



TRƯỜNG ĐẠI HỌC MỞ TP. HCM

Cơ hội học tập cho mọi người

Khoa Công Nghệ Thông Tin

Chương 2 **XẾP THỨ TỰ VÀ TÌM KIẾM**



www.ou.edu.vn

Nội dung chính

2.1 Xếp thứ tự

- Selection Sort
- Insert Sort
- Interchange Sort
- Bubble Sort
- Merge Sort

2.2 Tìm kiếm

- Tìm kiếm tuần tự
- Tìm kiếm nhị phân

2.3 Tổng kết chương

2.4 Bài tập chương 2

Tài liệu tham khảo



2.1

XẾP THỨ TỰ (SORT)

2.1 – XẾP THỨ TỰ (SORT) PHÁT BIỂU BÀI TOÁN

Cho một tập các số nguyên gồm n phần tử:


$$a_0, a_1, a_2, \dots, a_{n-1}$$


Hãy thực hiện sắp xếp n phần tử này theo thứ tự tăng dần như sau:

$$a_0, a_1, a_2, \dots, a_{n-1}$$

Với $a_0 \leq a_1 \leq a_2 \leq \dots \leq a_{n-1}$

2.1 – XẾP THỨ TỰ (SORT) MÔ HÌNH BÀI TOÁN

 **Đầu vào:** một danh sách đặc (các số nguyên)
gồm có n phần tử $a_0, a_1, a_2, \dots, a_{n-1}$.




 **Đầu ra:** một danh sách đặc (các số nguyên)
gồm có n phần tử: $a_0, a_1, a_2, \dots, a_{n-1}$ ($a_0 \leq a_1$
 $\leq a_2 \leq \dots \leq a_{n-1}$)

2.1 – XẾP THỨ TỰ (SORT) MÔ HÌNH BÀI TOÁN

KHAI BÁO MẢNG DANH SÁCH ĐẶC NHƯ SAU

```
# define MAX 100  
  
int a[MAX];  
  
int n; // n là tổng số phần tử hiện có trong danh  
sách,  $0 < n \leq \text{MAX}$ 
```

CÁC NỘI DUNG CHÍNH CỦA BÀI TOÁN

-  Ý tưởng giải thuật
-  Cài đặt chương trình
-  Đánh giá độ phức tạp của giải thuật

Phương pháp xếp thứ tự cơ bản: Selection Sort, Insertion Sort, Bubble Sort, Interchange Sort, Merge Sort

2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

PHƯƠNG PHÁP

Với một danh sách đặc $a[]$, có n phần tử từ $a[0]$ đến $a[n-1]$ như sau: $a[0], a[1], a[2], a[3], \dots, a[n-1]$

Phần tử:	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[n-1]$
Vị trí:	0	1	2	3	$n-1$

	0	1	2	3	4	5	6	7
a	40	60	15	50	90	20	10	70
								$n-1$

2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

- Để xếp thứ tự **danh sách đặc** trên bằng phương pháp chọn lựa trực tiếp, đầu tiên **xác định phần tử nhỏ nhất** trong danh sách các phần tử đang xét, và sau đó **hoán vị phần tử nhỏ nhất** với **phần tử ở vị trí đầu** đoạn danh sách các phần tử nằm bên phải phần tử nhỏ nhất vừa được hoán vị vào, **thực hiện bước lặp** này cho đến khi đoạn danh sách đang xét còn một phần tử.

2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

□ Bước 0:

- Xét danh sách từ vị trí 0 đến vị trí $n-1$, xác định phần tử nhỏ nhất (vị trí min_pos_0)
- Đổi chỗ hai phần tử tại vị trí min_pos_0 và vị trí 0

□ Bước 1:

- Xét danh sách từ vị trí 1 đến vị trí $n-1$, xác định phần tử nhỏ nhất (vị trí min_pos_1)
- Đổi chỗ hai phần tử tại vị trí min_pos_1 và vị trí 1

□ Bước i:

- Xét danh sách từ vị trí i đến vị trí $n-1$, xác định phần tử nhỏ nhất (vị trí min_pos_i)
- Đổi chỗ hai phần tử tại vị trí min_pos_i và vị trí i

2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

THUẬT TOÁN

$i=0;$

Bước 1:

- Tìm phần tử $a[\text{min_pos}]$ nhỏ nhất trong dãy hiện hành từ $a[i]$ đến $a[n-1]$
- Đổi chỗ $a[\text{min_pos}]$ và $a[i]$

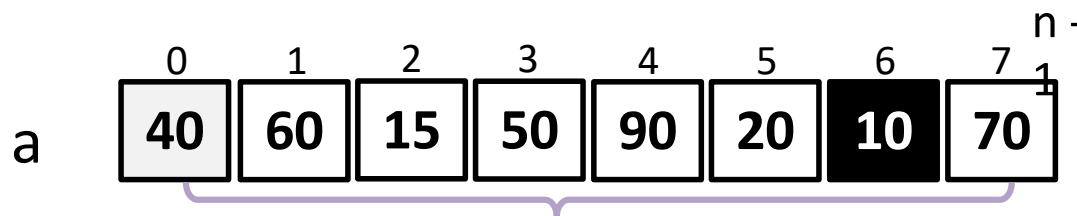
Bước 2: $i+1;$

- Nếu $i < n - 1$ thì lặp lại bước 1

2.1 – XẾP THỨ TỰ (SORT)

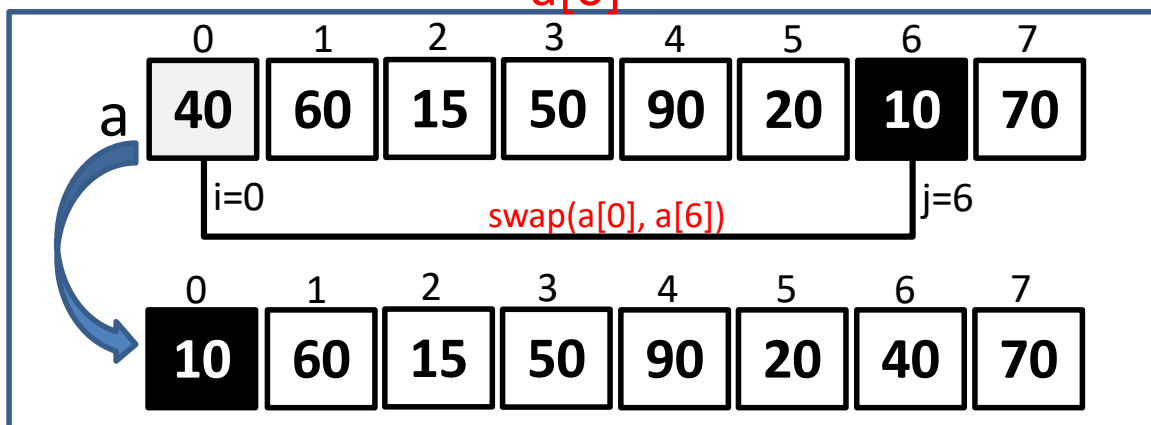
CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ $0 \rightarrow 7$



$a[6] = 10$ Là phần tử nhỏ nhất

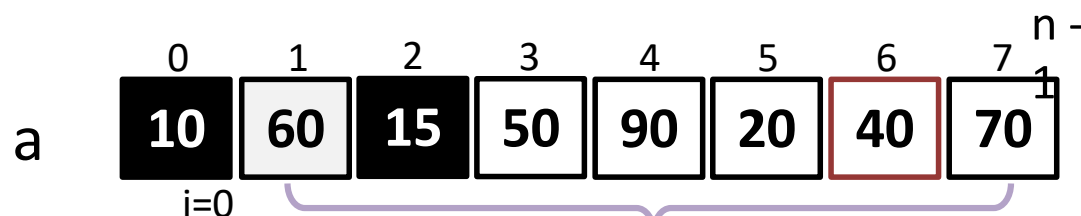
Đổi giá trị giữa $a[6]$ và $a[0]$



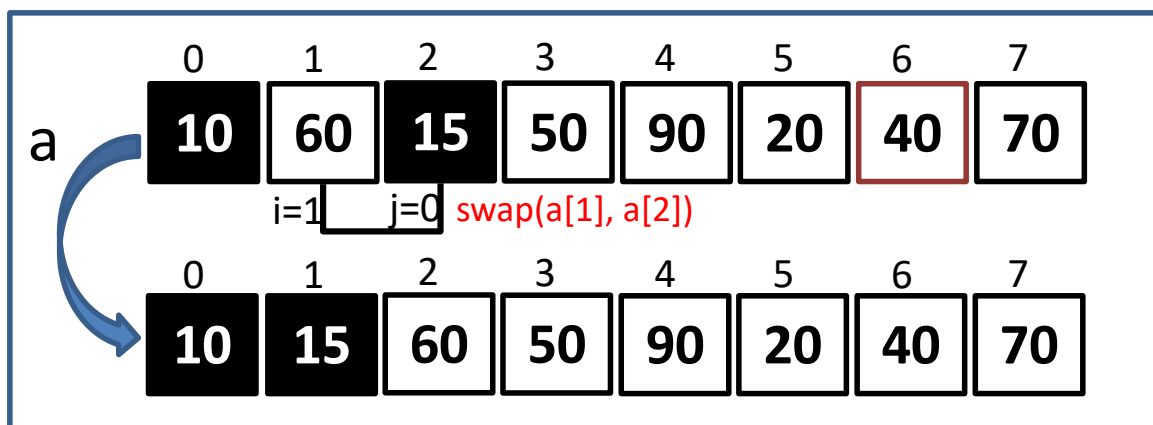
2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 1 → 7

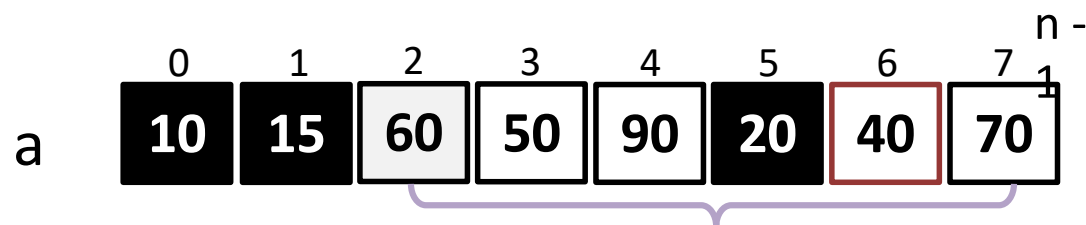


$a[2] = 15$ Là phần tử nhỏ nhất
Đổi giá trị giữa $a[1]$ và $a[2]$



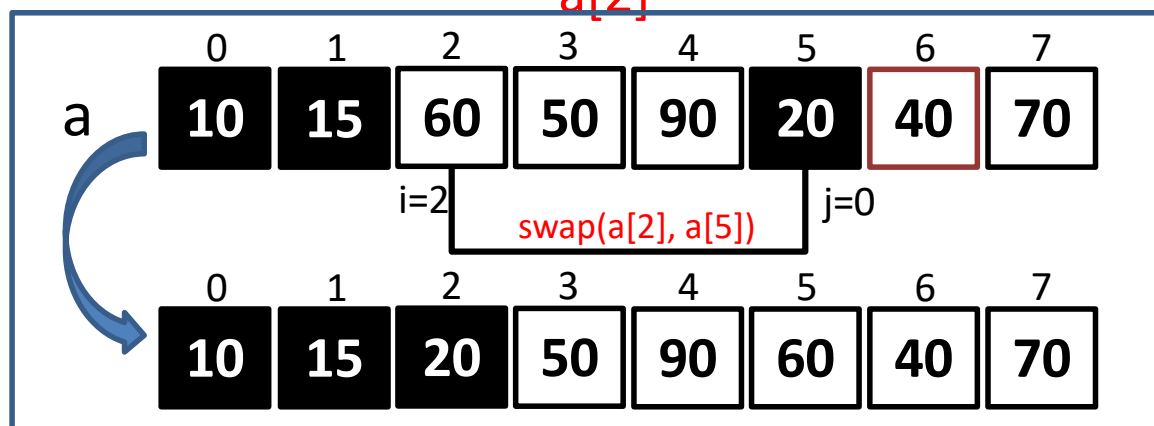
2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ $2 \rightarrow 7$



$a[5] = 20$ Là phần tử nhỏ nhất

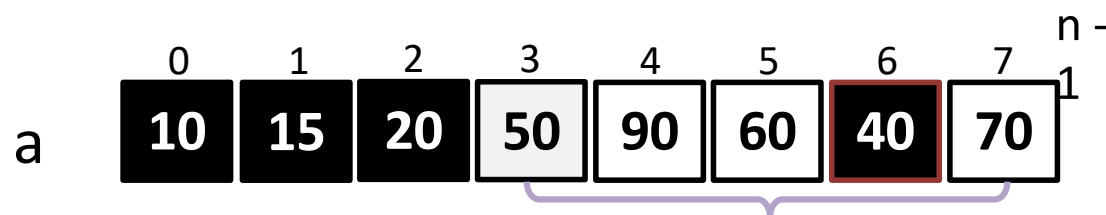
Đổi giá trị giữa $a[5]$ và $a[2]$



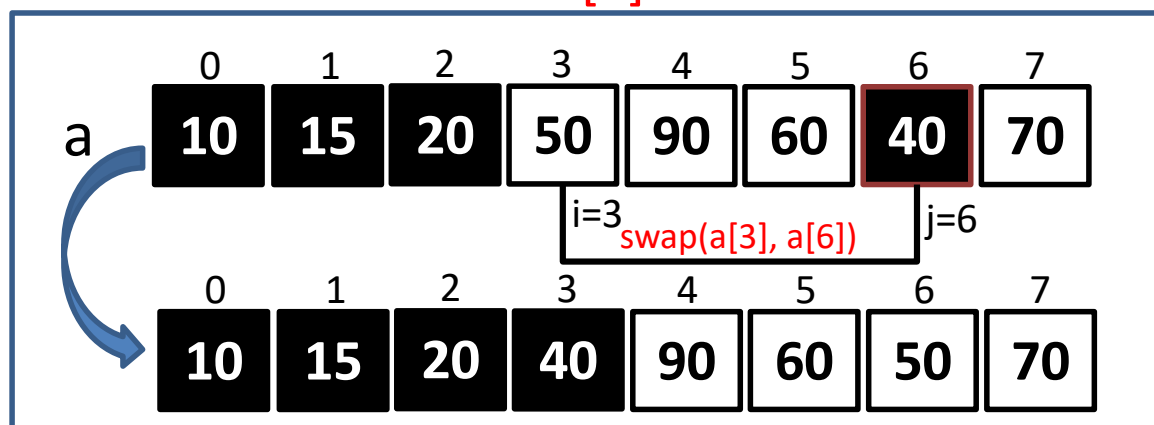
2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 3 → 7

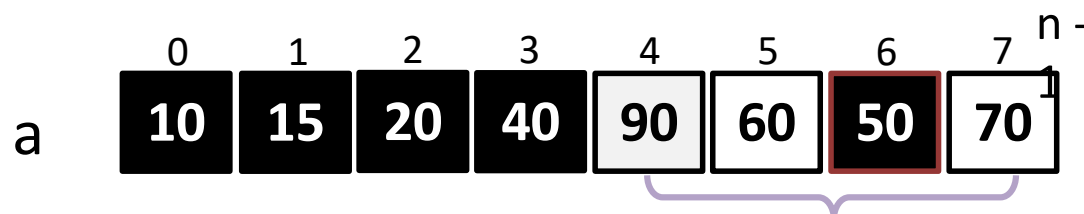


$a[6] = 40$ Là phần tử nhỏ nhất
Đổi giá trị giữa $a[6]$ và $a[3]$



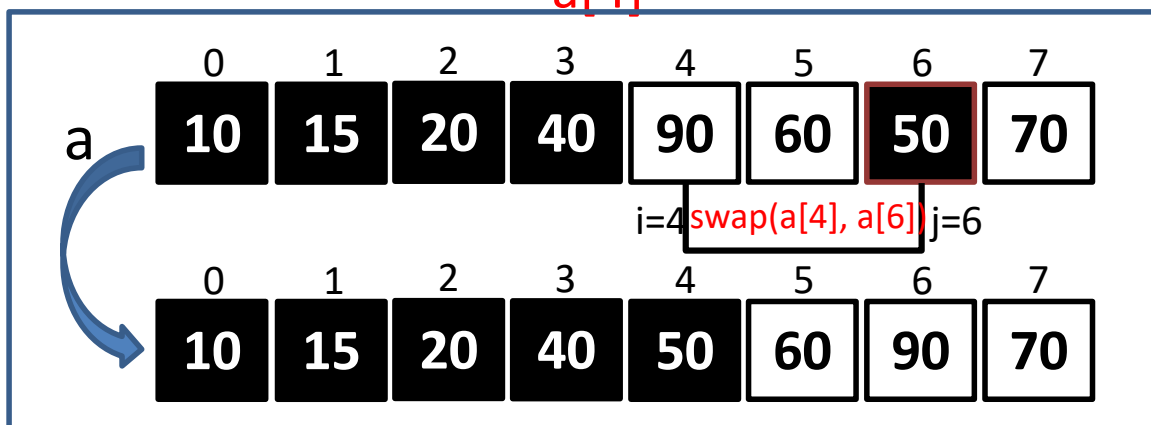
2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 4 → 7



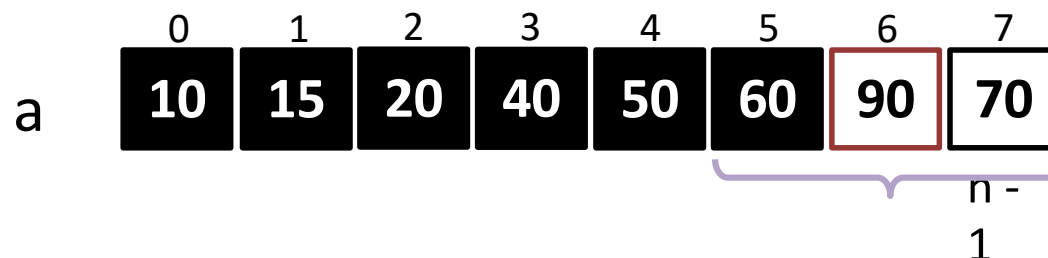
$a[6] = 50$ Là phần tử nhỏ nhất

Đổi giá trị giữa $a[6]$ và
 $a[4]$

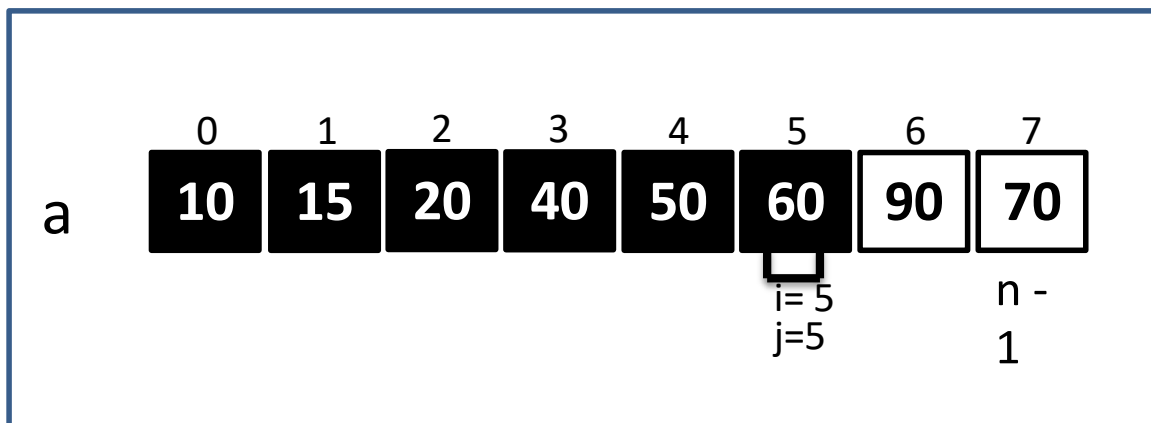


2.1 – XẾP THỨ TỰ (SORT) CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 5 → 7



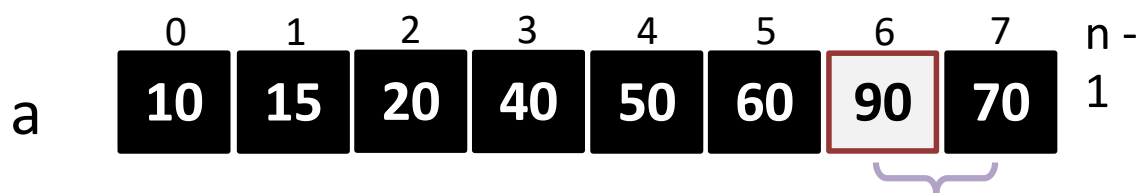
Vì vị trí nhỏ nhất là 5 trùng với vị trí cần sắp là 5, vì vậy không diễn ra động tác đổi chỗ



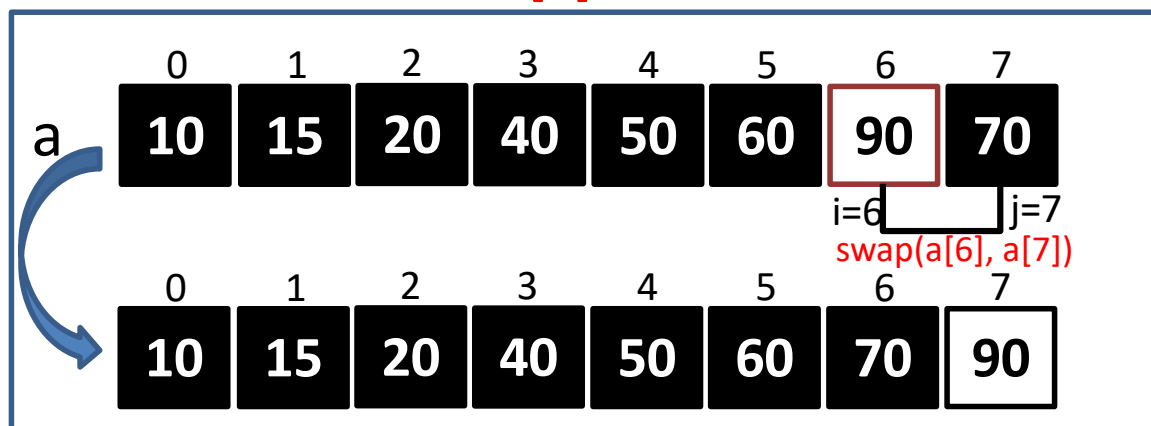
2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 5 → 7



$a[7] = 70$ Là phần tử nhỏ nhất
Đổi giá trị giữa $a[7]$ và $a[6]$



2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

CHƯƠNG TRÌNH

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

min_pos là vị trí chứa giá trị hiện nhỏ nhất

```
void swap(int &a, int &b)
{
    int c;
    c=a;
    a=b;
    b=c;
}
```

2.1 – XẾP THỨ TỰ (SORT)

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

$$\text{Số lần so sánh} = \sum_{i=1}^{n-1} (n - i) = \frac{n(n-1)}{2}$$

TRƯỜNG HỢP	SỐ LẦN SO SÁNH	SỐ LẦN HOÁN VỊ
Tốt nhất	$\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$

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

Selection sort

- Sắp xếp tăng dần dãy số sau:

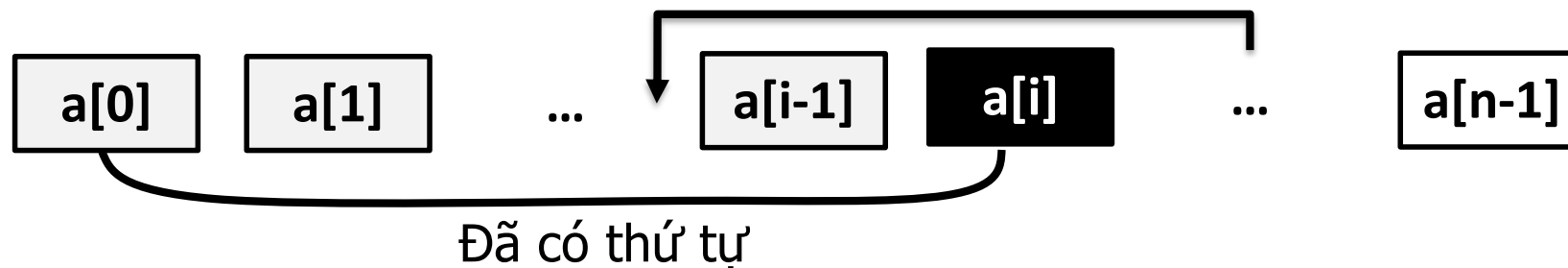
65 73 88 12 100 93 27

2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT

PHƯƠNG PHÁP

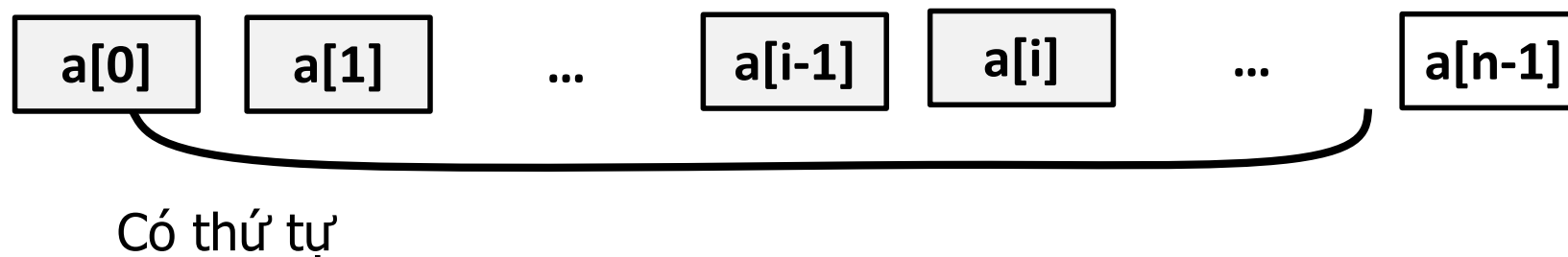
- Trong danh sách $a[0], a[1], \dots, a[n-1]$, giả sử các phần tử $a[0], a[1], \dots, a[i]$ đã có thứ tự.
- Ta tìm cách chèn phần tử $a[i]$ vào vị trí thích hợp của đoạn đã được sắp thứ tự $a[0], a[1], \dots, a[i-1]$, để có dãy mới $a[0], a[1], \dots, a[i]$ trở nên có thứ tự.

2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT



Chèn $a[i]$ vào vị trí thích hợp của đoạn danh sách có thứ tự $a[0]$, $a[1], \dots, a[i-1]$

Để đoạn $a[0], a[1], \dots, a[i-1], a[i]$ có thứ tự



2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT

THUẬT TOÁN

Bước 1: $i=1$; // đoạn $a[0]$, có 1 phần tử được xem là danh sách (danh sách có một phần tử) đã được xếp thứ tự

Bước 2: Thực hiện gán giá trị: $x = a[i]$;

Bước 3: Tìm j (j đi từ vị trí $i-1$ sang trái). j là vị trí phần tử ở đầu tiên mà $a[j]$ nhỏ hơn hoặc bằng x . Do đó $j+1$ là vị trí thích hợp chèn x vào. Tịnh tiến đoạn các phần tử từ $a[j+1]$ đến $a[i-1]$ sang phải 1 vị trí.

Bước 4: Thực hiện gán giá trị $a[j+1] = x$; // $j+1$ là vị trí thích hợp chèn x vào.

Bước 5: Xét vị trí i tiếp theo ($i++$) và **NẾU** $i < n$: **LẶP LẠI** bước 2

2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT

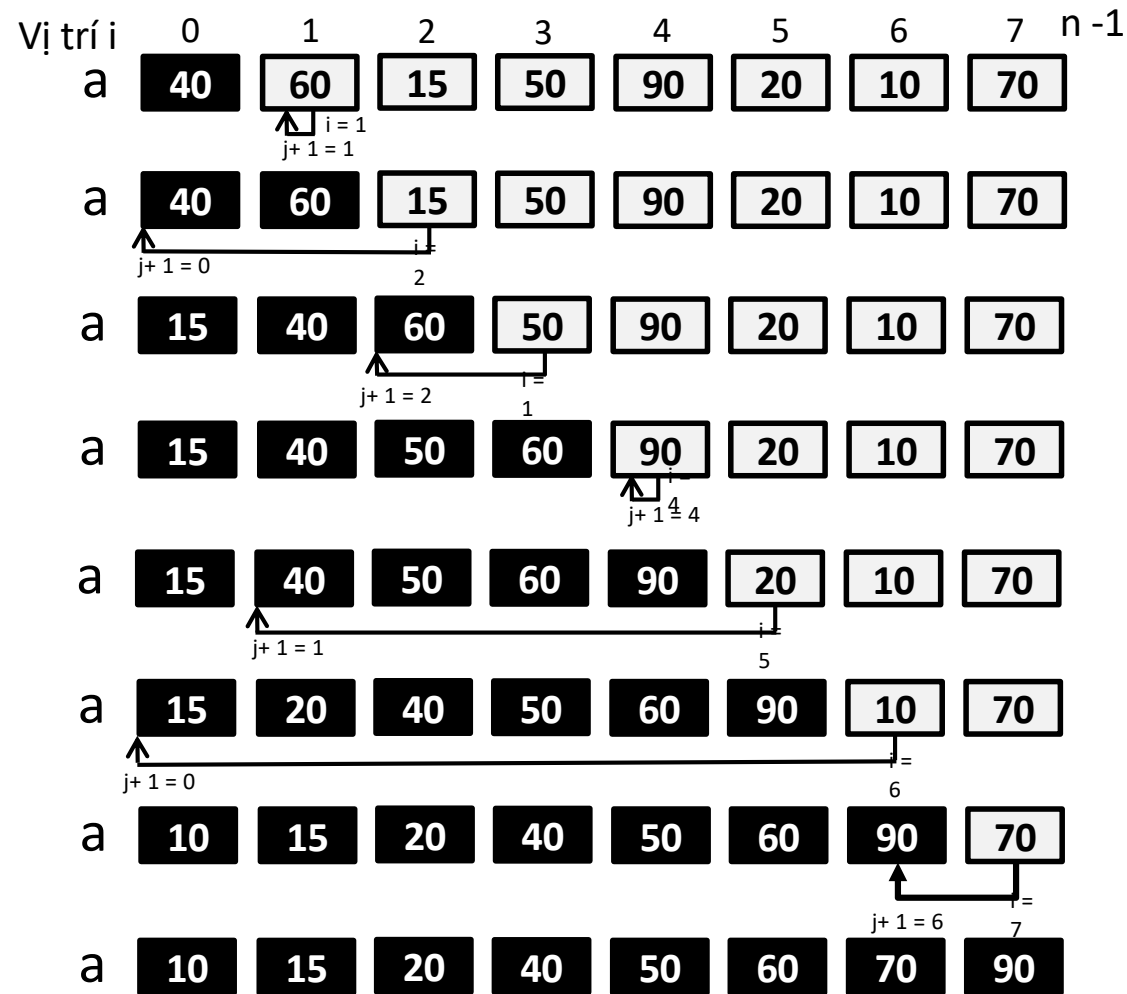
VÍ DỤ

- Cho mảng danh sách đặc $a[]$ như sau:

Phần tử:	40	60	15	50	90	20	10	70
Vị trí:	0	1	2	3	4	5	6	7

2.1 – XẾP THỨ TỰ (SORT)

CHÈN TRỰC TIẾP – INSERTION SORT



2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT

CHƯƠNG TRÌNH

```
void InsertionSort(int a[], int n)
{
    int x, i, j;
    for(int i=1; i<n; i++)
    {
        x = a[i]; // biến x lưu giá trị a[i]
        j = i-1;
        while(0<=j && x <a[j])
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1]=x;
    }
}
```

2.1 – XẾP THỨ TỰ (SORT) CHÈN TRỰC TIẾP – INSERTION SORT

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

TRƯỜNG HỢP	SỐ LẦN SO SÁNH	SỐ LẦN GÁN
Tốt nhất	$n-1$	$2(n-1)$
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n+1)}{2} - 1$

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

Insertion sort

- Sắp xếp tăng dần dãy số sau:

65 73 88 12 100 93 27



TRƯỜNG ĐẠI HỌC MỞ TP. HCM
Cơ hội học tập cho mọi người

2.1 – XẾP THỨ TỰ (SORT) **NỔI BỌT – BUBBLE SORT**

2.1 – XẾP THỨ TỰ (SORT) NỔI BẬT – BUBBLE SORT

PHƯƠNG PHÁP

- Với một danh sách đặc a , có n phần tử từ $a[0]$ đến $a[n-1]$ như sau: $a[0], a[1], a[2], a[3], \dots, a[n-1]$

Phần tử:	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[n-1]$
Vị trí:	0	1	2	3	$n-1$

	0	1	2	3	4	5	6	7
a	40	60	15	50	90	20	10	70

$n-1$
1

2.1 – XẾP THỨ TỰ (SORT) NỔI BẬT – BUBBLE SORT

Bắt đầu từ cuối danh sách, so sánh *các cặp phần tử kế nhau*.
Hoán vị hai phần tử trong cùng một cặp nếu phần tử nhỏ
đứng sau.

Tiếp tục so sánh các cặp phần tử để đưa phần tử nhỏ nhất về
đầu dãy. Sau đó sẽ không xét đến *phần tử nhỏ nhất này* ở
bước tiếp theo, 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

2.1 – XẾP THỨ TỰ (SORT) NỒI BỌT – BUBBLE SORT

THUẬT TOÁN

Bước 1: $i=0$;

Bước 2: $j = n-1$; // Bắt đầu từ vị trí cuối danh sách

Lặp lại trong khi ($j>i$):

Nếu $a[j] < a[j-1]$

$\text{swap}(a[j], a[j-1]);$

Bước 3: $i++$;

Nếu $i < n-1$: lặp lại bước 2.

2.1 – XẾP THỨ TỰ (SORT) NỔI BẬT – BUBBLE SORT

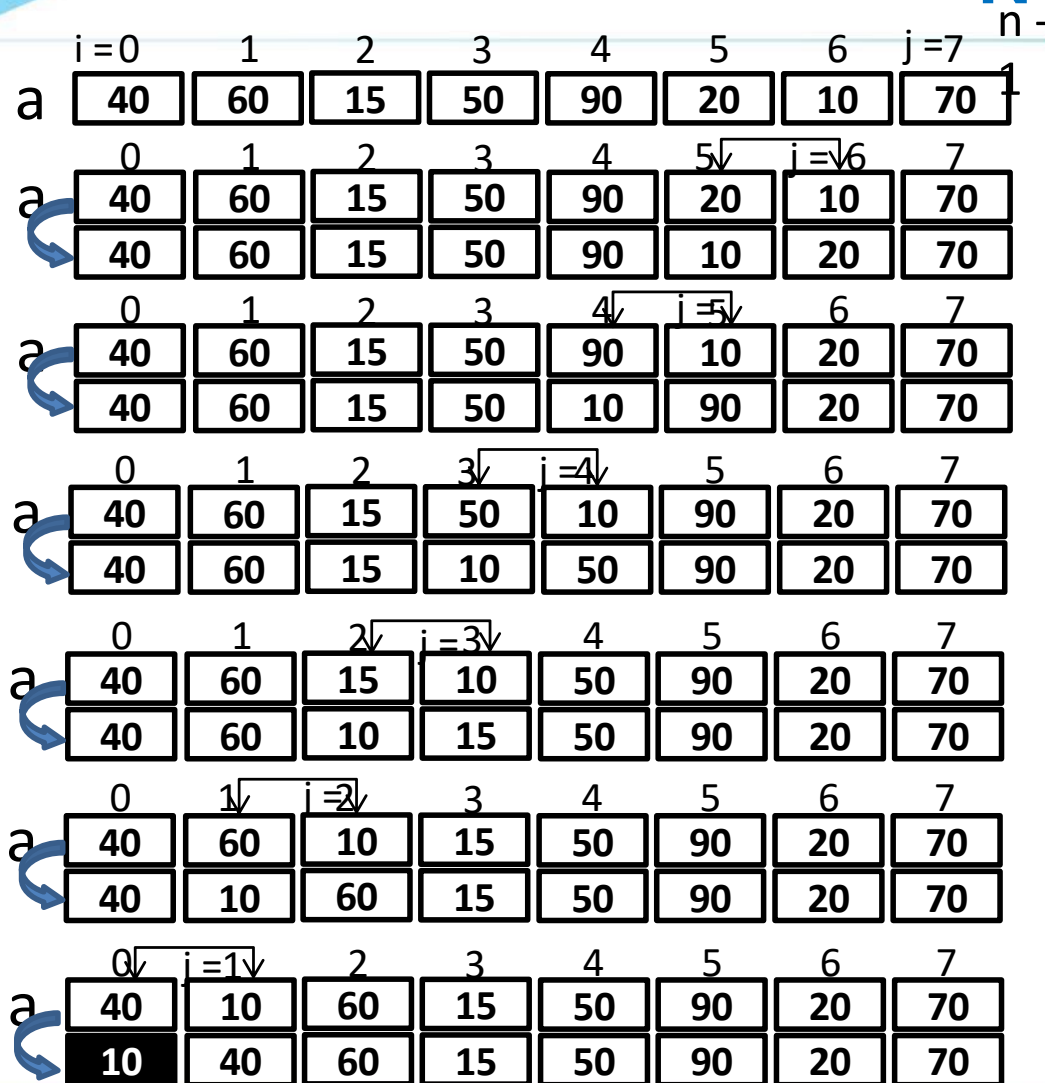
VÍ DỤ

- Cho mảng danh sách đặc $a[]$ như sau:

Phần tử:	40	60	15	50	90	20	10	70
Vị trí:	0	1	2	3	4	5	6	7

2.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT



$j = 7$, vì $a[j]$ lớn hơn $a[j-1]$ nên không đổi chỗ.

$j = 6$, vì $a[6]$ nhỏ hơn $a[5]$ nên đổi chỗ giá trị giữa $a[6]$ và $a[5]$

$j = 5$, vì $a[5]$ nhỏ hơn $a[4]$ nên đổi chỗ giá trị giữa $a[5]$ và $a[4]$

$j = 4$, vì $a[4]$ nhỏ hơn $a[3]$ nên đổi chỗ giá trị giữa $a[4]$ và $a[3]$

$j = 3$, vì $a[3]$ nhỏ hơn $a[2]$ nên đổi chỗ giá trị giữa $a[3]$ và $a[2]$

$j = 2$, vì $a[2]$ nhỏ hơn $a[1]$ nên đổi chỗ giá trị giữa $a[2]$ và $a[1]$

$j = 1$, vì $a[1]$ nhỏ hơn $a[0]$ nên đổi chỗ giá trị giữa $a[1]$ và $a[0]$

2.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

a

0	i=1	2	3	4	5	6	i=7
10	40	60	15	50	90	20	70

$j = 7$, vì $a[j]$ lớn hơn $a[j-1]$ nên không đổi chỗ.

a

0	1	2	3	4	5	i=6	7
10	40	60	15	50	90	20	70
10	40	60	15	50	20	90	70

$j = 6$, vì $a[6]$ nhỏ hơn $a[5]$ nên đổi chỗ giá trị giữa $a[6]$ và $a[5]$

a

0	1	2	3	4	5	6	7
10	40	60	15	50	20	90	70
10	40	60	15	20	50	90	70

$j = 5$, vì $a[5]$ nhỏ hơn $a[4]$ nên đổi chỗ giá trị giữa $a[5]$ và $a[4]$

a

0	1	2	3	i=4	5	6	7
10	40	60	15	20	50	90	70

$j = 4$, vì $a[4]$ lớn hơn $a[3]$ nên không đổi chỗ

a

0	1	2	3	4	5	6	7
10	40	60	15	20	50	90	70
10	40	15	60	20	50	90	70

$j = 3$, vì $a[3]$ nhỏ hơn $a[2]$ nên đổi chỗ giá trị giữa $a[3]$ và $a[2]$

a

0	1	2	3	4	5	6	7
10	40	15	60	20	50	90	70
10	15	40	60	20	50	90	70

$j = 2$, vì $a[2]$ nhỏ hơn $a[1]$ nên đổi chỗ giá trị giữa $a[2]$ và $a[1]$

2.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

	0	1	$i=2$	3	4	5	6	$j=7$
a	10	15	40	60	20	50	90	70
	10	15	40	60	20	50	70	90

$j = 7$, vì $a[7]$ nhỏ hơn $a[6]$ nên đổi chỗ giá trị giữa $a[7]$ và $a[6]$

	0	1	2	3	4	5	$i=6$	7
a	10	15	40	60	20	50	70	90

$j = 6$, vì $a[6]$ lớn hơn $a[5]$ nên không đổi chỗ

	0	1	2	3	4	$i=5$	6	7
a	10	15	40	60	20	50	70	90

$j = 5$, vì $a[5]$ lớn hơn $a[4]$ nên không đổi chỗ

	0	1	2	3	$i=4$	5	6	7
a	10	15	40	60	20	50	70	90
	10	15	40	20	60	50	70	90

$j = 4$, vì $a[4]$ nhỏ hơn $a[3]$ nên đổi chỗ giá trị giữa $a[4]$ và $a[3]$

	0	1	2	$i=3$	4	5	6	7
a	10	15	40	20	60	50	70	90
	10	15	20	40	60	50	70	90

$j = 3$, vì $a[3]$ nhỏ hơn $a[2]$ nên đổi chỗ giá trị giữa $a[3]$ và $a[2]$

2.1 – XẾP THỨ TỰ (SORT) NỔI BỌT – BUBBLE SORT

a

0	1	2	$j=3$	4	5	6	$j=7$
10	15	20	40	60	50	70	90

$j = 7$, vì $a[7]$ lớn hơn $a[6]$ nên không đổi chỗ

a

0	1	2	3	4	5	$j=6$	7
10	15	20	40	60	50	70	90

$j = 6$, vì $a[6]$ lớn hơn $a[5]$ nên không đổi chỗ

a

0	1	2	3	4	$j=5$	6	7
10	15	20	40	60	50	70	90
10	15	20	40	50	60	70	90

$j = 5$, vì $a[5]$ nhỏ hơn $a[4]$ nên đổi chỗ giá trị giữa $a[5]$ và $a[4]$

a

0	1	2	3	$j=4$	5	6	7
10	15	20	40	50	60	70	90

$j = 4$, vì $a[4]$ lớn hơn $a[3]$ nên không đổi chỗ

2.1 – XẾP THỨ TỰ (SORT) NỔI BỌT – BUBBLE SORT

a

0	1	2	3	$i=4$	5	6	$j=7$
10	15	20	40	60	50	70	90

$j = 7$, vì $a[7]$ lớn hơn $a[6]$ nên không đổi chỗ

a

0	1	2	3	4	5	$i=6$	7
10	15	20	40	60	50	70	90

$j = 6$, vì $a[6]$ lớn hơn $a[5]$ nên không đổi chỗ

a

0	1	2	3	4	$i=5$	6	7
10	15	20	40	60	50	70	90
10	15	20	40	50	60	70	90

$j = 5$, vì $a[5]$ nhỏ hơn $a[4]$ nên đổi chỗ giá trị giữa $a[5]$ và $a[4]$

2.1 – XẾP THỨ TỰ (SORT) NỒI BỌT – BUBBLE SORT

a

0	1	2	3	4	$j=5$	6	$j=7$
10	15	20	40	50	60	70	90

$j = 7$, vì $a[7]$ lớn hơn $a[6]$ nên không đổi chỗ

a

0	1	2	3	4	5	$j=6$	7
10	15	20	40	50	60	70	90

0	1	2	3	4	5	$j=6$	7
10	15	20	40	50	60	70	90

$j = 6$, vì $a[6]$ lớn hơn $a[5]$ nên không đổi chỗ

2.1 – XẾP THỨ TỰ (SORT) NỔI BỌT – BUBBLE SORT

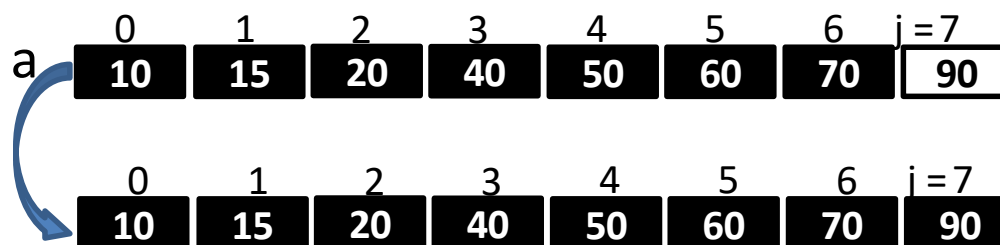
a

0	1	2	3	4	5	$i=6$	$j=7$
10	15	20	40	50	60	70	90

0	1	2	3	4	5	6	$j=7$
10	15	20	40	50	60	70	90

$j = 7$, vì $a[7]$ lớn hơn $a[6]$ nên không đổi chỗ

2.1 – XẾP THỨ TỰ (SORT) NỒI BỌT – BUBBLE SORT



2.1 – XẾP THỨ TỰ (SORT) NỔI BẬT – BUBBLE SORT

CHƯƠNG TRÌNH

```
void BubbleSort(int a[], int n)
{
    for(int i=0; i<n-1; i++)
        for(int j=n-1; j>i; j--)
            if(a[j-1] > a[j]); // xét điều kiện phần tử sau nhỏ hơn phần tử
                                trước
                                swap(a[j],a[j-1]) // hoán vị a[j] với a[j-1]
}
```

2.1 – XẾP THỨ TỰ (SORT) NỔI BẬT – BUBBLE SORT

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

TRƯỜNG HỢP	SỐ LẦN SO SÁNH	SỐ LẦN HOÁN VỊ
Tốt nhất	$\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$

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



2.1 – XẾP THỨ TỰ (SORT) **ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT**

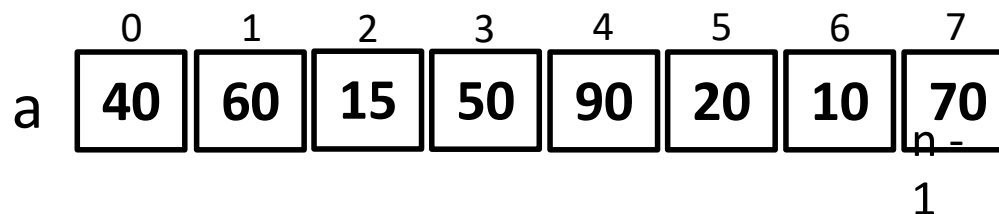
2.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

Ý TƯỞNG INTERCHANGE SORT

- Với một danh sách đặc $a[]$, có n phần tử từ $a[0]$ đến $a[n-1]$ như sau: $a[0], a[1], a[2], a[3], \dots, a[n-1]$

Phần tử:	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[n-1]$
Vị trí:	0	1	2	3	$n-1$



2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

Bắt đầu từ *phần tử ở vị trí đầu dãy*, so sánh *cặp phần tử* đầu dãy (tại vị trí 0) với các phần tử còn lại trong danh sách. Trong các *cặp so sánh*, nếu phần tử ở *vị trí sau nhỏ hơn phần tử ở vị trí trước* thì *hoán vị* hai phần tử này.

Lặp lại bước trên, cho các phần tử ở các vị trí tiếp theo (vị trí thứ 1, 2, ..., $n-2$)

2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

THUẬT TOÁN

Bước 1: Bắt đầu từ vị trí đầu tiên $i=0$ trong danh sách

Bước 2: Xét phần tử tại vị trí $j = i+1$

Lặp lại trong khi ($j \leq n-1$):

{

Nếu $a[j] < a[i]$ thì hoán vị $a[j]$ và $a[i]$

$j++$

}

Bước 3: $i++$;

Nếu $i < n-1$, lặp lại bước 2.

2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

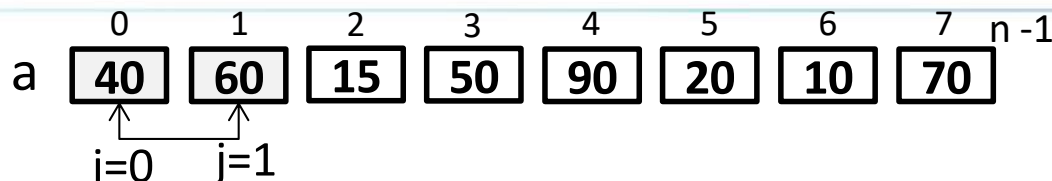
THÍ DỤ

- Cho mảng danh sách đặc $a[]$ như sau:

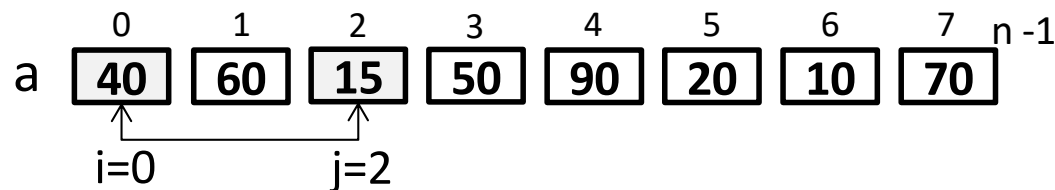
Phần tử:	40	60	15	50	90	20	10	70
Vị trí:	0	1	2	3	4	5	6	7

2.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

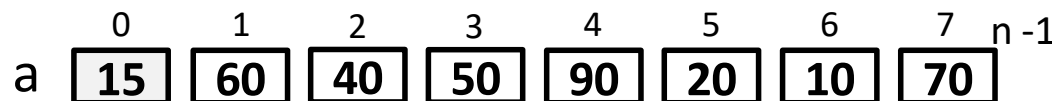


Cặp hai phần tử $a[i] = 40$ (với $i=0$) và $a[j] = 60$ (với $j=1$): $a[i] < a[j]$:
Không hoán vị

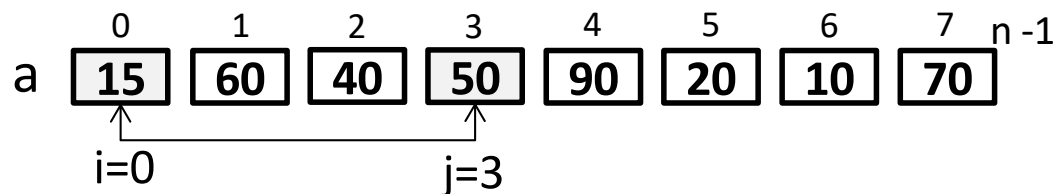


Cặp hai phần tử $a[i] = 40$ (với $i=0$) và $a[j] = 15$ (với $j=2$): $a[i] > a[j]$:
Hoán vị

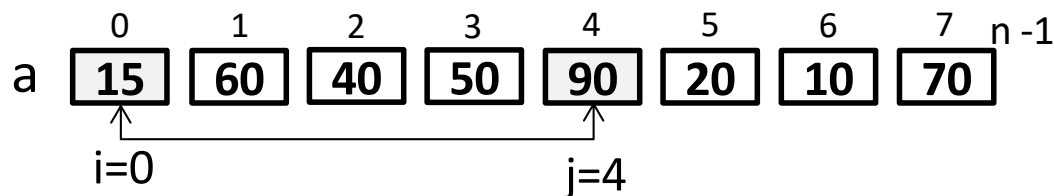
Kết quả sau khi hoán vị 40 và 15, được danh sách sau:



2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

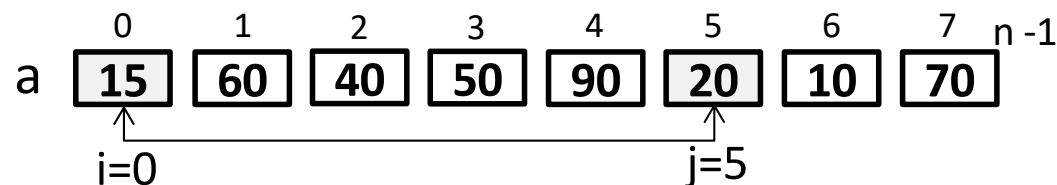


Cặp hai phần tử $a[i] = 15$ (với $i=0$) và $a[j] = 50$ (với $j=3$): $a[i] < a[j]$:
Không hoán vị

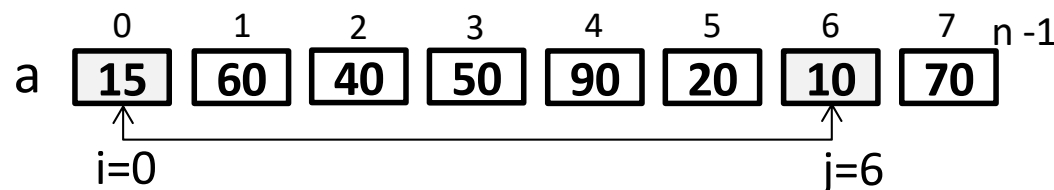


Cặp hai phần tử $a[i] = 15$ (với $i=0$) và $a[j] = 90$ (với $j=4$): $a[i] < a[j]$:
Không hoán vị

2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

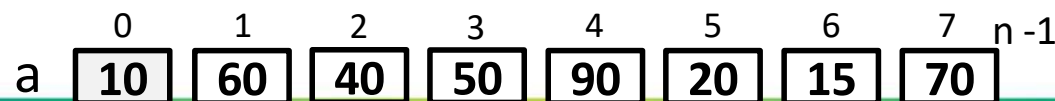


Cặp hai phần tử $a[i] = 15$ (với $i=0$) và $a[j] = 20$ (với $j=5$): $a[i] < a[j]$:
Không hoán vị



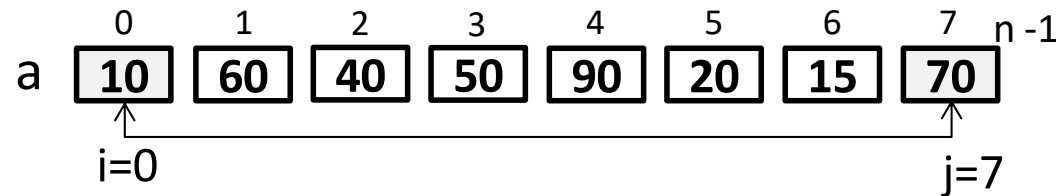
Cặp hai phần tử $a[i] = 15$ (với $i=0$) và $a[j] = 10$ (với $j=6$): $a[i] > a[j]$:
Hoán vị

Kết quả sau khi hoán vị 15 và 10, được danh sách sau

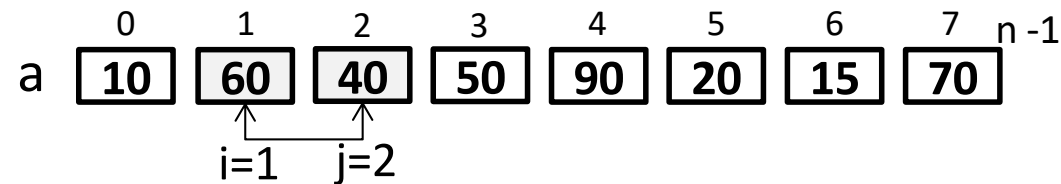


2.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

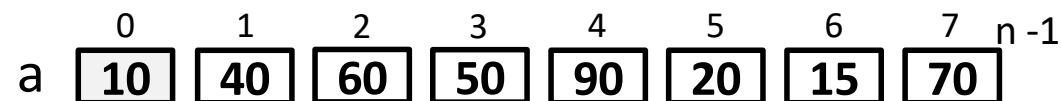


Cặp hai phần tử $a[i] = 10$ (với $i=0$) và $a[j] = 70$ (với $j=7$): $a[i] < a[j]$:
Không hoán vị



Cặp hai phần tử $a[i] = 60$ (với $i=1$) và $a[j] = 40$ (với $j=2$): $a[i] > a[j]$:
Hoán vị

Kết quả sau khi hoán vị 60 và 40, được danh sách sau



2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

Lặp lại các bước trên cho $i=1$, j đi từ 2 đến 7. Và tương tự cho trường hợp $i = 2, 3, 4, 5, 6$

Ta được dãy xếp thứ tự tăng dần:

a $\begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & n-1 \\ \boxed{10} & \boxed{15} & \boxed{20} & \boxed{40} & \boxed{50} & \boxed{60} & \boxed{70} & \boxed{90} \end{matrix}$

2.1 – XẾP THỨ TỰ (SORT) ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

CHƯƠNG TRÌNH

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

2.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

TRƯỜNG HỢP	SỐ LẦN SO SÁNH	SỐ LẦN HOÁN VỊ
Tốt nhất	$\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$

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



TRƯỜNG ĐẠI HỌC MỞ TP. HCM
Cơ hội học tập cho mọi người

Merge sort



www.ou.edu.vn

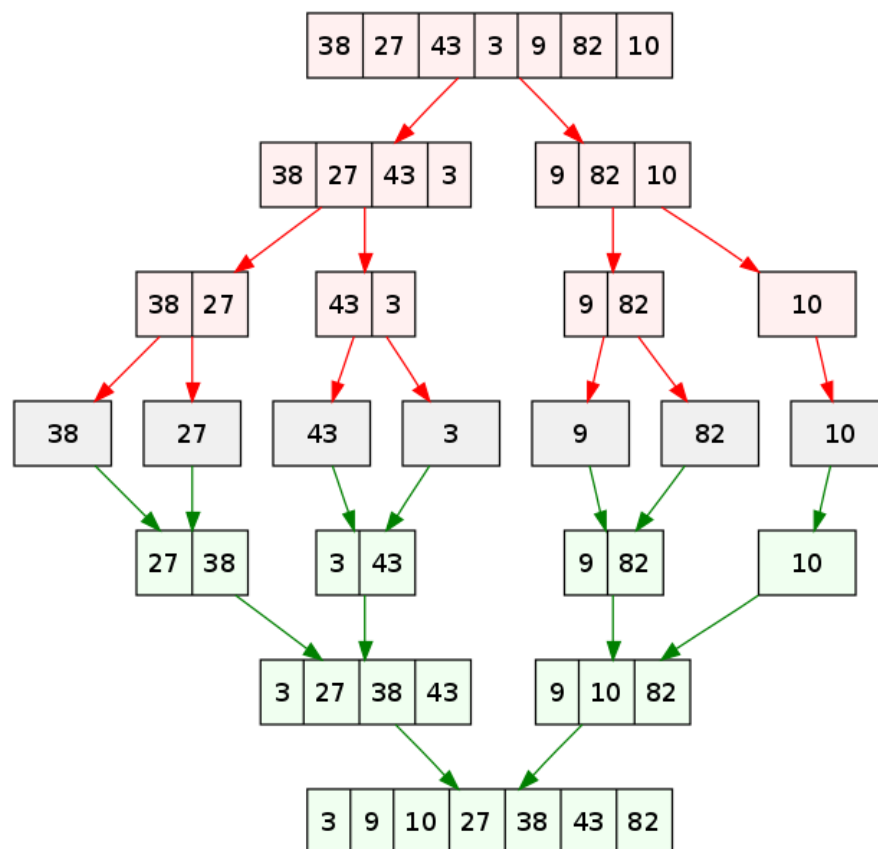
Ý tưởng

- Là thuật toán chia để trị.
- Bước 1: chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia.
- Bước 2: Sau cùng gộp các nửa đó thành mảng đã sắp xếp.

Ý tưởng

- Hàm `merge()` được sử dụng để gộp hai nửa mảng.
- Hàm `merge(arr, l, m, r)` gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là `arr[l...m]` và `arr[m+1...r]` sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

Ví dụ



- Giả sử ta có 2 mảng con lần lượt là:
arr1 = [1 9 10 10] , n1 = 4 // chiều dài của mảng con
arr2 = [3 5 7 9], n2 = 4
sort_arr = [] // Mảng lưu lại tiến trình gộp
- Khởi tạo i = 0, j = 0 tương ứng là chỉ số bắt đầu của arr1 và arr2
- Xét thấy arr1[i] < arr2[j] => chèn arr1[i] vào cuối mảng sort_arr, tăng i lên 1 đơn vị
=> sort_arr = [1], i = 1
- Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
=> sort_arr = [1, 3], i = 1, j = 1
- Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
=> sort_arr = [1, 3, 5], i = 1, j = 2
- Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
=> sort_arr = [1, 3, 5, 7], i = 1, j = 3
- Xét thấy arr1[i] = arr2[j] => chèn arr1[i] hoặc arr2[j] vào cuối mảng sort_arr
Giả sử chọn arr1, tăng i lên 1 đơn vị
=> sort_arr = [1, 3, 5, 7, 9], i = 2, j = 3
- Xét thấy arr1[i] > arr2[j] => chèn arr2[j] vào cuối mảng sort_arr, tăng j lên 1 đơn vị
=> sort_arr = [1, 3, 5, 7, 9, 9], i = 1, j = 4
- Do j >= n2, tiếp tục tăng i chừng nào i < n1 thì thêm phần tử ở arr1[i] vào sort_arr.
- Sau cùng ta được mảng đã sắp xếp là sort_arr = [1, 3, 5, 7, 9, 9, 10, 10]

Cài đặt

```
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    /* Copy dữ liệu sang các mảng tạm */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];
    /* Gộp hai mảng tạm vừa rồi vào mảng arr*/
    i = 0; // Khởi tạo chỉ số bắt đầu của mảng con đầu tiên
    j = 0; // Khởi tạo chỉ số bắt đầu của mảng con thứ hai
    k = l; // Khởi tạo chỉ số bắt đầu của mảng lưu kết quả
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
}
```

```
/* Copy các phần tử còn lại của mảng L vào arr nếu có */
while (i < n1)
{
    arr[k] = L[i];
    i++;
    k++;
}

/* Copy các phần tử còn lại của mảng R vào arr nếu có */
while (j < n2)
{
    arr[k] = R[j];
    j++;
    k++;
}
}
```

Cài đặt

```
void mergeSort(int arr[], int l, int r)
{
    // Tựa gọi đệ quy để chia đôi mảng
    // Tương tự (l+r)/2, nhưng cách này tránh tràn số khi l và r
    // lớn
    int m = l+(r-l)/2;
    // Gọi hàm đệ quy tiếp tục chia đôi từng nửa mảng
    mergeSort(arr, l, m);
    mergeSort(arr, m+1, r);
    merge(arr, l, m, r);
}
```

mergeSort(arr, 0, arr_size - 1);

Độ phức tạp giải thuật

- Trường hợp tốt: $O(n\log(n))$
- Trung bình: $O(n\log(n))$
- Trường hợp xấu: $O(n\log(n))$



2.2 TÌM KIẾM

2.2 – TÌM KIẾM



TÌM KIẾM TUẦN TỰ



TÌM KIẾM NHỊ PHÂN



2.2 – TÌM KIẾM



Trong cấu trúc danh sách đặc, thuật toán tìm kiếm một phần tử trong danh sách có hai thuật toán

- *Tìm kiếm tuần tự*, thuật toán này thường được áp dụng trong trường hợp danh sách chưa được xếp thứ tự
- *Tìm kiếm nhị phân*, thuật toán này được áp dụng trong trường hợp danh sách đã được xếp thứ tự

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ



Ý tưởng của thuật toán tìm kiếm tuần tự

- *Đi từng phần tử từ a_0, a_1, \dots , đến a_{n-1} .*
- *Mỗi lần đi đến thăm a_i kiểm tra X có và a_i có giống nhau không? Nếu có trả lời "có" và dừng chương trình, nếu không có thì đi tiếp đến phần tử tiếp theo (cho đến khi hết phần tử).*

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

- Với một danh sách đặc $a[]$, có n phần tử từ $a[0]$ đến $a[n-1]$ như sau: $a[0], a[1], a[2], a[3], \dots, a[n-1]$

Phần tử:	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[n-1]$
Vị trí:	0	1	2	3	$n-1$

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

Tìm phần tử x có trong danh sách trên, bằng phương pháp tìm kiếm tuần tự sau:

Bước 1: Bắt đầu từ vị trí $i=0$ trong danh sách

Bước 2: Nếu $(x == a[i])$ *tìm thấy* x trong danh sách tại vị trí i và kết thúc;

Ngược lại tăng i lên một giá trị

Lặp lại bước 2

THÍ DỤ 1:

- Với một danh sách đặc $a[]$ như sau:

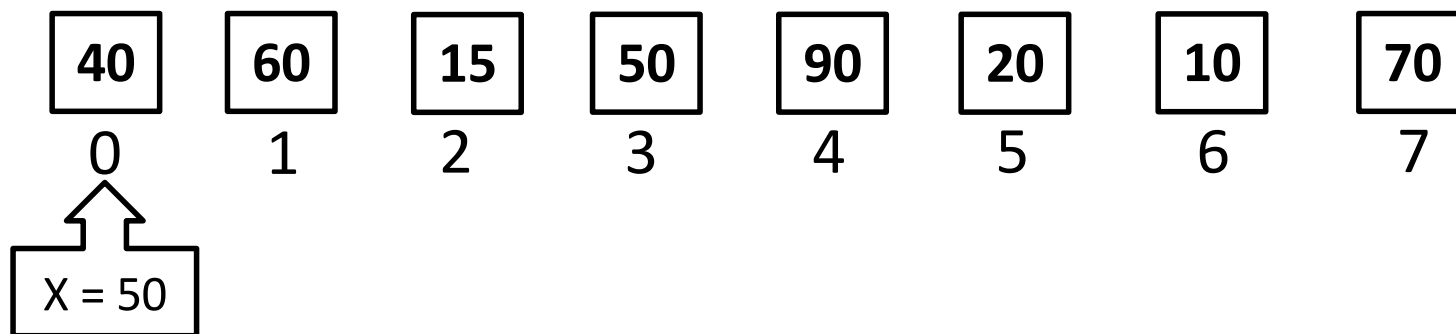
Phần tử:	40	60	15	50	90	20	10	70
Vị trí i:	0	1	2	3	4	5	6	7

Yêu cầu: Tìm giá trị $x = 50$ trong danh sách.

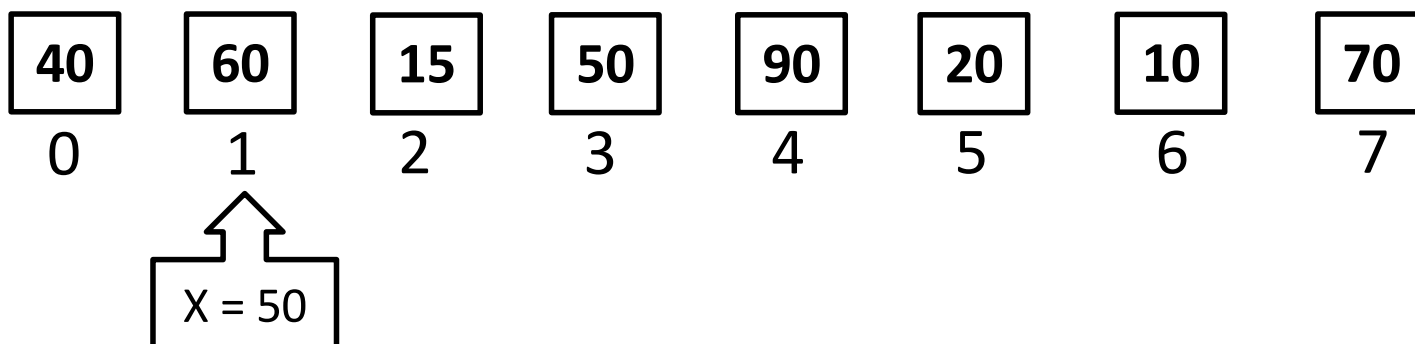
40	60	15	50	90	20	10	70
0	1	2	3	4	5	6	7

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

Đầu tiên so sánh giá trị x với $a[i]$ (với $i=0$)

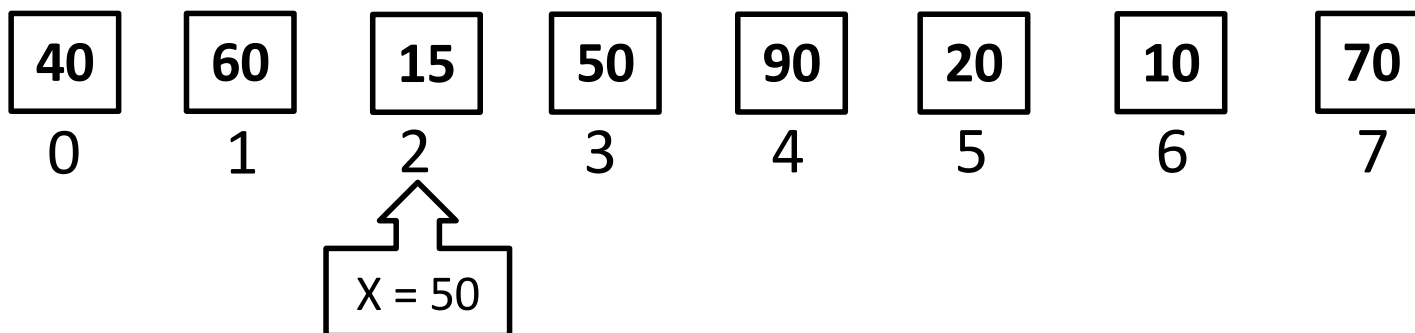


$a[i] \neq x$ (với $i = 0$). Tăng i lên một giá trị, so sánh $a[1]$ với x

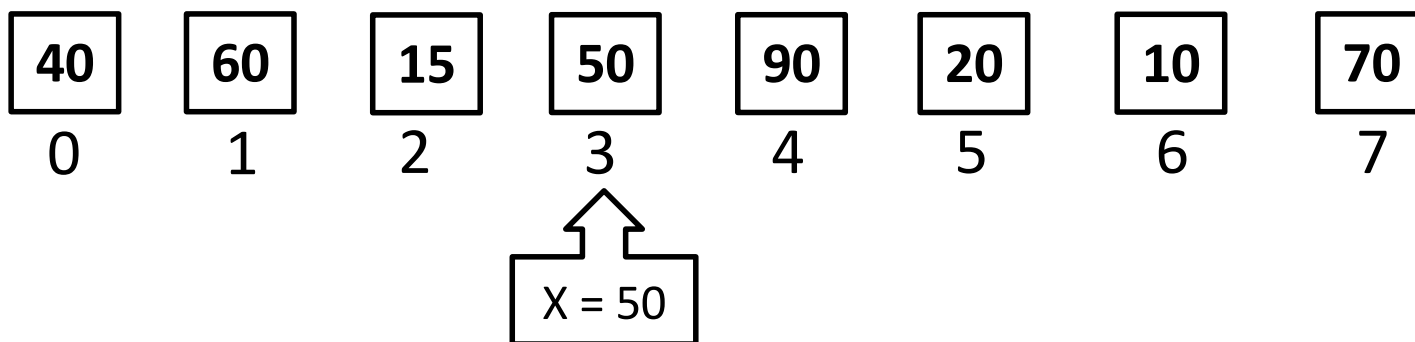


$a[i] \neq x$ (với $i = 1$). Tăng i lên một giá trị, so sánh $a[2]$ với x

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ



$a[i] \neq x$ (với $i = 2$). Tăng i lên một giá trị, so sánh $a[3]$ với x



$a[i] = x$ (với $i = 3$). Tìm thấy giá trị $x = 50$ tại vị trí $i = 3$ trong danh sách.
Kết thúc return giá trị 3 (vị trí tìm thấy $x = 50$ trong danh sách)

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

THÍ DỤ 2:

Xét danh sách sau:

Cho danh sách đặc $a[]$ như sau:

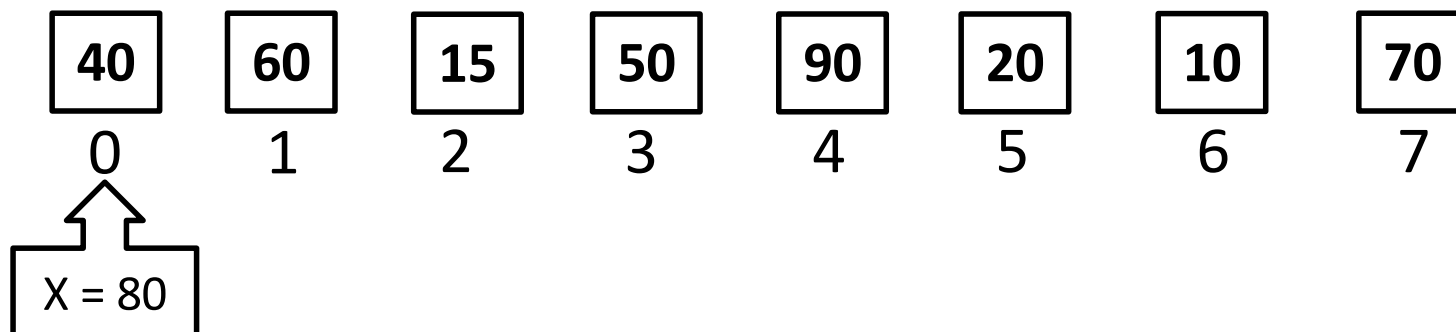
Phần tử:	40	60	15	50	90	20	10	70
Vị trí i:	0	1	2	3	4	5	6	7

Yêu cầu: Tìm giá trị $x = 80$ trong danh sách.

40	60	15	50	90	20	10	70
0	1	2	3	4	5	6	7

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

Đầu tiên so sánh giá trị x với $a[i]$ (với $i=0$)



$a[i] \neq x$ (với $i = 0$). Tăng i lên một giá trị, so sánh $a[1]$ với x

Lặp lại tương tự như trên cho trường hợp $i = 1, 2, 3, 4, 5, 6, 7$

Khi $i = 7$, mà $a[i]$ vẫn khác x . khi đó tăng i lên 1 giá trị ($i = 8, i = n$), kết thúc.

Tim không thấy, return giá trị -1

2.2 – TÌM KIẾM TÌM KIẾM TUẦN TỰ

CHƯƠNG TRÌNH

```
int Search(int a[], int n, int X)
{
    int i=0;
    while(i<n && a[i]!=X)
        i++;
    if(i < n)
        return i; // x trong danh sách a, và nằm ở vị trí i
    return -1; // không tìm thấy x trong danh sách a;
}
```

Độ phức tạp của phương pháp tìm tuần tự

ĐỘ PHỨC TẠP	SỐ LẦN SO SÁNH
Trường hợp tốt nhất	1
Trường hợp xấu nhất	n
Trường hợp trung bình	$\frac{n+1}{2}$

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

Tìm kiếm nhị phân được thực hiện trên một danh sách đã được xếp thứ tự

Với một mảng danh sách đặc $a[]$ có n phần tử đã có thứ tự tăng dần từ phần tử $a[0]$ đến $a[n-1]$ như sau: $a[0] \leq a[1] \leq a[2] \leq a[3] \leq \dots \leq a[n-1]$

Phần tử:	$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[n-1]$
----------	--------	--------	--------	--------	-----	-----	----------

Vị trí:	0	1	2	3	$n-1$
---------	---	---	---	---	-----	-----	-------

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

THUẬT TOÁN:

Tìm phần tử x có trong danh sách trên, bằng phương pháp tìm kiếm tuần tự sau:

Bước 1: $\text{left} = 0, \text{right} = n - 1;$

Bước 2:

Nếu $(x == a[(\text{left} + \text{right})/2])$: Tìm thấy thì kết thúc;

Ngược lại Nếu $x < a[(\text{left} + \text{right})/2]$ lặp lại bước 2 cho dãy từ vị trí left đến $(\text{left} + \text{right})/2 - 1;$

Ngược lại: lặp lại bước 2 cho dãy từ vị trí $(\text{left} + \text{right})/2 + 1$ đến right

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

THÍ DỤ 1:

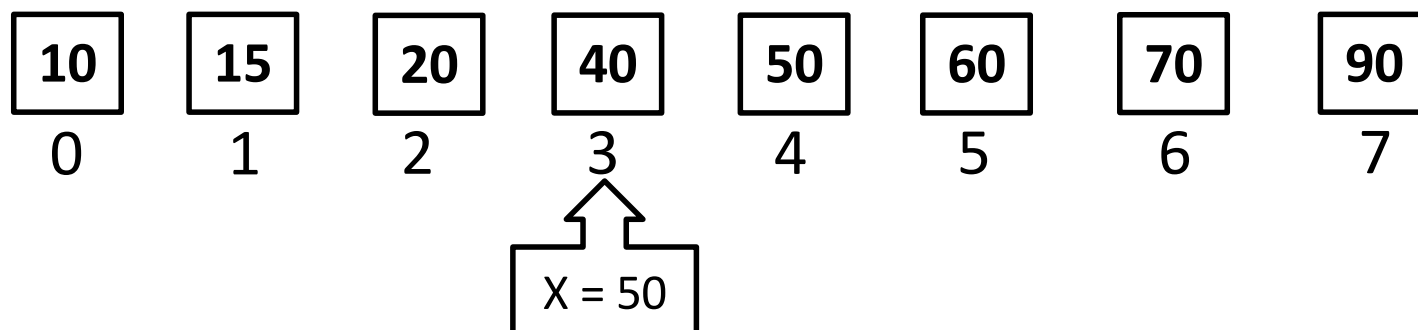
Yêu cầu: Tìm giá trị $x = 50$ trong danh sách sau:

10	15	20	40	50	60	70	90
0	1	2	3	4	5	6	7

left = 0, right = 7

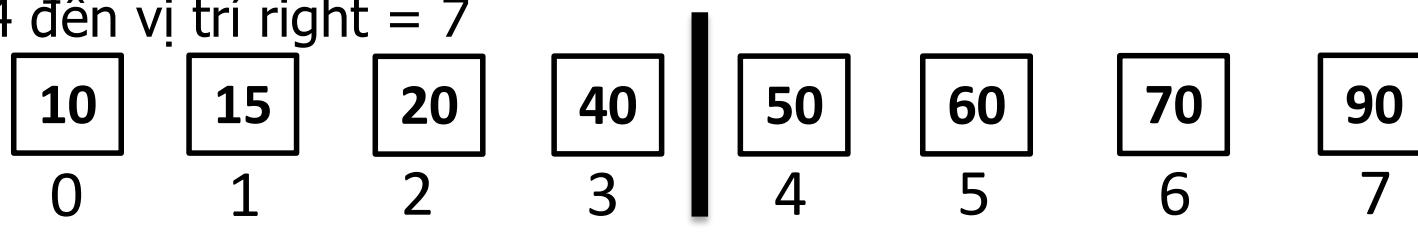
2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

Bước 1: So sánh x với phần tử tại vị trí $(\text{left} + \text{right})/2 = (0 + 7)/2 = 3$



Ta có $x = 50 > a[(\text{left} + \text{right})/2]$

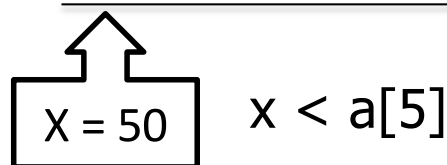
Bước 2: Giới hạn phạm vi tìm kiếm $x = 50$ trên đoạn từ $(\text{left} + \text{right})/2 + 1$
 $= 4$ đến vị trí $\text{right} = 7$



So sánh x với phần tử $a[(4+7)/2] = a[5] = 60$

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

10	15	20	40	50	60	70	90
0	1	2	3	4	5	6	7

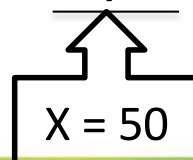


Giới hạn phạm vi tìm kiếm trong đoạn các phần tử từ: $a[4]$ đến $a[4]$

10	15	20	40	50	60	70	90
0	1	2	3	4	5	6	7

So sánh $x = 50$ với phần tử $a[(4+4)/2] = a[4]$

10	15	20	40	50	60	70	90
0	1	2	3	4	5	6	7



$x = a[4]$. Tìm thấy. Kết thúc

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

CHƯƠNG TRÌNH

```
int BinarySearch(int a[], int n, int X)
{
    int left=0, right=n-1, mid;
    while(left<=right)
    {
        mid=(left+right)/2;
        if(a[mid]==X) return mid;
        if(x>a[mid]) left=mid+1;
        else right=mid-1;
    }
    return -1; // không tìm thấy x trong danh sách a;
}
```

2.2 – TÌM KIẾM TÌM KIẾM NHỊ PHÂN

ĐỘ PHỨC TẠP

TRƯỜNG HỢP	ĐỘ PHỨC TẠP
Trường hợp tốt nhất	$O(n)$
Trường hợp xấu nhất	$O(\log n)$
Trường hợp trung bình	$O(\log n)$

Độ phức tạp: **$O(\log n)$**

2.3 – TỔNG KẾT CHƯƠNG 2



2.3 – Tổng kết chương 2



Trong chương xếp thứ tự và tìm kiếm, các **thuật toán xếp thứ tự** nội đã được trình bày như: *Selection Sort, Insertion Sort, Interchange Sort, Bubble Sort, Merge Sort.*



Thuật toán tìm kiếm tuần tự và tìm kiếm nhị phân

A photograph of a stack of five books with blue, purple, white, orange, and grey spines, leaning against an open silver laptop. The background is a bright, out-of-focus window. The text '2.4- Bài tập rèn luyện' is in a blue sans-serif font, and 'CHƯƠNG 2' is in a larger, bold blue sans-serif font, both centered over the image.

2.4- Bài tập rèn luyện **CHƯƠNG 2**

2.4 - Bài tập chương 2

CÂU HỎI



Câu 1: Trong các phương pháp xếp thứ tự đã học, phương pháp nào tối ưu nhất, và kém tối ưu nhất? Tại sao?



Câu 2: Trong các 2 phương pháp tìm kiếm đã học, trường hợp nào thì cả 02 phương pháp đều như nhau? Giải thích tại sao?



Câu 3: Ngoài các phương pháp xếp thứ tự đã học, hãy tìm hiểu thêm một phương pháp xếp thứ tự khác, giới thiệu sơ và giải thích.

2.4 - Bài tập chương 2

BÀI TẬP THỰC HÀNH

Bài 1: Quản lý danh sách đặc 100 phần tử kiểu số nguyên (int)

1.1 Khai báo cấu trúc danh sách.

1.2 Viết thủ tục nhập danh sách.

1.3 Viết thủ tục xuất danh sách

1.4 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán InsertionSort. Đánh giá độ phức tạp của thuật toán.

1.5 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán SelectionSort. Đánh giá độ phức tạp của thuật toán.

1.6 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán InterchangeSort. Đánh giá độ phức tạp của thuật toán.

2.4 - Bài tập chương 2

BÀI TẬP THỰC HÀNH

1.7 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán BubbleSort. Đánh giá độ phức tạp của thuật toán.

1.8 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán Merge Sort. Đánh giá độ phức tạp của thuật toán.

1.9 Viết thủ tục tìm kiếm một phần tử trong danh sách có thứ tự (dung phương pháp tìm kiếm tuần tự). Đánh giá độ phức tạp của thuật toán

1.10 Viết thủ tục tìm kiếm một phần tử trong danh sách có thứ tự (dung phương pháp tìm kiếm nhị phân). Đánh giá độ phức tạp của thuật toán

2.4 - Bài tập chương 2

BÀI TẬP LÀM THÊM

Bài 2: Một danh sách các phần tử được lưu trữ trong một danh sách đặc, có các phần tử sau: 40, 70, 20, 60, 90, 10, 50, 30. Yêu cầu:

2.1 Dùng phương pháp xếp thứ tự InsertionSort, mô tả từng bước quá trình xếp thứ tự dãy số trên (không lập trình). Tính độ phức tạp của quá trình xếp thứ tự danh sách trên.

2.2 Dùng phương pháp xếp thứ tự SelectionSort, mô tả từng bước quá trình xếp thứ tự dãy số trên (không lập trình). Tính độ phức tạp của quá trình xếp thứ tự danh sách trên.

2.3 Dùng phương pháp xếp thứ tự InterchangeSort, mô tả từng bước quá trình xếp thứ tự dãy số trên (không lập trình). Tính độ phức tạp của quá trình xếp thứ tự danh sách trên.

2.4 - Bài tập chương 2

BÀI TẬP LÀM THÊM

2.4 Dùng phương pháp xếp thứ tự BubbleSort, mô tả từng bước quá trình xếp thứ tự dãy số trên (không lập trình). Tính độ phức tạp của quá trình xếp thứ tự danh sách trên.

2.5 Dùng phương pháp xếp thứ tự MergeSort, mô tả từng bước quá trình xếp thứ tự dãy số trên (không lập trình). Tính độ phức tạp của quá trình xếp thứ tự danh sách trên.

2.6 Sau khi xếp thứ tự danh sách trên. Yêu cầu: Tính độ phức tạp của quá trình tìm kiếm giá trị 90 trong danh sách trên cho cả hai thuật toán tìm kiếm tuần tự và tìm kiếm nhị phân

Tài liệu tham khảo

- **Lê Xuân Trường**, (Chương 2) *Cấu trúc dữ liệu*, NXB Trường Đại học Mở TP-HCM, 2016.
- **Dương Anh Đức**, *Giáo trình cấu trúc dữ liệu & giải thuật (Chương 1)*, 2010, ĐH KHTN TP.HCM
- **Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Clifford Stein**, (Chapter 2, 3) *Introduction to Algorithms*, Third Edition, 2009.
- **Adam Drozdek**, (Chapter 9) *Data Structures and Algorithms in C++*, Fourth Edition, CENGAGE Learning, 2013.

KẾT THÚC CHƯƠNG 2



Trường Đại học Mở TP.HCM

Khoa Công Nghệ Thông Tin