CHƯƠNG 2: SẮP XẾP

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 bạn 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.
- Do đó 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 sự sắp xếp **dữ liệu được tổ chức trong bộ nhớ trong** của máy tính.
- Các đối tượng cần được sắp xếp là các mẩu tin gồm một hoặc nhiều trường. Một trong các trường được gọi là khóa (key), kiểu của nó là một kiểu có quan hệ thứ tự (như các kiểu số nguyên, số thực, chuỗi ký tự...).
- Danh sách các đối tượng cần sắp xếp sẽ là một mảng của các mẩu tin vừa nói ở trên.
- Mục đích của việc sắp xếp là tổ chức lại các mẩu tin sao cho các khóa của chúng được sắp thứ tự tương ứng với quy luật sắp xếp.
- Một cách mặc nhiên, quy luật sắp xếp là thứ tự không giảm. Khi cần sắp xếp theo thứ tự không tăng thì phải nói rõ.
- **Sắp xếp ngoài** là sự sắp xếp được sử dụng khi số lượng đối tượng cần sắp xếp lớn không thể lưu trữ trong bộ nhớ trong mà phải lưu trữ trên **bộ nhớ ngoài**.

Tổ chức dữ liệu và ngôn ngữ cài đặt

- Để trình bày các ví dụ minh họa chúng ta sẽ dùng C (Turbo C++, Version 3.0) làm ngôn ngữ thể hiện và sử dụng khai báo sau.

```
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;
}
```

- Cần thấy rằng thủ tục Swap lấy O(1) thời gian vì chỉ thực hiện 3 lệnh gán nối tiếp nhau.

Giải thuật sắp xếp chọn (Selection Sort)

- Bước 0, 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 1, 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 thứ 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].
- Sau n-1 bước này thì mảng đã được sắp xếp.

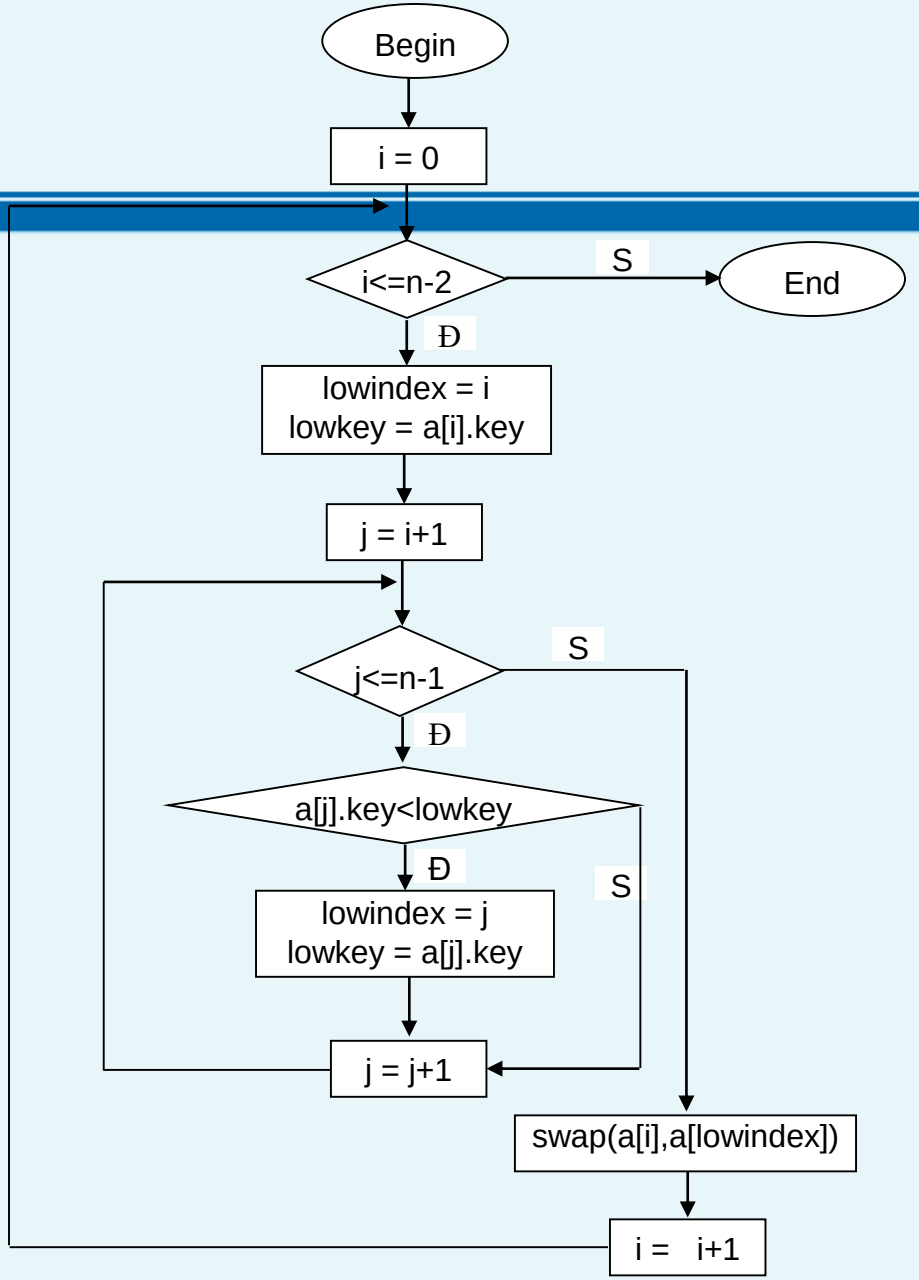
Phương pháp chọn phần tử

- Đầu tiên ta 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] nhỏ hơn khoá nhỏ nhất (a[j].key < lowkey) thì đặt lại lại khoá nhỏ nhất là khoá của a[j] (lowkey = a[j].key) và chỉ số của 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 0										
Bước 1										
Bước 2										
Bước 3										
Bước 4										
Bước 5										
Bước 6										
Bước 7										
Bước 8										
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Lưu đồ sắp xếp chọn



Chương trình sắp xếp chọn

	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 0										
Bước 1										

```
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]);
       }
  }
```

Đánh giá sắp xếp chọn

- Hàm Swap tốn O(1).
- Toàn bộ chương trình chỉ bao gồm lệnh /*1*/. Lệnh /*1*/ chứa các lệnh "đồng cấp" /*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 lệnh /*1*/. Mà lệnh /*1*/ có i chạy từ 0 đến n-2 nên ta có:

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

Giải thuật sắp xếp xen (Insertion Sort)

- Trước hết ta xem phần tử a[0] là một dãy 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.
- Sau n-1 bước thì kết thúc.

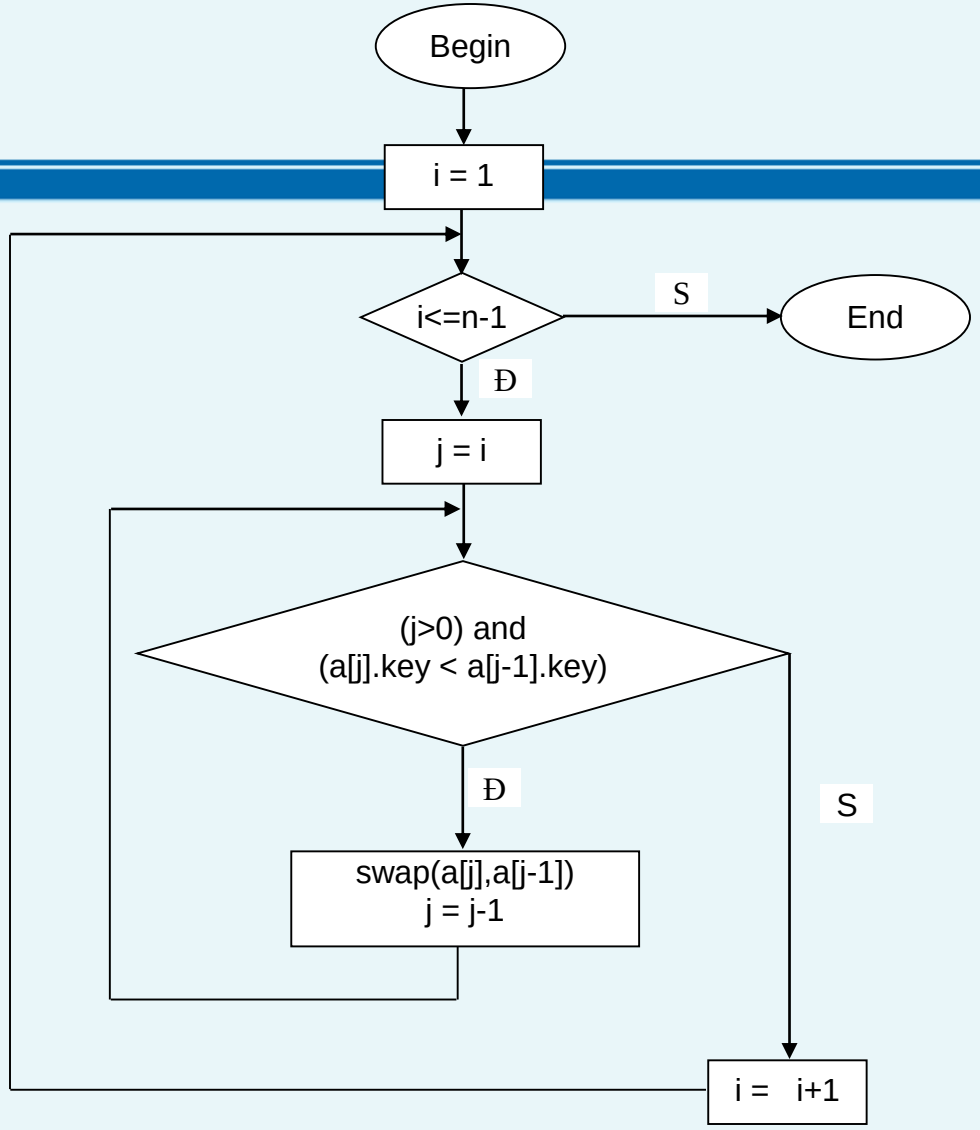
Phương pháp xen

- 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] 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										
Bước 2										
Bước 3										
Bước 4										
Bước 5										
Bước 6										
Bước 7										
Bước 8										
Bước 9										
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Lưu đồ sắp xếp xen



Chương trình sắp xếp xen

	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										
Bước 2										

```
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--;
              }
        }
}
```

Đánh giá sắp xếp xen

- Các lệnh /*4*/ và /*5*/ đều lấy 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*/ lấy i thời gian.
- Lệnh /*2*/ và /*3*/ là hai lệnh nối tiếp nhau, lệnh /*2*/ lấy O(1) nên cả hai lệnh này lấy i.
- Vòng lặp /*1*/ có i chạy từ 1 đến n-1 nên ta có:

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

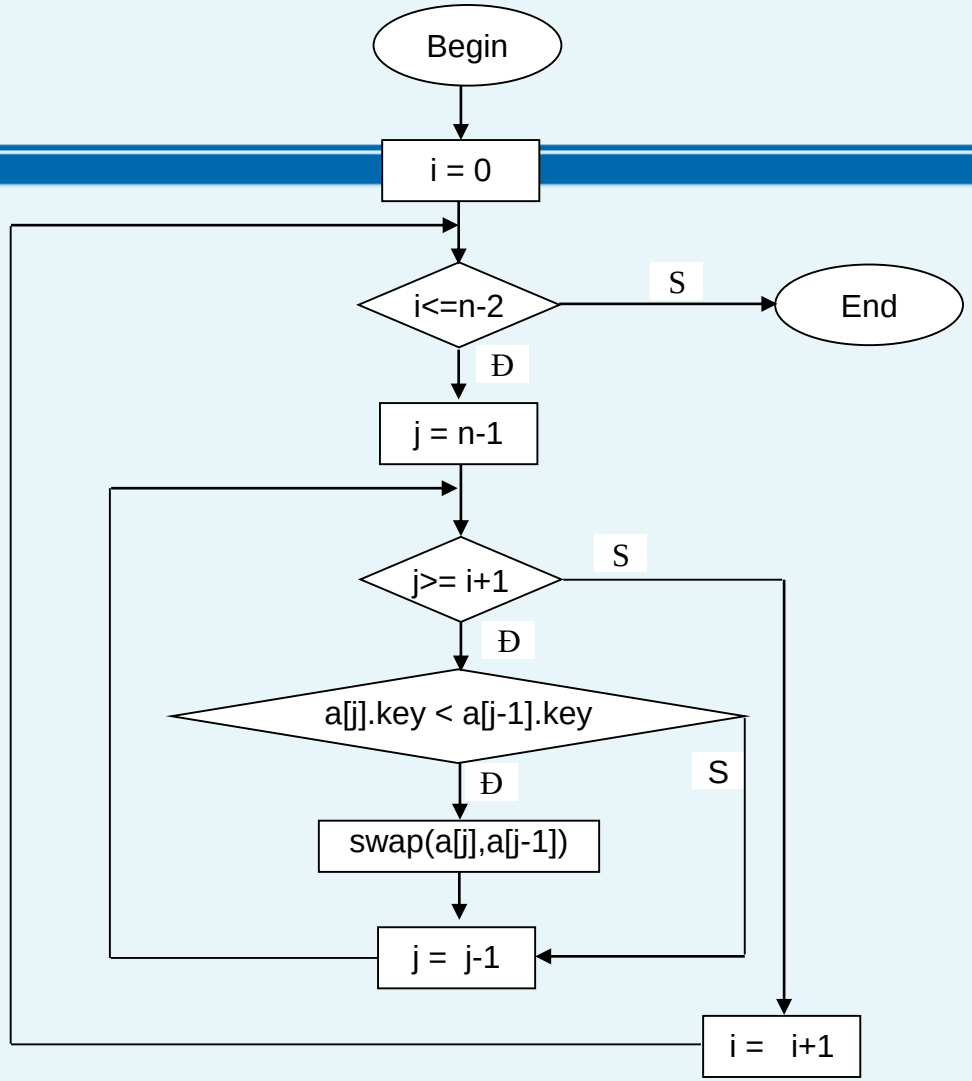
Giải thuật sắp xếp "nổi bọt" (Bubble Sort)

- Bước 1: Xét các phần tử a[j] (j giảm từ n-1 đến 1), so sánh khoá của a[j] với khoá của a[j-1]. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j] và a[j-1] cho nhau. Sau bước này thì a[0] có khoá nhỏ nhất.
- Bước 2: Xét các phần tử a[j] (j giảm từ n-1 đến 2), so sánh khoá của a[j] với khoá của a[j-1]. Nếu khoá của a[j] nhỏ hơn khoá của a[j-1] thì hoán đổi a[j] và a[j-1] cho nhau. Sau bước này thì a[1] có khoá nhỏ thứ 2.
- …
- Sau n-1 bước thì kết thúc.

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 0										
Bước 1										
Bước 2										
Bước 3										
Bước 4										
Bước 5										
Bước 6										
Bước 7										
Bước 8										
Kết quả	**2**	**2**	**3**	**5**	**6**	**9**	**9**	**10**	**10**	**12**

Lưu đồ sắp xếp nổi bọt



Chương trình sắp xếp "nổi bọt"

	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 0										
Bước 1										

```
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]);
}
```

Ý tưởng của QuickSort

- Chọn một giá trị khóa v làm chốt (pivot).
- 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**.
- Sắp xếp mảng con “bên trái” và mảng con “bên phải”.
- Sau khi đã sắp xếp được mảng con “bên trái” và mảng con “bên phải” thì mảng đã cho sẽ được sắp bởi vì tất cả các khóa trong mảng con “bên trái” đều nhỏ hơn các khóa trong mảng con “bên phải”.
- Việc sắp xếp các mảng con “bên trái” và “bên phải” cũng được tiến hành bằng phương pháp nói trên.
- Một mảng chỉ gồm một phần tử hoặc gồm nhiều phần tử có khóa bằng nhau thì đã có thứ tự.

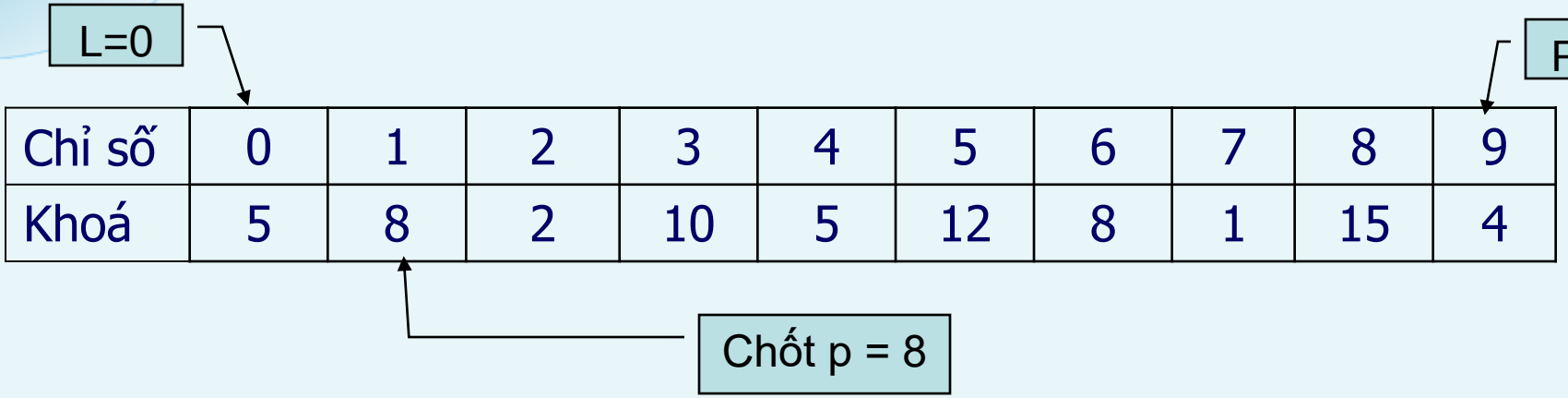
Phương pháp chọn chốt

- Chọn giá trị **khóa lớn nhất** trong hai phần tử có khóa khác nhau đầu tiên kể từ trái qua.
- Nếu mảng chỉ gồm một 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 (các phần tử có khoá bằng nhau).
  - Cho mảng gồm một phần tử có khoá là 6 thì không có chốt (do chỉ có một phần tử).

Phương pháp phân hoạch

- Để phân hoạch mảng ta dùng 2 "con nháy" L và R trong đó L từ bên trái và R từ bên phải.
- Ta cho L chạy sang phải cho tới khi gặp phần tử có khóa ≥ chốt
- Cho R chạy sang trái cho 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”.

Ví dụ về phân hoạch



Ví dụ về phân hoạch

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

Ví dụ về phân hoạch

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=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

Ví dụ về phân hoạch

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=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=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=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

Ví dụ về phân hoạch

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=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=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

Ví dụ về phân hoạch

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

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

0	1	2	3	4
5	4	2	1	5

5	6	7	8	9
12	8	10	15	8

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.

Ví dụ về QuickSort

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	4	2	1	5	12	8	10	15	8

Chốt p = 5

Ví dụ về QuickSort

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	10	15	8

Chốt p = 5

1	4	2	5	5

Chốt p = 4

Ví dụ về QuickSort

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	10	15	8

Chốt p = 5 | Chốt p = 12

1	4 2	2 4	5	5

Chốt p = 4 | xong

1	2	4

Chốt p = 2 | xong

1	2

xong | xong

Ví dụ về QuickSort

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

Chốt p = 4 | xong | Chốt p = 10 | Chốt p = 15

1	2	4	8	8	10

Chốt p = 2 | xong | xong | xong

1	2

xong | xong

Ví dụ về QuickSort

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

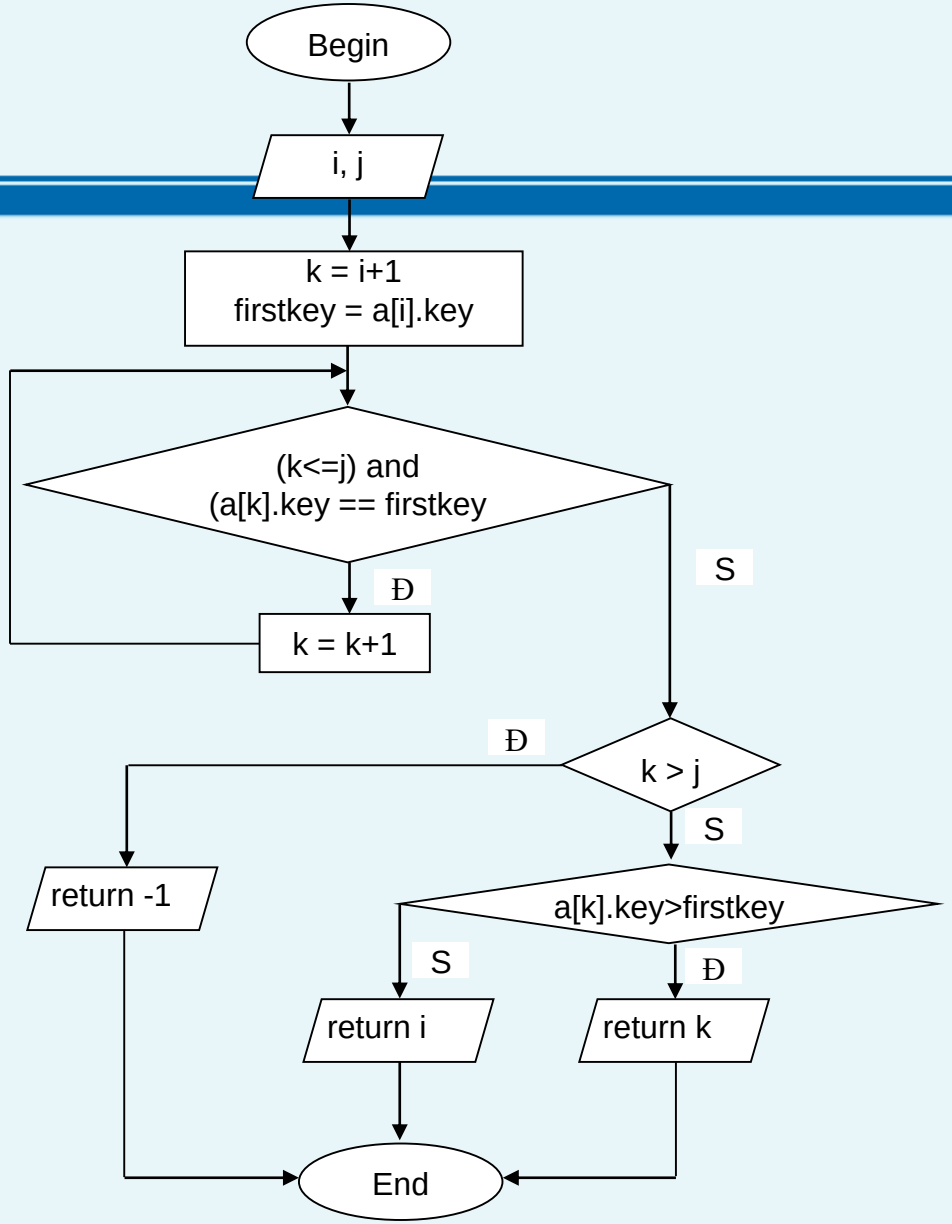
1	2	4	8	8	10	12	15	

Chốt p = 2 | xong | xong | xong | xong | xong

1	2

xong | xong

Lưu đồ hàm FindPivot



Chương trình hàm FindPivot

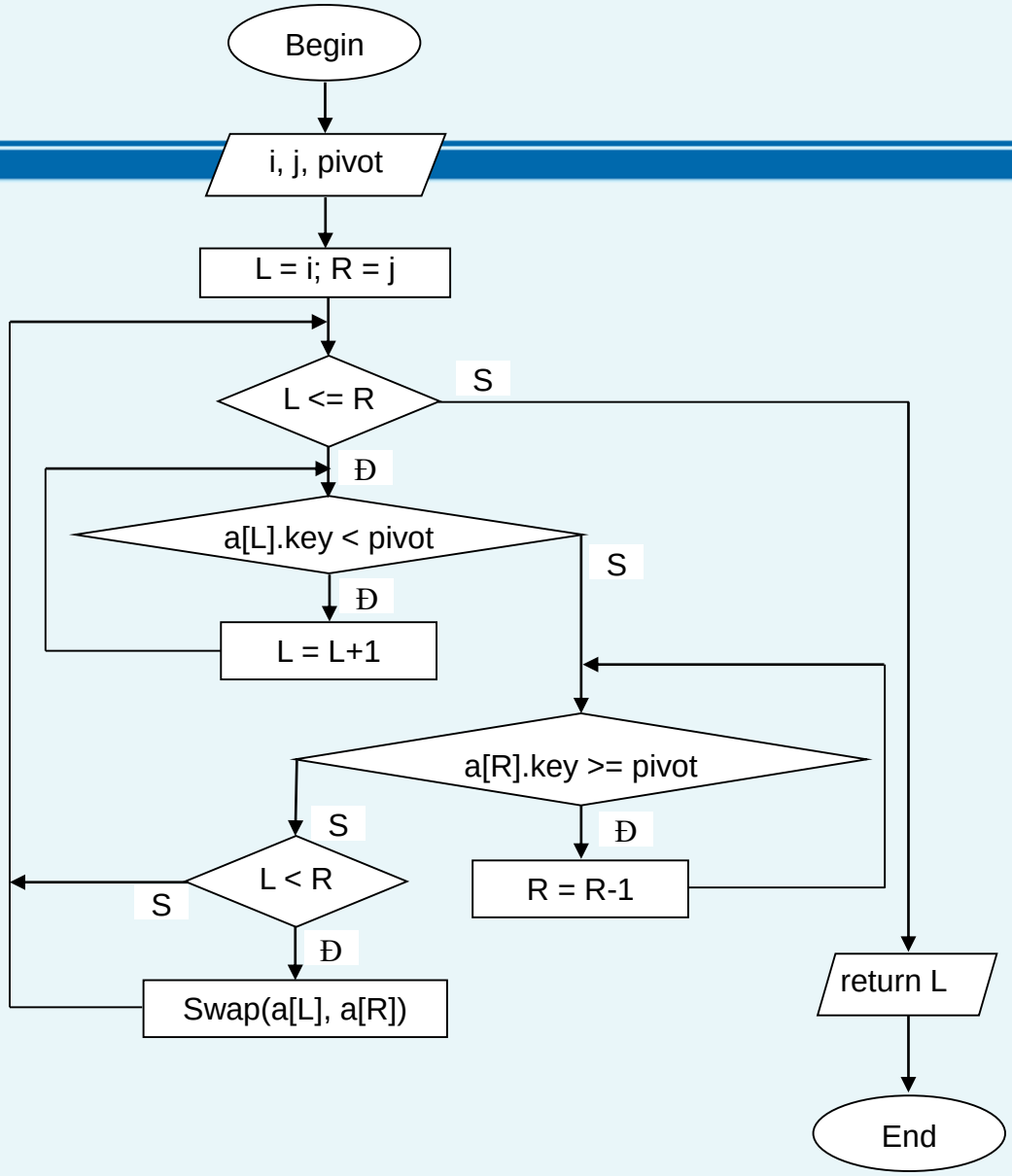
```
int FindPivot(recordtype a[], int i,int j)
{  keytype  firstkey;
   int k ;
   k = i+1;
   firstkey = a[i].key;
   while ( (k <= j) && (a[k].key == firstkey) ) k++;
   if (k > j)  return -1;
   else
      if (a[k].key>firstkey) return k; else return i;
}
```

Phân tích 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 là tốn nhiều thời gian nhất.
- Trong trường hợp xấu nhất thì k chạy từ i+1 đến j, tức là vòng lặp thực hiện j-i lần, mỗi lần O(1) do đó tốn 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



Hàm Partition

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

```
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*/
}
```

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*/ là 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 cộng việc duyệt này tốn j-i thời gian.
- Vòng lặp /*3*/ thực chất là để xét xem khi nào thì duyệt xong, do đó thời gian thực hiện của lệnh /*3*/ chính là thời gian thực hiện của hai lệnh /*4*/ và /*5*/ và do đó là j-i.
- Đặc biệt 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);
  }
}
```

Đánh giá QuickSort (Trường hợp xấu nhất)

- Giả sử các giá trị khóa của mảng khác nhau nên hàm FindPivot luôn tìm được chốt và đệ quy chỉ dừng khi kích thước bài toán bằng 1.
- Gọi T(n) là thời gian thực hiện việc QuickSort mảng có n phần tử.
- Thời gian tìm chốt và phân hoạch mảng là O(n) = n.
- Khi n = 1, thủ tục QuickSort chỉ làm một nhiệm vụ duy nhất là gọi hàm Findpivot với kích thước bằng 1, hàm này tốn thời gian O(1) =1.
- Trong trường hợp xấu nhất, phân hoạch lệch.
- Khi đó ta có thể thành lập phương trình đệ quy như sau:

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

Giải PT này ta được $T(n) = O(n^2)$

Đánh giá QuickSort (Trường hợp tốt nhất)

- Trong trường hợp tốt nhất khi ta chọn được chốt sao cho hai mảng con có kích thước bằng nhau và bằng n/2.
- Lúc đó ta có phương trình đệ quy như sau:

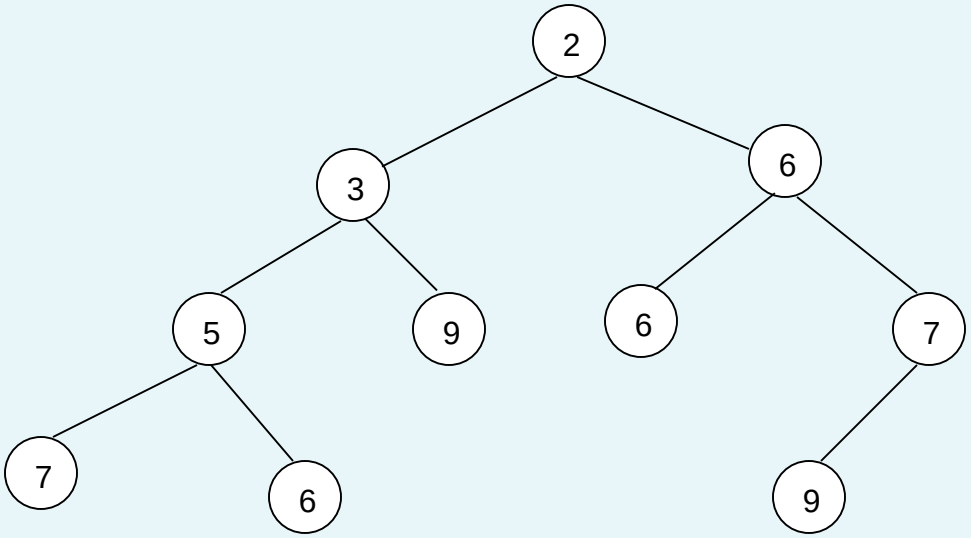
$$T(n) = \begin{cases} 1 & \text{nêu } n = 1 \\ 2T(\frac{n}{2}) + n & \text{nêu } n > 1 \end{cases}$$

Giải PT này ta được T(n) =O(nlogn)

HeapSort: Định nghĩa Heap

- Cây sắp thứ tự bộ phận hay còn gọi là heap là cây nhị phân mà giá trị tại mỗi nút (khác nút lá) đều không lớn hơn giá trị của các con của nó.
- Ta có nhận xét rằng nút gốc của cây sắp thứ tự bộ phận có giá trị nhỏ nhất.

Ví dụ về heap



HeapSort : Ý tưởng giải thuật

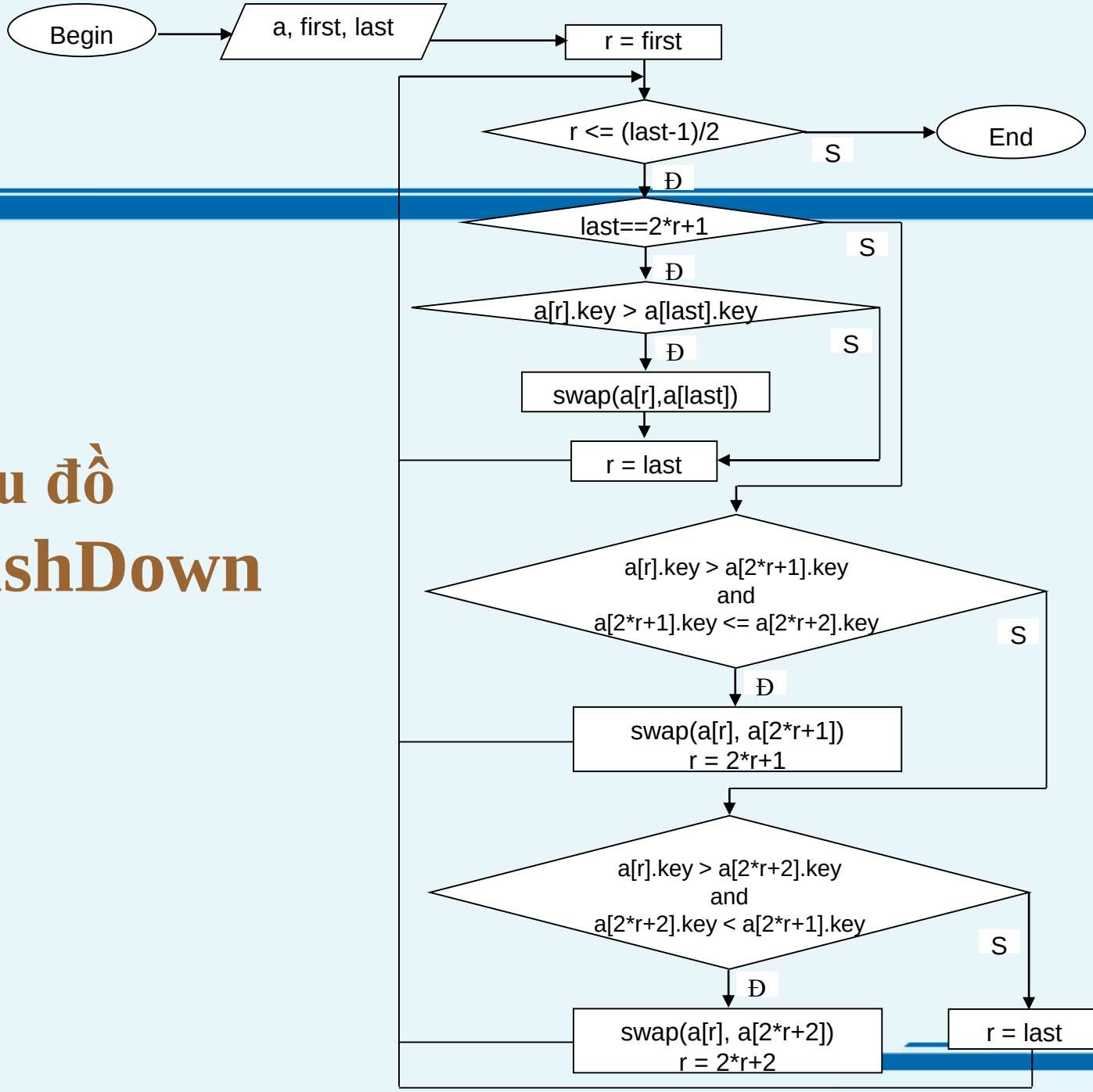
- (1) Xem mảng ban đầu là một cây nhị phân. Mỗi nút trên cây lưu trữ một phần tử mảng, trong đó a[0] là nút gốc và mỗi nút không là nút lá a[i] có con trái là a[2i+1] và con phải là a[2i+2]. Với cách tổ chức này thì cây nhị phân thu được sẽ có các nút trong là các nút a[0], …, a[(n-2)/2]. Tất cả các nút trong đều có 2 con, ngoại trừ nút a[(n-2)/2] có thể chỉ có một con trái (trong trường hợp n là một số chẵn).
- (2) Sắp xếp cây ban đầu thành một heap căn cứ vào giá trị khoá của các nút.
- (3) Hoán đổi nút gốc a[0] cho cho nút lá cuối cùng.
- (4) Sắp lại cây sau khi đã bỏ đi nút lá cuối cùng để nó trở thành một heap mới.
- Lặp lại quá trình (3) và (4) cho tới khi cây chỉ còn một nút. Nút này cùng với các nút lá đã bỏ đi tạo thành một mảng sắp theo thứ tự giảm.

Thiết kế hàm PushDown

- PushDown nhận vào 2 tham số first và last để đẩy nút first xuống.
- Giả sử a[first],..,a[last] đã đúng vị trí của một heap, ngoại trừ a[first]. PushDown dùng để đẩy phần tử a[first] xuống đúng vị trí của nó trong cây.
- Xét a[first], có các khả năng có thể xẩy ra:
  - Nếu a[firrst] chỉ có một con trái và nếu khoá của nó lớn hơn khoá của con trái (a[first].key > a[2*first+1].key) thì hoán đổi a[first] cho con trái của nó và kết thúc.
  - Nếu a[first] có khoá lớn hơn con trái của nó và khoá của con trái không lớn hơn khoá của con phải thì hoán đổi a[first] cho con trái của nó, việc này có thể gây ra tình trạng con trái sẽ không đúng vị trí nên phải xem xét lại con trái để có thể đẩy xuống.
  - Ngược lại, nếu a[first] có khoá lớn hơn khoá của con phải của nó và khoá của con phải nhỏ hơn khoá của con trái thì hoán đổi a[first] cho con phải của nó, việc này có thể gây ra tình trạng con phải sẽ không đúng vị trí nên phải tiếp tục xem xét con phải để có thể đẩy xuống.
  - Nếu tất cả các trường hợp trên đều không xẩy ra thì a[first] đã đúng vị trí.

CANTHO UNIVERSITY

Lưu đồ hàm PushDown



Chương trình hàm Pushdown

```
void PushDown(recordtype a[], int first,int last)
{ int  r= first;
  while (r <= (last-1)/2)
   if (last == 2*r+1) {
     if (a[r].key > a[last].key) Swap(a[r],a[last]);
     r = last;
   } else
   if ((a[r].key>a[2*r+1].key) && (a[2*r+1].key<=a[2*r+2].key))
   {
      Swap(a[r],a[2*r+1]);
      r = 2*r+1 ;
   } else
      if ((a[r].key>a[2*r+2].key) && (a[2*r+2].key<a[2*r+1].key))
      {
       Swap(a[r],a[2*r+2]);
       r = 2*r+2 ;
      }
      else
       r = last;
}
```

Phân tích hàm PushDown

- Ta xét PushDown(0,n-1), tức là PushDown trên cây có n nút.
- PushDown chỉ duyệt trên một nhánh nào đó của cây nhị phân, tức là sau mỗi lần lặp thì số nút còn lại một nửa. Một cách cụ thể, trước hết PushDown trên cây có n nút; Sau lần lặp thứ nhất, PushDown trên cây có n/2 nút; Sau lần lặp thứ hai, PushDown trên cây có n/4 nút;… Tổng quát, Sau lần lặp thứ i, PushDown trên cây có $n/2^i$ nút.
- Như vậy, trong trường hợp xấu nhất (luôn phải thực hiện việc đẩy xuống) thì lệnh lặp while phải thực hiện i lần sao chon/2^i = 1 tức là i=logn (i=logn là số lần lặp của lệnh while, trong trường hợp xấu nhất). Mà mỗi lần lặp chỉ thực hiện một lệnh IF với thân lệnh IF là lời gọi Swap và lệnh gán, do đó tốn O(1) = 1 đơn vị thời gian.
- Từ đó ta thấy PushDown lấy O(logn) để đẩy xuống một nút trong cây có n nút.

Chương trình hàm HeapSort

```
void HeapSort(recordtype a[], int n)
     {  int i;
/*1*/  for(i = (n-2)/2; i>=0; i--)
/*2*/          PushDown(a,i,n-1);
/*3*/  for(i = n-1; i>=2; i--) {
/*4*/          Swap(a[0],a[i]);
/*5*/          PushDown(a, 0, i-1);
        }
/*6*/  Swap(a[0],a[1]);
     }
```

Phân tích HeapSort

- Hàm PushDown lấy O(logn).
- Trong HeapSort,
  - Vòng lặp /*1*/-/*2*/ lặp (n-2)/2+1 lần mà mỗi lần lấy O(logn) nên thời gian thực hiện /*1*/-/*2*/ là O(n logn).
  - Vòng lặp /*3*/-/*5*/ lặp n-2 lần, mỗi lần lấy O(logn) nên thời gian thực hiện của /*3*/-/*5*/ là O(n logn).
- Thời gian thực hiện HeapSort là O(nlogn).

CANTHO UNIVERSITY

HeapSort: Trình bày bằng bảng

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**
Tạo Heap										
i=9										
i=8										
i=7										
i=6										