

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GIẢNG VIÊN: NGUYỄN VĂN GIANG

CHƯƠNG 1: MỞ ĐẦU

Mục tiêu:

Sau khi học xong người học hiểu được:

- Mục đích, vị trí, vai trò của môn học;
- Khái niệm cấu trúc dữ liệu, giải thuật và các yêu cầu chung khi xây dựng cấu trúc dữ liệu và giải thuật.

1.1. CẤU TRÚC DỮ LIỆU (Data Structure)

- CTDL là cách thức tổ chức dữ liệu sao cho phản ánh đúng thực tế bài toán.
- CTDL là tập hợp các biến và các kiểu dữ liệu được liên kết với nhau.

Ví dụ: CTDL của chương trình quản lý SV

```
typedef struct SV
{
    char MaSV[10];
    char TenSV[50];
    int NS;
    float Diem;
};
SV a[100];
```

- Một cấu trúc dữ liệu tốt phải thỏa:

- + Phản ánh đúng thực tế
- + Phù hợp với các thao tác xử lý trên dữ liệu đó
- + Tiết kiệm tài nguyên của hệ thống (thời gian, bộ nhớ):

1.2. GIẢI THUẬT (Algorithms)

- Giải thuật là trình tự các thao tác xử lý dữ liệu để giải quyết bài toán.
- Gọi n là kích thước của dữ liệu đầu vào, ta ký hiệu $T(n)$ là hàm biểu diễn thời gian thực hiện của giải thuật.
- Mức độ tăng lên của $T(n)$ khi n tăng lên gọi là độ phức tạp tính toán của giải thuật và được biểu diễn bằng hàm O (gọi là Big O).
- Các lớp độ phức tạp tính toán phổ biến:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

1.3. MỐI QUAN HỆ GIỮA CTDL VÀ GIẢI THUẬT

Data structures + Algorithms = Programs

- Để xác định được giải thuật phù hợp cần phải biết nó tác động đến những loại dữ liệu nào
- Khi chọn lựa một cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ được tác động lên nó

BÀI TẬP

Viết chương trình **DanhSachSoNguyen.cpp** thực hiện các công việc sau:

- Nhập mảng số nguyên
- Xuất mảng số nguyên
- Cho phép người dùng nhập một số x và kiểm tra xem trong mảng có số x hay không ?

Chỉ số i	0	1	2	3	4	5	6	7 (n-1)
Giá trị a[i]	7	2	0	9	8	1	4	3

Chỉ số i	0	1	2	3	4	5	6	7 (n-1)
Giá trị a[i]	7	2	0	9	8	1	4	3

```
void NhapMang (int a[],int &n)
```

```
{
```

```
    printf("Nhap so luong phan tu: ");
```

```
    scanf("%d",&n);
```

```
    for (int i=0;i<n;i++)
```

```
    {
```

```
        printf("a[%d] =",i);
```

```
        scanf("%d",&a[i]);
```

```
        printf("\n");
```

```
    }
```

```
}
```

```
}
```

CHƯƠNG 2: TÌM KIẾM VÀ SẮP XẾP

BÀI 1: TÌM KIẾM (SEARCH)

Mục tiêu:

Sau khi học xong người học hiểu được:

- Trình bày được ý tưởng của phương pháp tìm kiếm tuyến tính, tìm nhị phân;
- Mô phỏng và đánh giá được giải thuật của phương pháp tìm kiếm tuyến tính, tìm nhị phân;
- Cài đặt được các giải thuật tìm kiếm trên mảng.
- Vận dụng được các giải thuật tìm kiếm trong lập trình các ứng dụng

1. Bài toán tìm kiếm

INPUT:

- Dãy (a) gồm n phần tử $a_0, a_1, \dots, a_{n-1}$.
- x là giá trị của phần tử cần tìm

OUTPUT:

- k ($0 \leq k \leq n-1$) nếu $a_k = x$, (k là vị trí xuất hiện của x trong dãy (a))
- 1 nếu trong dãy không có phần tử nào bằng x

2. Tìm tuyến tính (Linear Search)

2.1. Ý tưởng của giải thuật

5

Khóa tìm

Vị trí = 2

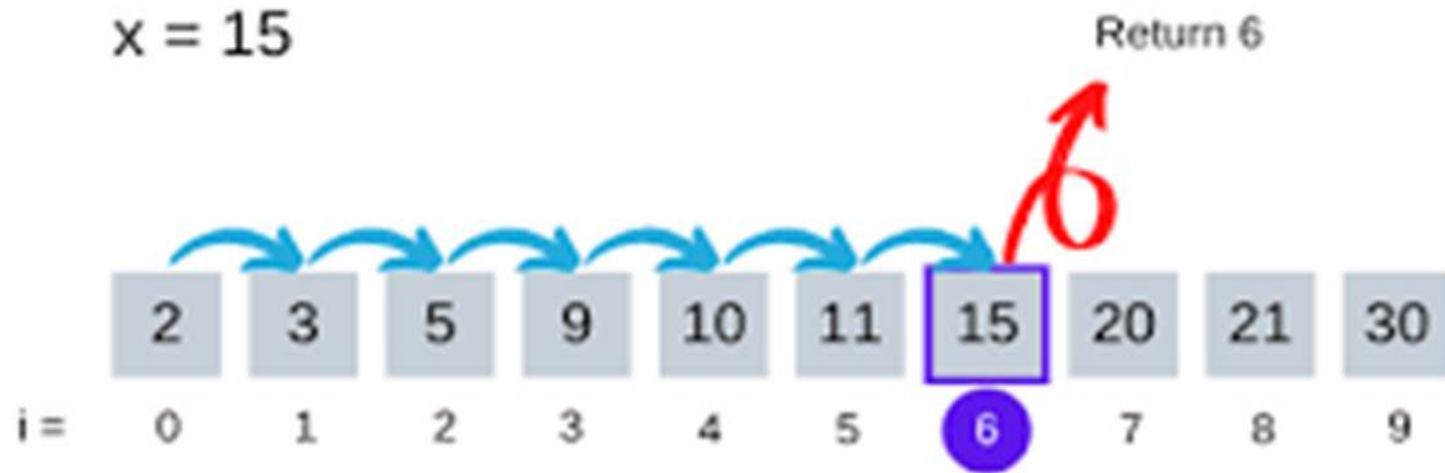
0	1	2	3	4	5	6	7
7	13	5	21	6	2	8	15



Tìm thành công

Số lần so sánh: 3

Linear Search



Lần lượt so sánh x với các $a[i]$ ($i=0,1,2,\dots,n-1$)

- Nếu có $a[i] = x$ thì dừng và trả về i
- Nếu không có $a[i]$ nào bằng x thì trả về -1

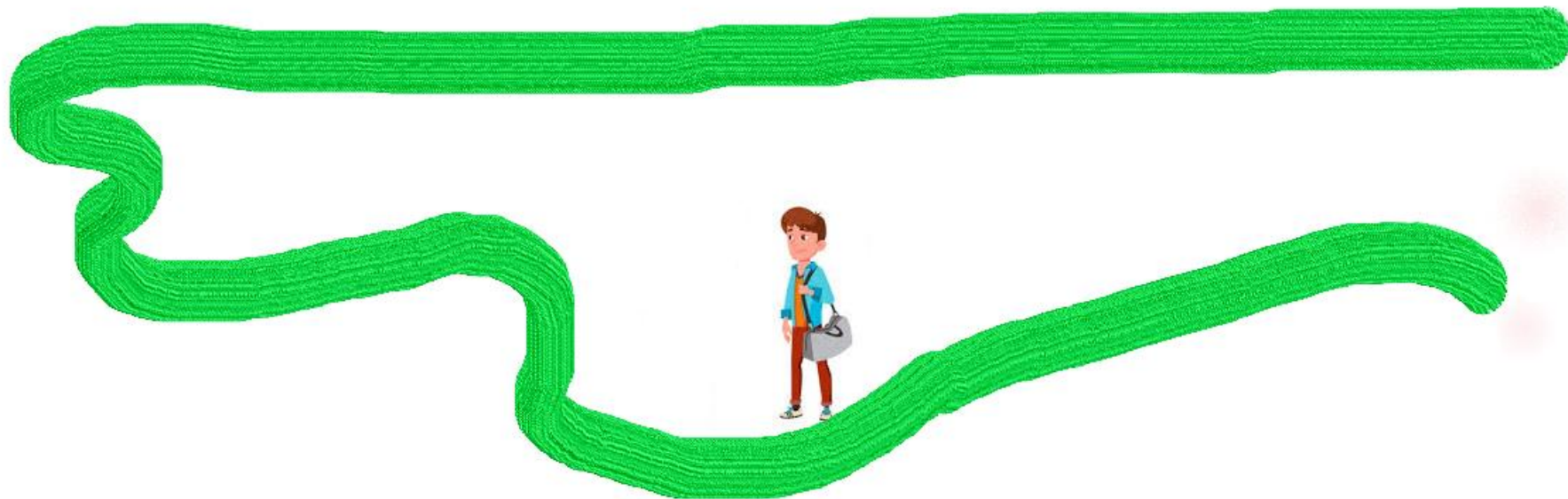
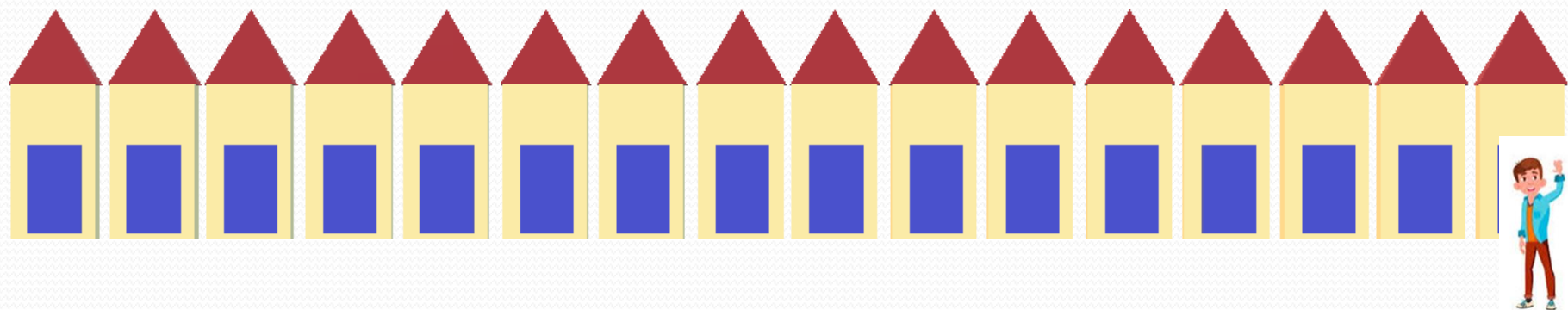
i	0	1	2	3	4	5	6	7(n-1)
a[i]	12	2	8	5	1	6	8	15

Lưu ý : Giải thuật tìm tuyến tính luôn trả về **vị trí xuất hiện đầu tiên** của x trong dãy số.

2.2. Cài đặt

```

int linearSearch (int a[], int n, int x)
{
    int i = 0;
    while (i<n&&a[i] != x)
        i++;
    if( i==n) return -1;
    return i;
}
  
```



2.3. Thuật toán cải tiến

```
int LinearSearch (int a[], int n, int x)
{
    int i = 0;
    a[n]=x;
    while (a[i]!=x)
        i++;
    if( i==n) return -1;
    return i;
}
```

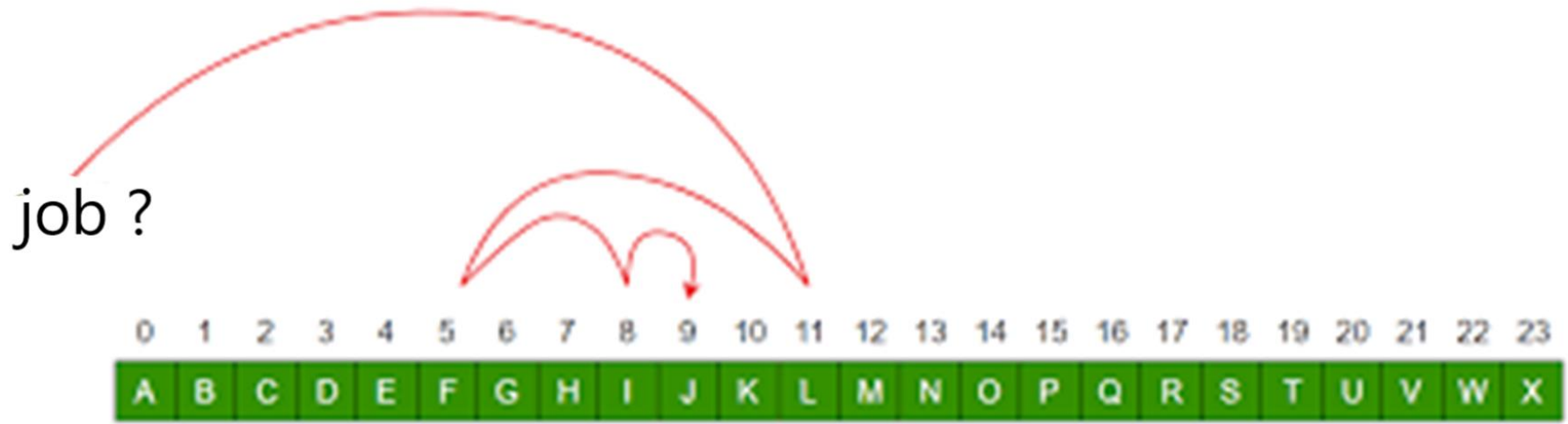
Đánh giá giải thuật:

Độ phức tạp của giải thuật là $O(n)$

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	$n + 1$	Không có x trong mảng
Trung bình	$(n+1)/2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

3. Tìm nhị phân (Binary Search)

3.1. Ý tưởng của giải thuật



Điều kiện áp dụng: **Dãy (a) đã được sắp thứ tự tăng dần**

Xét dãy: $a[l], a[l+1], \dots, a[r]$ ($l \leq r$)

- Gọi $m = (l+r)/2$ và so sánh $a[m]$ với x :
 - + Nếu $a[m] = x$: Trả về m
 - + Nếu $a[m] < x$: Thực hiện lại giải thuật với $l = m+1$
 - + Nếu $a[m] > x$: Thực hiện lại giải thuật với $r = m-1$
- Giải thuật dừng khi $l > r$ và trả về -1

i	0	1	2	3	4	5	6	7(n-1)
a[i]	2	3	5	7	8	8	10	15

Tìm $x=10$:

- $l = 0, r = 7: m = 3, a[3] = 7 < x$
- $l = 4, r = 7: m = 5, a[5] = 8 < x$
- $l = 6, r = 7: m = 6, a[6] = 10 = x$. Trả về 6.

i	0	1	2	3	4	5	6	7(n-1)
a[i]	2	3	5	7	8	8	10	15

Tìm $x=5$?

- $l = 0, r = 7: m = 3, a[3] = 7 > x$
- $l = 0, r = 2: m = 1, a[1] = 3 < x$
- $l = 2, r = 2: m = 2, a[2] = 5 = x$. Trả về 2

Tìm $x=4$?

- $l = 0, r = 7: m = 3, a[3] = 7 > x$
- $l = 0, r = 2: m = 1, a[1] = 3 < x$
- $l = 2, r = 2: m = 2, a[2] = 5 > x$. Trả về 2.
- $l = 2, r = 1: l > r$. Trả về -1

3.2. Cài đặt:

```
int binarySearch(int a[],int n,int x)
{
    int l=0,r=n-1,m;
    do
    {
        m=(l+r)/2;
        if (x==a[m]) return m;
        if (x<a[m]) r=m-1;
        else l=m+1;
    }while (l<=r);
    return -1;
}
```

Đánh giá giải thuật:

Số phần tử

2

4

8

16

32

n

Số lần chia để tìm x

1

2

3

4

5

$\log_2 n$

Độ phức tạp của giải thuật tìm nhị phân là $O(\log_2 n)$
 < Độ phức tạp của giải thuật tìm tuyến tính là $O(n)$

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 n / 2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

Viết tiếp chương trình **DanhSachSoNguyen.cpp**, thực hiện các công việc sau :

d. Cài đặt giải thuật tìm tuyến tính cải tiến và gọi hàm này để tìm số x có trong mảng không ? Nếu có thì trả về vị trí của số x , nếu không có thì xuất câu thông báo.

e. Cài đặt hàm tìm nhị phân và gọi hàm này để tìm số x có trong mảng không ? Nếu có thì trả về vị trí của số x , nếu không có thì xuất câu thông báo.

f. So sánh kết quả tìm kiếm của 2 hàm nếu $x=7$ và (a) : 1, 3, 7, 7, 8, 9, 11. Giải thích kết quả.

BÀI 2: SẮP XẾP (SORT)

Mục tiêu:

Sau khi học xong người học có khả năng:

- Trình bày được ý tưởng của phương pháp sắp xếp Chèn trực tiếp và Phân hoạch;
- Mô phỏng và đánh giá được giải thuật của phương pháp sắp xếp Chèn trực tiếp và Phân hoạch;
- Cài đặt được các giải thuật sắp xếp trên mảng.
- Vận dụng được các giải thuật sắp xếp trong lập trình các ứng dụng

1. Bài toán sắp xếp

Giả sử cần sắp xếp tăng dần

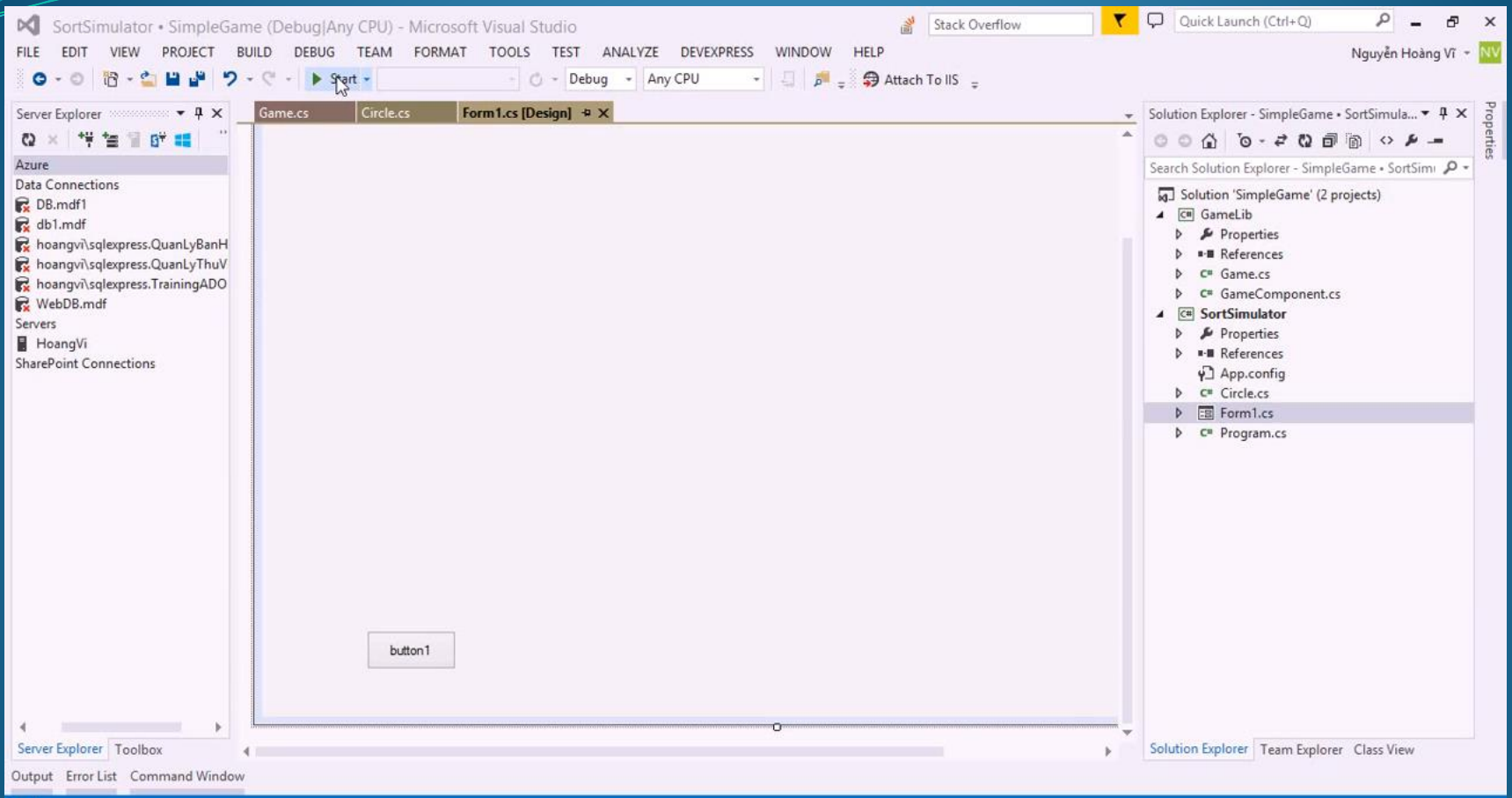
INPUT:

- Dãy (a) gồm n phần tử $a_0, a_1, \dots, a_{n-1}$.

OUTPUT:

- Dãy (a) thỏa : $a_0 \leq a_1 \leq \dots \leq a_{n-1}$

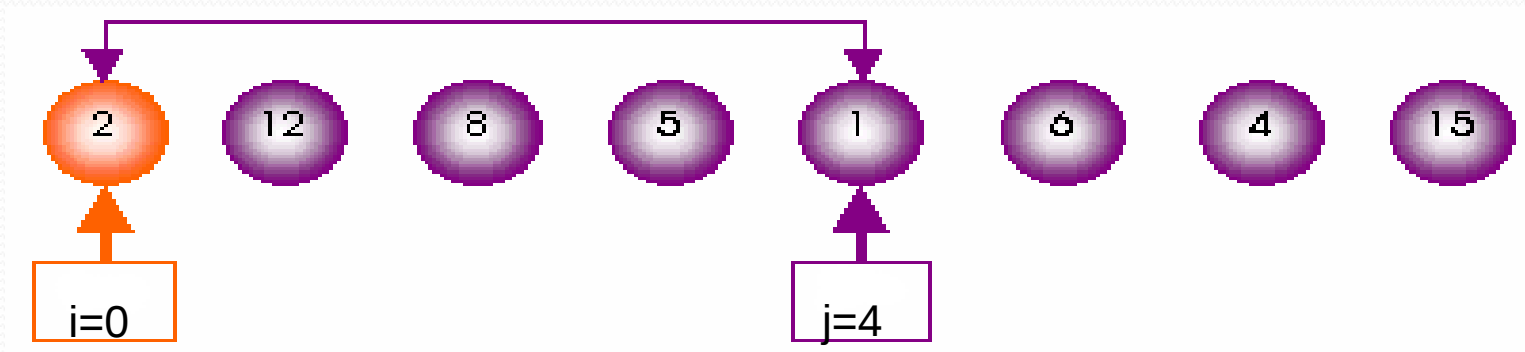
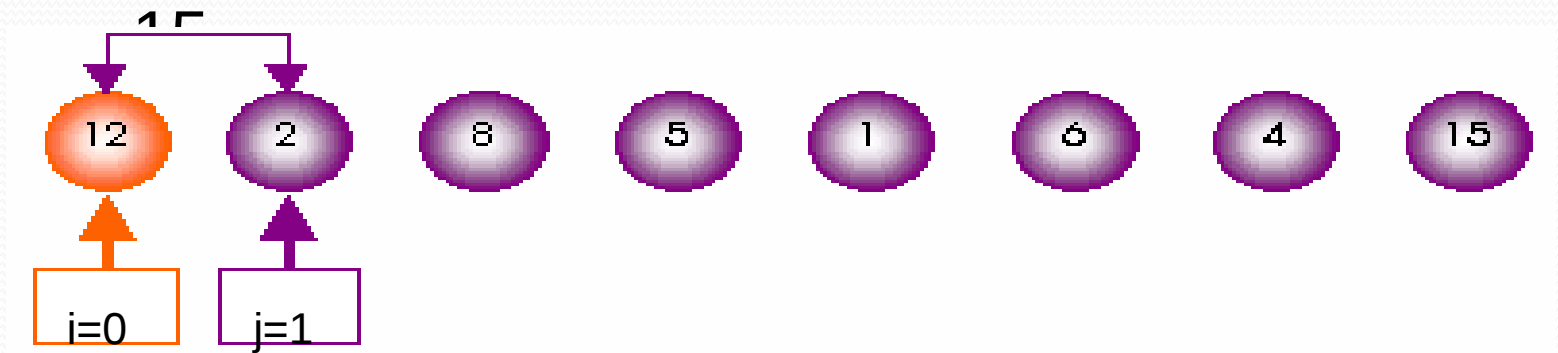
2. Phương pháp đổi chỗ trực tiếp (Interchange Sort)

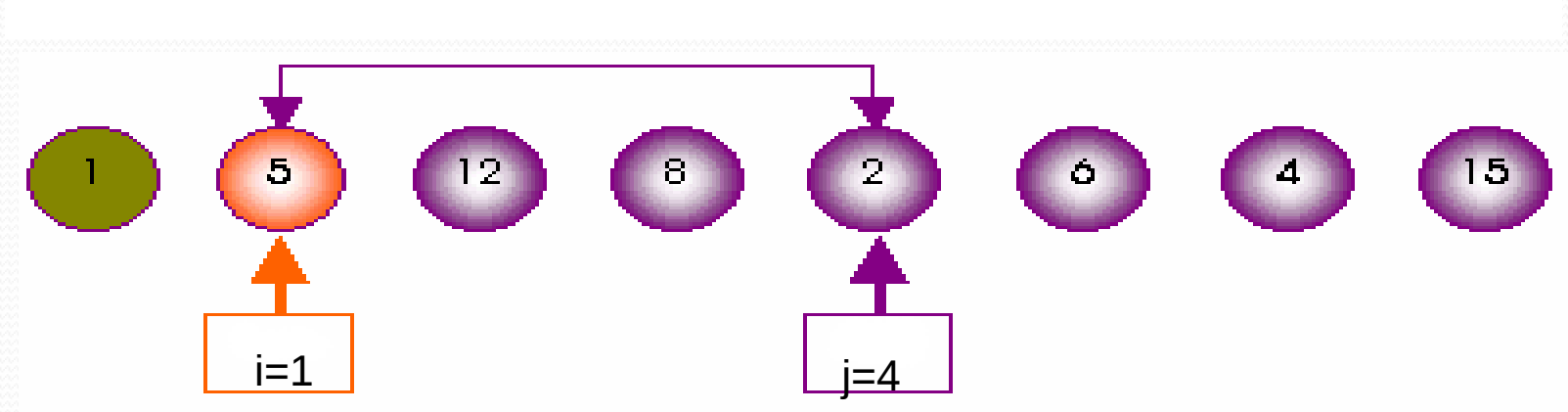
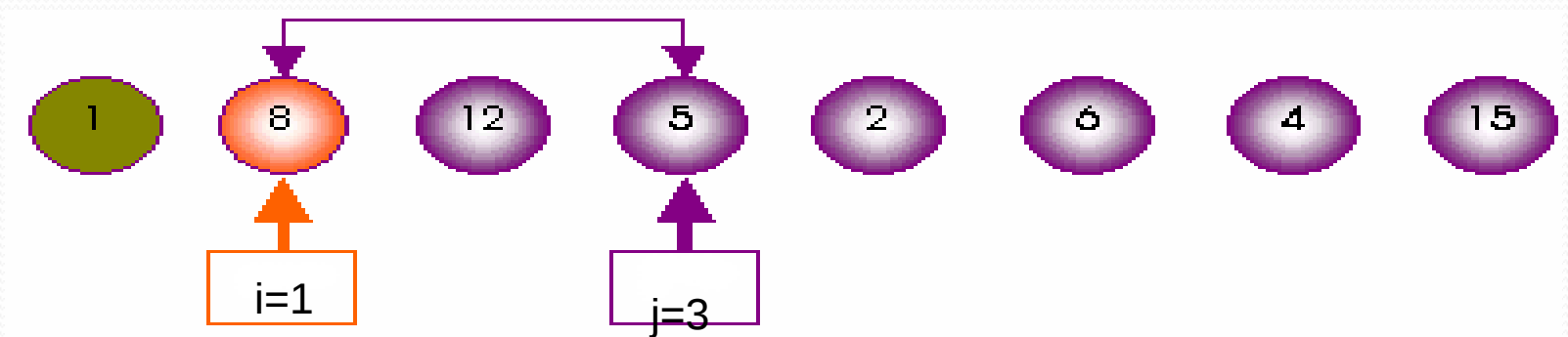
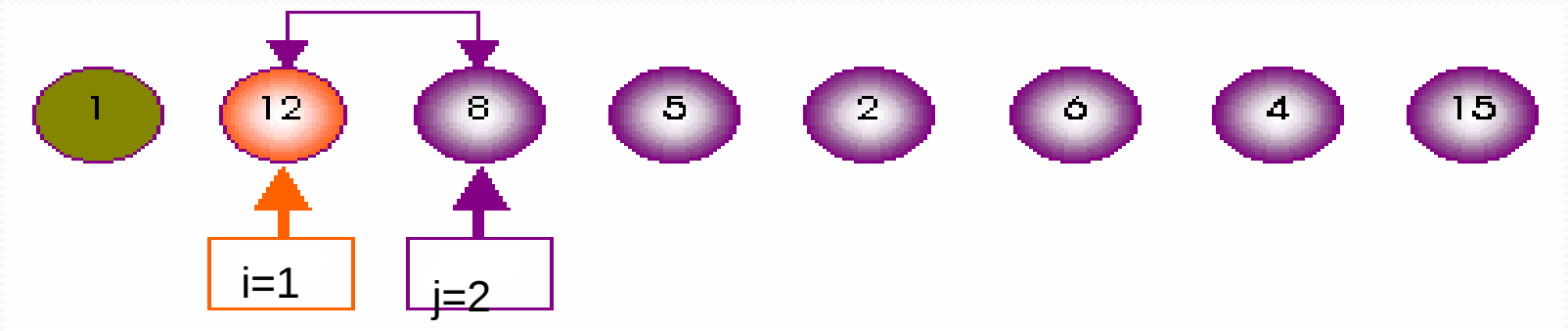


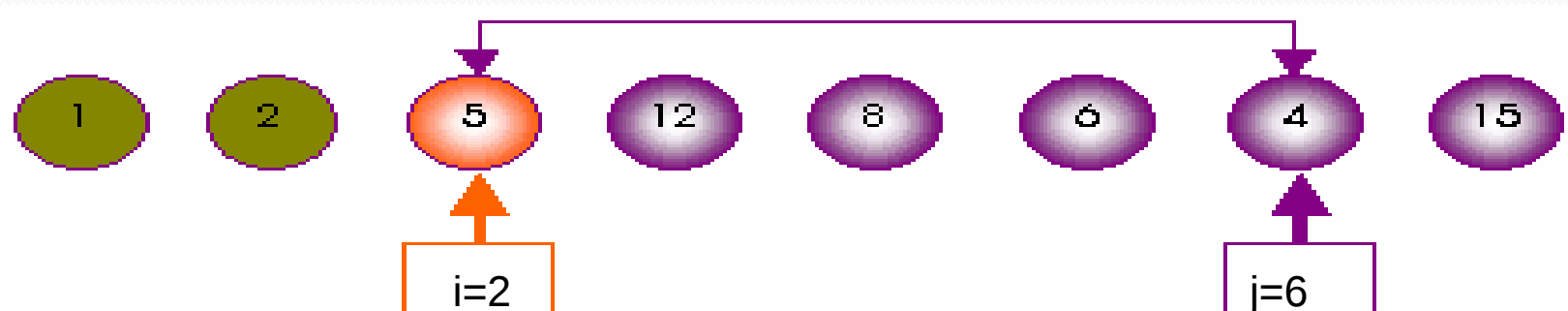
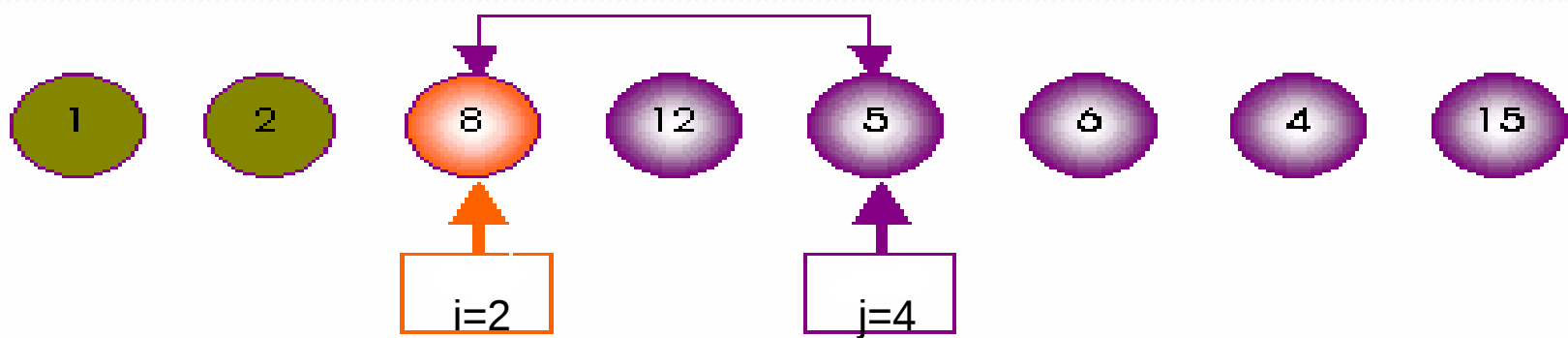
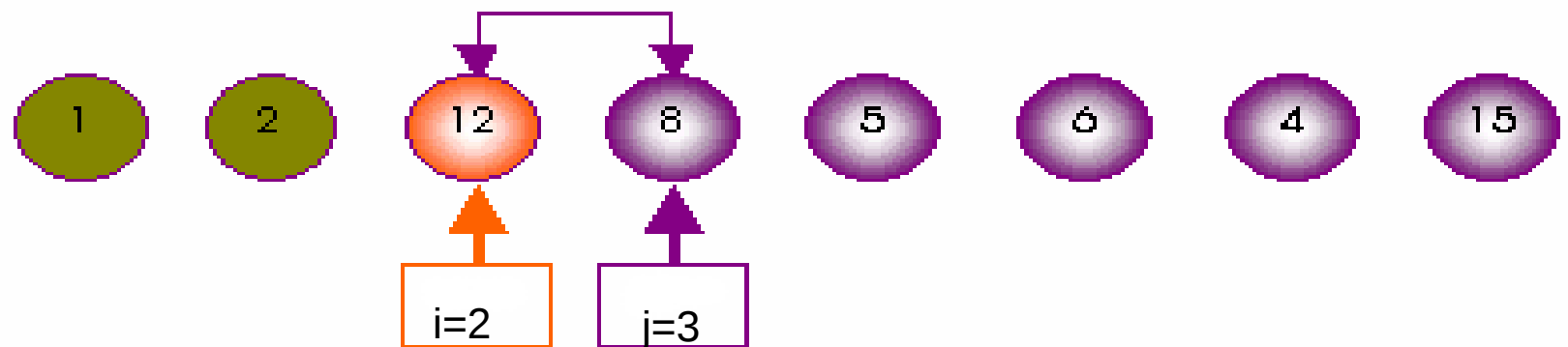
2.1. Ví dụ: Sắp tăng dần dãy số 4, 6, 5, 1, 2, 9, 3

•Cho dãy số a:

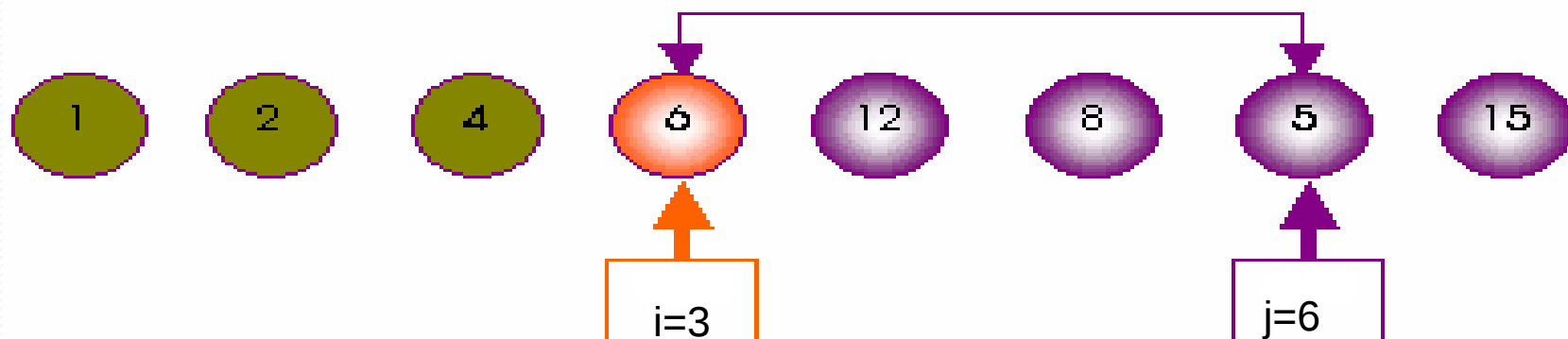
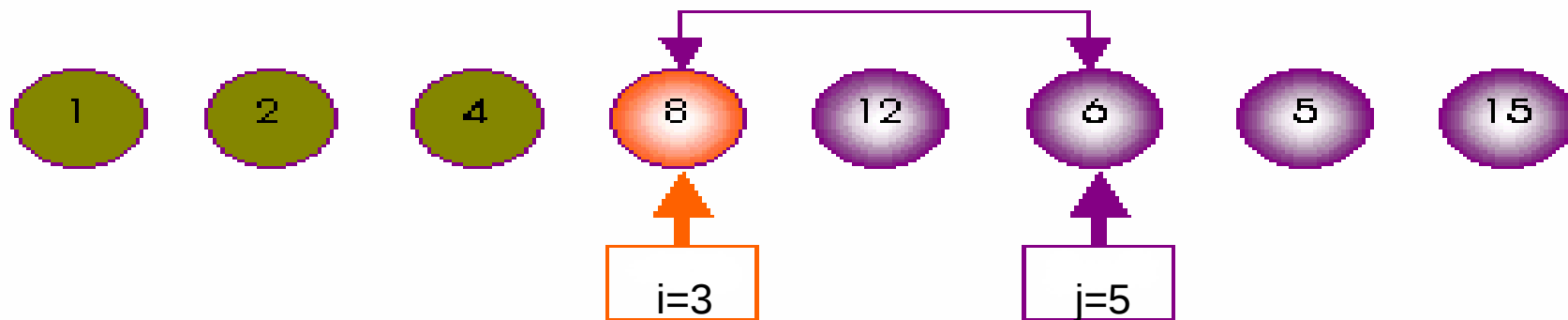
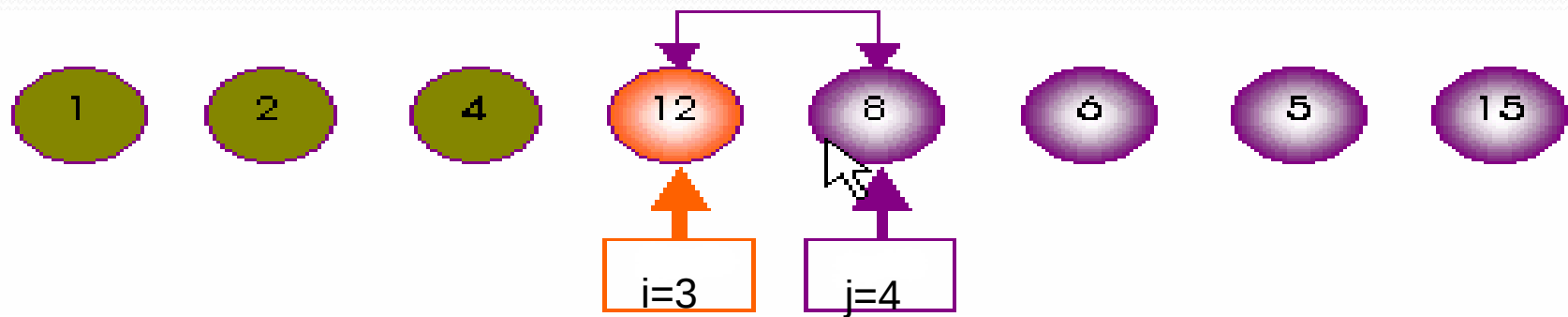
12 2 8 5 1 6 4



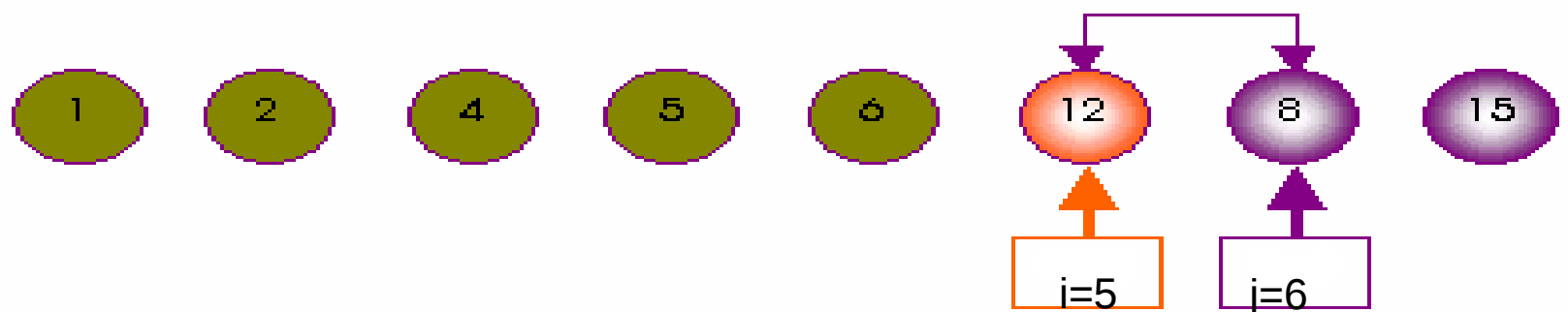
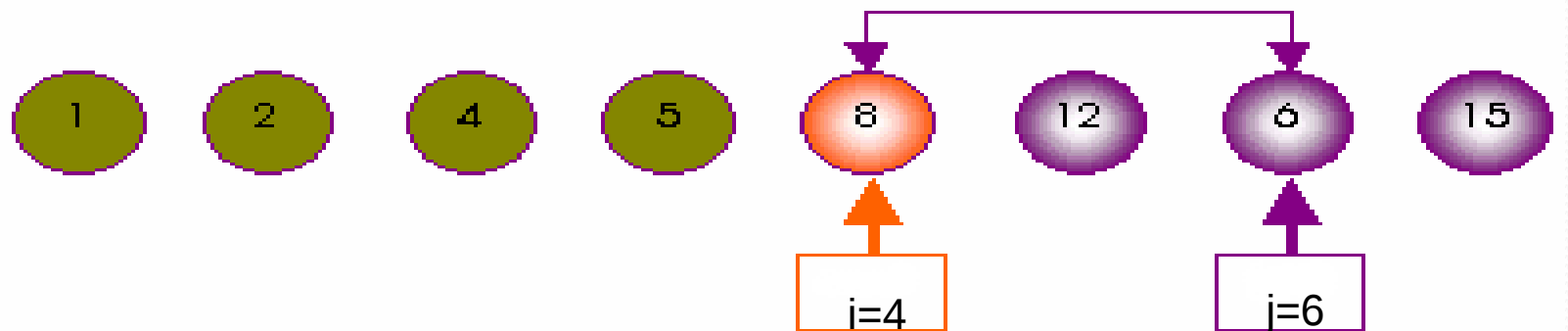
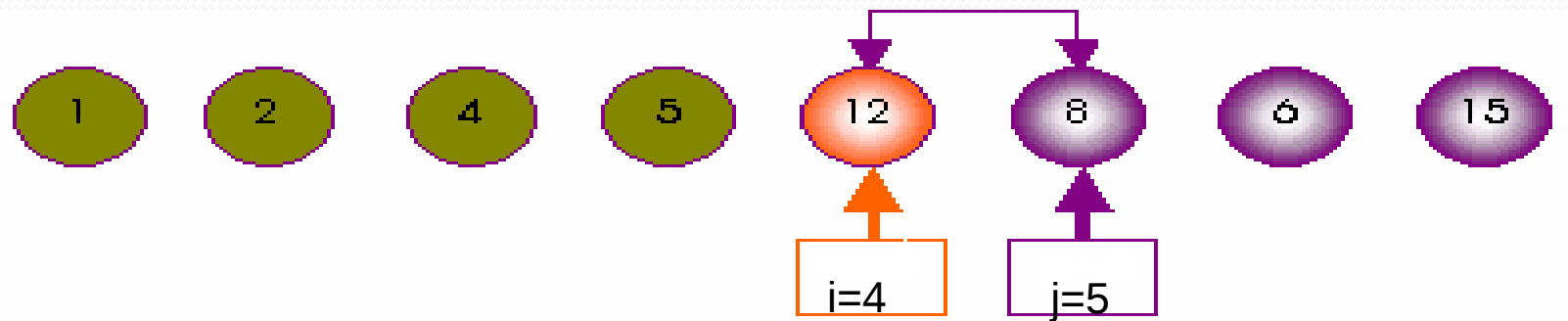


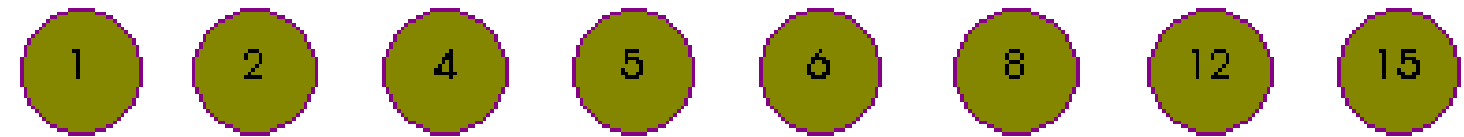
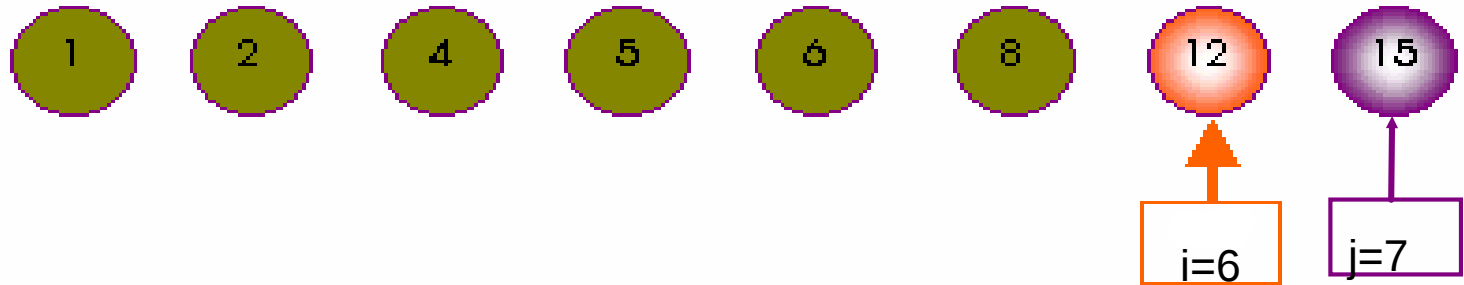


Đổi Chỗ Trục Tiếp – Interchange Sort



Đổi Chỗ Trực Tiếp – Interchange Sort





Bước	0	1	2	3	4	5	6
Bước 0	4	6	5	1	2	9	3
Bước 1	1	6	5	4	2	9	3
Bước 1	1	5	6	4	2	9	3
Bước 1	1	4	6	5	2	9	3
Bước 2	1	2	6	5	4	9	3
Bước 2	1	2	5	6	4	9	3
Bước 2	1	2	4	6	5	9	3
Bước 3	1	2	3	6	5	9	4
Bước 3	1	2	3	5	6	9	4
Bước 4	1	2	3	4	6	9	5
Bước 5	1	2	3	4	5	9	6
Kết quả	1	2	3	4	5	6	9

Trình bày rút gọn:

Bước	0	1	2	3	4	5	6
Bước 0	4	6	5	1	2	9	3
Bước 1	1	6	5	4	2	9	3
Bước 2	1	2	6	5	4	9	3
Bước 3	1	2	3	6	5	9	4
Bước 4	1	2	3	4	6	9	5
Bước 5	1	2	3	4	5	9	6
Kết quả	1	2	3	4	5	6	9

Sắp xếp dãy số sau tăng dần: 12, 2, 8, 5, 1, 6, 4

Bước	0	1	2	3	4	5	6(n-1)
0	12	2	8	5	1	6	4
0	2	12	8	5	1	6	4
1	1	12	8	5	2	6	4
1		8	12	5	2	6	4
1		5	12	8	2	6	4
2		2	12	8	5	6	4
2			8	12	5	6	4
2			5	12	8	6	4
3			4	12	8	6	5
3				8	12	6	5
3				6	12	8	5
4				5	12	8	6
4					8	12	6
5 (n-2)					6	12	8
KQ	1	2	4	5	6	8	12

Sắp xếp dãy số sau tăng dần: 12, 2, 8, 5, 1, 6, 4

Bước	0	1	2	3	4	5	6(n-1)
0	12	2	8	5	1	6	4
1	1	12	8	5	2	6	4
2		2	12	8	5	6	4
3			4	12	8	6	5
4				5	12	8	6
5 (n-2)					6	12	8
KQ	1	2	4	5	6	8	12
i				a[i]	a[j]		a[n-1]

2.2. Ý tưởng của giải thuật:

Giải thuật gồm $n-1$ bước (từ bước $i=0$ đến bước $i=n-2$).

Ở bước 0, ta phải sắp xếp sao cho $a[0]$ sẽ là phần tử nhỏ nhất dãy $a[0], \dots, a[n-1]$.

Ở bước 1, ta phải sắp xếp sao cho $a[1]$ sẽ là phần tử nhỏ nhất dãy $a[1], \dots, a[n-1]$

Tổng quát:

Ở bước i , ta phải sắp xếp sao cho $a[i]$ sẽ là phần tử nhỏ nhất dãy $a[i], \dots, a[n-1]$, bằng cách :

So sánh $a[i]$ với các $a[j]$ ($i+1 \leq j \leq n-1$), nếu thấy $a[i] > a[j]$ thì hoán vị $a[i]$ với $a[j]$.

Giải thuật gồm $n-1$ bước (bước $i = 0, \dots, n-2$).

Ở bước i , ta phải sắp xếp sao cho $a[i]$ sẽ là phần tử nhỏ nhất dãy $a[i], \dots, a[n-1]$, bằng cách :

So sánh $a[i]$ với các $a[j]$ ($i+1 \leq j \leq n-1$), nếu thấy $a[i] > a[j]$ thì hoán vị $a[i]$ với $a[j]$.

2.3. Cài đặt:

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

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

Đánh giá giải thuật:

- Vòng lặp **for j** lặp **(n - i - 1)** lần
- Số lần lặp của 2 vòng lặp for lồng nhau là:

$$\begin{aligned}
 & \sum_{i=0}^{n-2} (n - i - 1) \\
 &= (n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1 \\
 &= \frac{[(n-1)+1](n-1)}{2} = \frac{n(n-1)}{2}
 \end{aligned}$$

Độ phức tạp
của giải thuật
là : **$O(n^2)$**

3. Phương pháp phân hoạch (Quick Sort)

3.1. Ý tưởng của giải thuật

$a[\text{left}] \text{ ----- } x \text{ ----- } a[\text{right}]$
 $a[i] < x$ $a[j] > x$

$\text{left} \text{ ----- } \rightarrow i$

$j \leftarrow \text{-----} \text{right}$

$a[l]$ ----- x ----- $a[r]$
 $a[i] < x$ $a[j] > x$

| ----- $\rightarrow$ i

j $\leftarrow$ ----- r

Giả sử ta cần sắp tăng dần dãy $a[l], \dots, a[r]$

Bước 1: Chọn $x = a[(l+r)/2]$ làm mốc, $i=l$, $j=r$

Bước 2:

Trong khi $a[i] < x$ thì tăng i

Trong khi $a[j] > x$ thì giảm j

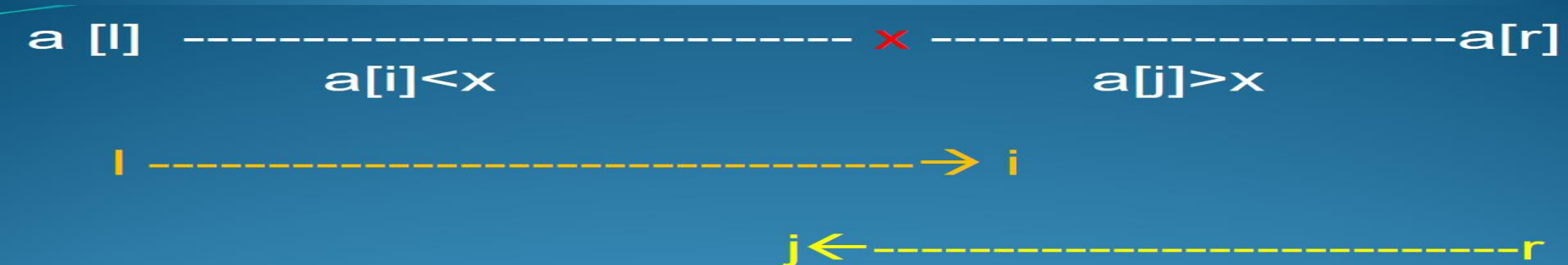
Nếu $i \leq j$ thì hoán vị $a[i]$, $a[j]$ rồi tăng i , giảm j

Nếu $i \leq j$ thì quay lại bước 2, ngược lại qua bước 3

Bước 3:

Nếu $l < j$ thì thực hiện lại giải thuật với $r=j$

Nếu $i < r$ thì thực hiện lại giải thuật với $l=i$



Ví dụ 1: Sắp dãy sau tăng dần bằng Quick Sort: 2, 9, 5, 4, 7, 6, 8

	-1	0	1	2	3	4	5	6
$l=0, r=6,$ $x=4$		i	i		j	j	j	j
		2	9	5	(4)	7	6	8
		2	(4)	i j		j	j	j
				5	9	7	6	8
$l=0,$ $r=1, x=2$	j	i j	i j					
		(2)	4					
$l=2,$ $r=6, x=7$				i	i		j	j
				5	9	(7)	6	8
				5	j	i j	i	
					6	(7)	9	8

	0	1	2	3	4	5	6
$l=2, r=3,$ $x=5$		j	i j (5)	i j 6			
$l=5, r=6,$ $x=9$						i (9)	j 8
						j 8	i (9)
Kết quả	2	4	5	6	7	8	9

Ví dụ 1: Sắp dãy sau tăng dần bằng Quick Sort: 4, 8, 5, 6, 9, 0, 7

	0	1	2	3	4	5	6
$l=0, r=6, x=a[3]=6$	<i>i</i> 4	<i>i</i> 8	5	<6>	9	<i>j</i> 0	<i>j</i> 7
	4	0	<i>i</i> 5	<i>i</i> <6> <i>j</i>	9 <i>j</i>	8	7
	4	0	5 <i>j</i>	<6>	<i>i</i> 9	8	7
$l=0, r=2, x=a[1]=0$	<i>i</i> 4	<0> <i>j</i>	5 <i>j</i>				
	<0> <i>j</i>	4 <i>i</i>	5				
$l=1, r=2, x=a[1]=4$		<i>i</i> <4>	5 <i>j</i>				
	<i>j</i>	<i>i</i> <4> <i>j</i>	<i>i</i> 5				
$l=4, r=6, x=a[5]=8$					<i>i</i> 9	<8>	7
					7	<i>i</i> <8> <i>j</i>	9
					<i>j</i> 7	<8>	<i>i</i> 9
ket qua:	0	4	5	6	7	8	9

3.3. Cài đặt:

```
void quickSort(int a[],int l,int r)
{
    int x=a[(l+r)/2],i=l,j=r;
    do
    {
        while (a[i]<x) i++;
        while (a[j]>x) j--;
        if (i<=j)
        {
            swap(a[i],a[j]); i++;j--;
        }
    }while (i<=j);
    if (l<j) quickSort(a,l,j);
    if (i<r) quickSort(a,i,r);
}
```

Đánh giá giải thuật:

Độ phức tạp của giải thuật là $O(n \log n)$

BÀI TẬP

Bổ sung chương trình `DanhSachSoNguyen.cpp`, thực hiện các công việc sau bằng 2 phương pháp : Interchange Sort và Quick Sort

g. Sắp xếp danh sách tăng dần.

h. Sắp xếp danh sách giảm