Chương 6

Một số thuật toán thông dụng

Nội dung

1. Các phương pháp sắp xếp
2. Các phương pháp tìm kiếm

Chương 2 Đánh giá độ phức tạp thuật toán

- **Bài toán sắp xếp**

- **CÁC PHƯƠNG PHÁP SẮP XẾP SƠ CẤP**
  - ✓ Sắp xếp chọn (Select Sort)
  - ✓ Sắp xếp chèn (Insert Sort)
  - ✓ Sắp xếp nổi bọt (Bubbe Sort)

- **CÁC PHƯƠNG PHÁP SẮP XẾP CAO CẤP**
  - ✓ Sắp xếp trộn (MergeSort)
  - ✓ Sắp xếp nhanh (QuickSort)
  - ✓ Sắp xếp vun đống (HeapSort)

Bài toán sắp xếp

- Cho một danh sách tuyến tính gồm các phần tử trong một tập sắp thứ tự toàn phần. Hãy sắp xếp các phần tử theo thứ tự tăng dần (hoặc giảm dần)

- Trường hợp nghiên cứu: Sắp xếp mảng: Sắp xếp mảng a[1..n] các số theo thứ tự tăng dần Sắp xếp các kí tự trong xâu theo thứ tự tăng dần

Các phương pháp sắp xếp

- Sắp xếp chọn: Lần lượt chọn phần tử nhỏ thứ nhất, thứ hai,..., thứ n xếp vào vị trí thứ nhất, thứ hai, thứ n.

- Sắp xếp chèn: Bắt đầu từ i=2, so sánh a[i] với các phần tử của dãy a[1..i-1] để chèn vào vị trí thích hợp.

- Sắp xếp nổi bọt: So sánh mỗi phần tử với phần tử đứng sau nó nếu lớn hơn thì đổi chỗ.

Phương pháp sắp xếp chọn

Selection sort

Selection sort

- **Nhận xét**
  - Mảng có thứ tự thì a[i]=min(a[i], a[i+1], …, a[n-1])
- **Ý tưởng**: mô phỏng một trong những cách sắp xếp tự nhiên nhất trong thực tế:
  - Chọn phần tử nhỏ nhất trong n phần tử ban đầu, đưa phần tử này về vị trí đúng là đầu dãy hiện hành
  - Xem dãy hiện hành chỉ còn n-1 phần tử của dãy ban đầu, bắt đầu từ vị trí thứ 2; lặp lại quá trình trên cho dãy hiện hành... đến khi dãy hiện hành chỉ còn 1 phần tử

Selection sort

- Thuật toán

Input: A[0..N-1]

Output: A[0..N-1] đã được sắp xếp không giảm

Procedure SelectSort(a[i]);

Begin

B1: i=1

B2: Tìm phần tử nhỏ nhất a[min] trong dãy từ a[i] đến a[n]

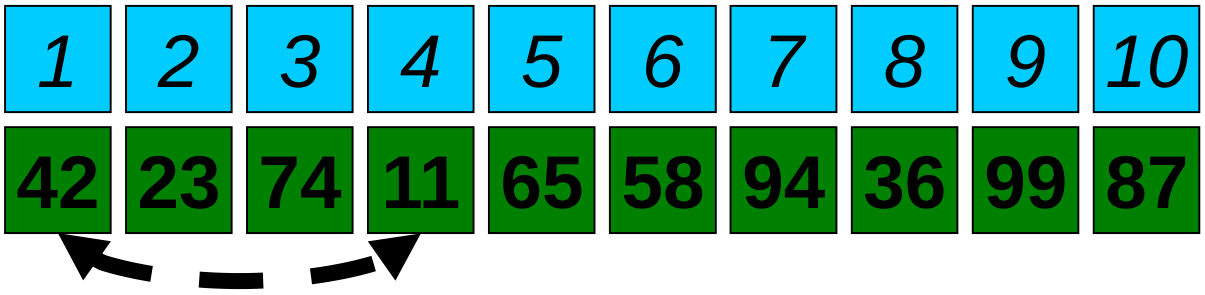
B3: Hoán vị a[min] với a[i]

B4: Nếu i<n thì i=i+1. lặp lại B2

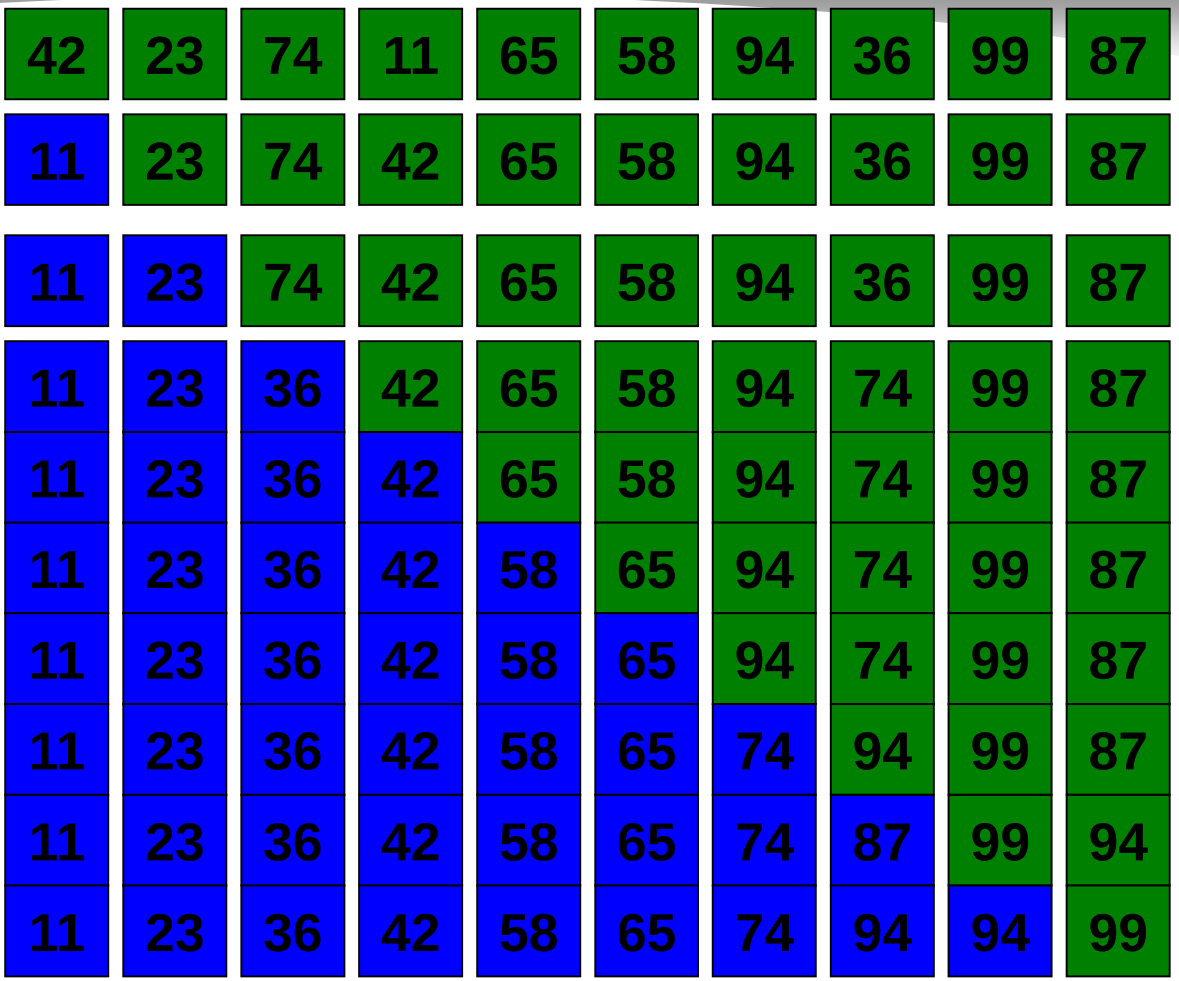
Ngược lại: Dừng

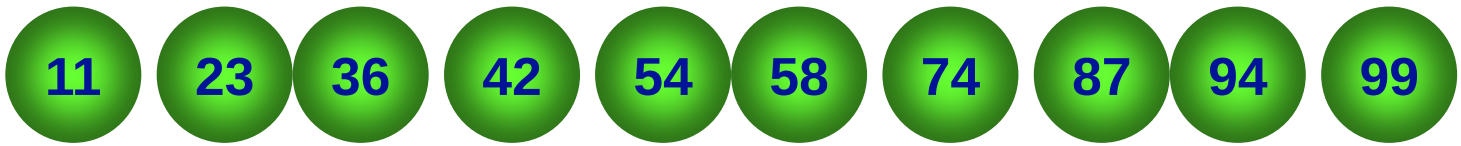
End;

Phương pháp sắp xếp chọn



SELECT SORT


42 23 74 11 65 58 94 36 99 87
11 23 74 42 65 58 94 36 99 87
11 23 74 42 65 58 94 36 99 87
11 23 36 42 65 58 94 74 99 87
11 23 36 42 65 58 94 74 99 87
11 23 36 42 58 65 94 74 99 87
11 23 36 42 58 65 94 74 99 87
11 23 36 42 58 65 74 94 99 87
11 23 36 42 58 65 74 87 99 94
11 23 36 42 58 65 74 94 94 99


11 23 36 42 54 58 74 87 94 99

Phương pháp sắp xếp chọn

- **Cài đặt**

```
void selection_sort(int A[],int n)
{ for(int i=0;i<=n-2;i++)
     {  m=i;
       for(int j=i+1;j<=n-1;j++)
        if(A[j]<A[m]) m=j;
         if(m!=i) swap(A[i],A[m]);
     }
}
```

Phương pháp sắp xếp chọn

- Ở lượt thứ i, cần (n-i) lần so sánh để xác định phần tử nhỏ nhất hiện hành
- Số lượng phép so sánh không phụ thuộc vào tình trạng của dãy số ban đầu
- Trong mọi trường hợp, số lần so sánh là:

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

Trường hợp	Số lần so sánh	Số phép gán
Tốt nhất	n(n-1)/2	0
Xấu nhất	n(n-1)/2	3n

Phương pháp sắp xếp chèn

Insertion sort

Phương pháp sắp xếp chèn

- Ý tưởng
  - ✓ Dựa trên ý tưởng việc sắp xếp quân bài
  - ✓ Chèn những quân bài đang cầm (xem xét) vào vị trí thích hợp
  - ✓ Ban đầu chỉ có một quân bài
  - ✓ Sau đó thêm các quân bài mới thì chèn quân bài đó vào vị trí thích hợp

Phương pháp sắp xếp chèn

- Nhận xét:
  - Mọi dãy **a[0] , a[1] ,..., a[n-1]** luôn có i-1 phần tử đầu tiên **a[0] , a[1] ,... ,a[i-2]** đã có thứ tự ($2 \leq i$)
- Ý tưởng chính:
  - Tìm cách chèn phần tử **ai** vào vị trí thích hợp của đoạn đã được sắp để có dãy mới **a[0] , a[1] ,... ,a[i-1]** trở nên có thứ tự
  - Vị trí này chính là pos thỏa :

$$\mathbf{a[pos-1] \leq a[i] < a[pos]} \quad (1 \leq pos \leq i)$$

Phương pháp sắp xếp chèn

Chi tiết hơn:

- *Dãy ban đầu* ***a[0], a[1],..., a[n-1]****, xem như đã có đoạn gồm một phần tử* ***a[0]*** *đã được sắp*
- *Thêm* ***a[1]*** *vào đoạn* ***a[0]*** *sẽ có đoạn* ***a[0] a[1]*** *được sắp*
- *Thêm* ***a[2]*** *vào đoạn* ***a[0] a[1]*** *để có đoạn* ***a[0] a[1] a[2]*** *được sắp*
- *Tiếp tục cho đến khi thêm xong* ***a[n-1]*** *vào đoạn* ***a[0] a[1] … a[n-1]*** *sẽ có dãy* ***a[0] a[1]… A[n-1]*** *được sắp*

Phương pháp sắp xếp chèn

- Thuật toán

Input: A[0..N-1] phần từ

Ouput: A[0..N-1] đã được sắp xếp không giảm

```
for i=2->N-1
        a. x=A[i];
        b. min=i-1;
        c.  while (min>=0 and A[min]>x)
              A[min]=A[min-1];
              min--;
        d. A[min+1]=x;
```

Phương pháp sắp xếp chèn

```
Procedure InsertSort(a[1..n]);
Begin
        For i:=2 to n do
        begin
                j:=i; v:=a[i];
                while j>1 and a[j]<a[j-1] do
                begin a[j]:=a[j-1],j=j-1; end;
                a[j]:=v;
        end;
End;
```

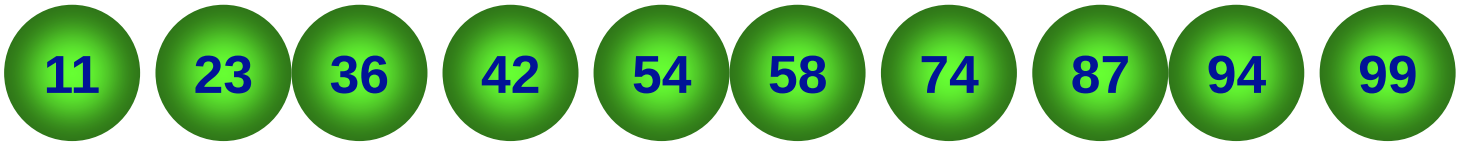
Phương pháp sắp xếp chèn

❖ **Cài đặt**

```
void insertion_sort(int A[],int n)
{
    for(int i=2;i<=n;i++)
    { x=A[i]; min =i-1;
      while((x<A[min])&&(min >0))
        {A[min]=A[min -1]; min --; }
      A[j+1]=x;
    }
}
```

Phương pháp sắp xếp chèn

1	2	3	4	5	6	7	8	9	10
42	23	74	11	65	58	94	36	99	87
23	42	74	11	65	58	94	36	99	87
11	23	42	74	65	58	94	36	99	87



Phương pháp sắp xếp chèn

- Các phép so sánh xảy ra trong mỗi vòng lặp tìm vị trí thích hợp min. Mỗi lần xác định vị trí min đang xét không thích hợp dời chỗ phần tử a[min -1] đến vị trí min
- Giải thuật thực hiện tất cả N-1 vòng lặp tìm min , do số lượng phép so sánh và dời chỗ này phụ thuộc vào tình trạng của dãy số ban đầu, nên chỉ có thể ước lược trong từng trường hợp như sau:

Trường hợp	Số phép so sánh	Số phép gán
Tốt nhất	$\sum_{i=1}^{n-1} 1 = n-1$	$\sum_{i=1}^{n-1} 2 = 2(n-1)$
Xấu nhất	$\sum_{i=1}^{n-1} (i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1$

Phương pháp sắp xếp nổi bọt

BUBBLE SORT

Phương pháp sắp xếp nổi bọt

Phương pháp nổi bọt:

- Dựa trên ý tưởng về các bọt khí trong cốc bia
- Hai bọt khí cạnh nhau thì bọt lớn hơn sẽ nổi lên trên
- Đến khi không còn bọt khí nào trái quy luật đó thì các bọt khí đã được sắp xếp

Phương pháp sắp xếp nổi bọt

Ý tưởng:

- Đi từ *cuối mảng đến đầu mảng*, nếu phần tử ở dưới < phần tử đứng trên nó thì sẽ được "đưa lên trên".
- Sau mỗi lần đi duyệt dãy, 1 phần tử sẽ được đưa lên đúng chỗ của nó. Đối với mảng M có N phần tử thì sau N-1 lần đi duyệt dãy → dãy M có thứ tự tăng

Phương pháp sắp xếp nổi bọt

Thuật toán:

B1: i = 1; // lần xử lý đầu tiên

B2: j=n; //duyệt từ cuối dãy về I

Trong khi i<j thì thực hiện

Nếu a[j]<a[j-1] thì hoán vị(a[j],a[j-1]) j=j-1;

B3:i=i+1;

Nếu i>n-1 thì dừng; // đã xét hết dãy số

Ngược lại: Lặp lại B2

Phương pháp sắp xếp nổi bọt

- **Cài đặt**

```
void bubble_sort(int A[],int n)
{
    for(int i=1;i<=n-1;i++)
        for(int j=n;j>i;j--)
            if(A[j]<A[j-1])  swap(A[j],A[j-1]);
}
```

Phương pháp sắp xếp nổi bọt

- Khuyết điểm:
  - Không nhận diện được tình trạng dãy đã có thứ tự hay có thứ tự từng phần
  - Các phần tử nhỏ được đưa về vị trí đúng rất nhanh, trong khi các phần tử lớn lại được đưa về vị trí đúng rất chậm

Các phương pháp sắp xếp cao cấp

- **Sắp xếp trộn**: dựa trên giải thuật cơ bản là trộn (hay hay sắp xếp kiểu hòa nhập) hai mảng con kề nhau đã được sắp thành một mảng con được sắp.

- **Sắp xếp nhanh**: (hay sắp xếp kiểu phân đoạn) dựa trên thủ tục phân chia một mảng thành hai nửa mảng trong đó tất cả các phần tử trong nửa mảng sau lớn hơn tất cả các phần tử trong nửa mảng trước

- **Sắp xếp kiểu vun đống**: là một kiểu sắp xếp chọn trong đó tạo một hàng đợi có ưu tiên dưới dạng đống để lần lượt lấy phần tử lớn nhất đặt vào cuối mảng.

QUICK SORT

Quick Sort

- QuickSort chia mảng thành hai danh sách bằng cách so sánh từng phần tử của danh sách với một phần tử được chọn được gọi là phần tử chốt.
- Những phần tử nhỏ hơn hoặc bằng phần tử chốt được đưa về phía trước và nằm trong danh sách con thứ nhất
- các phần tử lớn hơn chốt được đưa về phía sau và thuộc danh sách con thứ hai.
- Cứ tiếp tục chia như vậy tới khi các danh sách con đều có độ dài bằng 1.

Quick Sort

- Ý tưởng thuật toán sắp xếp Quick Sort
  - Bước 1: Phân chia dãy X[L .. R] thành 2 dãy con bằng cách:
    - ✓ Chọn giá trị P của 1 phần tử trên dãy X[L .. R] làm mốc phân hoạch.
    - ✓ Đổi chổ các phần tử X[i]>= P với các phần tử X[j]<= P, với i < j

Quick Sort

- ✓ Khi i > j, dãy ban đầu được phân thành 2 phần:
  1. X[L .. j] chứa giá trị <= P
  2. X[j .. i] chứa giá trị = P
  3. X[i .. R] chứa giá trị >= P

- Bước 2:
  - Nếu X[L .. j] có hơn 1 phần tử thì sắp xếp tăng dãy X[L .. j]
  - Nếu X[i .. R] có hơn 1 phần tử thì sắp xếp tăng dãy X[i .. R]

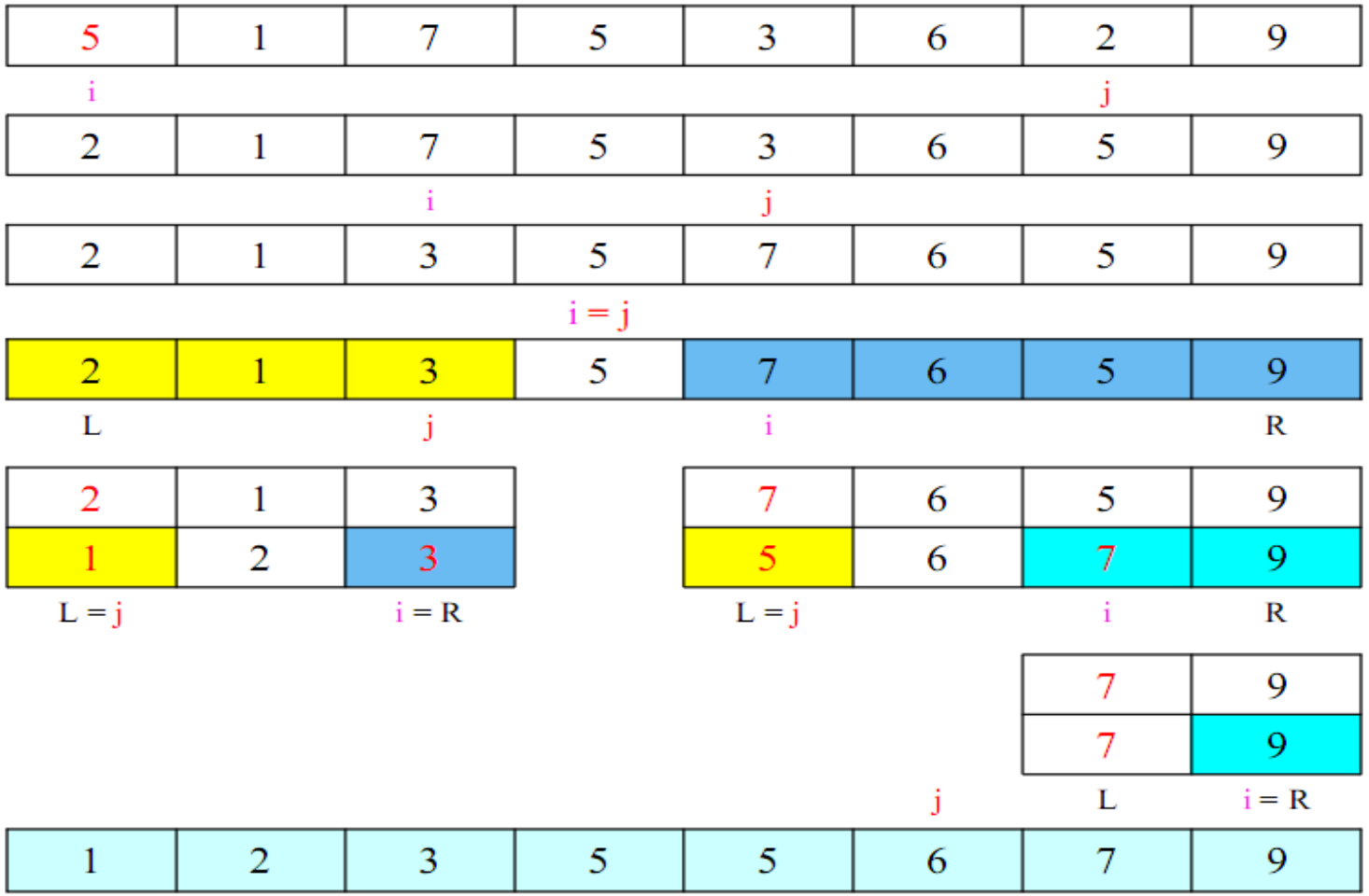
Quick Sort

- **Cài đặt**

```
void QuickSort(X[L .. R])
{i = L;j = R;P = X[i];
while (i <= j)
{
        while (X[i] < P) i++;
        while (X[j] > P) j--;
        if (i <= j)
        {if (X[i] <> X[j]) Hoanvi(X[i], X[j]);i++;  j--;}
}
 if (L < j) QuickSort(X[L ..j ]);
 if (i < R) QuickSort(X[i .. R]);}
```

Quick Sort

- Ví dụ



Quick Sort

Hiệu quả phụ thuộc vào việc chọn giá trị mốc:

- Trường hợp tốt nhất: mỗi lần phân hoạch đều chọn phần tử median làm mốc, khi đó dãy được phân chia thành 2 phần bằng nhau và cần $\log_2(n)$ lần phân hoạch thì sắp xếp xong
- Nếu mỗi lần phân hoạch chọn phần tử có giá trị cực đại (hay cực tiểu) là mốc ➔ dãy sẽ bị phân chia thành 2 phần không đều: một phần chỉ có 1 phần tử, phần còn lại gồm (n-1) phần tử, do vậy cần phân hoạch n lần mới sắp xếp xong

Các giải thuật tìm kiếm

- Tìm kiếm tuyến tính
- Tìm kiếm nhị phân

Tìm kiếm tuyến tính

Cài đặt

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

Tìm kiếm tuyến tính lính canh

- Cải tiến tìm kiếm tuyến tính:
  - Sử dụng kỹ thuật phần tử “lính canh”
  - cho thêm 1 phầntử a[n]=x, như vậy, bảo đảm luôn tìm thấy x trong mảng.

Tìm kiếm tuyến tính

Cài đặt:

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

Tìm kiếm nhị phân

Ý tưởng:

- Đối với những dãy đã có thứ tự (giả sử thứ tự tăng), các phần tử trong dãy có quan hệ:

  $a_{i-1} \le a_i \le a_{i+1}$

- Nếu x > ai thì x chỉ có thể xuất hiện trong đoạn [ai+1 , an-1] của dãy.
- Nếu x < ai thì x chỉ có thể xuất hiện trong đoạn [a0, ai-1] của dãy

Tìm kiếm nhị phân

- So sánh x với phần tử giữa mảng, “bằng” thì kết thúc nếu không thì tùy giá trị của x mà ta sẽ thu hẹp không gian tìm kiếm và lặp lại bước trên cho đến khi tìm thấy, hoặc không gian tìm rỗng.

Tìm kiếm nhị phân

- **Cài đặt**

```
int BinarySearch(int a[],int n,int x)
{ int left = 0, right = n-1, mid;
do{
        mid = (left + right)/2;
        if (x == a[mid])return mid;
        if (x<a[mid])right = mid -1;
        else left  = mid +1;
} while (left <= right);
return -1;}
```