sort

6.8. Giải thuật QuickSort chia 3

- Một phương pháp chia khác là chia danh sách thành 3 danh sách con, lần lượt nhỏ hơn, bằng và lớn hơn phần tử mốc.
- Giải thuật QuickSort sắp xếp dãy a_1, a_2, \dots, a_N dựa trên việc phân hoạch dãy ban đầu thành 3 đoạn:
 - *Đoạn 1*: Gồm các phần tử có giá trị bé hơn x
 - *Đoạn 2*: Gồm các phần tử có giá trị bằng x
 - *Đoạn 3*: Gồm các phần tử có giá trị lớn hơn xvới x là giá trị của một phần tử tùy ý trong dãy ban đầu.

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

6. Giải thuật Quick Sort

6.8. Giải thuật QuickSort chia 3

- Đoạn thứ 2 đã có thứ tự.
- Nếu các đoạn 1 và 3 chỉ có 1 phần tử \Rightarrow đã có thứ tự
 \Rightarrow dãy con ban đầu đã được sắp.
- Ngược lại, 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

$a_k \leq x$	$a_k = x$	$a_k \geq x$
--------------	-----------	--------------

6.9. Đánh giá giải thuật

- Chi phí trung bình $O(n \cdot \log_2 n)$
- Chi phí cho trường hợp xấu nhất $O(n^2)$
- Chi phí tùy thuộc vào cách chọn phần tử *pivot*:
 - Nếu chọn được phần tử có giá trị trung bình, ta sẽ chia thành 2 dãy bằng nhau ($\log_2 n$);
 - Nếu chọn đúng vào phần tử nhỏ nhất (hay lớn nhất)
→ $O(n^2)$

6. Giải thuật Quick Sort

6.10. Bài tập: Minh họa giải thuật Quick Sort dưới dạng bảng với pivot luôn là vị trí ở **giữa** của mảng

Diễn giải	L	a[L]	R	a[R]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
					12	2	8	5	1	6	4	15

7. THỰC HÀNH

Viết chương trình dạng menu để thực hiện các giải thuật sau:

- i. Đổi chỗ trực tiếp – Interchange Sort
- ii. Chọn trực tiếp – Selection Sort
- iii. Chèn trực tiếp – Insertion Sort
- iv. Nổi bọt – Bubble Sort
- v. Giải thuật lắc - Shaker Sort
- vi. Sắp xếp nhanh - Quick Sort
- vii. Sắp xếp trộn trực tiếp - Merge Sort
- viii. Sắp xếp với độ dài bước giảm dần - Shell Sort
- ix. Sắp xếp vun đống - Heap Sort
- x. Sắp xếp theo cơ số - Radix Sort
- xi. Sắp xếp bằng đếm phân khối - Distribution Counting
- xii. Sắp xếp tuyến tính – FlashSort
- xiii. Tìm kiếm tuyến tính
- xiv. Tìm kiếm nhị phân