TRƯỜNG ĐẠI HỌC MỞ TP. HCM
Cơ hội học tập cho mọi người
Khoa Công Nghệ Thông Tin

Chương 3
XẾP THỨ TỰ VÀ TÌM KIẾM

GV: Lê Ngọc Hiếu

TP.HCM - Tháng 06 -2019

Mở đầu

Kiến thức cần thiết khi tìm hiểu về XẾP THỨ TỰ, TÌM KIẾM:

- Các CTDL cơ bản: Danh sách đặc, DSLK, DS Hạn chế (Stack, Queue)
- Kiểu dữ liệu cơ bản, dữ liệu lưu trữ trong máy tính.
- Các kiến thức về cơ sở lập trình & kỹ thuật lập trình.

Kỹ năng cần có:

- Có thể sử dụng Visual Studio 2015
- Có thể lập trình C++

Mục tiêu dạy học

- Cung cấp cho người học các kiến thức xếp thứ tự như: *SelectionSort, InsertSort, InterChangeSort, BubbleSort, MergeSort*, … độ phức tạp của các thuật toán sắp xếp; và các thuật toán tìm kiếm phần tử.
- Rèn luyện kỹ năng *lập trình xếp thứ tự* danh sách, *tìm kiếm một phần tử* trong danh sách.
- Có khả năng áp dụng các thuật toán xếp thứ tự danh sách và tìm kiếm một phần tử trong danh sách vào các bài toán giải quyết các vấn đề thực tế.

Nội dung chính

3.1 Xếp thứ tự

- Selection Sort
- Insert Sort
- Interchange Sort
- Bubble Sort
- Merge Sort
- Quick Sort
- Heap Sort

3.2 Tìm kiếm

- Tìm kiếm tuần tự
- Tìm kiếm nhị phân

3.3 Tổng kết chương

3.4 Bài tập chương 3

Tài liệu tham khảo

3.1
XẾP THỨ TỰ
(SORT)

PHÁT BIỂU BÀI TOÁN

Cho một tập các số nguyên gồm n phần tử:

$$a_0, a_2, a_3, \ldots, a_{n-1}$$

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

$$a_0, a_2, a_3, \ldots, a_{n-1}$$

Với $a_0 \le a_2 \le a_3 \le \ldots \le a_{n-1}$

MÔ HÌNH BÀI TOÁN

- Đầu vào: một danh sách đặc (các số nguyên) gồm có n phần tử a_0, a_2, a_3, ..., a_{n-1}.
- Đầu ra: một danh sách đặc (các số nguyên) gồm có n phần tử: a_0, a_2, a_3, ..., a_{n-1} ($a_0 \le a_2 \le a_3 \le \ldots \le a_{n-1}$)

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<MAX
```

CÁC NỘI DUNG CHÍNH CỦA BÀI TOÁN

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

Ta tìm hiểu 7 phương pháp xếp thứ tự cơ bản: Selection Sort, Insertion Sort, Bubble Sort, Interchange Sort, Quick Sort, Heap Sort, Merge 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], ..., 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

3.1 – 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.

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

THUẬT TOÁN

i=0;

Bước 1:

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

Bước 2: i+1;

- Nếu i < n – 1 thì lặp lại bước 1

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 0 → 7

n - 1

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

a[6] = 10 Là phần tử nhỏ nhất

Đổi giá trị giữa a[6] và a[0]

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

i=0 swap(a[0], a[6]) j=6

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 1 → 7

	0	1	2	3	4	5	6	7 (n - 1)
a	10	60	15	50	90	20	40	70
	i=0							

a[2] = 15 Là phần tử nhỏ nhất

Đổi giá trị giữa a[1] và a[2]

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

swap(a[1], a[2])

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 2 → 7

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

a[5] = 20 Là phần tử nhỏ nhất

Đổi giá trị giữa a[5] và a[2]

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

i=2 swap(a[2], a[5]) j=0

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 3 → 7

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

a[6] = 40 Là phần tử nhỏ nhất

Đổi giá trị giữa a[6] và a[3]

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

i=3 swap(a[3], a[6]) j=6

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 4 → 7

n - 1

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

a[6] = 50 Là phần tử nhỏ nhất

Đổi giá trị giữa a[6] và a[4]

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

i=4 swap(a[4], a[6]) j=6

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

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 5 → 7

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

n - 1

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

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

i= 5
j=5

n - 1

CHỌN LỰA TRỰC TIẾP – SELECTION SORT

Tìm giá trị nhỏ nhất từ 5 → 7

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

a[7] = 70 Là phần tử nhỏ nhất

Đổi giá trị giữa a[7] và a[6]

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

i=6 j=7

swap(a[6], a[7])

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

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; // min_pos là vị trí chứa giá tri hiện nhỏ nhất
        swap(a[min_pos], a[i]);
    }
}
```

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

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

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

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)**

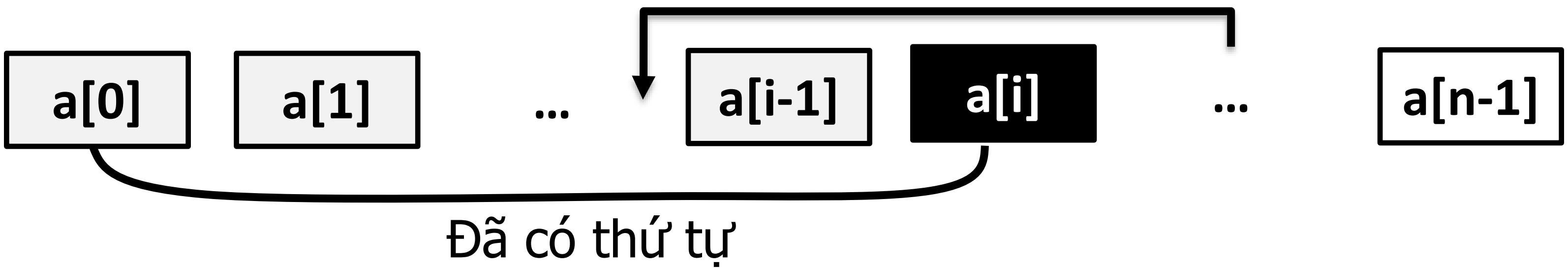
CHÈN TRỰC TIẾP – INSERTION SORT

PHƯƠNG PHÁP

- Trong danh sách a[0], a[1], ..., a[n-1], giả sử các phần tử a[0], a[1], ..., 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], ..., a[i-1], để có dãy mới a[0], a[1], ..., a[i] trở nên có thứ tự.

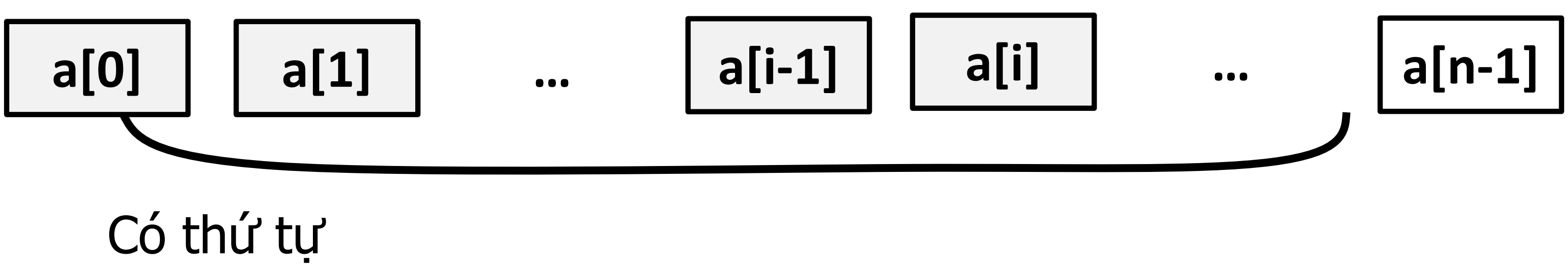
3.1 – XẾP THỨ TỰ (SORT)

CHÈN TRỰC TIẾP – INSERTION SORT



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

Để đoạn a[0], a[1], …, a[i-1], a[i] có thứ tự



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

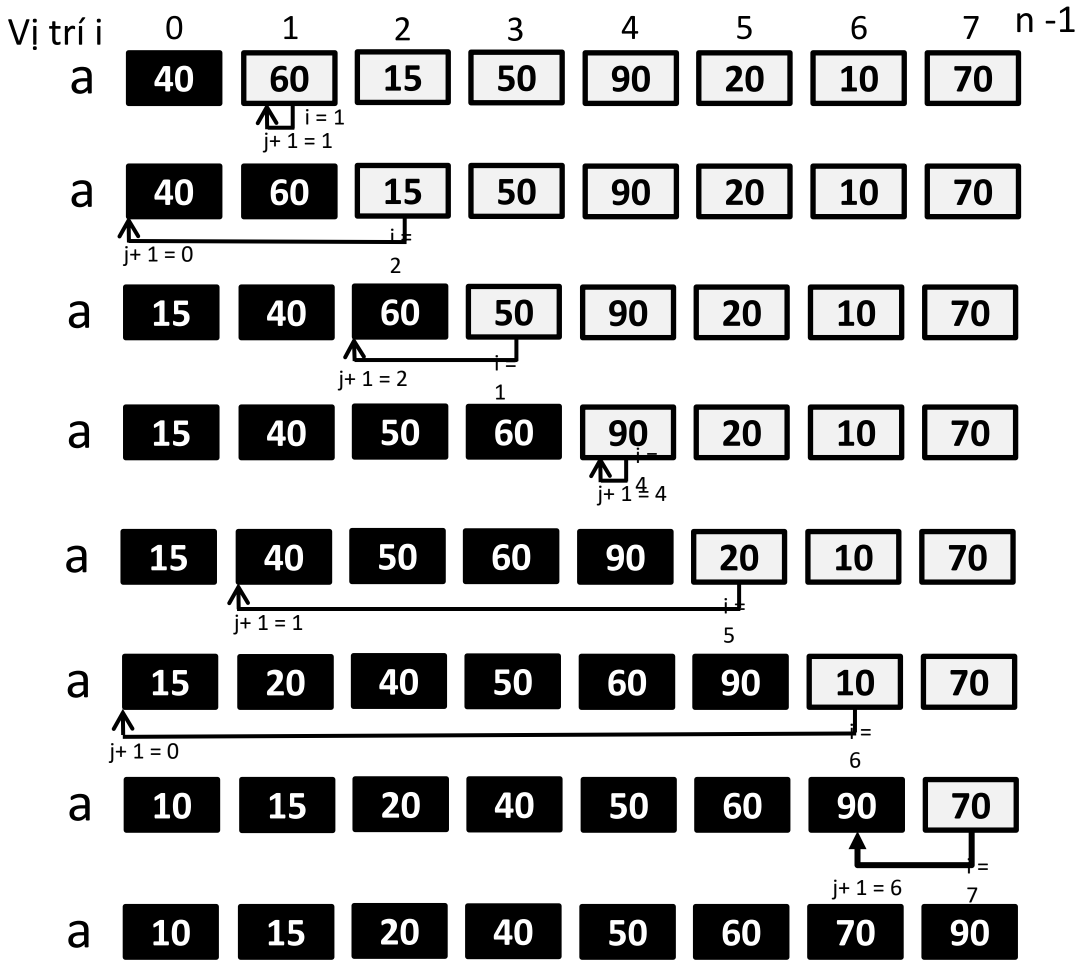
CHÈN TRỰC TIẾP – INSERTION SORT

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

CHÈN TRỰC TIẾP – INSERTION SORT



3.1 – 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;
    }
}
```

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)**

3.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

NỔI BỌT – 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], …, 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

3.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – 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.

Lập lại xử lý trên cho đến khi không còn cặp phần tử nào để xét

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]

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

Bước 3: i++;

Nếu i<n-1: lặp lại bước 2.

NỔI BỌT – 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

3.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

	0	1	2	3	4	5	j =6	7
a	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]

	0	1	2	3	4	j =5	6	7
a	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]

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

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

	0	1	2	j =3	4	5	6	7
a	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]

	0	1	j =2	3	4	5	6	7
a	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]

3.1 – 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	j =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	j =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	j =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 đỗi chỗ giá trị giữa a[4] và a[3]

	0	1	2	j =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]

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

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

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

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

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

	0	1	2	3	4	j =5	6	7
a	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]

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

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

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

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

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

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

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

	0	1	2	3	4	j =5	6	7
a	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]

3.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

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

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

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

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

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

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

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

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

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

3.1 – XẾP THỨ TỰ (SORT)

NỔI BỌT – BUBBLE SORT

	0	1	2	3	4	5	6	j = 7
a	**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**

NỔI BỌT – 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]

}
```

NỔI BỌT – 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)**

3.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP– INTERCHANGE 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], ..., 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

3.1 – 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 *hóan 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)

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<=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.

ĐỔI CHỖ TRỰC TIẾP– INTERCHANGE 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

ĐỔI CHỖ TRỰC TIẾP– INTERCHANGE SORT

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

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

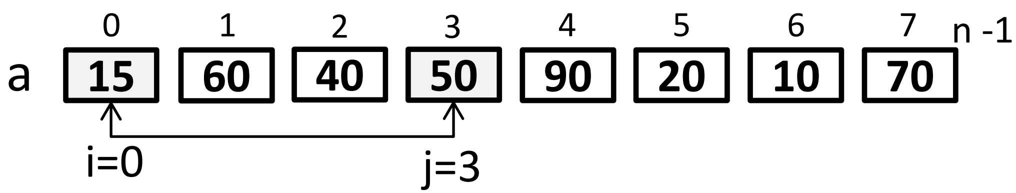
	0	1	2	3	4	5	6	7 n -1
a	**40**	**60**	**15**	**50**	**90**	**20**	**10**	**70**
	i=0		j=2					

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

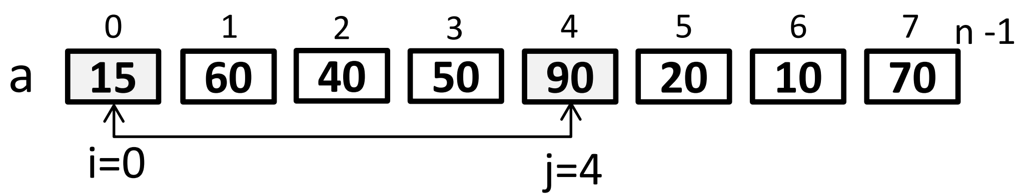
Kết quả sau khi hoán vị 40 và 15, được danh sách sau:

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

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

3.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP– INTERCHANGE SORT

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

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

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

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

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

3.1 – XẾP THỨ TỰ (SORT)

ĐỔI CHỖ TRỰC TIẾP– INTERCHANGE SORT

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

i=0 j=7

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

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

i=1 j=2

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

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

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:

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

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]
}
```

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)**

3.1 – XẾP THỨ TỰ (SORT)

PHÂN HOẠCH – QUICK SORT

Ý TƯỞNG

- 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], ..., a[n-1]

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

Ý TƯỞNG (tt)

Phân hoạch các phần tử trong danh sách trên thành hai (02) đoạn

Đoạn 1: Chứa các phần tử nhỏ hơn hoặc bằng x (a[k] <= x, k=0..j)

Đoạn 2: Chứa các phần tử lớn hơn hoặc bằng x (a[k] >= x, k=i..n-1)

a[k] <= x	a[k] >= x

PHÂN HOẠCH – QUICK SORT

Với x là giá trị phần tử bất kỳ trong danh sách (thông thường x là giá trị của phần tử nằm ở vị trí giữa danh sách, hoặc x là giá trị phần tử đầu của danh sách đang xét)

Nhận xét:

Nếu đoạn 1 có một (01) phần tử: Đoạn 1 có thứ tự

Nếu đoạn 2 có một (01) phần tử: Đoạn 2 có thứ tự

Nếu đoạn 1 có thứ tự và đoạn 2 có thứ tự, thì danh sách trên có thứ tự.

Ý tưởng của phương pháp xếp thứ tự QuickSort là: Sau khi phân hoạch danh sách ban đầu thành 12 đoạn (đoạn 1, đoạn 2), tiếp tục phân hoạch đoạn 1, đoạn 2, và lặp lại việc phân hoạch trên các đoạn con này cho đến khi mỗi đoạn con còn một (01) phần tử

THUẬT TOÁN

Vị trí đầu danh sách left=0, vị trí cuối danh sách right = n-1

Trong khi left < right

Phân hoạch danh sách các phần tử: a[left], …, a[right] thành 2 đoạn:

+ Đoạn 1: a[left], …, a[j] (các phần tử <=x)

+ Đoạn 2: a[i], …, a[right] (các phần tử >x)

Thực hiện tương tự các bước nêu trên cho mỗi đoạn con 1 và đoạn con 2, cho đến khi mỗi đoạn con còn một (01) phần tử

PHÂN HOẠCH – QUICK SORT

THÍ DỤ:

- Với một danh sách đặc a, có n phần tử từ a[0] đến a[7] như sau: a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7].

Vị trí:	0	1	2	3	4	5	6	7
Phần tử:	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Giá trị:	**40**	**60**	**15**	**50**	**90**	**20**	**10**	**70**

Phân hoạch danh sách trên, với left = 0, right = 7

Chọn x = a[(left+right)/2] = a[3] = 50

3.1 – XẾP THỨ TỰ (SORT)
PHÂN HOẠCH – QUICK SORT

x = a[3] = 50

left=0 right=7

a

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

i = left -> <- j=right

a[i] = a[0] = 40 < x, i++ (i tăng lên 1 giá trị)
a[i] = a[1] = 60 > x, dừng i (i lúc này = 1)

a[j] = a[7] = 70 > x, j --, (giảm j xuống 1 giá trị)
a[j] = a[6] = 10 < x, dừng j (lúc này j =6)

Left=0 right=7

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

swap(a[1], a[6])

i = 1 j=6

Hoán vị giá trị giữa a[i] = 60 và a[j] = 10, tang i lên giá trị, giảm j xuống 1 giá trị

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

PHÂN HOẠCH – QUICK SORT

x = a[3] = 50

left=0 right=7

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

a[i] = a[2] = 10 < x, i++ (i tăng lên 1 giá trị)
a[i] = a[3] = 50 > x, dừng i (i lúc này = 3)

a[j] = a[5] = 20 < x, dừng j (lúc này j = 5)

Left=0 right=7

40	**10**	**15**	**50**	**90**	**20**	**60**	**70**
0	1	2	3	4	5	6	7
			i = 3	swap(a[3], a[5])	j=5		

Hoán vị giá trị giữa a[i] = 50 và a[j] = 20, tăng i lên một giá trị, giảm j xuống một giá trị

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

3.1 – XẾP THỨ TỰ (SORT)
PHÂN HOẠCH – QUICK SORT

x = a[3] = 50

left=0 right=7

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

i = 4 j=4

a[i] = a[4] = 90 > x, dừng i (lúc này i = 4)
a[j] = a[5] = 90 > x, j-- (giảm j một giá trị, lúc này j = 3)

Left=0 right=7

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

j=4 i = 2

j < i dừng

Dãy ban đầu đã được phân hoạch thành 2 đoạn con. Mỗi đoạn là một dãy con như sau:

Dãy con 1: gồm các phần tử a[0], a[1], a[2], a[3]	40	10	15	20	**Dãy con 1**
Dãy con 2: gồm các phần tử a[4], a[5], a[6], a[7]	90	50	60	70	**Dãy con 2**

Trên dãy con 1, ta thực hiện lặp lai việc phân hoạch như trên với vi trí đầu dãy là left = 0, vị trí cuối dãy right = 3

Trên dãy con 2, ta thực hiện lặp lai việc phân hoạch như trên với vi trí đầu dãy là left = 4, vị trí cuối dãy right = 7

Tiếp tục thực hiện phân hoạch các dãy con cho đến khi mỗi dãy còn tối đa một phần tử, khi đó danh sách ban đầu được xếp thứ tự.

THUẬT GIẢI QUICK SORT

```
void QuickSort(int a[], int left, int right)
{
        int x = a[(left+right)/2];
        int i=left;
        int j=right;
        while(i<j)
        {
                while(a[i]<x) i++;
                while(a[j]>x) j--;
                if(i<=j){
                        swap(a[i],a[j])
                        i++; j--;
                }
        }
        if(left<j) QuickSort(a,left,j);
        if(i<right) QuickSort(a,i,right);
}
```

Gọi thực thi thủ tục QuickSort: **QuickSort(a, 0, n-1);**

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

TRƯỜNG HỢP	ĐỘ PHỨC TẠP
Tốt nhất	**O(n log n)**
Xấu nhất	O(n^2)

Độ phức tạp của thuật toán: **O(n log n)**

3.1 – XẾP THỨ TỰ (SORT)
HEAP SORT

ĐỊNH NGHĨA HEAP

- Một danh sách các phần tử là một Heap (Heap Max) khi và chỉ khi:

  Với mọi phần tử a[i] bất kì trong danh sách (i=0…n-1),

  Luôn có: a[i]>=a[2*i+1], và a[i]>=a[2*i+2]

- Một danh sách các phần tử là một Heap (Heap Min) khi và chỉ khi:

  Với mọi phần tử a[i] bất kì trong danh sách (i=0…n-1)

  Luôn có: a[i] <=a[2*i+1], và a[i]<=a[2*i+2].

HỆ QUẢ

🕮 Trong một Heap (Heap Max) phần tử đầu Heap là phần tử lớn nhất.

THUẬT TOÁN

- Bước 1: Tạo **Heap**
- Bước 2: Hoán vị phần tử đầu Heap (a[0]) với phần tử cuối *Heap* đang xét
- Bước 3: trong dãy đang xét, giới hạn (không quan tâm) phần tử cuối dãy (vừa thay thế giá trị phần tử đầu *Heap*). Kết quả là dãy đang xét giảm đi một phần tử bên phải. Tạo Heap ban đầu lại cho dãy các phần tử đang xét (trong trường hợp này thực chất chỉ xét lại vị trí a[0], các vị trí còn lại đã thỏa tính chất Heap trước đó)
- Bước 4: Lặp lại bước 2 trong khi đãy đang xét còn nhiều hơn 1 phần tử.

Ý TƯỞNG TẠO HEAP Ở BƯỚC 1

Chia dãy ban đầu a[0], a[1], ..., a[n-1], thành hai phần:

Nữa dãy bên trái: a[0], a[1], ..., a[(n/2)-1].

Nữa dãy bên phải: a[n/2], ..., a[n-1]: các phần tử nữa dãy bên phải này thỏa tính chất các phần tử trong Heap, vì với mọi vị trí i trong nữa dãy này không tồn tại vị trí 2*i+1. Do đó, chúng ta không cần xem xét các phần tử trong nữa dãy bên phải này khi tạo Heap ban đầu

Ý TƯỞNG TẠO HEAP Ở BƯỚC 1 (TT)

Để tạo Heap ban đầu, đầu tiên ta thực hiện tại vị trí **i = (n/2) - 1**

So sánh **a[i]** với **a[2*i+1]** và **a[2*i+2]**, nếu không thỏa tính chất Heap (max) hoán vị **a[i]** với phần tử max(**a[2*i+1], a[2*i+2]**)

Sau đó, giảm i một giá trị, và lặp lại việc so sánh a[i] với a[2*i+1] và a[2*i+2], thực hiện hoán vị như trên nhằm đảm bảo phần tư tại vị trí i thỏa tính chất Heap. Lặp lại bước này cho đến khi i là vị trí đầu dãy. Khi đó, danh sách dãy các phần tử là một Heap

THÍ DỤ:

- Với một danh sách đặc a[], có n phần tử từ a[0] đến a[6] như sau:

  a[0], a[1], a[2], a[3], a[4], a[5], a[6].

Vị trí:	0	1	2	3	4	5	6
Phần tử:	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Giá trị:	**40**	**60**	**15**	**50**	**90**	**20**	**10**

BƯỚC 1: TẠO HEAP BAN ĐẦU

Chi dãy trên thành 2 đoạn, bao gồm:

Nữa dãy bên trái chứa các phần tử sau a[0],…., a[(n/2)-1]: 40, 60, 15

Nữa dãy bên phải chứa các phần tử sau a[n/2],…., a[n-1]: 50, 90, 20, 10

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

3.1 – XẾP THỨ TỰ (SORT)
HEAP SORT

Tại vị trí cuối cùng của nữa dãy con bên trái i=2, so sánh a[i] =a[2] = 15 với hai phần tử tại vị trí 2*i+1 = 5 và vị trí 2*i+2 = 6

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

swap(a[2], a[5])

Giá trị lớn nhất của a[2], a[5], a[6] là a[5] = 20. Thực hiện hoán vị a[2] và a[5]

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

Tiếp tục giảm i xuống 1 giá trị (i=1), và so sánh a[1], a[3], a[4]

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

swap(a[1], a[4])

Giá trị lớn nhất của a[1], a[3], a[4] là a[4] = 90. Thực hiện hoán vị a[1] và a[4]

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

3.1 – XẾP THỨ TỰ (SORT)
HEAP SORT

Tiếp tục giảm i xuống 1 giá trị (i=0). So sánh a[0], a[1], a[2]

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

swap(a[0], a[1])

Giá trị lớn nhất của a[0], a[1], a[2] là a[1] = 90. Thực hiện hoán vị a[0] và a[1]

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

Xét lại tính lan truyền tại vị trí a[1], sau khi a[1] nhận giá trị mới là 40 (thay thế giá trị 90 trước đó; giá trị a[1] = 90 trước đó thỏa tính của Heap so với a[3] và a[4])

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

swap(a[1], a[4])

Giá trị lớn nhất của a[1], a[3], a[4] là a[4] = 60. Thực hiện hoán vị a[1] và a[4]
Việc tạo Heap (Heap Max) ban đầu hoàn tất. Ta được một Heap sau:

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

3.1 – XẾP THỨ TỰ (SORT)
HEAP SORT

BƯỚC 2: Hoán vị phần tử a[0] và phần tử cuối Heap đang xét. Ta có kết quả sau:

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

BƯỚC 3: Trong dãy đang xét, giới hạn phần tử cuối dãy. Ta được dãy sau:

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

Tạo Heap ban đầu lại cho dãy các phần tử đang xét từ a[0], a[1], …,a[5]

Trong trường hợp này thực chất chỉ xét lại vị trí a[0] (và sự lan truyền nếu có), các vị trí còn ại từ a[1],…, a[5] đã thỏa tính chất Heap trước đó.

BƯỚC 4: Sau khi dãy từ a[0], a[1], …., a[5] là một Heap, hoán vị a[0] và a[5]. Tiếp túc xét lại dãy từ a[0] đến a[4]….*Lặp lại* bước này cho đến khi danh sách được xếp thứ tự tăng dần.

CÀI ĐẶT THUẬT GIẢI HEAPSORT

```
void shift(int a[], int i, int n)
{
        int j = 2*i+1;
        if (j>=n) // nếu vị trí j không tồn tại trong danh sách đang xét thì thoát khỏi chương trình
                return;
        if (j+1 <n) // nếu tồn tại vị trí j+1 trong danh sách đang xét thì thoát khỏi chương trình
                if ( a[j]<a[j+1] ) // nếu vị trí j không tồn tại  phần tử a[j] <a[j+1]
                        j++;

        if (a[i] >= a[j] )
                return;
        else {
                int x = a[i];
                a[i] = a[j];
                a[j] = x;
                shift(a, j, n);
        }
}
```

3.1 – XẾP THỨ TỰ (SORT)
HEAP SORT

Gọi thực thi thủ tục HeapSort: **HeapSort(a, n);**

```
void HeapSort(int a[], int n)
{
        int i = n/2;
        while (i >=0) // tạo heap ban đầu
        {
                shift(a, i, n-1);       i - -;
        }
        int right=n-1; // right là vị trí cuối Heap đang xét
        while (right>0)
        {
                swap(a[0], a[right]); // hoán vị phần tử a[0] cho phần tử cuối Heap đang xét
                right - -; // giới hạn lại phần tử cuối đang xét
                if (right > 0) // Kiểm tra dãy đang xét còn nhiều hơn 1 phần tử
                        shift(a, 0, right); // tạo Heap lại tại vị trí 0
        }

}
```

ĐỘ PHỨC TẠP CỦA THUẬT TOÁN

O(n log n)

3.2
TÌM KIẾM

3.2 – TÌM KIẾM

- TÌM KIẾM TUẦN TỰ
- TÌM KIẾM NHỊ PHÂN

3.2 – 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

TÌM KIẾM TUẦN TỰ

- Ý tưởng của thuật toán tìm kiếm tuần tự
  - *Đi từng phần tử từ a_0, a_1, ..., 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).*

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], ..., a[n-1]

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

n -
1

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

TÌM KIẾM TUẦN TỰ

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

TÌM KIẾM TUẦN TỰ

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

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

X = 50

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

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

X = 50

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

TÌM KIẾM TUẦN TỰ

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

X = 50

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

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

X = 50

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)

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

TÌM KIẾM TUẦN TỰ

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

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

X = 80

a[i] !=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 tang I lên 1 giá trị (i = 8, i = n), kết thúc.

Tim không thấy, return giá trị -1

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	$\frac{n+1}{2}$

TÌM KIẾM NHỊ PHÂN

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] <=a[1]<=a[2]<=a[3]<=…<=a[n-1]

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

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: left = 0, right = n -1;

Bước 2:

Nếu (x==a[(left+right)/2]): *Tìm thấy* thì kết thúc;

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

Ngược lại: lặp lại bước 2 cho dãy từ vị trí (left+right)/2+1 đến right

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

TÌM KIẾM NHỊ PHÂN

Bước 1: So sánh x với phần tử tại vị tri trí (left+right)/2 = (0 + 7)/2 = 3

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

X = 50

Ta có x = 50 > a[(left+right)/2]

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

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

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

TÌM KIẾM NHỊ PHÂN

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

X = 50

x < a[5]

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 = 50

x = a[4]. Tìm thấy. Kết thúc

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;
}
```

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)**

3.3 – TỔNG KẾT
CHƯƠNG 3

3.3 – Tổng kết chương 3

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

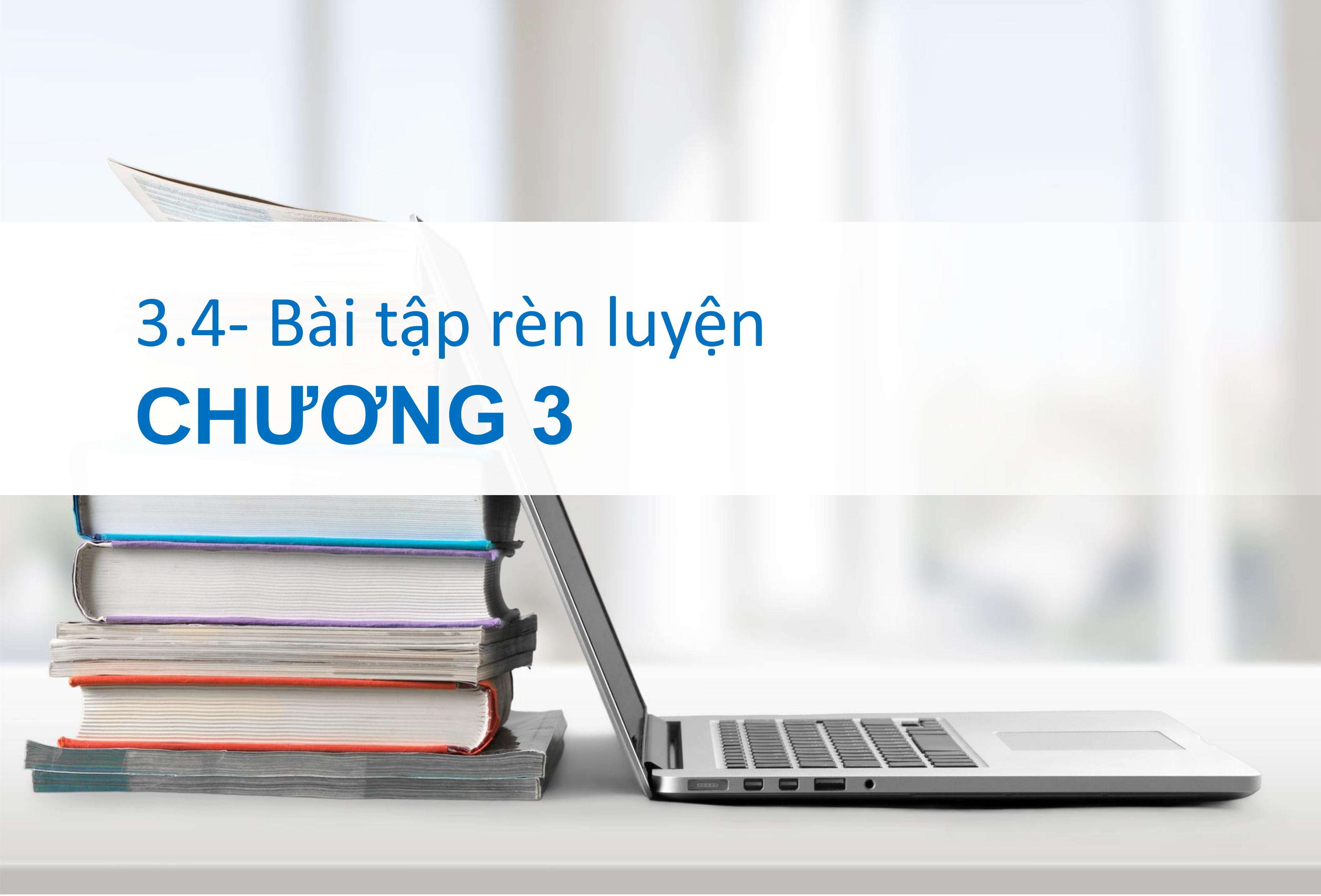
- Thuật toán tìm kiếm tuần tự và tìm kiếm nhị phân

3.3 – Tổng kết chương 3

ĐỘ PHỨC TẠP

THUẬT TOÁN SẮP XẾP	ĐỘ PHỨC TẠP
SelectionSort	$O(n^2)$
InsertionSort	$O(n^2)$
InterchangeSort	$O(n^2)$
BubbleSort	$O(n^2)$
QuickSort	O(n log n)
HeapSort	O(n log n)

THUẬT TOÁN TÌM KIẾM	ĐỘ PHỨC TẠP
Tìm kiếm tuần tự	O(n)
Tìm kiếm nhị phân	O(log n)



3.4- Bài tập rèn luyện

CHƯƠNG 3

3.4 - Bài tập chương 3

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.

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.

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 QuickSort. Đánh giá độ phức tạp của thuật toán.

1.9 Viết thủ tục sắp xếp danh sách theo thứ tự tăng dần bằng thuật toán HeapSort. 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 tuần tự). Đánh giá độ phức tạp của thuật toán

1.12 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

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). Tín 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). Tín 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). Tín độ phức tạp của quá trình xếp thứ tự danh sách trên.

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). Tín độ phức tạp của quá trình xếp thứ tự danh sách trên.

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

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

2.7 Sau khi xếp thú 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

Hướng dẫn

- Tất cả sinh viên phải trả lời các **câu hỏi** của chương, làm **bài tập thực hành** tại phòng máy (**bài làm thêm** ở nhà, và **bài nâng cao** khuyến khích hoàn tất) và nộp bài qua LMS của trường.
- Câu hỏi chương 3 làm trên file WORD; trong bài làm ghi rõ họ tên, lớp, bài tập chương và các thông tin cần thiết.
- Khuyến khích sử dụng tiếng Anh trong bài tập.

⇒ **Ngày nộp:** trước khi học chương 5

⇒ **Cách nộp:** sử dụng ***github*** để nộp bài, sau đó nộp lên LMS của trường.

Tài liệu tham khảo

- **Lê Xuân Trường**, (Chuong 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, Cliffrod Stein**, *(Chapter 2, 3) Introduction to Algorithms*, Third Edition, 2009.
- **Adam Drozdek**, *(Chapter 9) Data Structures and Algorithms in C*++, Fourth Edtion, CENGAGE Learning, 2013.

Phụ lục – Thuật ngữ tiếng Anh

#	Tiếng Anh	Phiên Âm	Tiếng Việt
1	Sort	/ sɔ:t /	Sắp xếp / Xếp
2	Selection Sort	/ sɪ'lekʃn sɔ:t/	PP Xếp thứ tự chọn trực tiếp
3	Insertion Sort	/ ɪn'sɜ:ʃn sɔ:t /	PP Xếp thứ tự chèn trực tiếp
4	Bubble Sort	/ 'bʌbl sɔ:t /	PP Xếp thứ tự nổi bọt
5	Interchange Sort	/ɪntər'tʃeɪndʒ sɔ:t /	PP Xếp thứ tự đổi chỗ
6	Quick Sort	/ kwɪk sɔ:t /	PP Xếp thứ tự phân hoạch
7	Heap Sort	/ hi:p sɔ:t /	PP Xếp thứ tự theo đống
8	Search	/ sɜ:tʃ /	Tìm kiếm
9	Sequential Search	/ sɪ'kwenʃl sɜ:tʃ /	Tìm kiếm tuần tự
10	Binary Search	/ 'baɪnəri sɜ:tʃ /	Tìm kiếm nhị phân

KẾT THÚC CHƯƠNG 3

Trường Đại học Mở TP.HCM

Khoa Công Nghệ Thông Tin