CANTHO UNIVERSITY

CHƯƠNG 2: SẮP XẾP

Tuần 4
– Sắp xếp chọn, xen, nổi bọt
– Sắp xếp nhanh (QUICK SORT)

Bộ môn CÔNG NGHỆ PHẦN MỀM
Khoa Công nghệ Thông tin & Truyền thông
ĐẠI HỌC CẦN THƠ

Mục tiêu

Sau khi hoàn tất bài học này, sinh viên cần phải:

- **Hiểu** các giải thuật sắp xếp.
- **Vận dụng** được giải thuật để minh họa việc sắp xếp.
- **Hiểu** các lưu đồ của các giải thuật sắp xếp.
- **Hiểu** các chương trình sắp xếp.
- **Hiểu** được việc đánh giá các giải thuật.

Tầm quan trọng của bài toán sắp xếp

- **Sắp xếp** một danh sách các đối tượng theo một thứ tự nào đó là một bài toán thường được vận dụng trong các ứng dụng tin học.
- **Sắp xếp** là một yêu cầu không thể thiếu trong khi thiết kế các phần mềm.
- Việc nghiên cứu các **phương pháp sắp xếp** là rất cần thiết để vận dụng trong khi lập trình.

Sắp xếp trong và sắp xếp ngoài

- **Sắp xếp trong** là sắp xếp **dữ liệu được tổ chức ở bộ nhớ trong** của máy tính.
- *Đối tượng sắp xếp*: các mẩu tin gồm một hoặc nhiều trường.

Có 1 **trường khóa** (key) với kiểu quan hệ thứ tự (*kiểu số nguyên, số thực, chuỗi ký tự...*). Danh sách đối tượng sắp xếp là mảng các mẩu tin trên.

- *Mục đích sắp xếp*: tổ chức lại các mẩu tin sao cho khóa của chúng được sắp thứ tự tương ứng với quy luật sắp xếp.
- *Quy luật sắp xếp*: mặc nhiên theo thứ tự không giảm.

- **Sắp xếp ngoài** là sắp xếp khi số lượng đối tượng cần sắp xếp lớn, không thể lưu trữ trong bộ nhớ trong mà phải lưu trữ ở **bộ nhớ ngoài**.

Tổ chức dữ liệu và ngôn ngữ cài đặt

- Ví dụ minh họa được thể hiện trên ngôn ngữ **C: Turbo C++ (*Version 3.0*)**
- Sử dụng khai báo:

```
typedef  int keytype;
typedef  float othertype;
typedef  struct recordtype {
    keytype key;
    othertype otherfields;
    };
```

Tổ chức dữ liệu và ngôn ngữ cài đặt (tt)

```
void Swap(recordtype &x, recordtype &y)
{
    recordtype temp;
    temp = x;
    x = y;
    y = temp;
}
```

- ***Lưu ý***: thủ tục **Swap tốn O(1)** thời gian vì thực hiện 3 lệnh gán nối tiếp.

Các thuật toán Sắp xếp

- Các thuật toán sắp xếp đơn giản (độ phức tạp $O(n^2)$):
  - Sắp xếp **chọn** (Selection Sort)
  - Sắp xếp **xen** (Insertion Sort)
  - Sắp xếp **nổi bọt** (Bubble Sort)
- Các thuật toán sắp xếp phức tạp (độ phức tạp O(n logn))
  - Sắp xếp **phân đoạn/nhanh** (Quick Sort)
  - Sắp xếp **vun đống** (Heap Sort)
  - *Trường hợp dữ liệu đặc biệt (Bin Sort) (độ phức tạp O(n))*

Thuật toán sắp xếp chọn (Selection Sort)

Bài toán: Cho mảng A = {a[0], a[1], ..., a[n-1]}, hãy sắp xếp mảng theo chiều không giảm.

Ý tưởng: Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là vị trí đầu tiên của mảng hiện hành. Sau đó không quan tâm đến nó nữa, xem mảng hiện hành chỉ còn n-1 phần tử của mảng ban đầu. Bắt đầu từ vị trí thứ 2, lặp lại quá trình trên cho mảng hiện hành đến khi chỉ còn 1 phần tử.

Do mảng ban đầu có n phần tử, vậy tóm tắt ý tưởng thuật toán là thực hiện n-1 lượt việc đưa phần tử nhỏ nhất trong mảng hiện hành về vị trí đúng ở đầu mảng.

Thuật toán sắp xếp chọn (Selection Sort)

Giải thuật :

- **Bước 1**: chọn phần tử có khóa nhỏ nhất trong **n** phần tử từ a[0] đến a[n-1] và hoán vị nó với phần tử **a[0]**.
- **Bước 2**: chọn phần tử có khóa nhỏ nhất trong **n-1** phần tử từ a[1] đến a[n-1] và hoán vị nó với **a[1]**.
- Tổng quát ở **bước i**: chọn phần tử có khoá nhỏ nhất trong **n-i** phần tử từ a[i] đến a[n-1] và hoán vị nó với **a[i]**.
- **Bước n-1**: mảng đã được sắp xếp.

Phương pháp chọn phần tử

- Đầu tiên, đặt khoá nhỏ nhất là khoá của a[i] : lowkey = a[i].key
  và chỉ số của phần tử có khoá nhỏ nhất là i : lowindex = i
- Xét các phần tử a[j] (*với j từ i+1 đến n-1*), nếu khoá của a[j] < khoá nhỏ nhất (**a[j].key < lowkey**) thì
  Đặt lại khoá nhỏ nhất là khoá của a[j] : lowkey = a[j].key
  và chỉ số phần tử có khoá nhỏ nhất là j : lowindex = j
- Khi đã xét hết các a[j] (*j>n-1*) thì phần tử có khoá nhỏ nhất là a[lowindex].

Ví dụ sắp xếp chọn

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	6	**5**	2	10	12	9	10	9	3
Bước 2		**2**	5	**6**	10	12	9	10	9	3
Bước 3			**3**	6	10	12	9	10	9	**5**
Bước 4				**5**	10	12	9	10	9	**6**
Bước 5					**6**	12	9	10	9	**10**
Bước 6						**9**	**12**	10	9	10
Bước 7							**9**	10	12	**10**
Bước 8								**10**	12	10
Bước 9									**10**	**12**
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

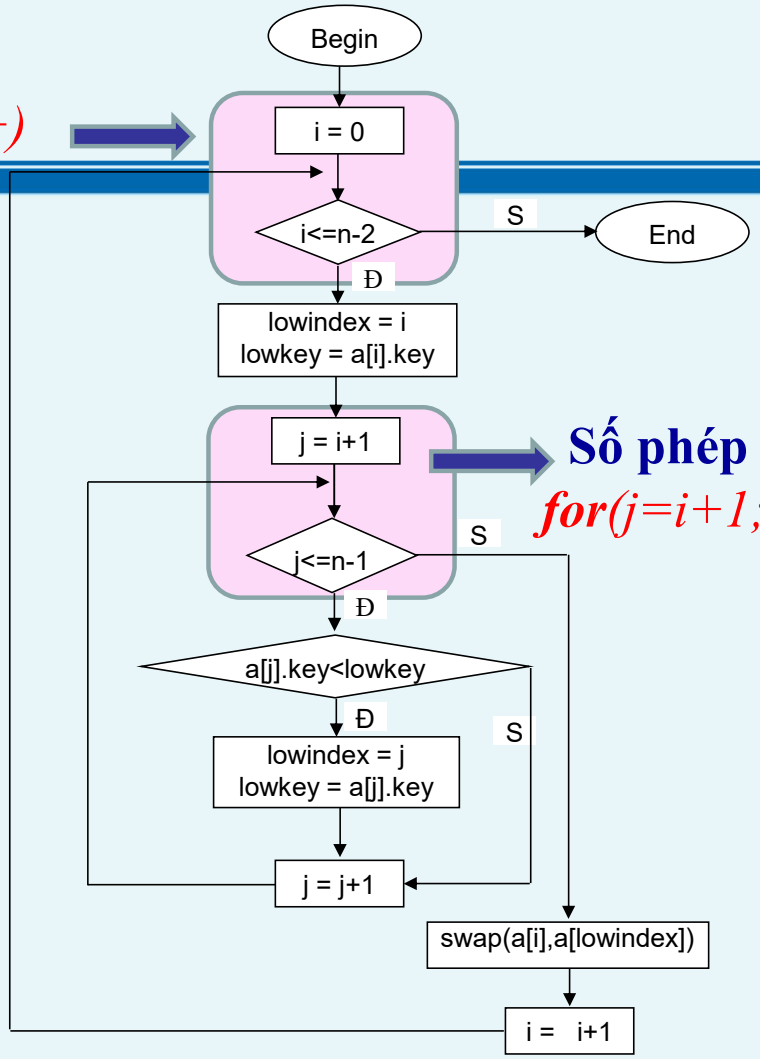
Góc trên bên phải

CANTHO UNIVERSITY

Lưu đồ thuật toán sắp xếp chọn

Số bước chọn

for(i=0; i<=n-2; i++)



Số phép so sánh

for(j=i+1; j<=n-1; j++)

Chương trình sắp xếp chọn

```
void SelectionSort (recordtype a[ ], int n){
  int i,j, lowindex;
  keytype lowkey;
{1}.  for(i=0; i<=n-2; i++){
{2}.    lowkey = a[i].key;
{3}.    lowindex = i;
{4}.    for(j=i+1; j<=n-1; j++)
{5}.      if(a[j].key < lowkey) {
{6}.        lowkey = a[j].key;
{7}.        lowindex = j;
          }
{8}.    Swap(a[i],a[lowindex]);
      }
}
```

→ $T(n) = O(n^2)$

Đánh giá sắp xếp chọn

- Hàm Swap tốn O(1).
- Toàn bộ chương trình chỉ bao gồm vòng for {1}. Vòng for {1} chứa các lệnh nối tiếp {2}, {3}, {4} và {8}, trong đó các lệnh {2}, {3} và {8} đều tốn thời gian O(1).
- Lệnh {6} và {7} đều tốn O(1) nên lệnh {5} tốn O(1).
- Vòng lặp {4} thực hiện n-i-1 lần, vì j chạy từ i+1 đến n-1, mỗi lần lấy O(1), nên lấy O(n-i-1) thời gian.
- Gọi T(n) là thời gian thực hiện của chương trình, thì T(n) là thời gian thực hiện vòng for {1}. Mà lệnh {1} có i chạy từ 0 đến n-2 nên ta có:

$$T(n) = \sum_{i=0}^{n-2} (n - i - 1) = \frac{n(n-1)}{2} = O(n^2)$$

Thuật toán sắp xếp xen (Insertion Sort)

Bài toán: Cho mảng A = {a[0], a[1], ..., a[n-1]}, hãy sắp xếp mảng theo chiều không giảm.

Ý tưởng: Bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự, người chơi bài rút lần lượt từ quân bài thứ 2, so với các quân đứng trước nó để chèn vào vị trí thích hợp.

Xét mảng con gồm k phần tử đầu. Với i = 1, mảng gồm một phần tử đã được sắp. Giả sử trong mảng i-1 phần tử đầu đã được sắp, để sắp xếp một phần tử ta tìm vị trí thích hợp của nó trong mảng. Vị trí thích hợp đó là đứng trước phần tử lớn hơn nó và sau phần tử nhỏ hơn hoặc bằng nó.

Thuật toán sắp xếp xen (Insertion Sort)

Giải thuật : Trước hết ta xem phần tử a[0] là một mảng đã có thứ tự.

- **Bước 1**: xen phần tử **a[1]** vào danh sách đã có thứ tự a[0] sao cho **a[0], a[1]** là một danh sách có thứ tự.
- **Bước 2**: xen phần tử **a[2]** vào danh sách đã có thứ tự a[0], a[1] sao cho **a[0], a[1], a[2]** là một danh sách có thứ tự.
- Tổng quát **bước i** : xen phần tử **a[i]** vào danh sách đã có thứ tự a[0], a[1], … a[i-1] sao cho **a[0], a[1],.. a[i]** là một danh sách có thứ tự.
- **Bước n-1**: mảng đã được sắp thứ tự.

Phương pháp xen phần tử

- Phần tử đang xét a[j] sẽ được xen vào vị trí thích hợp trong danh sách các phần tử đã được sắp trước đó a[0], a[1], .. , a[j-1]
- So sánh khoá của a[j] với khoá của a[j-1] đứng ngay trước nó.
- Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] (a[i] < a[j-1]) thì hoán đổi a[j-1] và a[j] cho nhau và tiếp tục so sánh khoá của a[j-1] (lúc này a[j-1] chứa nội dung của a[j]) với khoá của a[j-2] đứng ngay trước nó.

Ví dụ sắp xếp xen

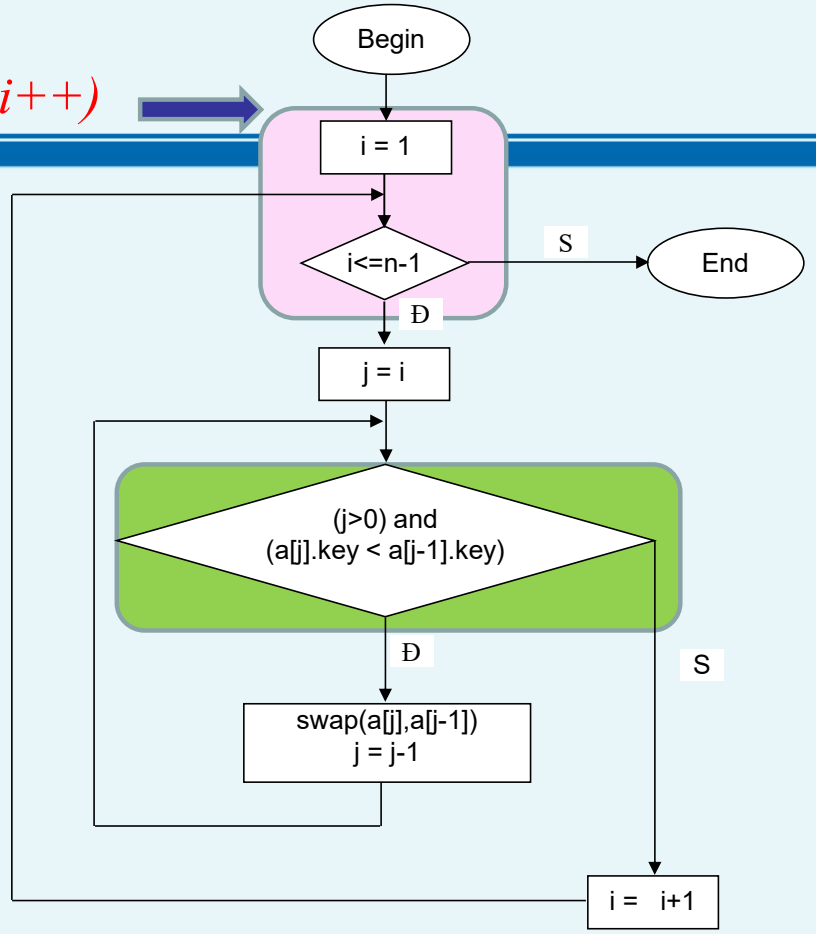
Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	**3**
Bước 1	5	**6**								
Bước 2	**2**	5	6							
Bước 3	2	**2**	5	6						
Bước 4	2	2	5	6	**10**					
Bước 5	2	2	5	6	10	**12**				
Bước 6	2	2	5	6	**9**	10	12			
Bước 7	2	2	5	6	9	10	**10**	12		
Bước 8	2	2	5	6	9	**9**	10	10	12	
Bước 9	2	2	**3**	5	6	9	9	10	10	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Góc dưới bên trái

Số bước xen

for(i=1; i<=n-1; i++)

Lưu đồ thuật toán sắp xếp xen



Chương trình sắp xếp xen

```
void InsertionSort(recordtype a[], int n){
   int i,j;
{1}  for(i=1; i<=n-1; i++){
{2}        j=i;
{3}        while ((j>0)&&(a[j].key<a[j-1].key)) {
{4}              Swap(a[j],a[j-1]);
{5}              j--;
           }
     }
}
```

→ **T(n) = O(n^2)**

Đánh giá sắp xếp xen

- Các lệnh {4} và {5} đều tốn O(1).
- Vòng lặp {3}, trong trường hợp xấu nhất, chạy i lần (j giảm từ i đến 1), mỗi lần tốn O(1) nên {3} tốn O(i) thời gian.
- Lệnh {2} và {3} là hai lệnh nối tiếp, lệnh {2} tốn O(1) nên cả hai lệnh này tốn thời gian max là O(i).
- Vòng lặp {1} có i chạy từ 1 đến n-1 nên ta có:

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

Thuật toán sắp xếp "nổi bọt" (Bubble Sort)

Bài toán: Cho mảng A = {a[0], a[1], ..., a[n-1]}, hãy sắp xếp mảng theo chiều không giảm.

Ý tưởng: Xuất phát từ cuối (hoặc đầu) mảng, đổi chổ các cặp phần tử kế cận để đưa phần tử nhỏ (lớn) hơn trong cặp phần tử đó về vị trí đúng đầu (cuối) mảng hiện hành. Sau đó sẽ không xét đến nó ở vị trí tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu mảng là i. Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét.

Thuật toán sắp xếp "nổi bọt" (Bubble Sort)

Giải thuật:

- **Bước 1**: Xét các a[j] (j giảm từ n-1 đến 1), so sánh khoá của a[j] với khoá của a[j-1]. Nếu **a[j] < a[j-1]** thì đổi a[j] và a[j-1]. Sau bước này thì a[0] có khoá nhỏ nhất.
- **Bước 2**: Xét các a[j] (j giảm từ n-1 đến 2), so sánh khoá của a[j] với khoá của a[j-1]. Nếu **a[j] < a[j-1]** thì đổi a[j] và a[j-1]. Sau bước này thì a[1] có khoá nhỏ thứ 2.
- …
- **Bước n-1**: mảng đã được sắp thứ tự

Ví dụ sắp xếp "nổi bọt"

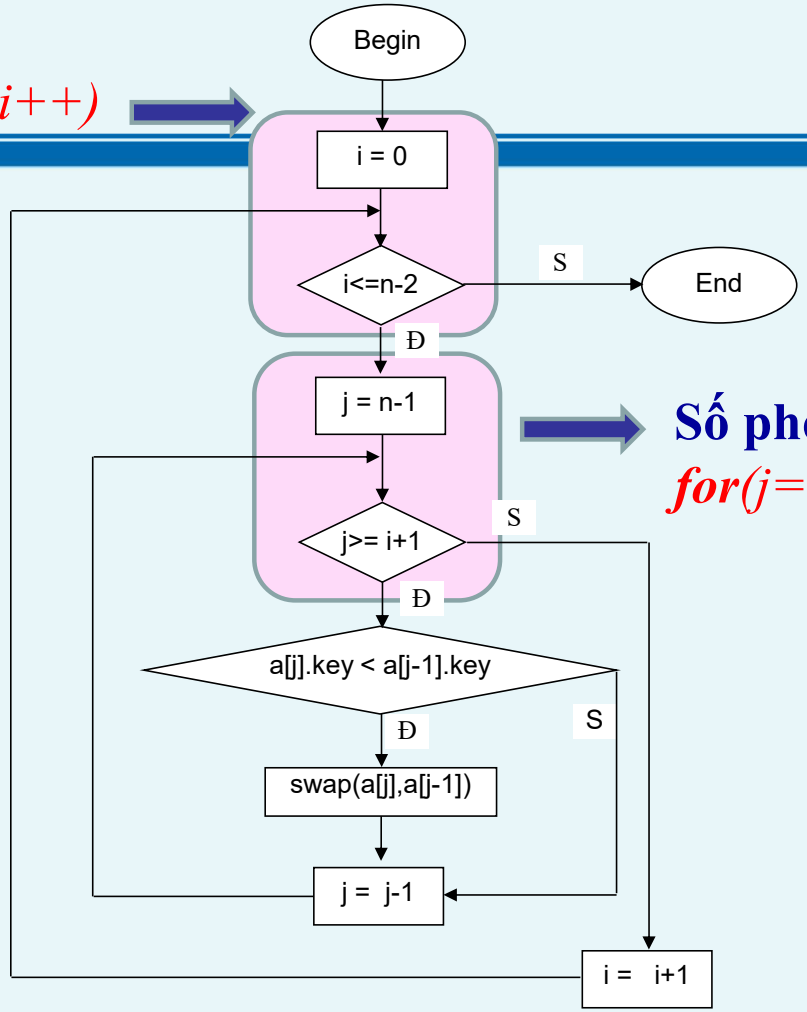
Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
Ban đầu	5	6	2	2	10	12	9	10	9	3
Bước 1	**2**	5	6	2	3	10	12	9	10	9
Bước 2		**2**	5	6	3	9	10	12	9	10
Bước 3			**3**	5	6	9	9	10	12	10
Bước 4				**5**	6	9	9	10	10	12
Bước 5					**6**	9	9	10	10	12
Bước 6						**9**	9	10	10	12
Bước 7							**9**	10	10	12
Bước 8								**10**	10	**12**
Bước 9									**10**	12
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Góc trên bên phải

Lưu đồ thuật toán sắp xếp nổi bọt

Số bước nổi bọt

for(i=0; i<=n-2; i++)



Số phép so sánh

for(j=n-1; j>=i+1; j--)

Chương trình sắp xếp "nổi bọt"

```
void BubbleSort(recordtype a[ ], int n) {
        int i,j;
{1}     for(i= 0; i<= n-2; i++)
{2}         for(j=n-1;j>=i+1; j--)
{3}             if (a[j].key < a[j-1].key)
{4}                 Swap(a[j],a[j-1]);
        }
```

→ $T(n) = O(n^2)$

Đánh giá sắp xếp "nổi bọt"

- Vòng lặp {3} tốn O(1)
- Vòng lặp {2} thực hiện (n – i -1) bước (giảm từ n – 1 đến i+1), mỗi lần tốn O(1) nên vòng lặp {2} tốn O(n – i - 1) thời gian.
- Vòng lặp {1} có i chạy từ 0 đến n-2 nên ta có:

$$T(n)=\sum_{i=0}^{n-2}(n-i-1)=\frac{n(n-1)}{2}=O(n^2)$$

Đánh giá chung về các thuật toán sắp xếp đơn giản $T(n) = O(n^2)$

(1) Sắp xếp chọn (Selection Sort): for – for

- Ở lượt thứ i, bao giờ cũng cần (n-i) phép so sánh để xác định phần tử nhỏ nhất
  → T(n) **không phụ thuộc** vào tình trạng của mảng ban đầu.

(2) Sắp xếp xen (Insetion Sort): for – *while*

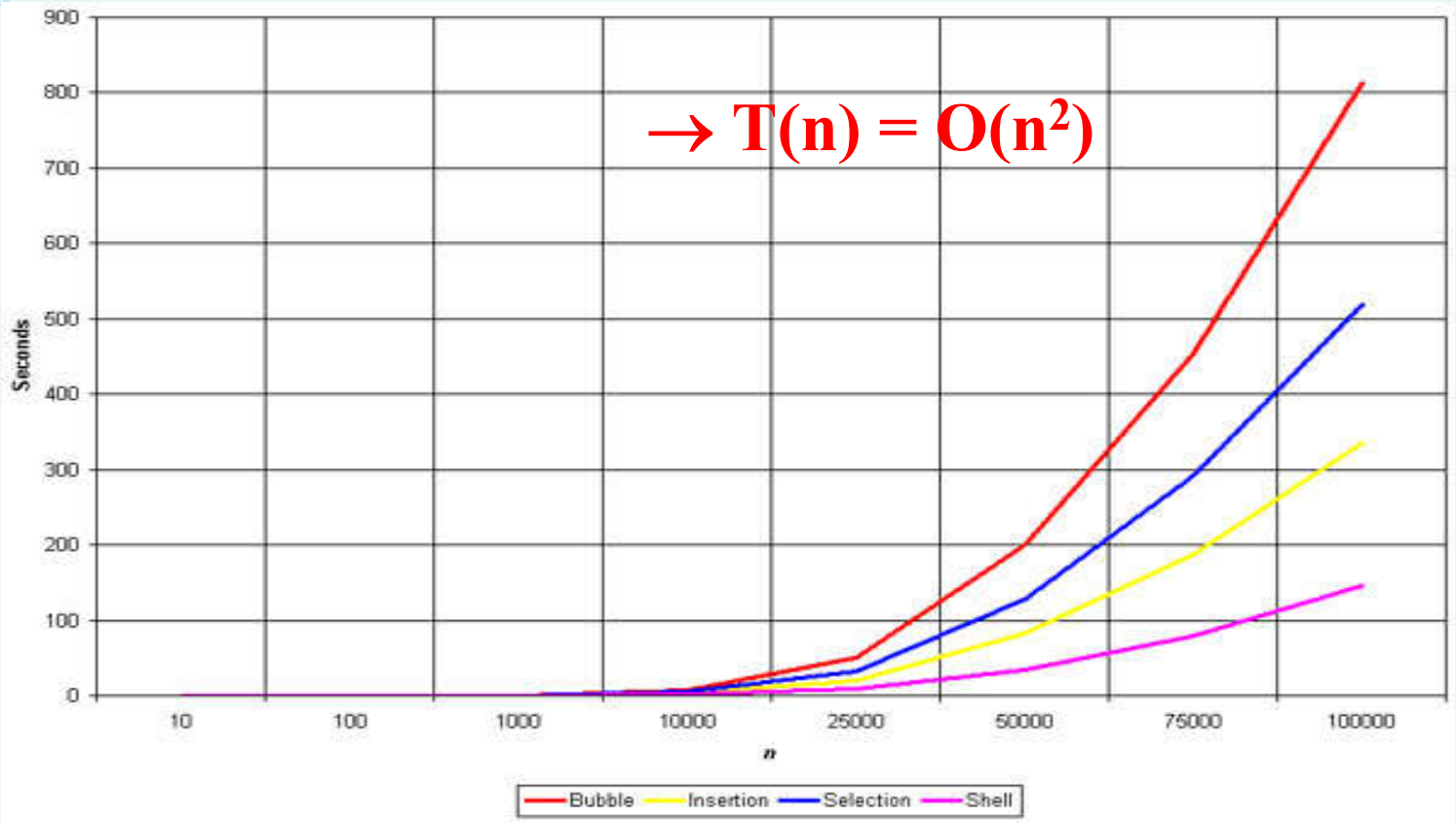
- Số lượng phép so sánh và đổi chỗ xảy ra khác nhau trong mỗi vòng lặp while
  → T(n) ***phụ thuộc*** vào tình trạng của mảng ban đầu.

(3) Sắp xếp nổi bọt (Bubble Sort): for – for

- Số lượng phép so sánh ở mỗi bước xác định theo số lần lặp trong vòng for
  → T(n) **không phụ thuộc** vào tình trạng của mảng ban đầu.

- *Các phần tử nhỏ được đưa về vị trí đúng rất chậm trong khi các phần tử lớn lại được đưa về vị trí đúng rất nhanh.*

Biểu đồ so sánh hiệu quả



Bài tập 1

Trong 3 thuật toán sắp xếp đơn giản (chọn, xen, nổi bọt), thuật toán nào thực hiện sắp xếp nhanh nhất với một **mảng đã có thứ tự** ? Giải thích tại sao ?

CANTHO UNIVERSITY

Bài tập 2

Thực hiện 3 thuật toán sắp xếp đơn giản (**chọn, xen, nổi bọt**) để minh họa việc sắp xếp mảng 12 phần tử có khóa là các số nguyên sau:

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
Ban đầu	**50**	**10**	**143**	**0**	**10**	**120**	**90**	**10**	**90**	**140**	**20**	**30**
Bước 1												
Bước 2												
Bước 3												
…..												
Kết quả												

Các thuật toán Sắp xếp

- Các thuật toán sắp xếp đơn giản (độ phức tạp $O(n^2)$):
  - Sắp xếp **chọn** (Selection Sort)
  - Sắp xếp **xen** (Insertion Sort)
  - Sắp xếp **nổi bọt** (Bubble Sort)

- Các thuật toán sắp xếp phức tạp (độ phức tạp O(n logn))
  - Sắp xếp **phân đoạn/nhanh** (Quick Sort)
  - Sắp xếp **vun đống** (Heap Sort)
  - *Trường hợp dữ liệu đặc biệt (Bin Sort) (độ phức tạp O(n))*

Thuật toán QuickSort

- Phát triển bởi C.A.R. Hoare (1960) dựa trên nguyên tắc chia danh sách cần sắp xếp thành 2 danh sách con.
- Khác với sắp xếp trộn (MergeSort: chia thành 2 danh sách con có kích thước tương đối bằng nhau nhờ chỉ số đứng giữa), **QuickSort** chia danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn gọi là **phần tử chốt.** Những phần tử ***nhỏ hơn*** phần tử chốt được đưa về phía trước và nằm trong danh sách con bên trái, các phần tử ***lớn hơn hoặc bằng*** chốt được đưa về phía sau và thuộc danh sách con bên phải.
- Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1. Sắp xếp mảng con "bên trái" và mảng con "bên phải".

Ý tưởng của QuickSort

- Ý tưởng của thuật toán này là "**chia để trị**"
- **Bước 1**: Chọn một phần tử khóa v làm **phần tử chốt (pivot)**
- **Bước 2**: Phân hoạch dãy a[0] .. a[n-1] thành hai mảng con "bên trái" và "bên phải". Mảng con "bên trái" bao gồm các phần tử có **khóa nhỏ hơn chốt.** Mảng con "bên phải" bao gồm các phần tử có **khóa lớn hơn hoặc bằng chốt**.
- **Bước 3**: Sau khi phân hoạch thành 2 mảng, thực hiện lại bước 2 ở từng mảng - Chọn pivot ở từng mảng và tiếp tục phân chia thành các mảng nhỏ hơn, sắp xếp đến khi mảng được sắp xếp hoàn toàn (*mảng chỉ gồm 1 phần tử hoặc gồm nhiều phần tử có khóa bằng nhau thì đã có thứ tự*)

Phương pháp chọn chốt

Chọn giá trị **khóa lớn nhất** trong 2 phần tử có khóa **khác nhau đầu tiên** kể từ trái qua.

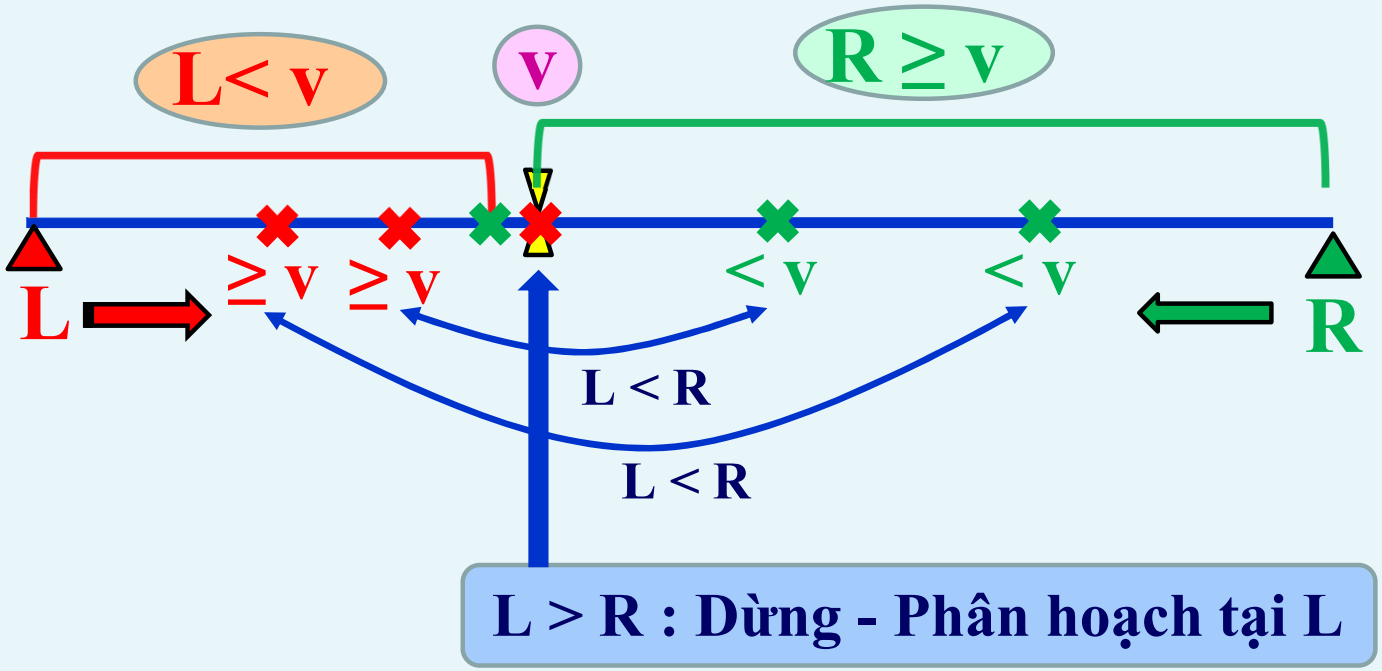
- Nếu mảng chỉ gồm **1 phần tử** hay gồm **nhiều phần tử có khóa bằng nhau** thì *không có chốt*.
- **Ví dụ:** Chọn chốt trong các mảng sau
  - Cho mảng gồm các phần tử có khoá là **6, 6, 5, 8, 7, 4**, ta chọn chốt là **6** (khoá của phần tử đầu tiên).
  - Cho mảng gồm các phần tử có khoá là **6, 6, 7, 5, 7, 4,** ta chọn chốt là **7** (khoá của phần tử thứ 3).
  - Cho mảng gồm các phần tử có khoá là **6, 6, 6, 6, 6, 6** thì *không có chốt* (vì các phần tử có khoá bằng nhau).
  - Cho mảng gồm 1 phần tử có khoá là **6** thì *không có chốt* (vì chỉ có 1 phần tử).

Phương pháp phân hoạch

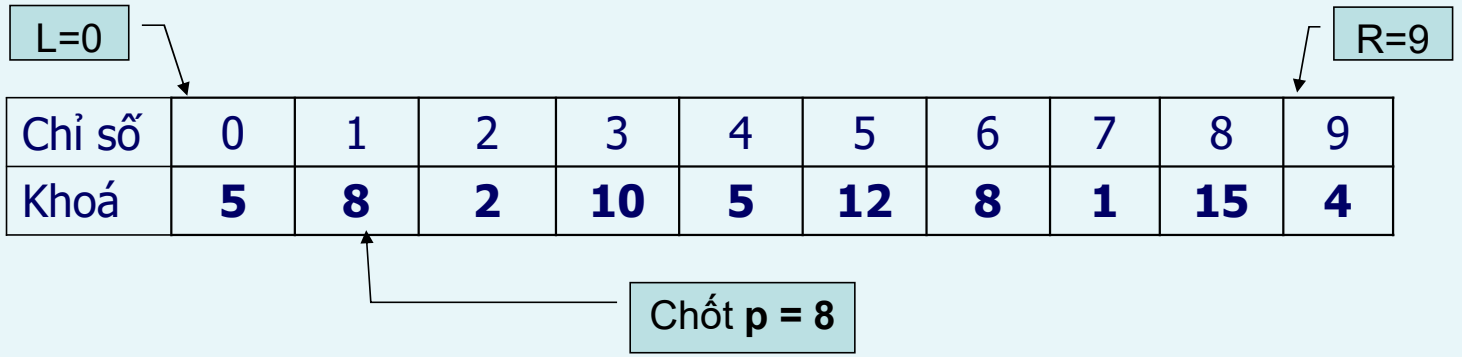
- Để phân hoạch mảng: dùng 2 "con nháy" L và R, trong đó L đi từ bên **trái** và R đi từ bên **phải.**
- Cho L chạy **sang phải** tới khi gặp phần tử có **khóa ≥ chốt**
- Cho R chạy **sang trái** tới khi gặp phần tử có **khóa < chốt**
- Tại chỗ dừng của L và R: nếu L < R thì hoán vị a[L], a[R].
- Lặp lại quá trình dịch sang phải, sang trái của 2 "con nháy" L và R cho đến khi L > R.
- Khi đó L sẽ là điểm phân hoạch, cụ thể là a[L] là phần tử đầu tiên của mảng con "bên phải".

Phương pháp phân hoạch

Chốt v = phần tử > trong 2 phần tử ≠ đầu tiên



L< v Ví dụ về phân hoạch R ≥ v



L< v Ví dụ về phân hoạch R ≥ v

L=1

R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**8**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**4**

Chốt **p = 8**

L< v Ví dụ về phân hoạch R ≥ v

L=1

R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=2

R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**8**

Chốt **p = 8**

L< v Ví dụ về phân hoạch R ≥ v

L=3

R=9

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=3

R=8

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=3

R=7

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=3

R=7

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

L< v Ví dụ về phân hoạch R ≥ v

L=4 R=7

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=5

R=7

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

L=5

R=6

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

L< v Ví dụ về phân hoạch R ≥ v

L=5 R=5

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt p = 8

Ví dụ về phân hoạch

L< v

R ≥ v

R=4

L=5

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

Ví dụ về phân hoạch

L< v

R ≥ v

R=4

L=5

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**4**	**2**	**1**	**5**	**12**	**8**	**10**	**15**	**8**

Chốt **p = 8**

Điểm phân hoạch L =5

0	1	2	3	4
5	**4**	**2**	**1**	**5**

Mảng con “bên trái”

5	6	7	8	9
12	**8**	**10**	**15**	**8**

Mảng con “bên phải”

Giải thuật QuickSort

- Để sắp xếp mảng a[i]..a[j] ta làm các bước sau:
  - **Xác định chốt** trong mảng a[i]..a[j],
  - **Phân hoạch** mảng a[i]..a[j] đã cho thành hai mảng con a[i]..a[k-1] và a[k]..a[j].
  - Sắp xếp mảng a[i]..a[k-1] (Đệ quy).
  - Sắp xếp mảng a[k]..a[j] (Đệ quy).
- Đệ quy sẽ dừng khi không còn tìm thấy chốt.

CANTHO UNIVERSITY

Ví dụ về QuickSort

L< v

R ≥ v

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**8**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**4**

Chốt p = 8

5 1	4	2	1 5	5	12 8	8	10	15	8 12

Chốt p = 5

Chốt p = 12

1	4 2	2 4	5	5	8	8	10	15 12	12 15

Chốt p = 4

xong

Chốt p = 10

Chốt p = 15

1	2	4			8	8	10	12	15

Chốt p = 2

xong

xong

xong

xong

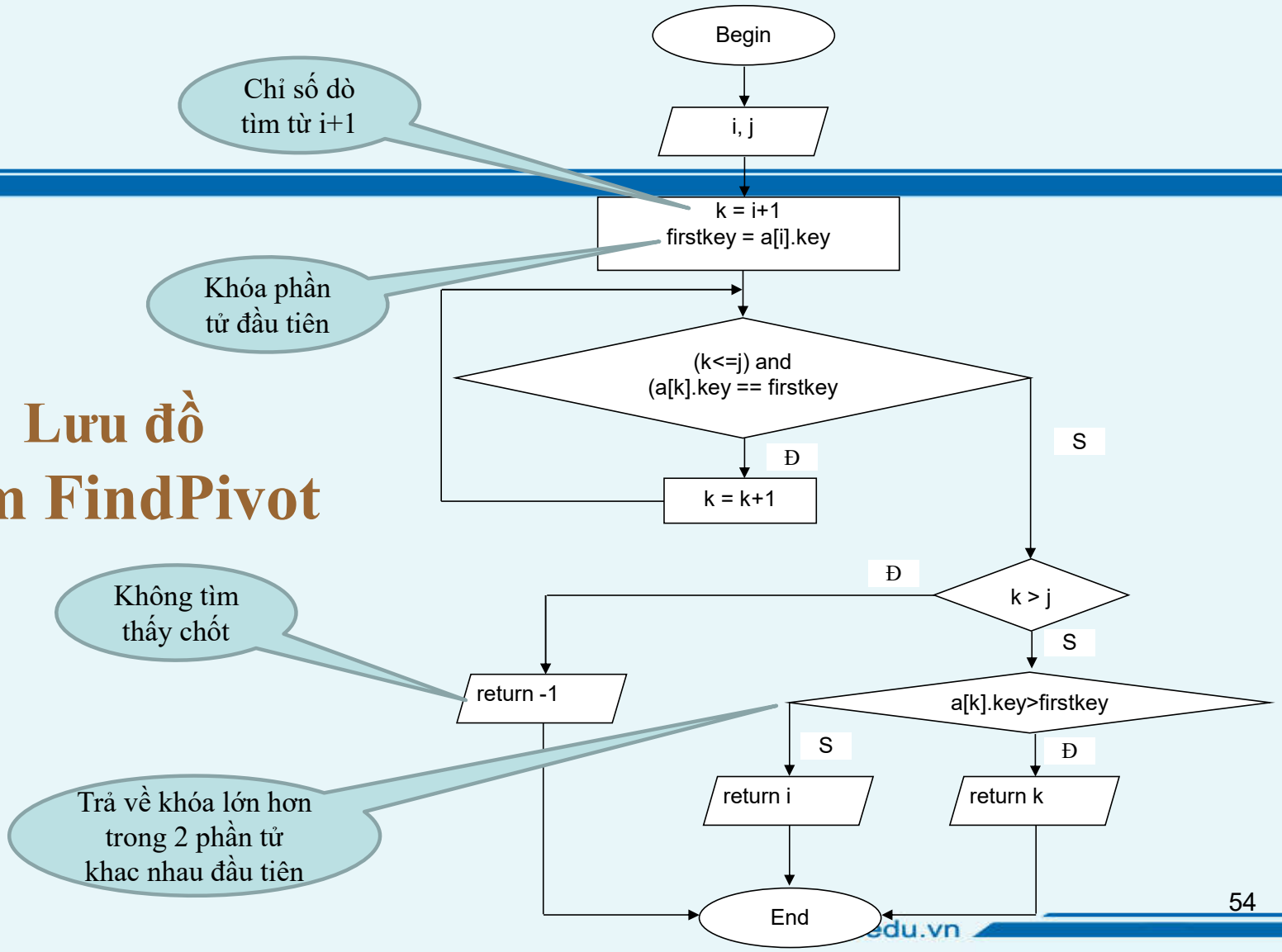
xong

1	2

xong xong

CANTHO UNIVERSITY

Lưu đồ hàm FindPivot



Chương trình hàm FindPivot

```
int FindPivot(recordtype a[ ], int i,int j) {
  keytype  firstkey;
  int k ;
{1}.    k = i+1;
{2}.    firstkey = a[i].key;
{3}.    while ( (k <= j) && (a[k].key == firstkey) ) k++;
{4}.    if (k > j)  return -1;
        else
{5}.      if (a[k].key>firstkey) return k; else return i;
}
```

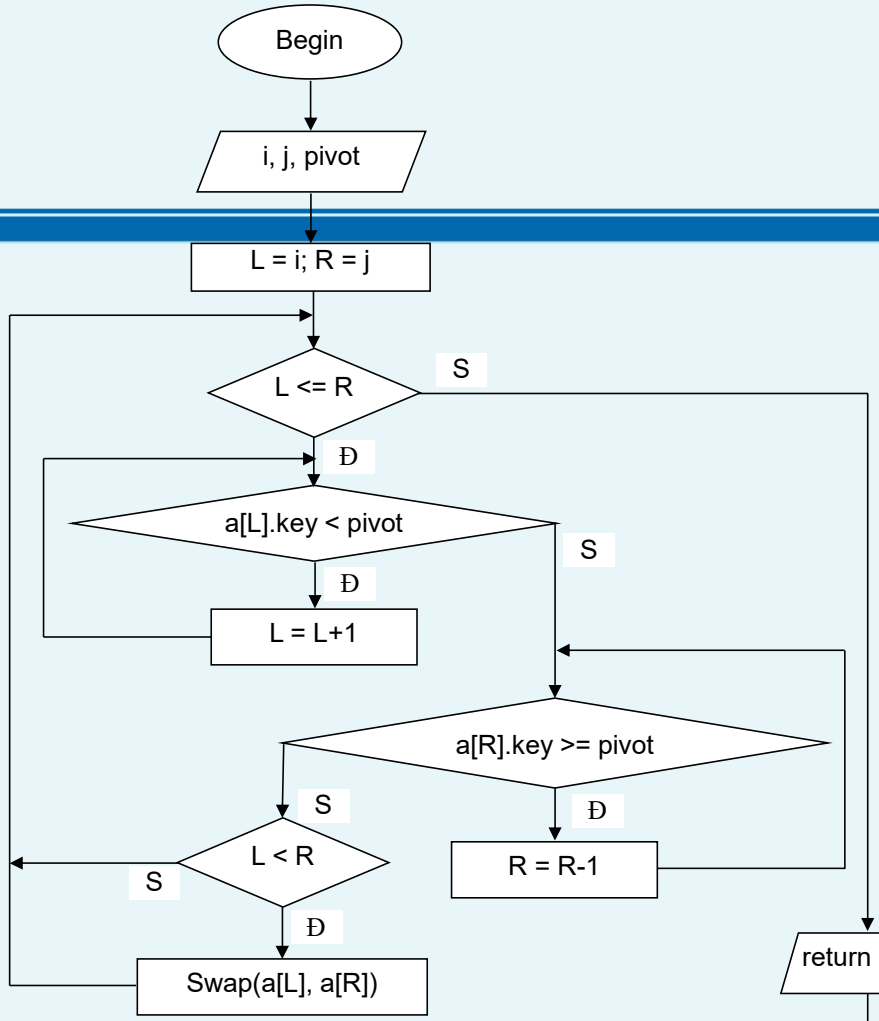
→ **T(n) = O(n)**

Độ phức tạp của hàm FindPivot

```
int FindPivot(recordtype a[], int i,int j)
{       keytype  firstkey;
        int k ;
{1}     k = i+1;
{2}     firstkey = a[i].key;
{3}     while ( (k <= j) && (a[k].key ==
            firstkey) ) k++;
{4}     if (k > j)  return -1;
        else
{5}       if (a[k].key>firstkey) return k;
          else return i;
}
```

- {1}, {2}, {3} và {4} nối tiếp nhau.
- Lệnh ***While*** tốn nhiều thời gian nhất.
- Trong trường hợp xấu nhất (*không tìm thấy chốt*): k chạy từ i+1 đến j, tức vòng lặp thực hiện j-i lần, mỗi lần O(1), do đó tốn O(j-i).
- Đặc biệt khi i=0 và j=n-1, thì thời gian thực hiện là n-1 hay T(n) = O(n).

Lưu đồ hàm Partition



Chỉ số đầu tiên của mảng "bên phải"

Hàm Partition

```
int Partition(recordtype a[], int i,int j, keytype pivot)
    {   int L,R;
{1}   L = i;
{2}   R = j;
{3}   while (L <= R) {
{4}     while (a[L].key < pivot)  L++;
{5}     while (a[R].key >= pivot) R--;
{6}     if (L<R) Swap(a[L],a[R]);
      }
{7}   return L;  /*Tra ve diem phan hoach*/
    }
```

→ **T(n) = O(n)**

Phân tích hàm Partition

```
int Partition(recordtype a[], int i,  int j,
    keytype pivot)
     {     int L,R;
{1}      L = i;
{2}      R = j;
{3}      while (L <= R)
{4}     {  while (a[L].key < pivot)  L++;
{5}       while (a[R].key >= pivot)  R--;
{6}      if (L<R) Swap(a[L],a[R]);
         }
{7}      return L;
     }
```

- {1}, {2}, {3} và {7} nối tiếp nhau
- Thời gian thực hiện của {3} lớn nhất.
- Các lệnh {4}, {5} và {6} là thân của lệnh {3}, trong đó lệnh {6} lấy O(1).
- Lệnh {4} và lệnh {5} thực hiện việc di chuyển L sang phải và R sang trái cho đến khi L và R gặp nhau, thực chất là duyệt các phần tử mảng, mỗi phần tử một lần, mỗi lần tốn O(1) thời gian. Tổng việc duyệt tốn O(j-i) thời gian.
- Vòng lặp {3} thực chất để xét xem khi nào thì duyệt xong, do đó thời gian thực hiện của {3} chính là thời gian thực hiện của lệnh {4} và {5}, là O(j-i).
- Khi i=0 và j=n-1, ta có: T(n) = O(n).

Hàm QuickSort

Chỉ số	0	1	2	3	4	5	6	7	8	9
Khoá	**5**	**8**	**2**	**10**	**5**	**12**	**8**	**1**	**15**	**4**

```
void QuickSort(recordtype a[], int i,int j)
{ keytype pivot;
  int pivotindex, k;
  pivotindex = FindPivot(a,i,j);
  if (pivotindex != -1) {
     pivot = a[pivotindex].key;
     k = Partition(a,i,j,pivot);
     QuickSort(a,i,k-1);
     QuickSort(a,k,j);
  }
}
```

pivot: Biến giữ giá trị chốt

k : Biến giữ giá trị điểm phân hoạch

Đánh giá QuickSort

* **Trường hợp xấu nhất** : *Phân hoạch* ***lệch***

Phương trình đệ quy :

$$T(n) = \begin{cases} 1 & \text{nêu } n = 1 \\ T(n-1) + T(1) + n & \text{nêu } n > 1 \end{cases} \rightarrow \mathbf{T(n) = O(n^2)}$$

* **Trường hợp trung bình → tốt nhất** : *Phân hoạch* ***đều***

Phương trình đệ quy :

$$T(n) = \begin{cases} 1 & \text{nêu } n = 1 \\ 2T(\frac{n}{2}) + n & \text{nêu } n > 1 \end{cases}$$

→ **T(n) = O(nlogn)**

CANTHO UNIVERSITY

Bài tập 3

Minh họa việc sắp xếp mảng gồm 12 phần tử có khóa là các số nguyên sau bằng cách sử dụng **thuật toán QuickSort**:

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
Ban đầu	**50**	**10**	**143**	**0**	**10**	**120**	**90**	**10**	**90**	**140**	**20**	**30**

QuickSort

L< v

v : >, 2, ≠

R ≥ v

Chỉ số	0	1	2	3	4	5	6	7	8	9	10	11
Khoá ban đầu	50 30	10	143 20	0	10	120 10	90	10 120	90	140	20 143	30 50
						v = 50						
Cấp 1	30 10	10	20	0	10	10 30	90	120 50	90	140	143	50 120
			v = 30						**v = 120**			
Cấp 2	10	10	20 10	0	10 20	30	90 50	50 90	90	140	143 120	120 143
		v = 20				*xong*		**v = 90**			**v = 143**	
Cấp 3	10 0	10	10	0 10	20		50	90	90	140 120	120 140	143
		v = 10			*xong*		*xong*	*xong*		**v = 140**		*xong*
Cấp 4	0	10	10	10						120	140	
	xong		*xong*							*xong*	*xong*	
Kết quả	**0**	**10**	**10**	**10**	**20**	**30**	**50**	**90**	**90**	**120**	**140**	**143**

Bài tập 4

Có một biến thể của QuickSort như sau: *Chọn chốt là khóa của phần tử* ***nhỏ nhất*** *trong hai phần tử có khóa khác nhau đầu tiên.*

- Mảng con bên trái gồm các phần tử có khóa ***nhỏ hơn hoặc bằng*** *chốt (≤ v)*

- Mảng con bên phải gồm các phần tử có khóa ***lớn hơn*** *chốt (> v)*

Hãy viết lại các thủ tục cần thiết cho biến thể này và minh họa bằng việc sắp xếp mảng trên.

Khóa / Bước	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
Ban đầu	**50**	**10**	**143**	**0**	**10**	**120**	**90**	**10**	**90**	**140**	**20**	**30**

QuickSort và QuickSort biến thể

QuickSort	$L < v$	$v : >, 2, \neq$	$R \geq v$
QuickSort biến thể	$L \leq v$	$v : <, 2, \neq$	$R > v$

Hàm FindPivot – QuickSort biến thể

v : <, 2, ≠

```
int FindPivot(recordtype a[ ], int i,int j) {
  keytype  firstkey;
  int k ;
{1}.    k = i+1;
{2}.    firstkey = a[i].key;
{3}.    while ( (k <= j) && (a[k].key == firstkey) ) k++;
{4}.    if (k > j)  return -1;
        else
{5}.        if (a[k].key < firstkey) return  k ; else return i ;
  }
```

CANTHO UNIVERSITY

Hàm FindPivot – QuickSort biến thể

```
int FindPivot(recordtype a[ ], int i,int j) {
  keytype  firstkey;
  int k ;
{1}.    k = i+1;
{2}.    firstkey = a[i].key;
{3}.    while ( (k <= j) && (a[k].key == firstkey) ) k++;
{4}.    if (k > j)  return -1;
        else
{5}.        if (a[k].key  >  firstkey) return i else return k
  }
```

v : <, 2, ≠

Hàm Partition – QuickSort biến thể

L ≤ v

R > v

```
int Partition(recordtype a[], int i,int j, keytype pivot)
     {   int L,R;
{1}    L = i;
{2}    R = j;
{3}    while (L <=  R) {
{4}       while (a[L].key ≤ pivot)  L++;
{5}       while (a[R].key  >  pivot) R--;
{6}       if (L < R) Swap(a[L], a[R]);
          }
{7}    return L;  /*Tra ve diem phan hoach*/
     }
```

QuickSort biến thể

L ≤ v | **v : <, 2, ≠** | **R > v**

Chi số	0	1	2	3	4	5	6	7	8	9	10	11
Khoá ban đầu	50 10	10	143 10	0	10 143	120	90	10 50	90	140	20	30
						v = 10						
Cấp 1	10 0	10	10	0 10	143 30	120	90	50	90	140 20	20 140	30 143
		v = 0							**v = 120**			
Cấp 2	0	10	10	10	30	120 20	90	50	90	20 120	140	143
	xong	*xong*							**v = 30**		**v =140**	
Cấp 3					30 20	20 30	90 50	50 90	90	120	140	143
					v = 20			**v =50**			*xong*	*xong*
Cấp 4					20	30	50	90	90	120		
					xong	*xong*	*xong*	**v=90**				
Cấp 5								90	90	120		
								xong		*xong*		
Kết quả	**0**	**10**	**10**	**10**	**20**	**30**	**50**	**90**	**90**	**120**	**140**	**143**