MỘT SỐ THUẬT TOÁN SẮP XẾP TÌM KIẾM CƠ BẢN TRONG LẬP TRÌNH

A. CÁC PHƯƠNG PHÁP SẮP XẾP CƠ BẢN

I. Định nghĩa bài toán sắp xếp

Sắp xếp là quá trình xử lý một danh sách các phần tử (hoặc các mẫu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu giữ tại mỗi phần tử.

Tại sao cần phải sắp xếp các phần tử thay vì để nó ở dạng tự nhiên (chưa có thứ tự) vốn có ? Ví dụ của bài toán tìm kiếm với phương pháp tìm kiếm nhị phân và tuần tự đủ để trả lời câu hỏi này.

Khi khảo sát bài toán sắp xếp, ta sẽ phải làm việc nhiều với một khái niệm gọi là *nghịch thế*.

Khái niệm nghịch thế:

Xét một mảng các số a 0 , a 1 , . a n .

Nếu có i<j và a i > a j , thì ta gọi đó là một nghịch thế.

Mảng chưa sắp xếp sẽ có nghịch thế.

Mảng đã có thứ tự sẽ không chứa nghịch thế. Khi đó a_0 sẽ là phần tử nhỏ nhất rồi đến a_1, a_2,…

a_0 **?** a_1 **?** . **?** a_n

Như vậy, để sắp xếp một mảng, ta có thể tìm cách giảm số các nghịch thế trong mảng này bằng cách hoán vị các cặp phần tử a_i, a_jnếu có i<j và $a_i > a_j$ theo một qui luật nào đó.

Cho trước một dãy số **a1 , a2 ,... , aN**được lưu trữ trong cấu trúc dữ liệu mảng a[N];

Sắp xếp dãy số **a1 , a2 ,... ,aN**là thực hiện việc bố trí lại các phần tử sao cho hình thành được dãy mới **ak1 , ak2 ,... ,akN** có thứ tự (giả sử xét thứ tự tăng) nghĩa là **aki ? aki-1.** Mà để quyết định được những tình huống cần thay đổi vị trí các phần tử trong dãy, cần dựa vào kết quả của một loạt phép so sánh. Chính vì vậy, hai thao tác so sánh và gán là các thao tác cơ bản của hầu hết các thuật toán sắp xếp.

Khi xây dựng một thuật toán sắp xếp cần chú ý tìm cách giảm thiểu những phép so sánh và đổi chỗ không cần thiết để tăng hiệu quả của thuật toán. Đối với các dãy số được lưu trữ trong bộ nhớ chính, nhu cầu tiết kiệm bộ nhớ được đặt nặng, do vậy những thuật toán sắp xếp đòi hỏi cấp phát thêm vùng nhớ để lưu trữ dãy kết quả ngoài vùng nhớ lưu trữ dãy số ban đầu thường ít được quan tâm. Thay vào đó, các thuật toán sắp xếp trực tiếp trên dãy số ban đầu - gọi là các thuật toán sắp xếp tại chỗ - lại được đầu tư phát triển. Phần này giới thiệu một số giải thuật sắp xếp từ đơn giản đến phức tạp có thể áp dụng thích hợp cho việc sắp xếp nội.

II. Các phương pháp sắp xếp

Sau đây là một số phương pháp sắp xếp thông dụng sẽ được đề cập đến trong giáo trình này:

1. Chọn trực tiếp - Selection sort

2. Chèn trực tiếp - Insertion sort
3. Nổi bọt - Bubble sort

Trong đó, chúng ta sẽ lần lượt khảo sát các thuật toán trên. các thuật toán như Interchange sort, Bubble sort, Shaker sort, Insertion sort, Selection sort là những thuật toán đơn giản dễ cài đặt nhưng chi phí cao . Các thuật toán Shell sort, Heap sort, Quick sort, Merge sort phức tạp hơn nhưng hiệu suất cao hơn nhóm các thuật toán đầu. cả hai nhóm thuật toán trên đều có một điểm chung là đều được xây dựng dựa trên cơ sở việc so sánh giá trị của các phần tử trong mảng (hay so sánh các khóa tìm kiếm). Riêng phương pháp Radix sort đại diện cho một lớp các thuật toán sắp xếp khác hẳn các thuật toán trước. Lớp thuật toán này không dựa trên giá trị của các phần tử để sắp xếp.

III. Các thuật toán sắp xếp

1. Chọn trực tiếp - Selection sort

- **Ý tưởng**

Ta thấy rằng, nếu mảng có thứ tự, phần tử a_i luôn là $\min(a_i, a_{i+1}, ., a_{n-1})$. Ý tưởng của thuật toán chọn trực tiếp 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; sau đó không quan tâm đến nó nữa, 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. Dãy 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 dãy hiện hành về vị trí đúng ở đầu dãy.

- **Thuật toán**:

Bước 1: i = 1;

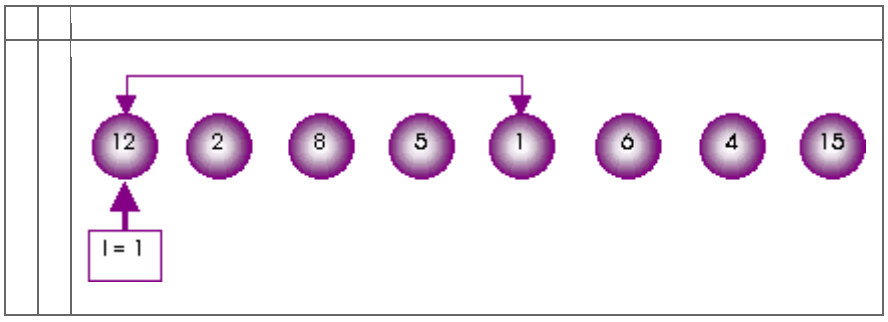
Bước 2: Tìm phần tử a[min] nhỏ nhất trong dãy hiện hành từ a[i] đến a[N]

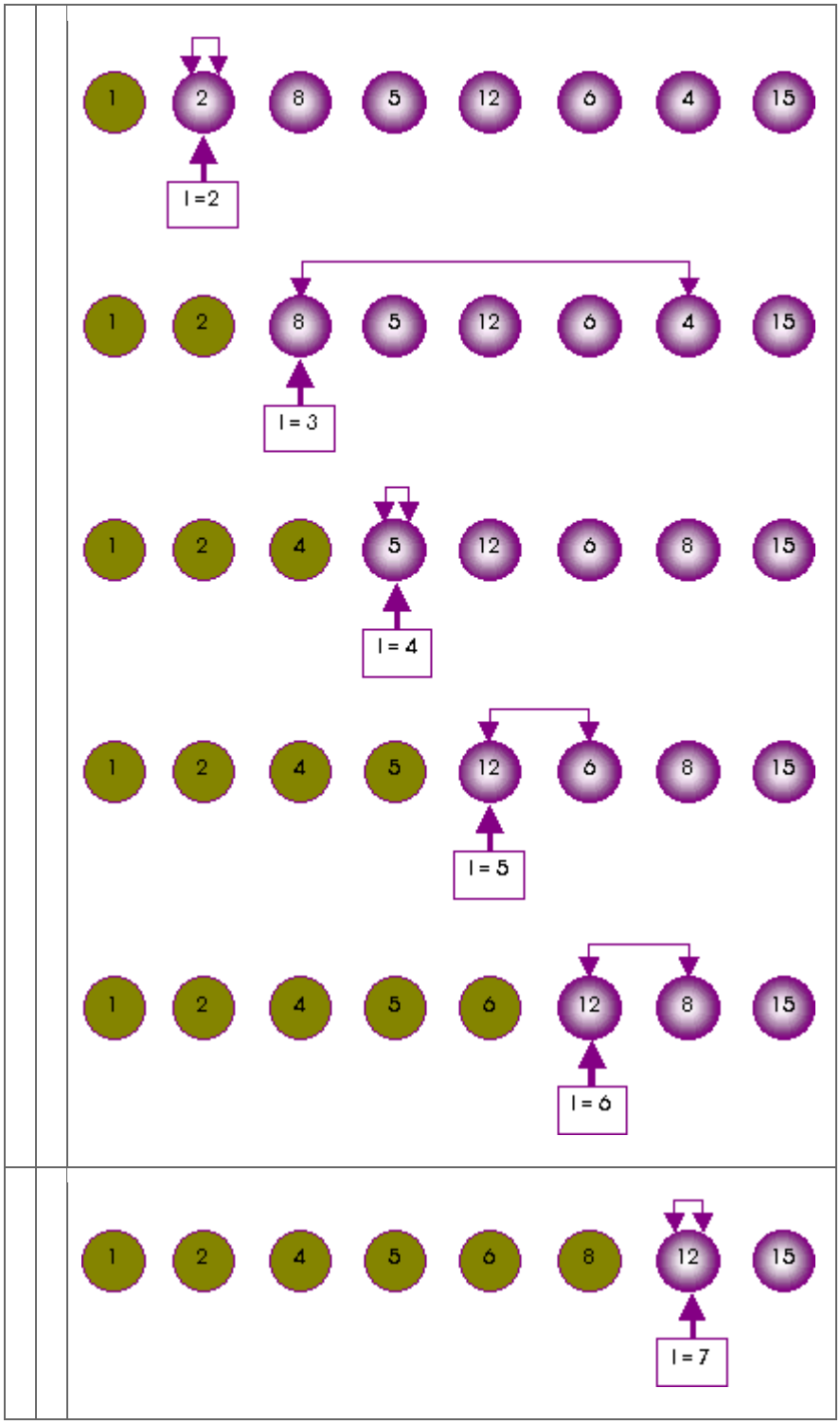
Bước 3 : Hoán vị a[min] và a[i]

Bước 4 : Nếu i ◆ N-1 thì i = i+1; Lặp lại Bước 2 Ngược lại: Dừng. //N-1 phần tử đã nằm đúng vị trí.

Ví dụ

Cho dãy số a: 12 2 8 5 1 6 4 15





- **Chương trình**

```
Procedure SelectionSort;
Var I,j,k: integer;
   min: Real;
Begin
   for i:=1 to n-1 do
```

```
        Begin
          k:=1; min:= a[i];
          for j:=i+1 to n do
          If a[j]<min then
          Begin
            min:= a[j];
            k:=j;
          End;
          Swap(a[i],a[k]);
        End;
    End;
```

- **Đánh giá giải thuật**

Đối với giải thuật chọn trực tiếp, có thể thấy rằng ở lượt thứ i, bao giờ cũng cần (n-i) lần so sánh để xác định phần tử nhỏ nhất hiện hành. Số lượng phép so sánh này không phụ thuộc vào tình trạng của dãy số ban đầu, do vậy trong mọi trường hợp có thể kết luận :

Số lần so sánh =

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

Số lần hoán vị (một hoán vị bằng 3 phép gán) lại phụ thuộc vào tình trạng ban đầu của dãy số, ta chỉ có thể ước lược trong từng trường hợp như sau :

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

2. Chèn trực tiếp - Insertion sort

- **Ý tưởng**

Giả sử có một dãy **a1 , a2 ,... ,an** trong đó iphần tử đầu tiên **a1 , a2 ,... ,ai-1** đã có thứ tự. Ý tưởng chính của giải thuật sắp xếp bằng phương pháp chèn trực tiếp là tìm cách chèn phần

tử **ai** vào vị trí thích hợp của đoạn đã được sắp để có dãy mới **a1 , a2 ,... ,ai** trở nên có thứ tự. Vị trí này chính là vị trí giữa hai phần tử **ak-1**và **ak** thỏa **ak-1 ≤ ai <ak**(1≤k≤i).

Cho dãy ban đầu **a1 , a2 ,... ,an**, ta có thể xem như đã có đoạn gồm một phần tử **a1** đã được sắp, sau đó thêm **a2** vào đoạn **a1** sẽ có đoạn **a1 a2** được sắp; tiếp tục thêm **a3** vào đoạn **a1 a2** để có đoạn **a1 a2 a3** được sắp; tiếp tục cho đến khi thêm xong **aN** vào đoạn **a1 a2 ...aN-1** sẽ có dãy **a1 a2.... aN** được sắp. Các bước tiến hành như sau :

- **Thuật toán**

Bước 1: i = 2; // giả sử có đoạn a[1] đã được sắp

Bước 2: x = a[i]; Tìm vị trí pos thích hợp trong đoạn a[1] đến a[i-1] để chèn a[i] vào

Bước 3: Dời chỗ các phần tử từ a[pos] đến a[i-1] sang phải 1 vị trí để dành chỗ cho a[i]

Bước 4: a[pos] = x; // có đoạn a[1]..a[i] đã được sắp

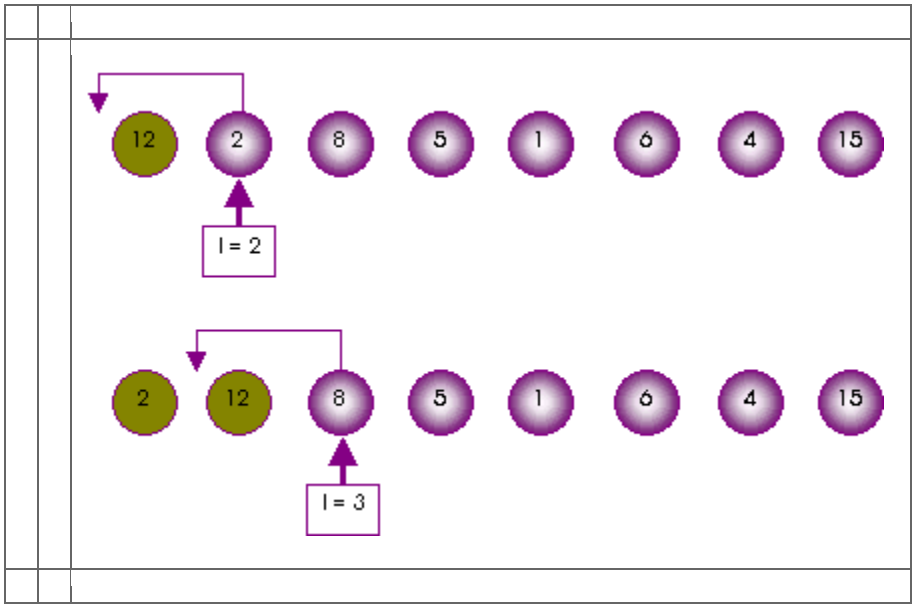
Bước 5: i = i+1;

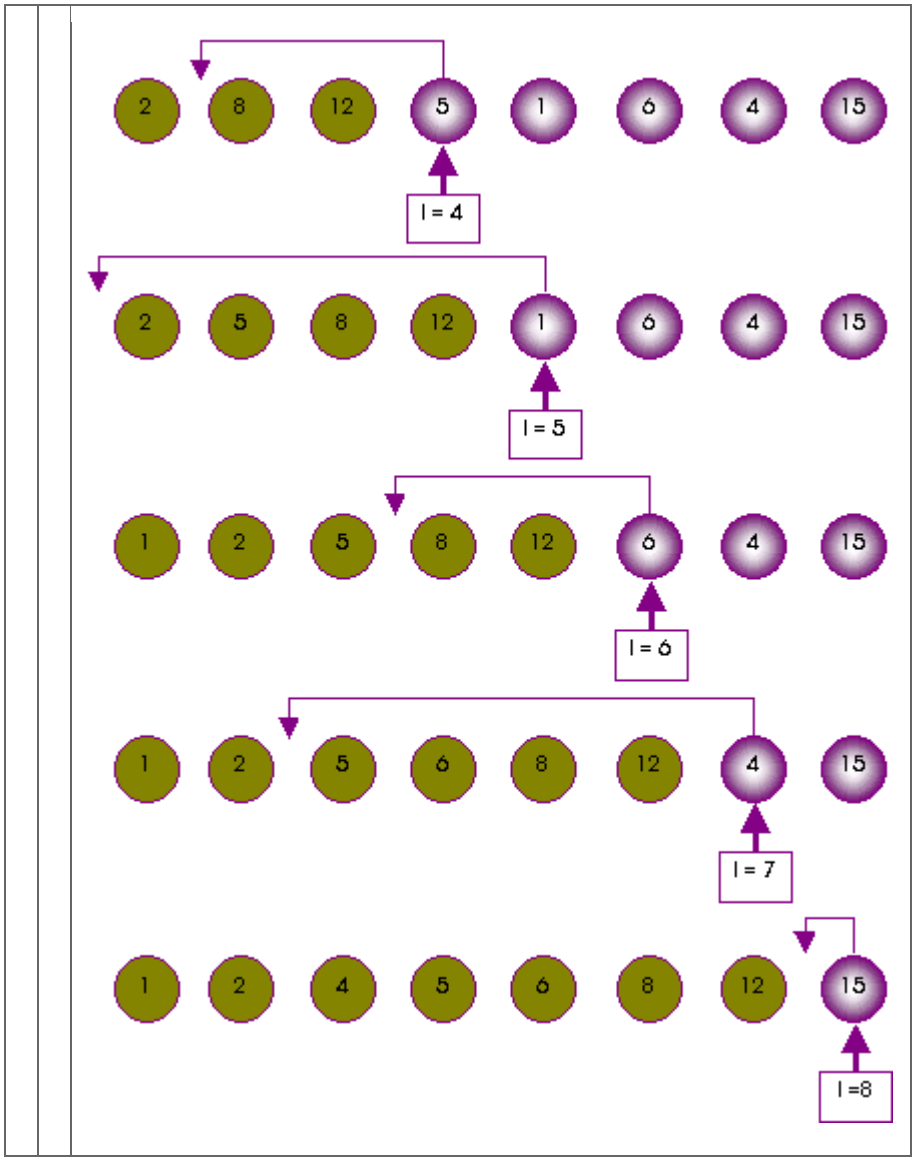
Nếu i≤ n : Lặp lại Bước 2.

Ngược lại : Dừng.

Ví dụ

Cho dãy số a: 12 2 8 5 1 6 4 15





- **Chương trình**

```
Procedure InsertionSort;
Var i,j: Integer;
Begin
   For i:=2 to n do
   Begin
      j:=i;
     While (j>1) anh (a[j]<a[j-1]) do
     Begin
        Swap (a[j],a[j-1]);
        j:=j-1;
```

```
        End;
    End;
End;
```

- **Nhận xét**

Khi tìm vị trí thích hợp để chèn a[i] vào đoạn a[0] đến a[i-1], do đoạn đã được sắp xếp, nên có thể sử dụng giải thuật tìm kiếm nhị phân để thực hiện việc tìm vị trí pos.

- **Đánh giá giải thuật**

Đối với giải thuật chèn trực tiếp, các phép so sánh xảy ra trong mỗi vòng lặp **while** tìm vị trí thích hợp **pos**, và mỗi lần xác định vị trí đang xét không thích hợp, sẽ dời chỗ phần tử a[pos] tương ứng. Giải thuật thực hiện tất cả N-1 vòng lặp **while** , 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	$\sum_{i=1}^{n-1} (i-1) = \frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1} (i+1) = \frac{n(n+1)}{2} - 1$

3. Nổi bọt - Bubble sort

- **Ý tưởng**

Ý tưởng chính của giải thuật là xuất phát từ cuối (đầu) dãy, đổi chỗ các cặp phần tử kế cận để đưa phần tử nhỏ (lớn) hơn trong cặp phần tử đó về vị trí đúng đầu (cuối) dãy hiện hành, sau đó sẽ không xét đến nó ở bước tiếp theo, do vậy ở lần xử lý thứ i sẽ có vị trí đầu dãy là i . Lặp lại xử lý trên cho đến khi không còn cặp phần tử nào để xét. Các bước tiến hành như sau :

- **Thuật toán**

Bước 1 : i = 1; // lần xử lý đầu tiên

Bước 2 : j = N; //Duyệt từ cuối dãy ngược về vị trí i

Trong khi (j < i) thực hiện:

Nếu a[j]<a[j-1]: a[j]<a[j-1];//xét cặp phần tử kế cận

j = j-1;

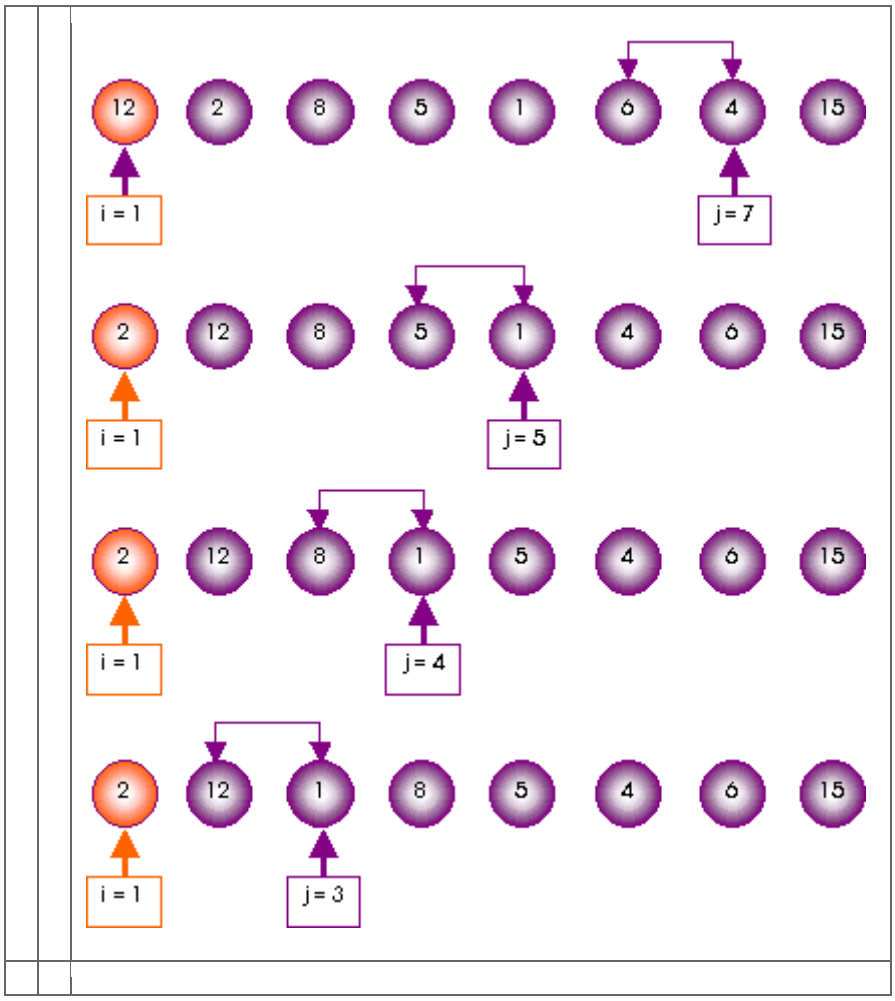
Bước 3 : i = i+1; // lần xử lý kế tiếp

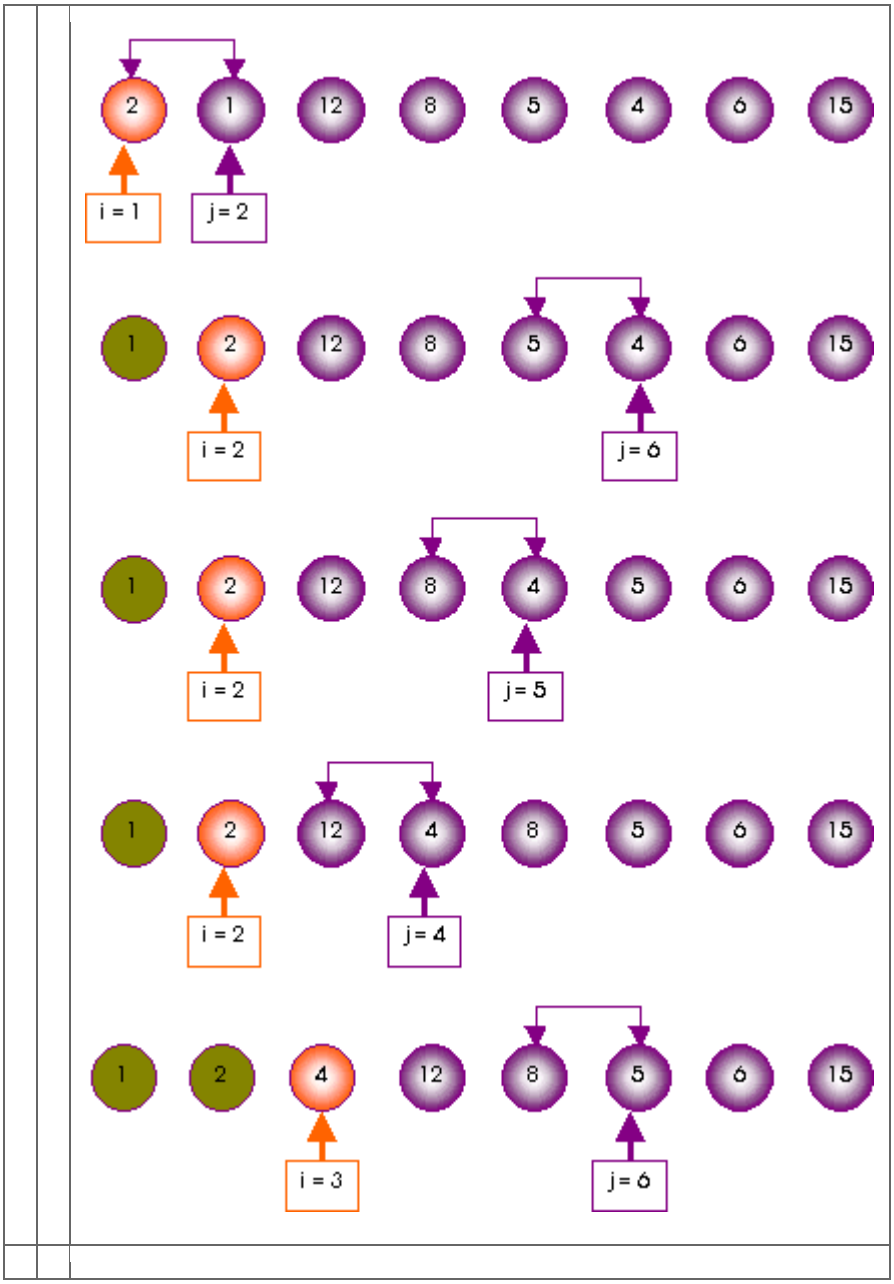
Nếu i >N-1: Hết dãy. Dừng

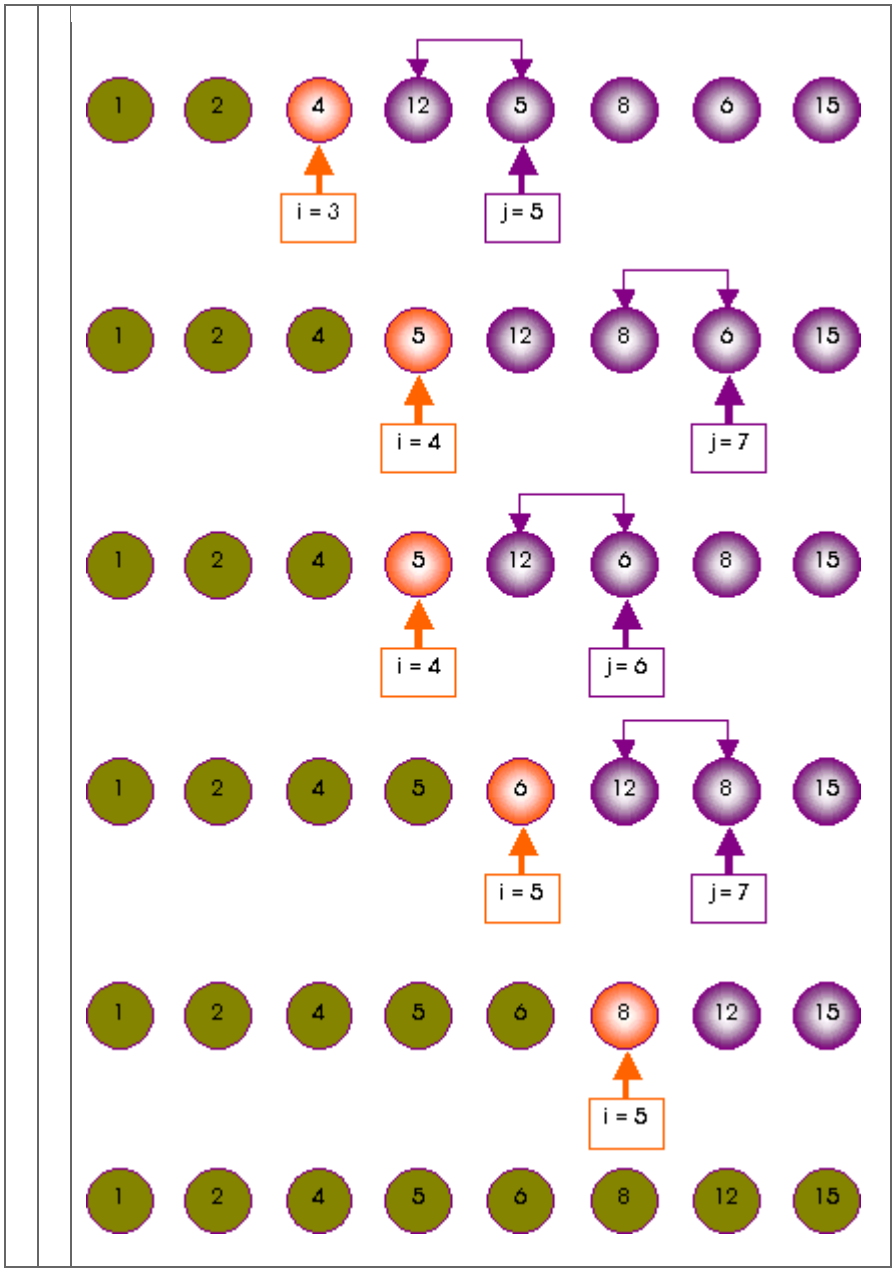
Ngược lại : Lặp lại Bước 2.

Ví dụ

Cho dãy số a: 12 2 8 5 1 6 4 15




2
1
12
8
5
4
6
15
i = 1
j = 2
1
2
12
8
5
4
6
15
i = 2
j = 6
1
2
12
8
4
5
6
15
i = 2
j = 5
1
2
12
4
8
5
6
15
i = 2
j = 4
1
2
4
12
8
5
6
15
i = 3
j = 6



- **Chương trình**

```
Procedure Bubblesort;
Var i,j: integer;
Begin
    for i:=1 to n-1 do
      for j:=i+1 to n do
        if A[i]>A[j] then
          begin
            tam:=A[i];
            A[i]:=A[j]
```

```
            A[j]:=tam
        end;
    End;
```

- **Đánh giá giải thuật**

Đối với giải thuật nổi bọt, số lượng các phép so sánh xảy ra không phụ thuộc vào tình trạng của dãy số ban đầu, nhưng số lượng phép hoán vị thực hiện tùy thuộc vào kết qủa so sánh, có thể ước lược trong từng trường hợp như sau :

Trường hợp	Số lần so sánh	Số lần hoán vị
Tốt nhất	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$	0
Xấu nhất	$\frac{n(n-1)}{2}$	$\sum_{i=1}^{n-1}(n-i+1)=\frac{n(n-1)}{2}$

- **Nhận xét**

BubbleSort có các khuyết điểm sau: 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.

B. CÁC PHƯƠNG PHÁP TÌM KIẾM CƠ BẢN

I. Định nghĩa bài toán tìm kiếm

Trong hầu hết các hệ lưu trữ, quản lý dữ liệu, thao tác tìm kiếm thường được thực hiện nhiều nhất để khai thác thông tin :

Ví du: tra cứu từ điển, tìm sách trong thư viện...

Do các hệ thống thông tin thường phải lưu trữ một khối lượng dữ liệu lớn, nên việc xây dựng các giải thuật cho phép tìm kiếm nhanh sẽ có ý nghĩa rất lớn. Nếu dữ liệu trong hệ thống đã được tổ chức theo một trật tự nào đó, thì việc tìm kiếm sẽ tiến hành nhanh chóng và hiệu quả hơn:

Ví dụ: các từ trong từ điển được sắp xếp theo từng vần, trong mỗi vần lại được sắp xếp theo trình tự alphabet; sách trong thư viện được xếp theo chủ đề ...

Vì thế, khi xây dựng một hệ quản lý thông tin trên máy tính, bên cạnh các thuật toán tìm kiếm, các thuật toán sắp xếp dữ liệu cũng là một trong những chủ đề được quan tâm hàng đầu.

Hiện nay đã có nhiều giải thuật tìm kiếm và sắp xếp dược xây dựng, mức độ hiệu quả của từng giải thuật còn phụ thuộc vào tính chất của cấu trúc dữ liệu cụ thể mà nó tác động đến. Dữ liệu được lưu trữ chủ yếu trong bộ nhớ chính và trên bộ nhớ phụ, do đặc điểm khác nhau của thiết bị lưu trữ, các thuật toán tìm kiếm và sắp xếp được xây dựng cho các cấu trúc lưu trữ trên bộ nhớ chính hoặc phụ cũng có những đặc thù khác nhau. Chương này sẽ trình bày các thuật toán sắp xếp và tìm kiếm dữ liệu được lưu trữ trên bộ nhớ chính - gọi là các giải thuật *tìm kiếm*

II. Các giải thuật tìm kiếm

Có 2 giải thuật thường được áp dụng để tìm kiếm dữ liệu là *tìm kiếm tuần tự (dãy số chưa được sắp xếp)* và *tìm nhị phân.* Để đơn giản trong việc trình bày giải thuật, bài toán được đặc tả như sau:

Tập dữ liệu được lưu trữ là dãy số **a1, a2, ... ,aN**.

Giả sử chọn cấu trúc dữ liệu mảng để lưu trữ dãy số này trong bộ nhớ chính, có khai báo :

int a[N];

Lưu ý các bản cài đặt trong giáo trình sử dụng ngôn ngữ C, do đó chỉ số của mảng mặc định bắt đầu từ 0, nên các giá trị của các chỉ số có chênh lệch so với thuật toán, nhưng ý nghĩa không đổi

Khoá cần tìm là **x**, được khai báo như sau: **int x;**

III. Thuật toán tìm kiếm

1. Tìm kiếm tuần tự

- **Ý tưởng**

Tìm tuyến tính là một kỹ thuật tìm kiếm rất đơn giản và cổ điển. Thuật toán tiến hành so sánh **x** lần lượt với phần tử thứ nhất, thứ hai, ... của mảng **a** cho đến khi gặp được phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy **x.**

- **Thuật toán**

Bước 1:

i = 1; // bắt đầu từ phần tử đầu tiên của dãy

Bước 2: So sánh a[i] với x, có 2 khả năng :

a[i] = x : Tìm thấy. Dừng

a[i] != x : Sang Bước 3.

Bước 3 :

i = i+1; // xét tiếp phần tử kế trong mảng

Nếu i >N: Hết mảng,không tìm thấy.Dừng

Ngược lại: Lặp lại Bước 2.

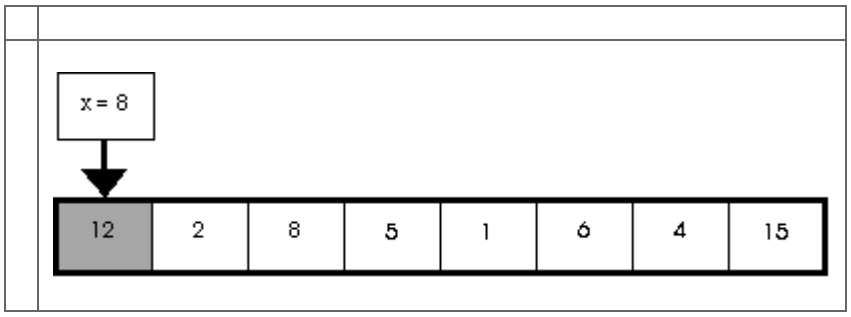
Ví dụ

Cho dãy số a:

12 2 8 5 1 6 4 15

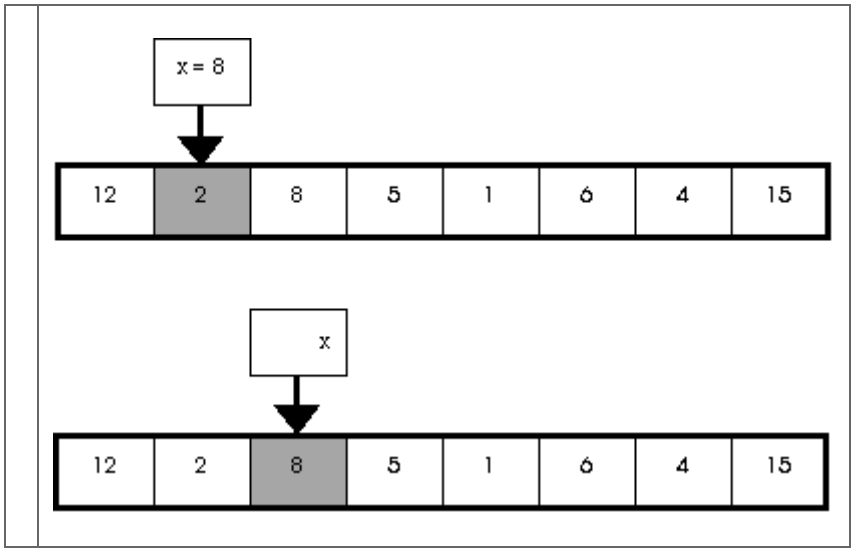
Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau :

Hình 2.2



i = 1

Hình 2.3



i = 2

i = 3

Dừng.

- **Chương trình**

```
Function TimKiem(x, N: Integer; A:Mang):Integer;
Var i:Integer;
Begin
        I:=1;
        While (I <= N) and (X<>A[I]) do I:=I+1;
        If I <= N Then Timkiem:=I Else Timkiem:=0;
End;
```

- **Đánh giá giải thuật**

Có thể ước lượng độ phức tạp của giải thuật tìm kiếm qua số lượng các phép so sánh được tiến hành để tìm ra x. Trường hợp giải thuật tìm tuyến tính, có:

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	n+1	Phần tử cuối cùng có giá trị x
Trung bình	(n+1)/2	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

Vậy giải thuật tìm tuyến tính có độ phức tạp tính toán cấp n: T(n) = O(n)

- **NHẬN XÉT**

Giải thuật tìm tuyến tính không phụ thuộc vào thứ tự của các phần tử mảng, do vậy đây là phương pháp tổng quát nhất để tìm kiếm trên một dãy số bất kỳ.

Một thuật toán có thể được cài đặt theo nhiều cách khác nhau, kỹ thuật cài đặt ảnh hưởng đến tốc độ thực hiện của thuật toán.

2. Tìm kiếm nhị phân

- Ý tưởng

Đối với những dãy số đã có thứ tự (giả sử thứ tự tăng), các phần tử trong dãy có quan hệ $a_{i-1} \leq a_i \leq a_{i+1}$, từ đó kết luận được nếu $x > a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_{i+1}, a_N]$ của dãy , ngược lại nếu $x < a_i$ thì x chỉ có thể xuất hiện trong đoạn $[a_1, a_{i-1}]$ của dãy . Giải thuật tìm nhị phân áp dụng nhận xét trên đây để tìm cách giới hạn phạm vi tìm kiếm sau mỗi lần so sánh x với một phần tử trong dãy. Ý tưởng của giải thuật là tại mỗi bước tiến hành so sánh x với phần tử nằm ở vị trí giữa của dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này để quyết định giới hạn dãy tìm kiếm ở bước kế tiếp là nửa trên hay nửa dưới của dãy tìm kiếm hiện hành. Giả sử dãy tìm kiếm hiện hành bao gồm các phần tử $a_{left} .. a_{right}$, các bước tiến hành như sau :

- **Thuật toán**

Bước 1: left = 1; right = N; // tìm kiếm trên tất cả các phần tử

Bước 2:

mid = (left+right)/2; // lấy mốc so sánh

So sánh a[mid] với x, có 3 khả năng :

a[mid] = x: Tìm thấy. Dừng

a[mid] > x: //tìm tiếp x trong dãy con $a_{left} .. a_{mid-1}$:

right =midle - 1;

a[mid] < x: //tìm tiếp x trong dãy con amid +1 .. aright :

left = mid + 1;

Bước 3:

Nếu left ≤ right //còn phần tử chưa xét ⇒ tìm tiếp.

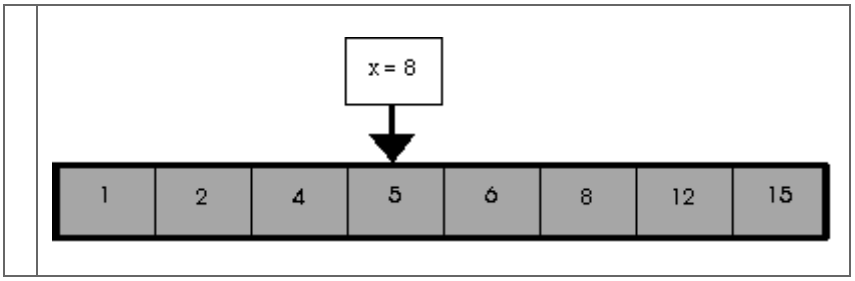
Lặp lại Bước 2.

Ngược lại: Dừng; //Đã xét hết tất cả các phần tử.

Ví dụ

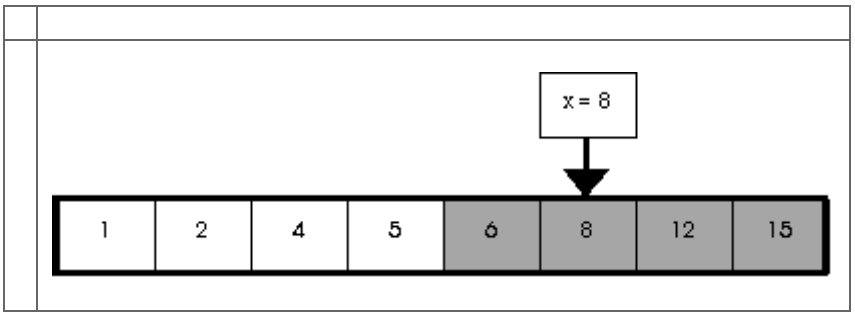
Cho dãy số a gồm 8 phần tử:

1 2 4 5 6 8 12 15



Nếu giá trị cần tìm là 8, giải thuật được tiến hành như sau:

left = 1, right = 8, midle = 4



left = 5, right = 8, midle = 6

Dừng.

- **Chương trình**

```
Function TimKiemNhiPhan(X, N: Integer; A: Mang):Integer;
Var dau,cuoi,giua:Integer;
    Found:Boolean;
Begin
        dau:=1; {điểm mút trái của khoảng tìm kiếm}
        cuoi:=N; {điểm mút phải của khoảng tìm kiếm}
        Found:=False; {chưa tìm thấy}
        While (dau <=cuoi) and (Not Found) Do
        Begin
            giua:=(dau + cuoi) Div 2;
            If X = A[giua] Then Found:=True {đã tìm thấy}
            Else
                If X > A[giua] Then dau:=giua+1
                Else cuoi:=giua-1;
        End;

    If Found Then TimKiemNhiPhan:= giua Else TimKiemNhiPhan:=0;
End;
```

- **Đánh giá giải thuật**

Trường hợp giải thuật tìm nhị phân, có bảng phân tích sau :

Trường hợp	Số lần so sánh	Giải thích
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 n/2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

Vậy giải thuật tìm nhị phân có độ phức tạp tính toán cấp n: $T(n) = O(\log_2 n)$

- **NHẬN XÉT**

Giải thuật tìm nhị phân dựa vào quan hệ giá trị của các phần tử mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được cho những dãy đã có thứ tự.

Giải thuật tìm nhị phân tiết kiệm thời gian hơn rất nhiều so với giải thuật tìm tuyến tính do $T_{\text{nhị phân}}(n) = O(\log_2 n) < T_{\text{tuần tự}}(n) = O(n)$.Tuy nhiên khi muốn áp dụng giải thuật tìm nhị phân cần phải xét đến thời gian sắp xếp dãy số để thỏa điều kiện dãy số có thứ tự. Thời gian này không nhỏ, và khi dãy số biến động cần phải tiến hành sắp xếp lại . Tất cả các nhu cầu đó tạo ra khuyết điểm chính cho giải thuật tìm nhị phân. Ta cần cân nhắc nhu cầu thực tế để chọn một trong hai giải thuật tìm kiếm trên sao cho có lợi nhất