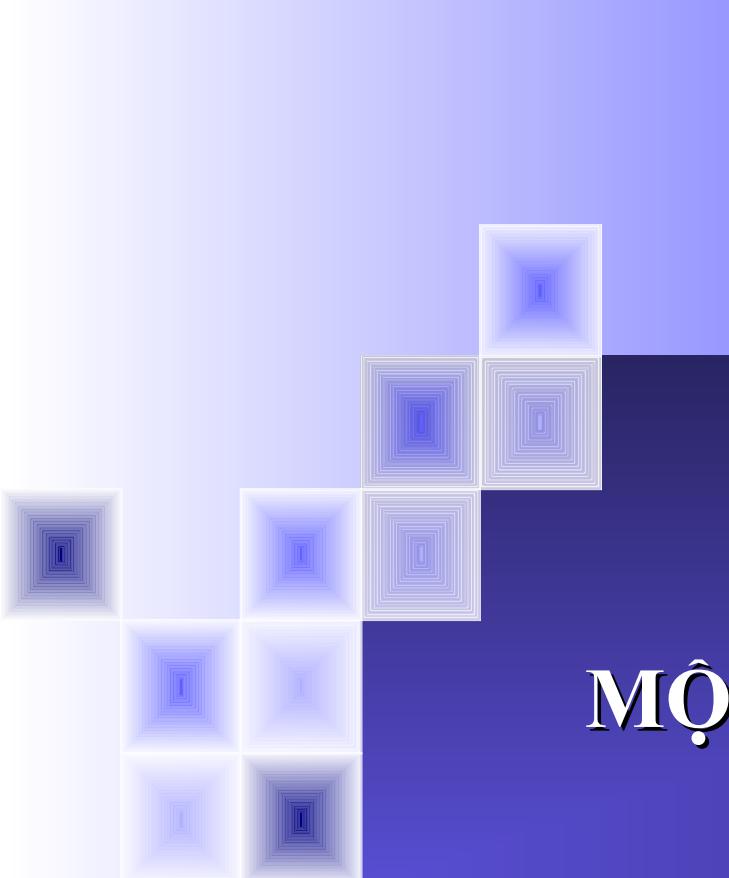




# PHÂN TÍCH VÀ THIẾT KẾ THUẬT TOÁN

*Dành cho SV ngành CNTT*



## Chương 2

# MỘT SỐ THUẬT TOÁN CƠ BẢN

# Nội dung

- Các thuật toán sắp xếp sơ cấp
  - SX chèn
  - SX chọn
  - SX nổi bọt
  - SX đếm
- Các thuật toán tìm kiếm sơ cấp
  - Tìm kiếm tuần tự
  - Tìm kiếm nhị phân

# 1. Các thuật toán sắp xếp sơ cấp

- **Bài toán SX**: SX là quá trình xử lý một danh sách các phần tử (mẩu tin) để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên thông tin lưu trữ của các phần tử
- Thông thường, tiêu chuẩn để SX là trường khóa (key)
- Để thuận tiện cho việc trình bày cũng như tập trung vào thuật toán ta sẽ sắp xếp mảng số nguyên theo thứ tự tăng.

# 1.1 SX chèn

## ➤ Mô tả 1:

For i:=2 to n do

*chèn A[i] vào vị trí thích hợp trong các phần tử A[1],  
..., A[i-1]*

## ➤ Chi tiết hơn:

- ❑ *Dãy ban đầu  $a_1, a_2, \dots, a_n$ , xem như đã có đoạn gồm một phần tử  $a_1$  đã được sắp.*
- ❑ *Thêm  $a_2$  vào đoạn  $a_1$  sẽ có đoạn  $a_1 a_2$  được sắp*
- ❑ *Thêm  $a_3$  vào đoạn  $a_1 a_2$  để có đoạn  $a_1 a_2 a_3$  được sắp*
- ❑ *Tiếp tục cho đến khi thêm xong  $a_N$  vào đoạn  $a_1 a_2 \dots a_{N-1}$  sẽ có dãy  $a_1 a_2 \dots a_N$  được sắp.*

# 1.1 SX chèn (tt)

➤ Tìm vị trí chèn j, với  $1 \leq j \leq i$ :

- Ở bước thứ i ta có các phần tử từ A[1] đến A[i-1] đã được sắp thứ tự
- Để chèn A[i] vào vị trí thích hợp trong các phần tử A[1],..., A[i-1] ta sẽ tìm vị trí j nhỏ nhất thỏa mãn A[i] < A[j] và chèn A[i] vào vị trí j

# 1.1 SX chèn (tt)

## ➤ Mô tả 2:

Procedure SXChen(A[1..n])

For i:=2 to n do

j:=i-1;

x:=A[i];

While (j>0) and (x<A[j]) do

j:=j-1; //j là vị trí cần chèn

// Dịch chuyển

For h:=i downto j-1 do A[h]:=A[h-1];

//Thực hiện chèn

A[j]:=x;

# 1.1 SX chèn (tt)

➤ Mô tả 3: Trong quá trình tìm vị trí j ta đồng thời dịch các phần tử lớn hơn A[i] sang phải một vị trí

Procedure SXChen(A[1..n])

For i:=2 to n do

j:=i-1;      x:=A[i];

While (j>0) and (x<A[j]) do

A[j]:= A[j-1]      //A[j+1]:= A[j];

j:=j-1;

A[j+1] :=x;            //Thực hiện chèn

# 1.1 SX chèn (tt)

Ví dụ:

$$A = (3, 6, 2, 8, 4, 5)$$

i=2    x=6    j=1    3, 6, 2, 8, 4, 5

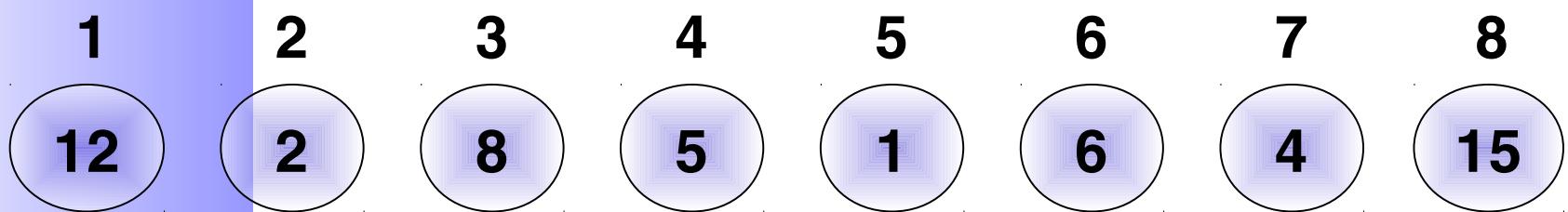
i=3    x=2    j=0    2, 3, 6, 8, 4, 5

i=4    x=8    j=3    2, 3, 6, 8, 4, 5

i=5    x=4    j=2    2, 3, 4, 6, 8, 5

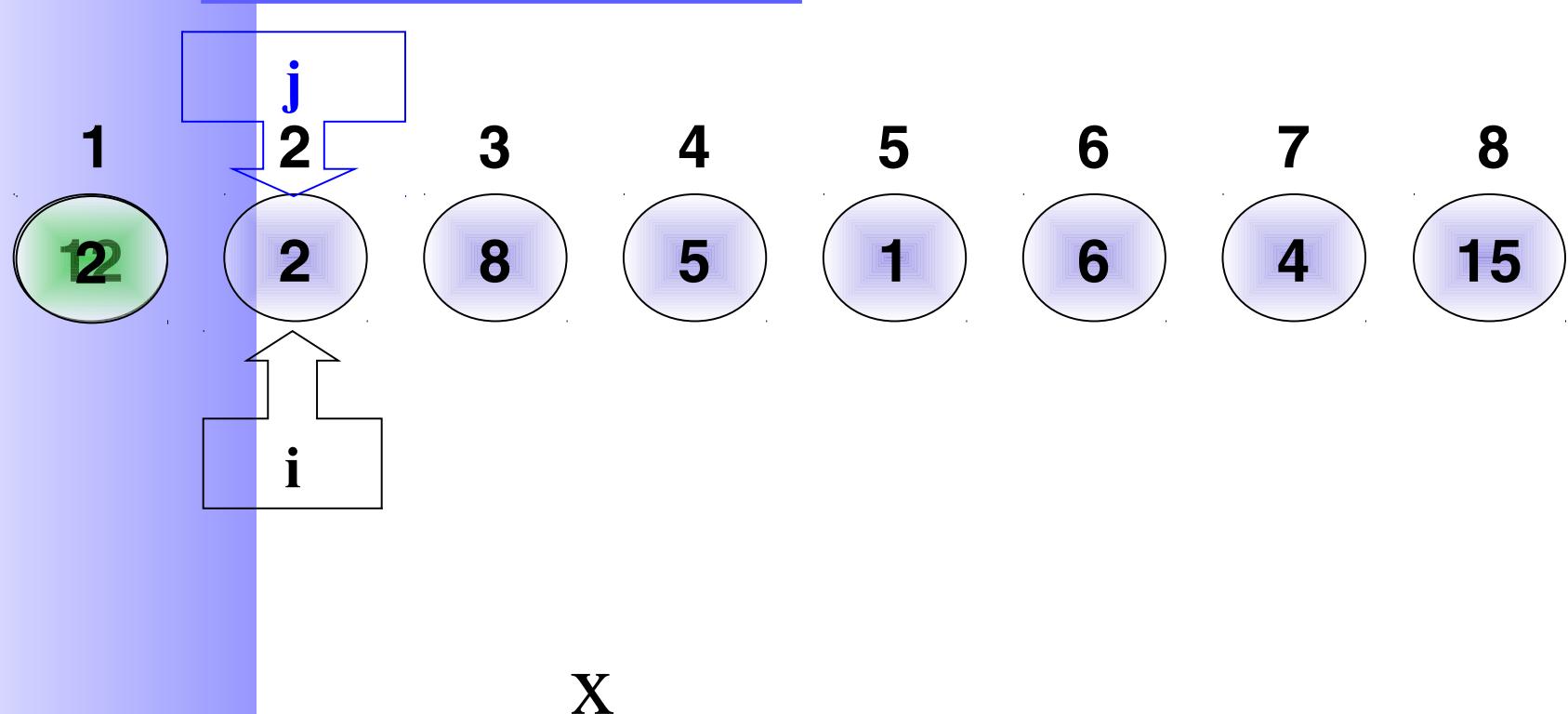
i=6    x=5    j=3    2, 3, 4, 5, 6, 8

# *1.1 Insertion Sort – Minh họa*



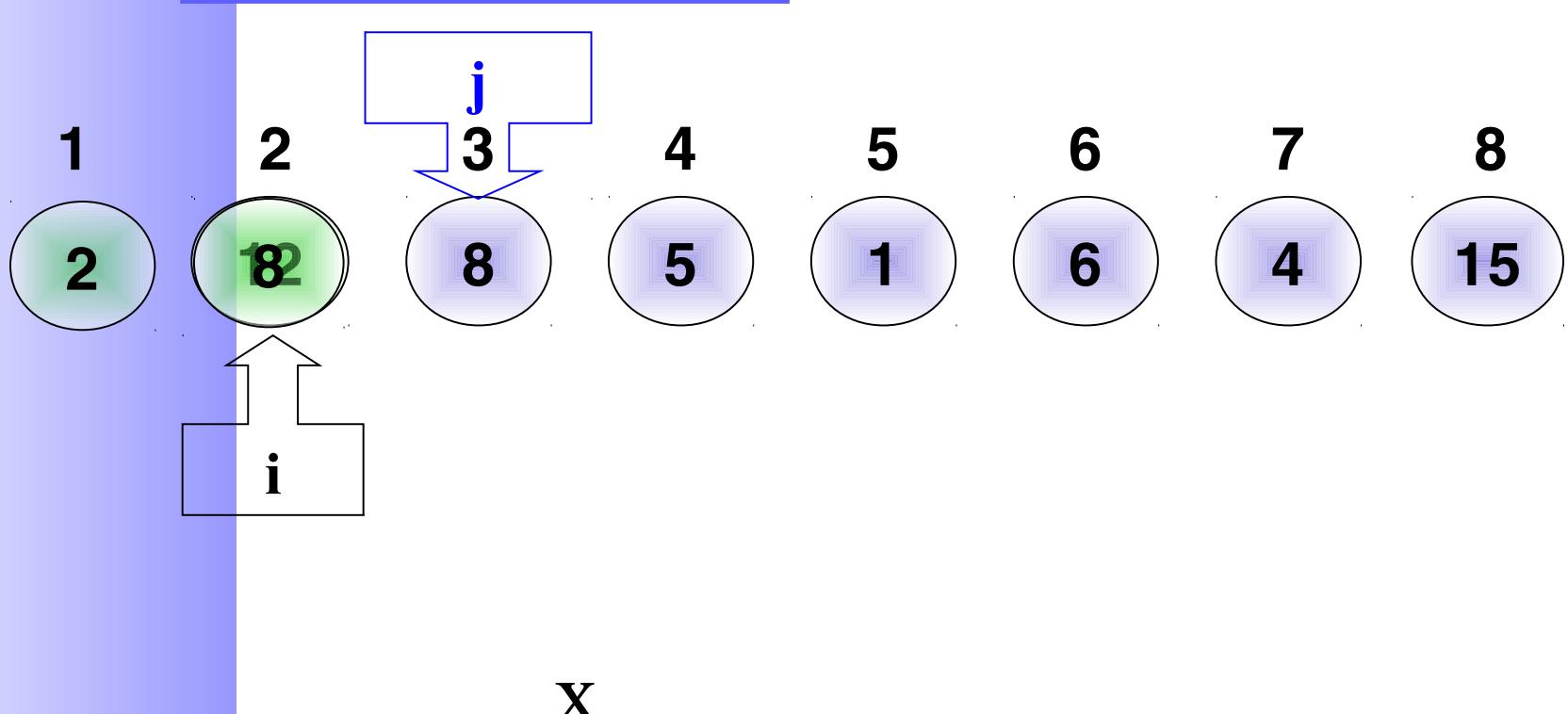
# 1.1 Insertion Sort – Minh họa

Insert  $a_2$  into (1, 2)



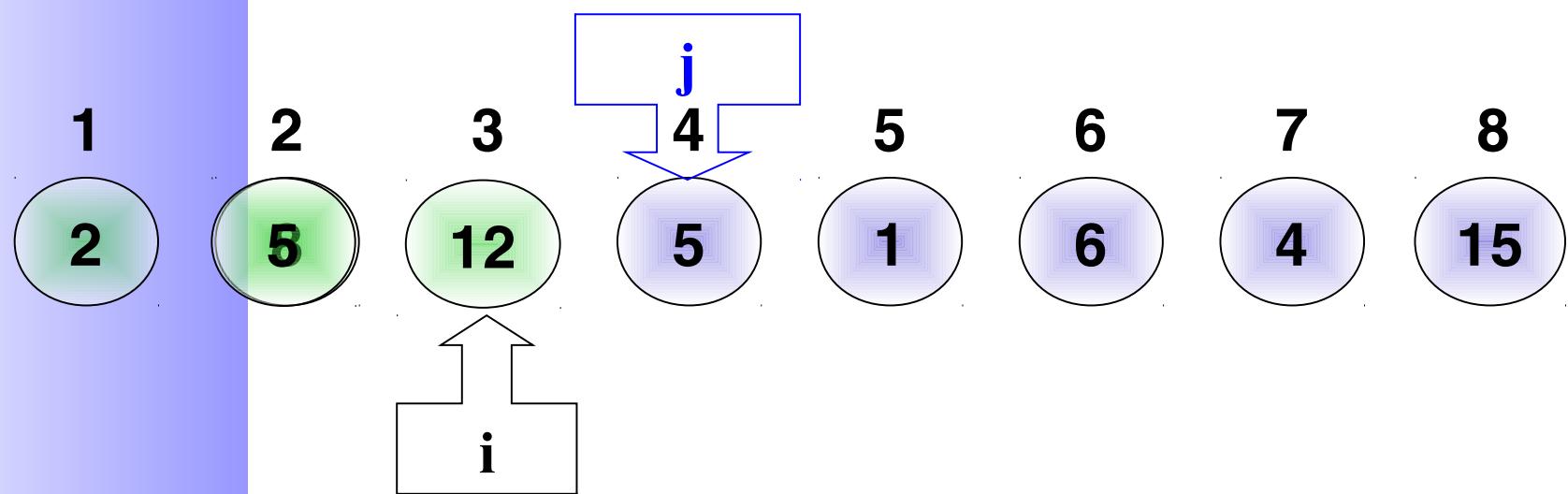
# 1.1 Insertion Sort – Minh họa

Insert  $a_3$  into (1, 3)



# 1.1 Insertion Sort – Minh họa

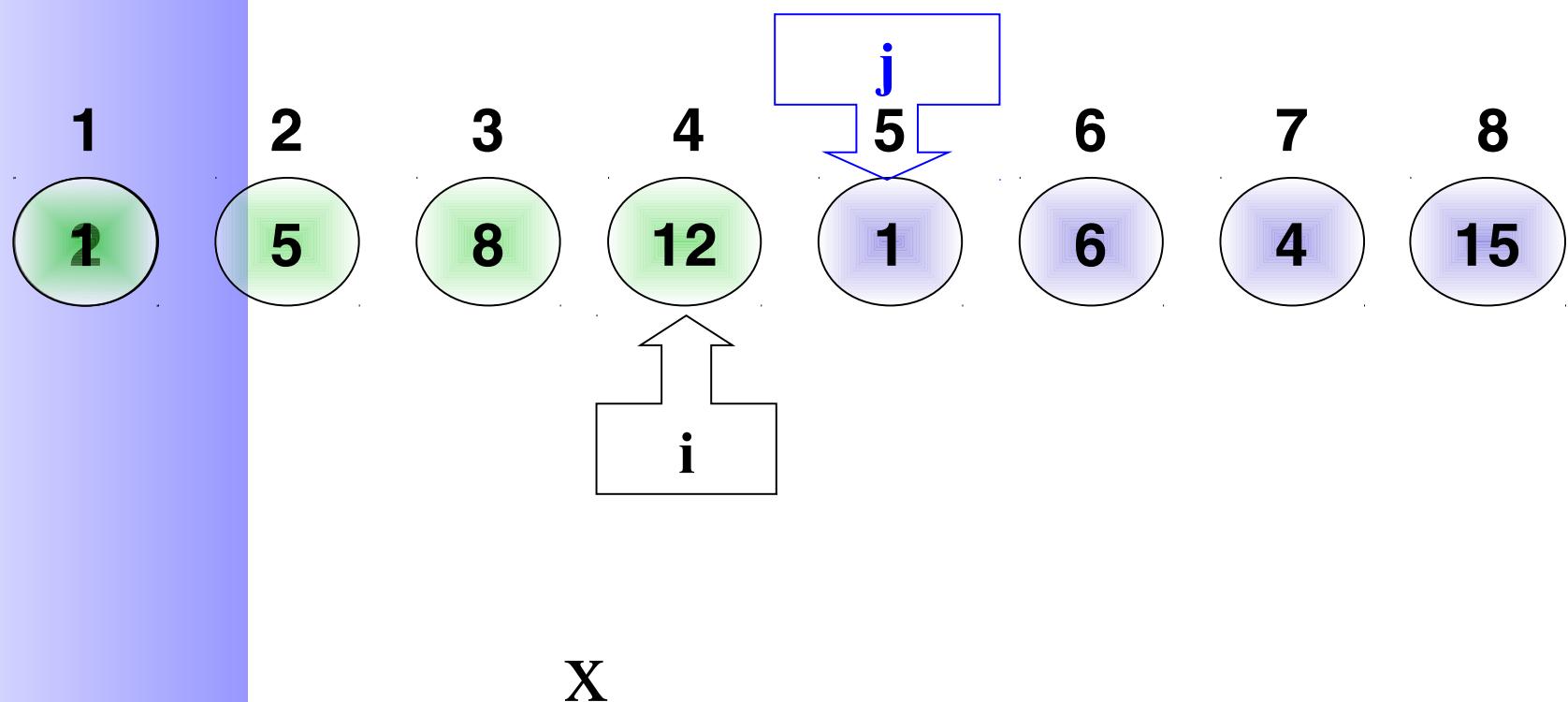
Insert  $a_4$  into (1, 4)



X

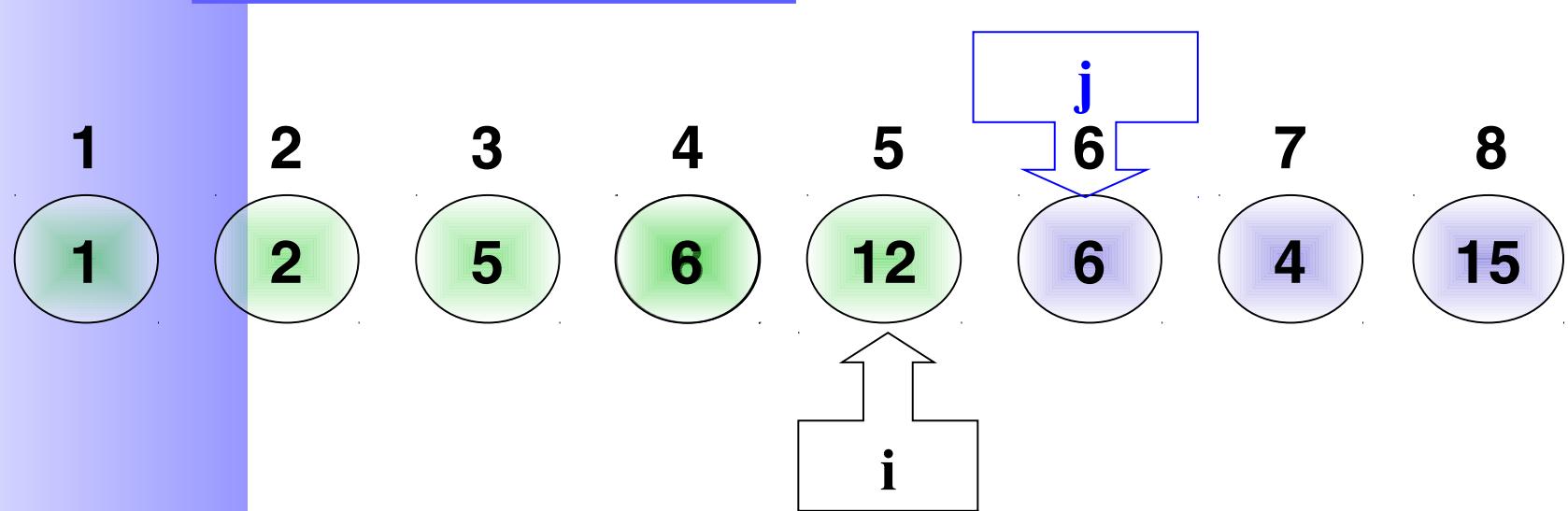
# 1.1 Insertion Sort – Minh họa

Insert  $a_5$  into (1, 5)



# 1.1 Insertion Sort – Minh họa

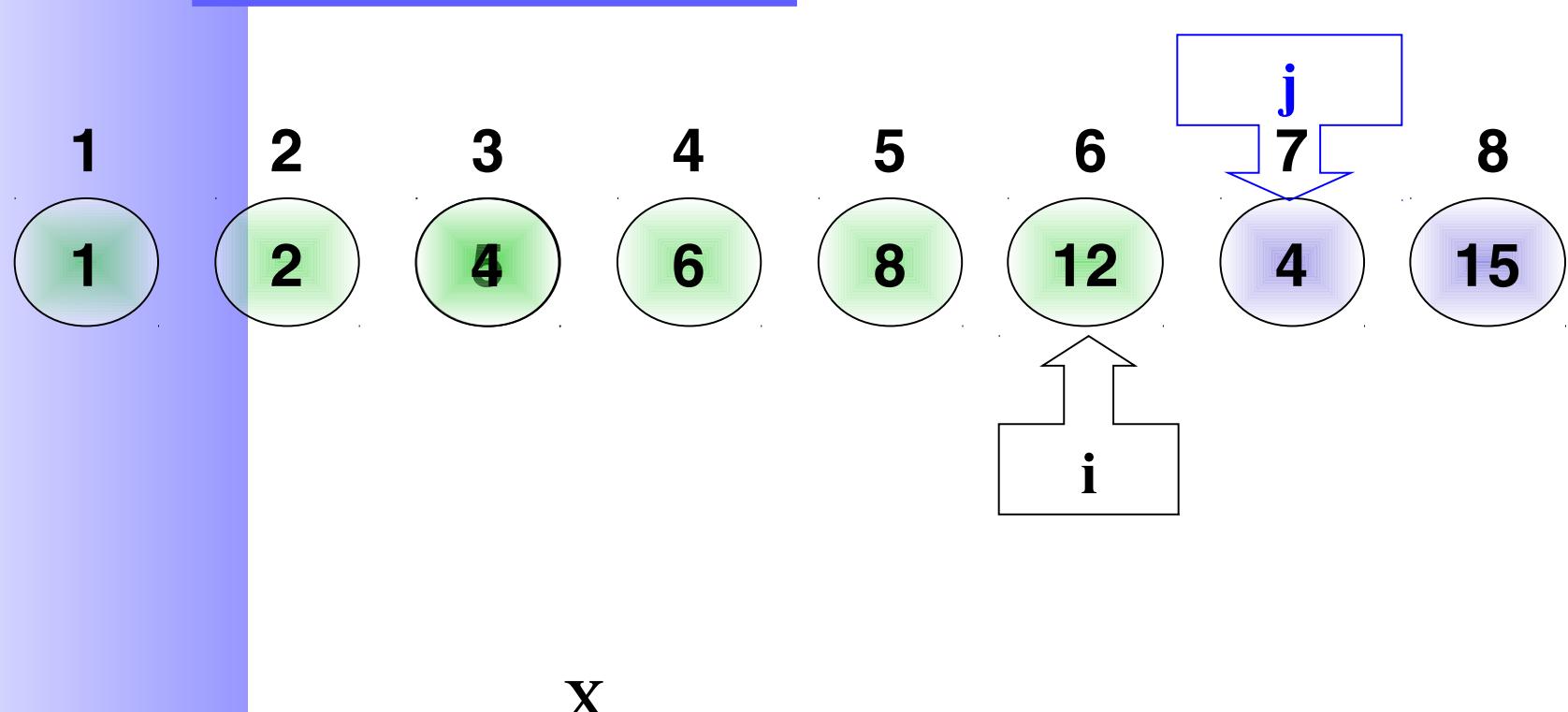
Insert  $a_6$  into (1, 6)



X

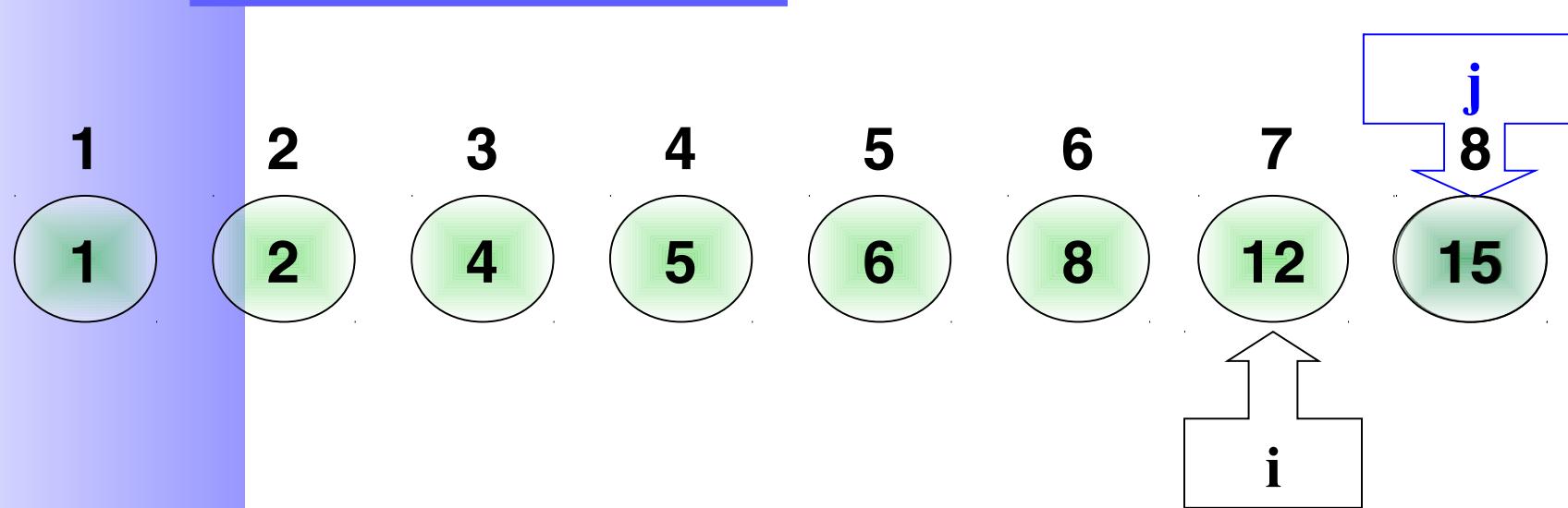
# 1.1 Insertion Sort – Minh họa

Insert  $a_7$  into (1, 7)

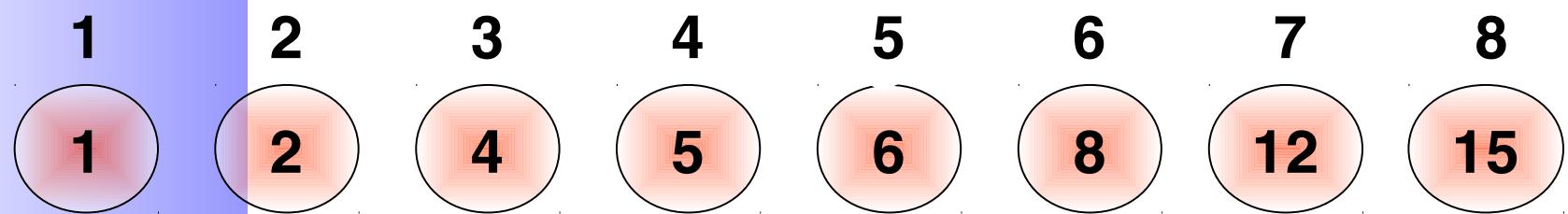


# 1.1 Insertion Sort – Minh họa

Insert  $a_8$  into (1, 8)



# *1.1 Insertion Sort – Minh họa*



## 1.2 SX Chọn

For i:=1 to n-1 do

*Chọn phần tử nhỏ nhất chưa đúng vị trí và đặt nó vào vị trí i*

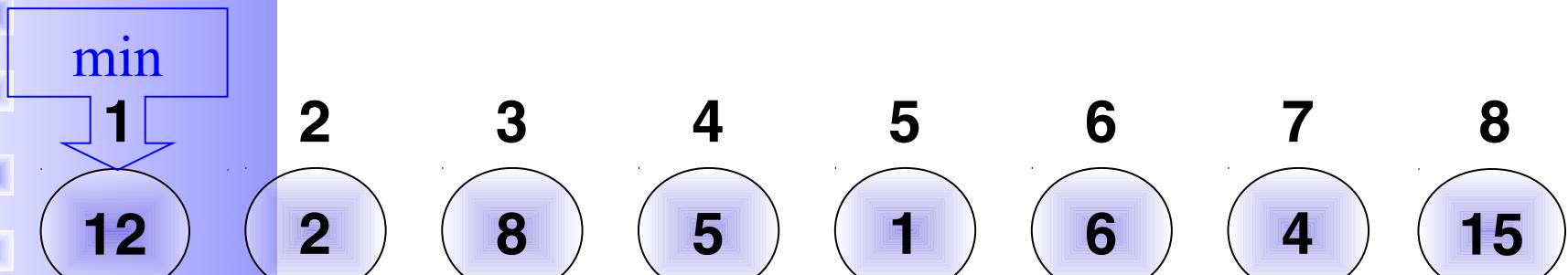
### Mô tả chi tiết:

- Bước 1 : i = Vị trí đầu;
- Bước 2 : Tìm phần tử  $a[min]$  nhỏ nhất trong dãy hiện hành từ  $a[i]$  đến  $a[N]$
- Bước 3 : Nếu  $min \neq i$ : Hoán vị  $a[min]$  và  $a[i]$
- Bước 4 : Nếu i chưa là Vị trí cuối
  - ❑ i = Vị trí kế(i);
  - ❑ Lặp lại Bước 2
- Ngược lại: Dừng. //N phần tử đã nằm đúng vị trí.

# 1.2 Selection sort – Minh họa

Find MinPos(1, 8)

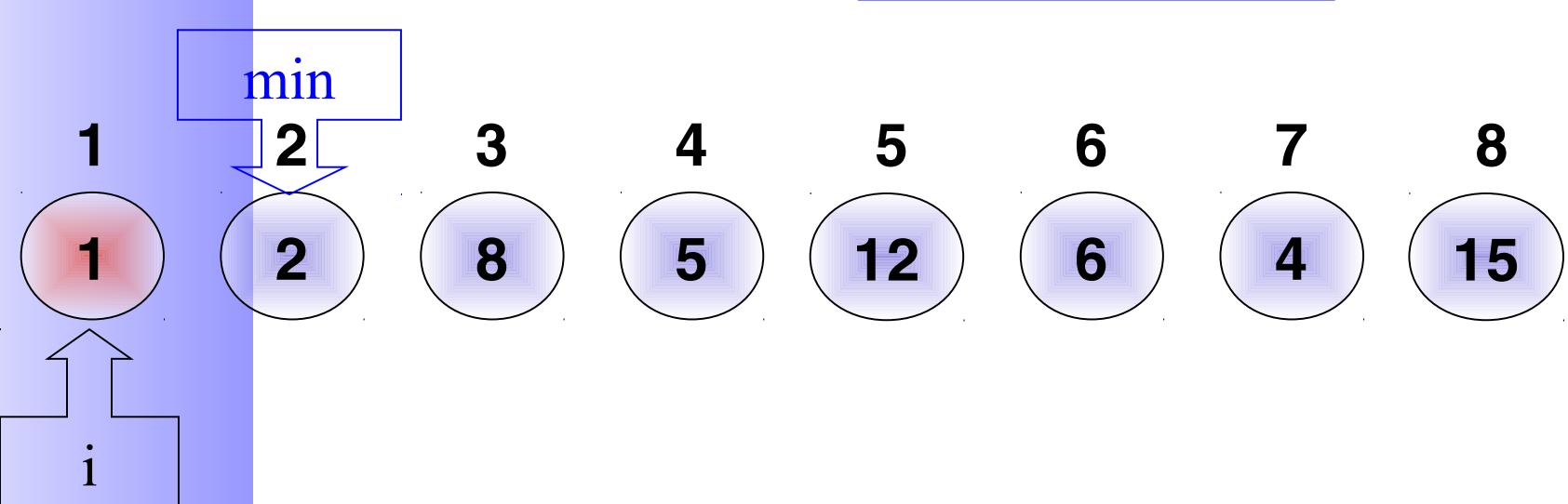
Swap( $a_i$ ,  $a_{min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(2, 8)

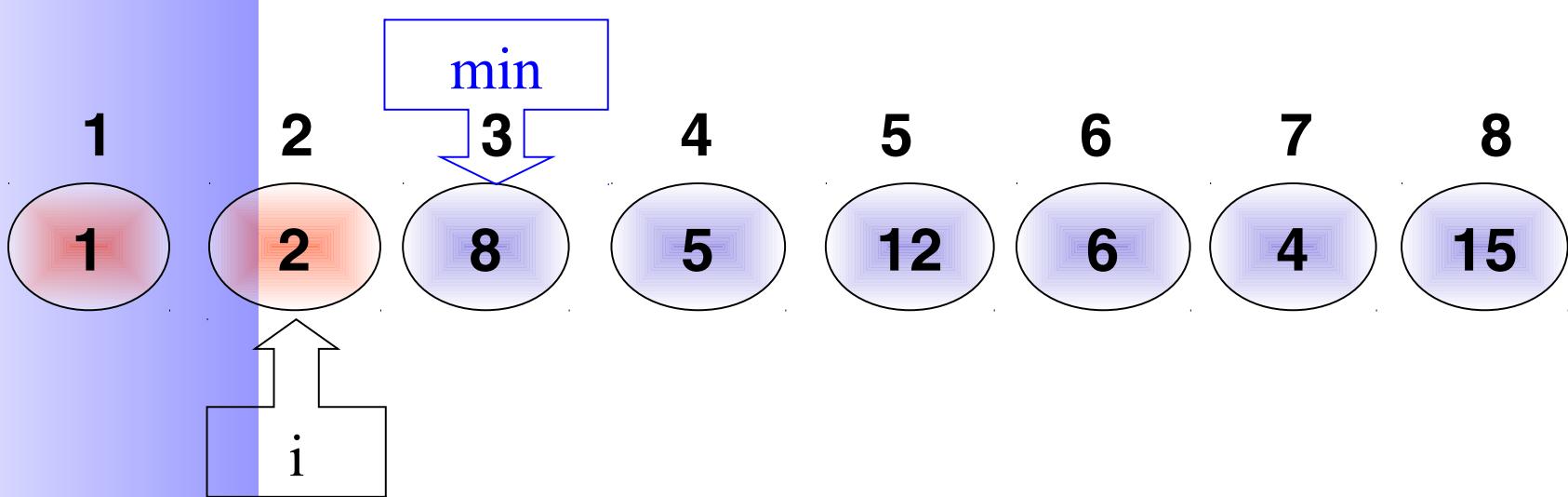
Swap( $a_i$ ,  $a_{min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(3, 8)

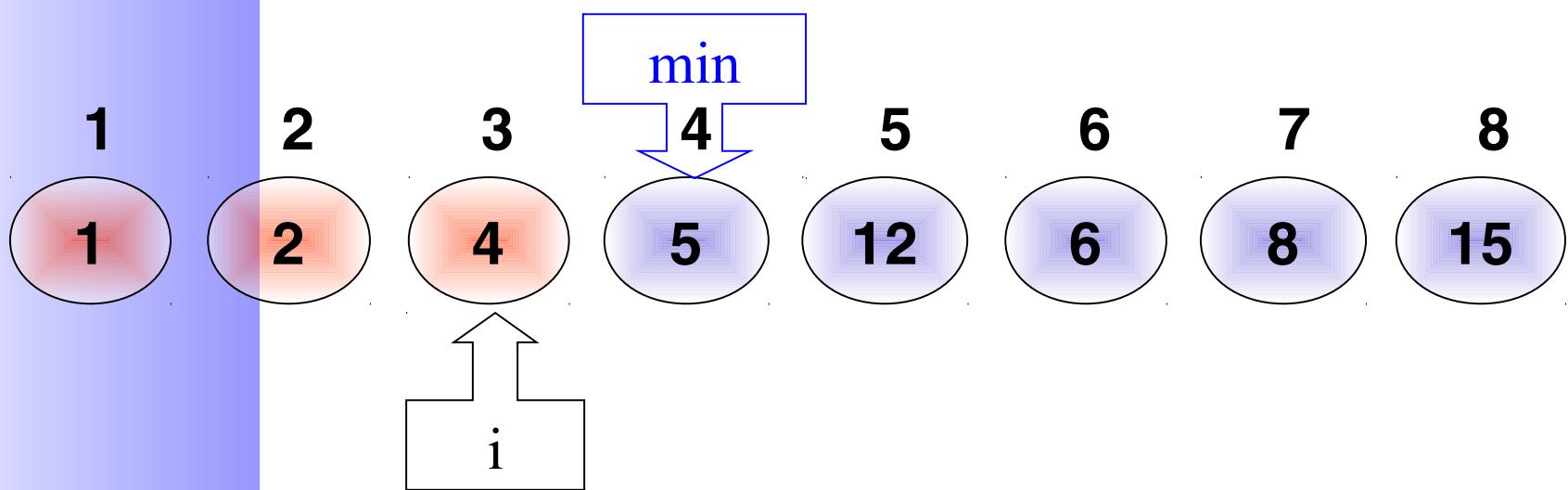
Swap( $a_i$ ,  $a_{\min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(4, 8)

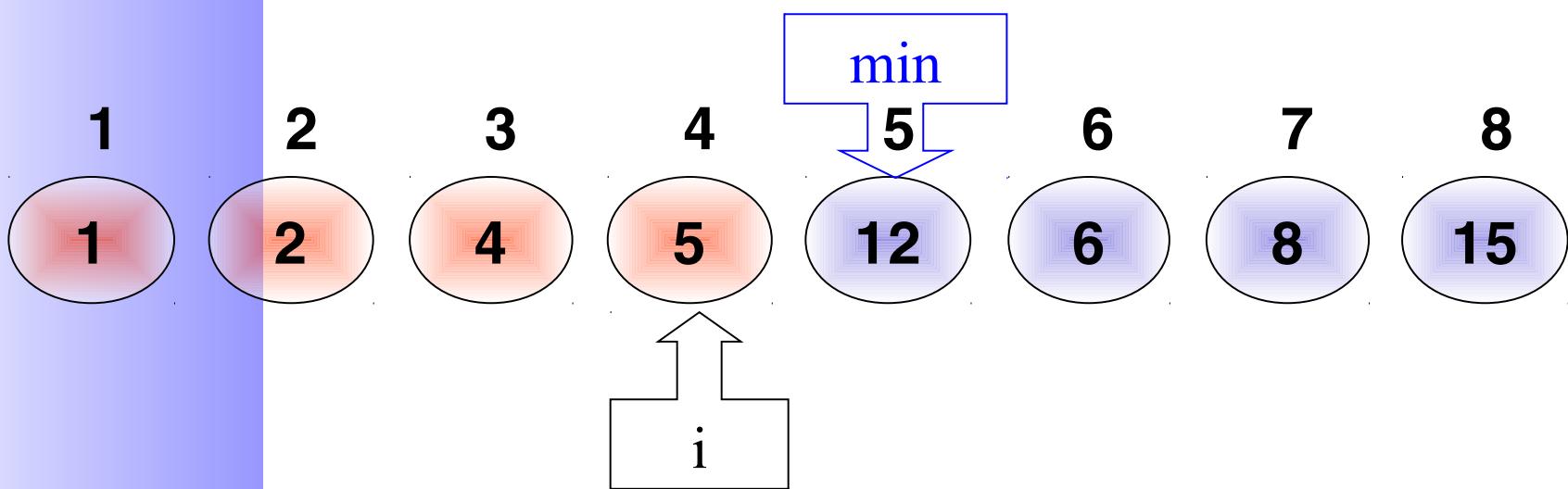
Swap( $a_i$ ,  $a_{min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(5, 8)

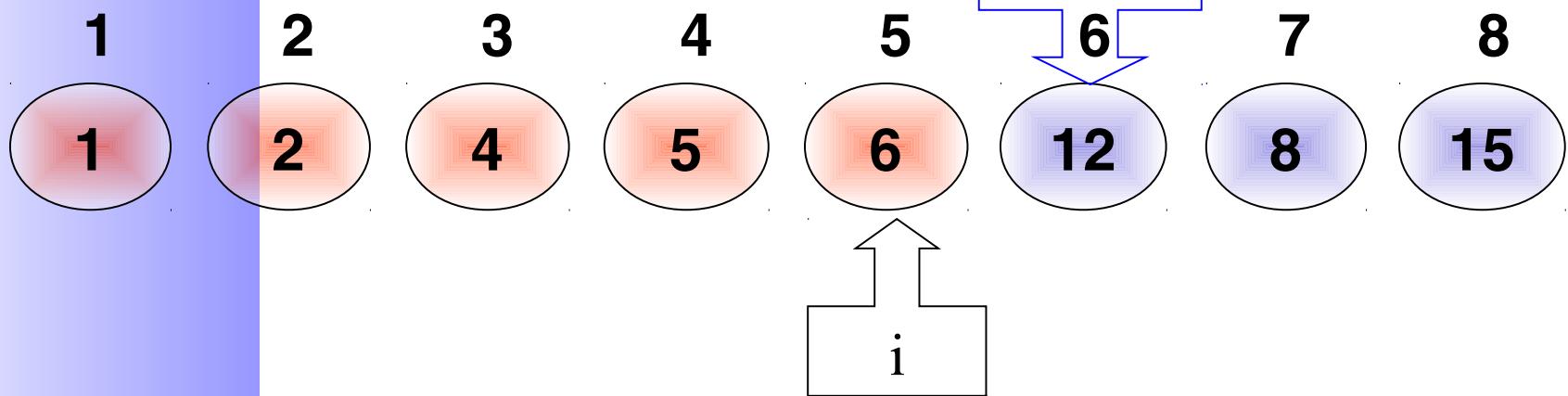
Swap( $a_i$ ,  $a_{\min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(6, 8)

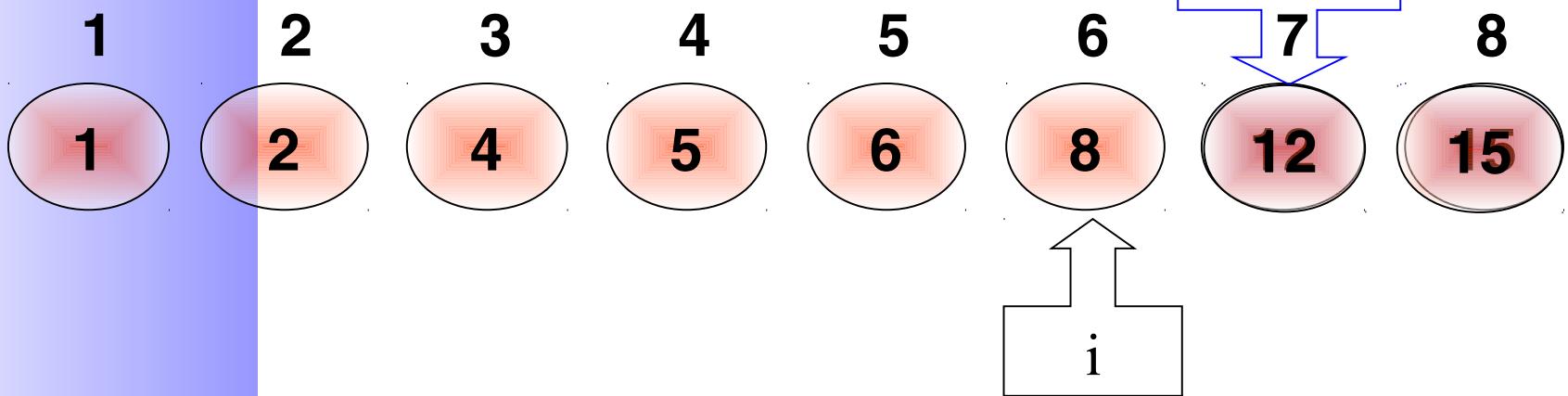
Swap( $a_i$ ,  $a_{\min}$ )



# 1.2 Selection sort – Minh họa

Find MinPos(7, 8)

Swap( $a_i$ ,  $a_{\min}$ )



## 1.2 SX Chọn (tt)

Procedure SXChon;

For i:=1 to n-1 do

    min:=i;

    For j:=i+1 to n do

        If A[j] < A[min] then

            min:= j;

{phân tử tại vị trí min là nhỏ nhất chưa được sắp}

Swap(A[i], A[min])

# 1.3 SX nổi bọt

- Ý tưởng: Ta sẽ so sánh và đổi chỗ của hai phần tử liền nhau đúng không đúng vị trí cho tới khi tất cả các phần tử được sắp thứ tự
- Mô tả chi tiết:

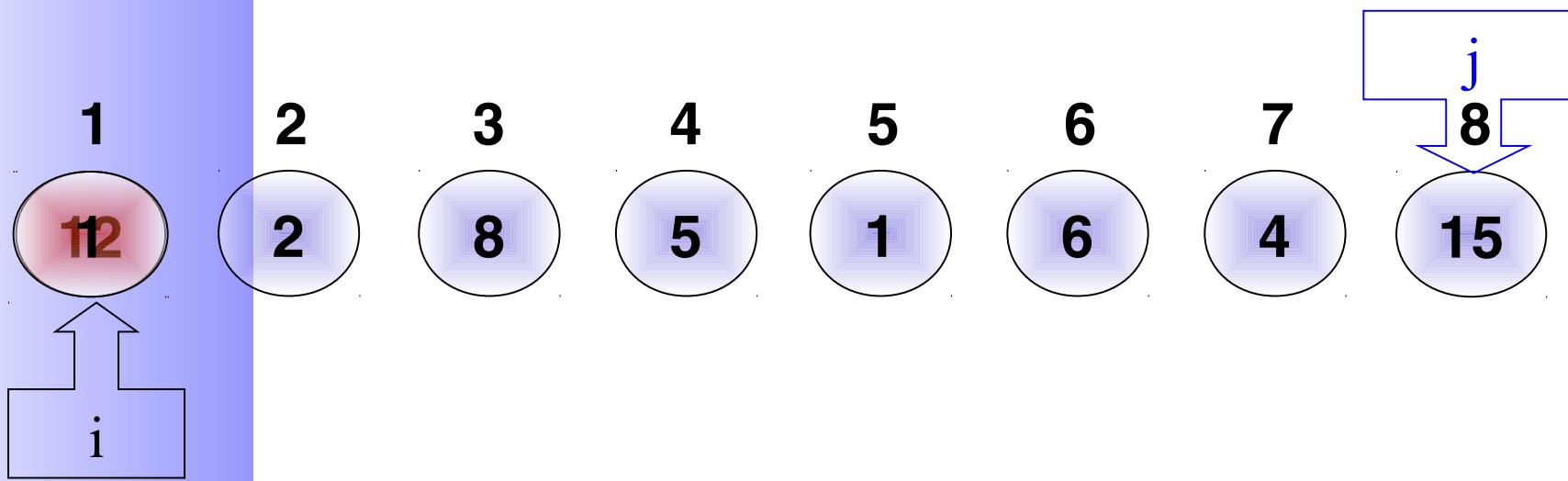
Procedure SXNoibot;

For i:=1 to n-1 do

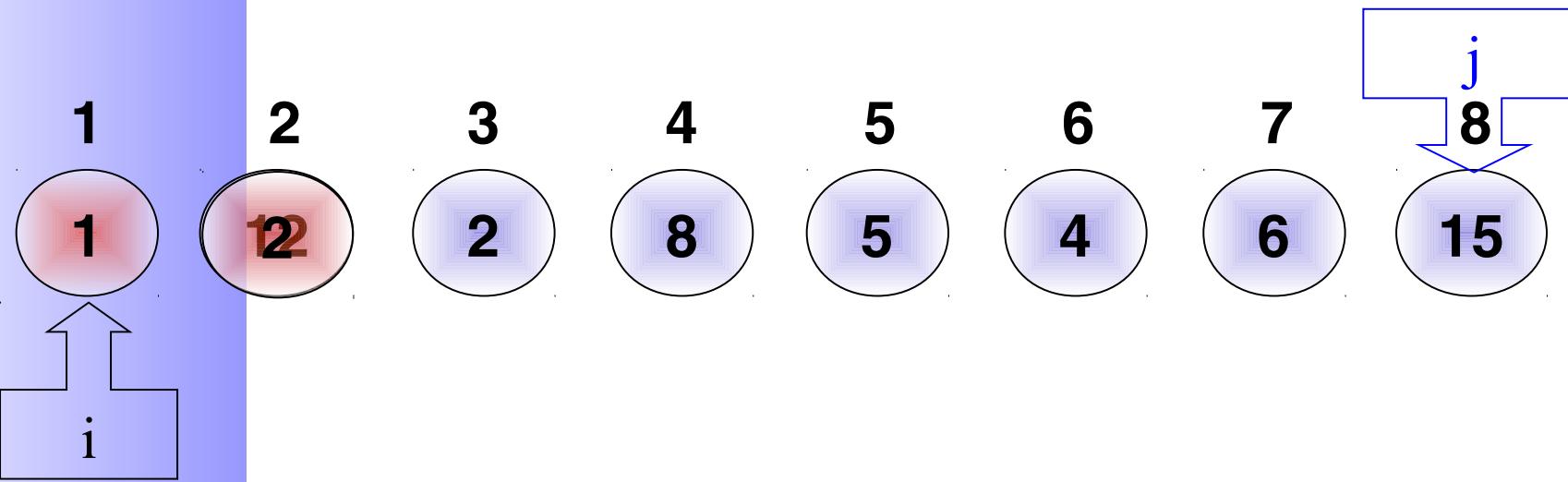
For j:=n downto i+1 do

If A[j] < A[j-1] then Swap(A[j-1], A[j])

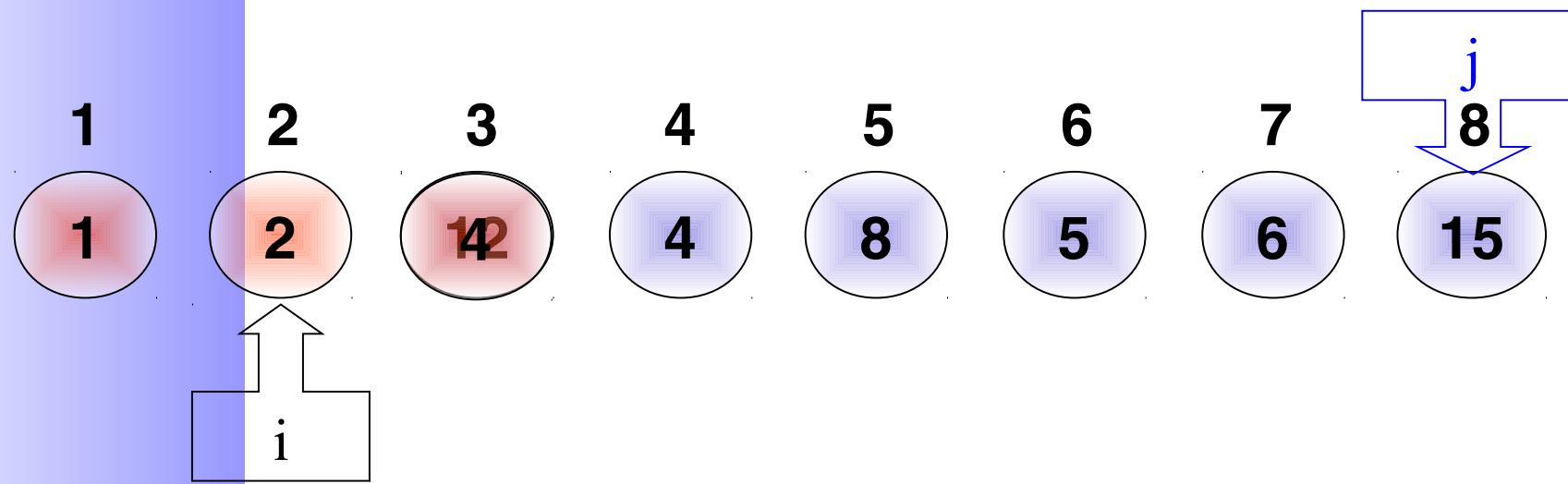
# *1.3 Bubble Sort – Ví dụ*



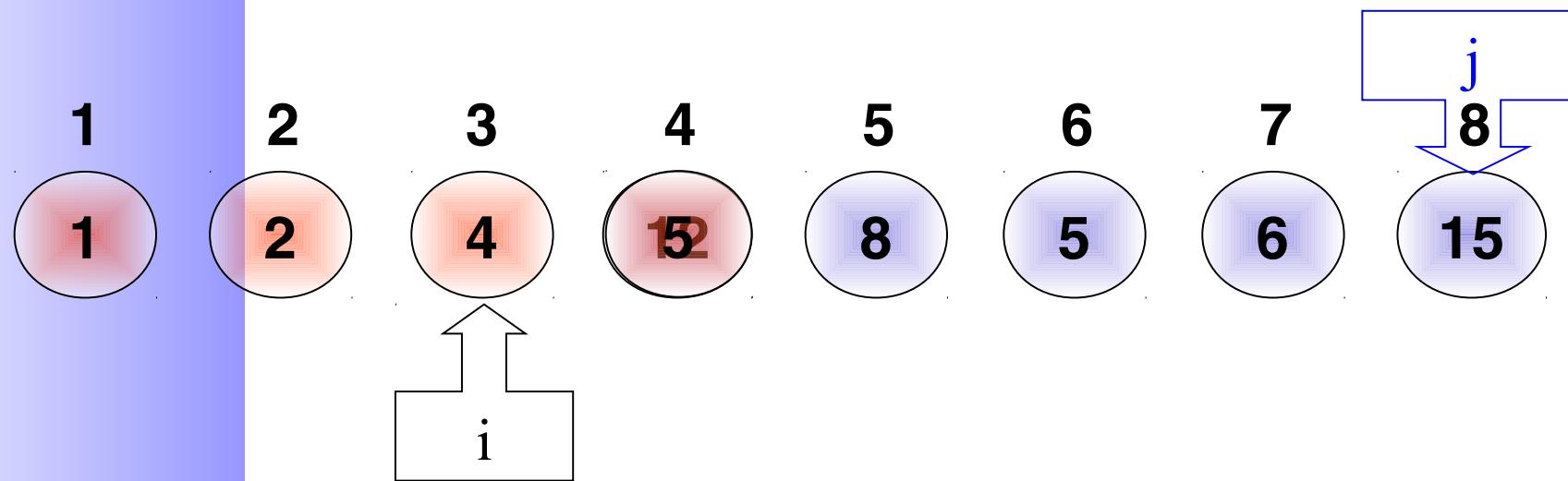
# 1.3 Bubble Sort – Ví dụ



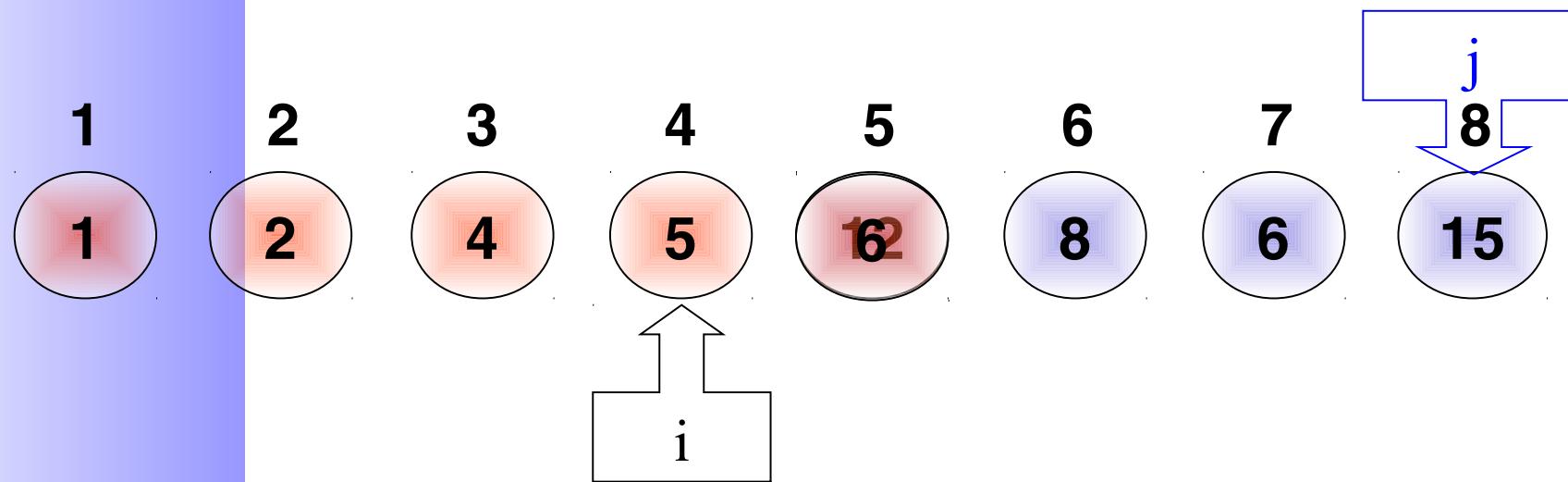
# 1.3 Bubble Sort – Ví dụ



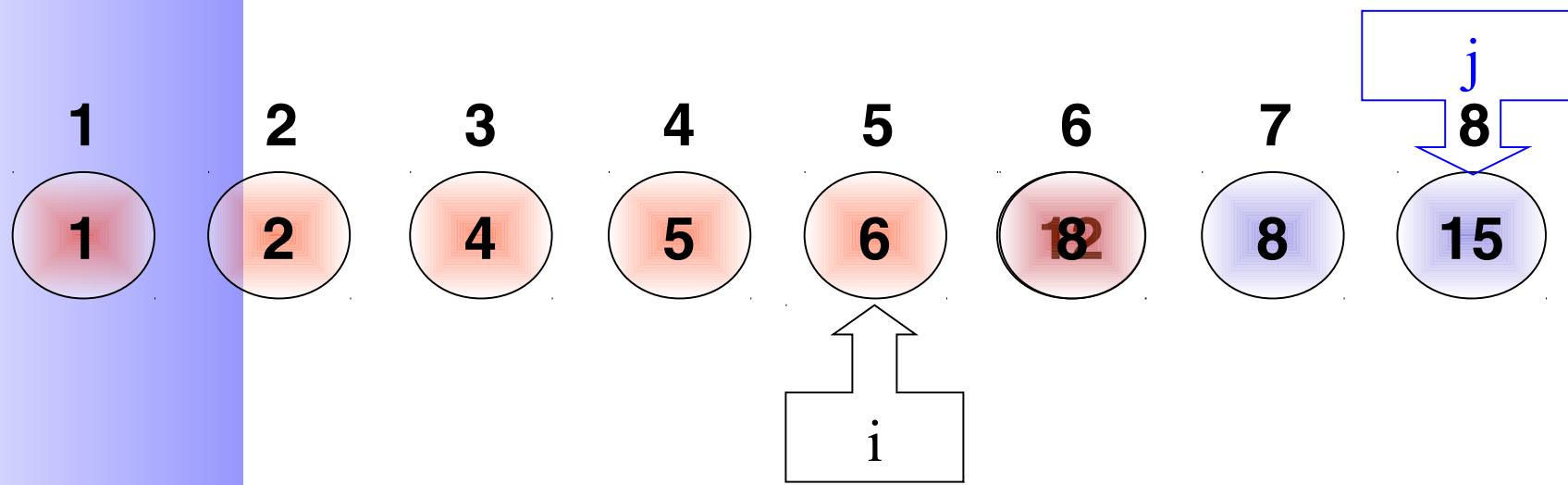
# 1.3 Bubble Sort – Ví dụ



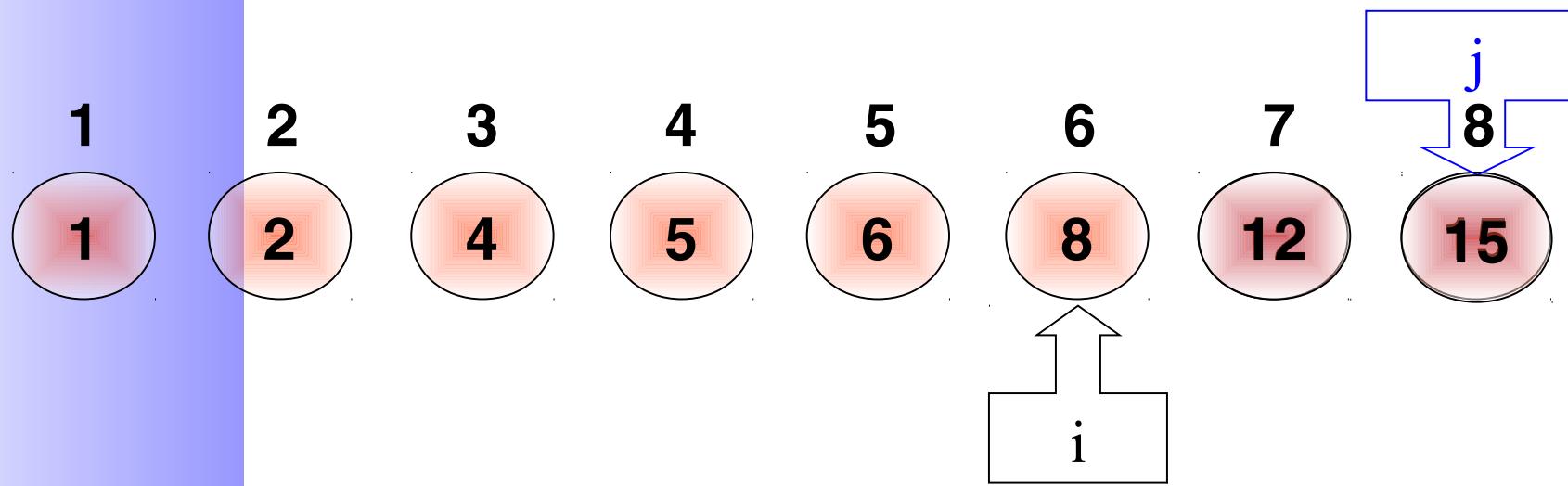
# 1.3 Bubble Sort – Ví dụ



# 1.3 Bubble Sort – Ví dụ



# 1.3 Bubble Sort – Ví dụ



# 1.3 Bubble sort – minh họa 2

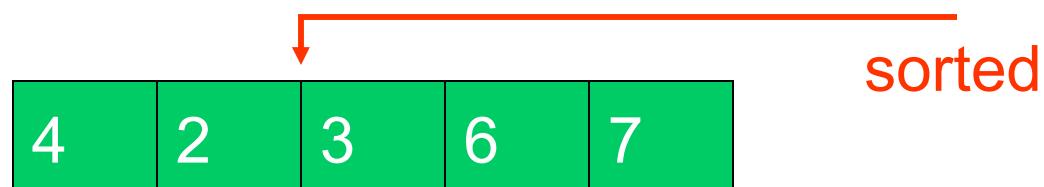
Bước 1



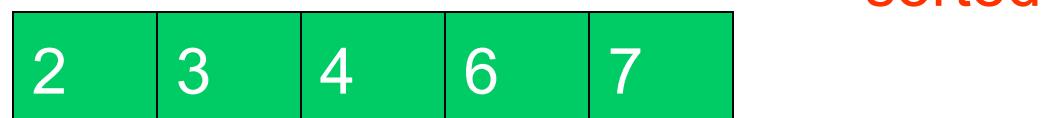
Bước 2



Bước 3



Bước 4



# 1.4 SX đếm

## ➤ Ý tưởng:

- ❑ Đếm số phần tử nhỏ hơn hoặc bằng  $A[i]$
- ❑ Nếu có  $j$  phần tử nhỏ hơn hoặc bằng  $A[i]$  thì  $A[i]$  sẽ có vị trí thứ  $j+1$  trong dãy đã sắp thứ tự.

## ➤ Chi tiết:

- ❑ Khởi tạo  $\text{đếm}[i] = 0$  với  $i$  chạy từ 1 đến  $n$
- ❑ For  $i:=1$  to  $n$  do
  - $\text{đếm}[i]:= \text{số phần tử bé hơn hoặc bằng } a[i]$
- ❑ For  $i:=1$  to  $n$  do  $B[\text{đếm}[i]+1]:=a[i]$
- ❑ Return  $B$

# 1.4 SX đếm (tt)

➤ Mô tả 2:

Procedure Sxdem;

    For i:=1 to n do Count[i]:=0;

    For i:=n downto 2 do

        For j:=i-1 downto 1 do

            If a[i] < a[j] then Count[j]:=Count[j]+1

            Else Count[i]:=Count[i]+1;

    For i:=1 to n do s[Count[i]+1]:=a[i];

    Return s;

➤ Đánh giá độ phức tạp:

## 1.5 Bài tập

1. Đánh giá độ phức tạp trong trường hợp xấu nhất của các thuật toán sắp xếp sơ cấp đã nêu.
2. Trong các thuật toán sắp xếp sơ cấp trên, thuật toán nào có tính ổn định? Giải thích.
3. Chạy từng bước các thuật toán sắp xếp đã nêu trên các mảng cụ thể có kích thước 10.
4. Mô tả một thuật toán thích hợp sắp xếp một mảng các bit. Cho biết độ phức tạp của thuật toán được sử dụng.

## 2. Tìm kiếm

➤ **Bài toán:** Tìm một phần tử  $x$  có trong mảng  $A[1..n]$  hay không?

*Tổng quát hơn:* Tìm một phần tử trong mảng  $A[1..n]$  có tính chất  $P$  hay không?

➤ **Ví dụ:**

Cho một mảng các số nguyên

Tìm xem trong mảng:

- Có số nguyên  $x$  hay không?
- Có số nguyên tố nào không?
- Có số nào chia hết cho 7 không?

## 2.1 Tìm tuần tự

- Mô tả 1: Duyệt (so sánh mỗi phần tử mảng với T) toàn bộ mảng cho tới khi gặp một phần tử bằng T hoặc đã duyệt hết mảng
- Mô tả 2:
  - ❑ Bước 1:  $i = \text{Vị trí đầu};$
  - ❑ Bước 2: Nếu  $a[i] = x : \text{Tìm thấy. Dừng, vị trí xuất hiện: } i$
  - ❑ Bước 3 :  $i = \text{Vị trí kế}(i); // \text{xét tiếp phần tử kế trong mảng}$
  - ❑ Bước 4: Nếu  $i > \text{Vị trí cuối: } // \text{Hết mảng}$   
                    Không tìm thấy. Dừng.

Ngược lại: Lặp lại Bước 2.

## 2.1 Tìm tuần tự (tt)

5

Target key

position = 2

0	1	2	3	4	5	6	7
7	13	5	21	6	2	8	15



return success

Số lần so sánh: 3

## 2.1 Tìm tuần tự (tt)

9

Target key

0	1	2	3	4	5	6	7
7	13	5	21	6	2	8	15



return not\_present

Số lần so sánh: 8

## 2.1 Tìm tuần tự (tt)

➤ Mô tả 3:

Procedure TimTT(A,T);

    Found:= False;

    For i:=1 to n do

        If A[i]=T then

            Found:=True;

            Exit;

➤ Cách khác: dùng vòng lặp không xác định

➤ Dùng vòng lặp không xác định với “*lính canh*”

- ❑ Đặt thêm một phần tử có giá trị x vào cuối mảng
- ❑ Bảo đảm luôn tìm thấy x trong mảng
- ❑ Sau đó dựa vào vị trí tìm thấy để kết luận.

## 2.2 Tìm nhị phân

- **Bài toán:** Tương tự bài toán tìm tuần tự, ở đây sử dụng giả thiết mảng A đã được sắp thứ tự tăng
- **Ý tưởng tìm nhị phân:** 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

## 2.2 Tìm nhị phân (tt)

➤ Mô tả 1:

Lặp:

- ❑ Chia đôi phạm vi tìm kiếm
- ❑ So sánh p.tử cần tìm với p.tử ở giữa p.vi tìm kiếm G
  - Nếu  $x > G$  thì tiếp tục tìm ở nửa bên phải
  - Nếu  $x < G$  thì tiếp tục tìm ở nửa bên trái
  - Nếu  $x = G$  thì Ghi nhận tìm thấy
- ❑ Lặp lại cho đến khi tìm thấy hoặc phạm vi tìm rỗng

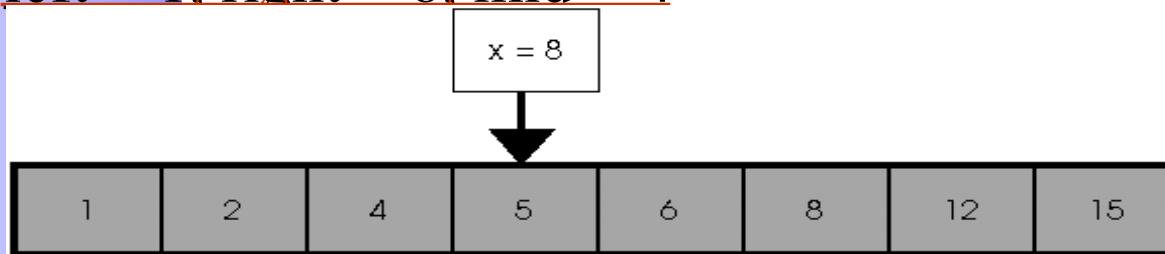
## 2.2 Tìm nhị phân (tt)

- Ví dụ: Cho dãy số a gồm 8 phần tử:

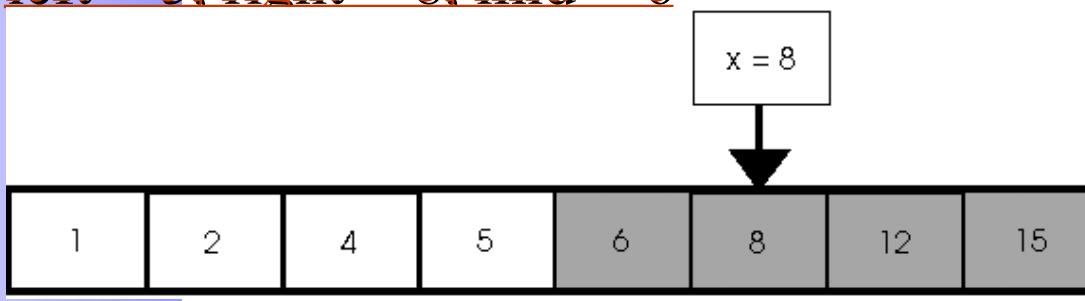
1      2      4      5      6      8      12      15

Giá trị cần tìm là 8

- left = 1, right = 8, mid = 4



- left = 5, right = 8, mid = 6



# ... Minh họa tìm kiếm nhị phân

10

Target key

position = 3

Khóa cần tìm ~~hỗn hợp~~ hoặc bằng

0	1	2	3	4	5	6	7	8	9
2	5	8	10	12	13	15	18	21	24

bottom                          middle                          top

return success

Số lần so sánh: 7

## 2.2 Tìm nhị phân (tt)

### ➤ Mô tả chi tiết

Procedure TimNP(A,T);

    d:=1; c:=n; co:=false;

    While (d<=c) and not co do

        g:=(d+c) div 2;

        If T < A[g] then c:=g-1 {tiếp tục tìm ở nửa trái phạm vi}

        Else

            If T> A[g] then d:=g+1 {tiếp tục tìm ở nửa phải phạm vi }

            Else co:=true

    If co then

        Phản tử bằng T có tại vị trí g trong mảng

    Else Không có T trong mảng.

## 2.3 Bài tập

1. Đánh giá độ phức tạp của các thuật toán tìm kiếm sơ cấp
2. Mô tả thuật toán *Tìm phần tử lớn thứ nhì* trong một dãy tùy ý có n phần tử và đánh giá độ phức tạp của thuật toán.
3. Chạy từng bước thuật toán tìm nhị phân trên một mảng cụ thể có 16 phần tử, xét hai trường hợp: phần tử cần tìm có trong mảng và không có trong mảng.
4. Mô tả thuật toán *Tìm phần tử xuất hiện nhiều lần nhất* trong một dãy tùy ý có n phần tử và đánh giá độ phức tạp của thuật toán.



ĐỆ QUY

# 1. Thuật toán đệ quy

- **Đệ quy (recursion):** là một kĩ thuật định nghĩa một khái niệm trực tiếp hoặc gián tiếp theo chính nó.
- **Thuật toán đệ quy:** là thuật toán có yêu cầu thực hiện lại chính thuật toán đó với mức độ dữ liệu thấp hơn

Thuật toán đệ quy luôn có 2 phần:

- ❑ *Phần cơ sở: cho trường hợp không cần thực hiện lại thuật toán, bảo đảm tính dừng*
- ❑ *Phần đệ quy: yêu cầu thực hiện lại chính nó*

## 2. Một số bài toán

- Tính giai thừa
- Tính UCLN
- Fibonaci
- Bài toán Tháp Hà Nội

## 2.1 Bài toán 1 – Tính giai thừa

➤  $N! = \begin{cases} 1 & \text{nếu } N=0 \\ N*(N-1)! & \text{nếu } N>0 \end{cases}$

➤ Function GiaiThua(n)

If n=0 then GiaiThua:=1  
Else GiaiThua:=n\*GiaiThua(n-1);

## 2.2 Bài toán 2 – Tìm UCLN

- Tìm UCLN của hai số tự nhiên a và b cho trước
  - ❑ Cơ sở: Nếu  $a=b$  hoặc  $b=1$  thì  $\text{UCLN}=b$
  - ❑ Đệ qui: Nếu  $a>b$  thì  $\text{UCLN}(a,b)=\text{UCLN}(a-b, b)$
- Function  $\text{UCLN}(a, b); \quad \{ \text{giả thiết } b \neq 0 \}$ 
  - ❑ If ( $a=b$ ) or ( $b=1$ ) then  $\text{UCLN} := b$  //phản cơ sở
  - ❑ Else
    - If  $a < b$  then
      - ❑ Đổi vai trò a, b để luôn có  $a \geq b$
    - $\text{UCLN} := \text{UCLN}(a-b, b)$

## 2.3 Bài toán 3 - Tính Fibonaci thứ n

$F(n) = 1$  nếu  $n \leq 2$

$F(n) = F(n - 1) + F(n - 2)$  nếu  $n > 2$

```
function F(n: Integer): Integer;  
begin  
    if n ≤ 2 then F := 1 {Phân neo}  
    else F := F(n - 1) + F(n - 2); {Phân đệ quy}  
end;
```

## 2.4 Bài toán Tháp Hà Nội

- **Bài toán tháp Hà nội.** Ngôi đền Benares có n đĩa bằng vàng:
- ❑ Có bán kính khác nhau
  - ❑ Chồng lên nhau ở một chiếc cọc
  - ❑ Theo thứ tự đĩa lớn ở dưới, đĩa nhỏ ở trên. Các nhà sư lần lượt chuyển các đĩa sang một cọc khác theo quy tắc sau:
    - Khi chuyển một đĩa phải đặt vào một trong 03 cọc
    - Mỗi lần chỉ chuyển đúng một đĩa trên cùng tại một cọc và đặt vào trên cùng ở cọc chuyển đến
    - Đĩa lớn hơn không được phép đặt lên đĩa nhỏ hơn

## 2.4 Bài toán Tháp Hà Nội (tt)

➤ Mô tả 1:

Procedure Chuyen(n,A,B,C)

Nếu  $n = 1$  thì chuyển đĩa từ cọc A qua cọc C

Nếu  $n > 1$  thì:

- + Chuyển  $(n - 1)$  đĩa từ A sang B (C làm trung gian).
- + Chuyển 1 đĩa từ A sang C (B: trung gian)
- + Chuyển  $(n - 1)$  đĩa từ B sang C (A: trung gian)

## 2.4 Bài toán Tháp Hà Nội (tt)

Function HaNoi(n, A, B, C)

If n=1 then Writeln( “A -> C”)

Else

    HaNoi(n -1, A, C, B);

    HaNoi(1, A, B, C);

    HaNoi(n -1, B, A, C);

➤ Đánh giá độ phức tạp:

- Số lần chuyển đĩa là  $2^n - 1$
- CM: dùng pp quy nạp

### 3. Đánh giá thuật toán đệ qui

➤ **Ưu điểm:**

- ❑ Mạnh, rõ ràng, chặt chẽ, dễ hiểu
- ❑ Thiết kế TT đơn giản

➤ **Nhược điểm:**

- ❑ Tốn rất nhiều thời gian và bộ nhớ
- ❑ Dễ phát sinh chạy vô hạn.

➤ Chính vì vậy, trong lập trình người ta cố tránh sử dụng thủ tục đệ quy nếu thấy không cần thiết

## 4. Khử đệ qui

- Sử dụng các vòng lặp và ngăn xếp để khử đệ qui
- Phần cơ sở chính là điều kiện dừng của vòng lặp
- Dùng biến và ngăn xếp để lưu các kết quả trung gian

**Bài tập:** Khử đệ qui cho các thuật toán trong mục 2



## Chương 3

# KỸ THUẬT THIẾT KẾ THUẬT TOÁN

# Nội dung

- PP Chia để trị
- PP Quy hoạch động
- PP Tham lam



# KỸ THUẬT CHIA ĐỂ TRỊ

*Divide and Conquer*

# 1. Giới thiệu

**Tư tưởng của kỹ thuật Chia để trị:**

- Chia một bài toán cần giải ra thành những bài toán con nhỏ hơn có cùng một loại vấn đề
- Giải từng bài toán con đó một cách lần lượt và độc lập
- Tổng hợp các lời giải con thu được thành lời giải của bài toán ban đầu.

# 1. Giới thiệu (tt)

## ➤ Mô tả tổng quát

Function DQ(x); {trả lại một lời giải với đầu vào x}

If (x đủ nhỏ) then Giải quyết trực tiếp

Else

Tách x thành các đầu vào nhỏ hơn  $x_1, \dots, x_k$ ;

For  $i:=1$  to  $k$  do  $y_i := DQ(x_i)$ ;

Kết hợp các  $y_i$  để thu được  $y$

Return  $y$ ;

# 1. Giới thiệu (tt)

## ➤ Một số nhận xét:

- ❑ “x đủ nhỏ”: trường hợp bài toán đơn giản, có thể thấy lời giải ngay, hoặc có thể giải bằng các thuật toán đơn giản đã biết
- ❑ Để áp dụng được kỹ thuật chia để trị cần có một số điều kiện:
  - Phải có khả năng tách đầu vào thành những đầu vào nhỏ hơn
  - Có khả năng kết hợp các lời giải con lại một cách hiệu quả
- ❑ Các thuật toán chia để trị, do cách tiếp cận, thường có mô tả tự nhiên dạng đệ qui

## 2. Một số bài toán

- Tìm nhị phân
- Merge sort
- QuickSort
- Hoán đổi hai phần của một dãy số
- Tính lũy thừa

## 2.1 Tìm nhị phân

- Bài toán: Giả sử  $T[1..n]$  là mảng đã sắp thứ tự tăng và  $x$  là một phần tử cho trước. Tìm vị trí chèn  $x$  vào  $T$  sao cho  $T$  vẫn giữ được tính thứ tự
- Tư tưởng: Theo cách tiếp cận chia để trị:
  - ❑ Nếu mảng  $T$  đủ nhỏ (1 phần tử) thì ta sử dụng thuật toán đơn giản để tìm ra vị trí chèn
  - ❑ Nếu  $T$  lớn ( $>1$  p.tử) thì ta chia mảng làm 2 phần, vì mảng sắp thứ tự nên ta sẽ quyết định tìm bên phải hoặc bên trái dựa vào phần tử ở giữa

## 2.1 Tìm nhị phân (tt)

Procedure TimNP2(T,i,j,x);

{thủ tục này được gọi khi  $T[i] < x \leq T[j]$  và  $i \leq j$ }

If ( $i=j$ ) or ( $T[i]=x$ ) then vị trí:= i

Else

g:=(i+j+1) div 2;

if  $x \leq T[g]$  then TimNP2(T,i,g,x)

else TimNP2(T,g,j,x);

Return vị trí

Bài tập: Cài đặt chương trình trên.

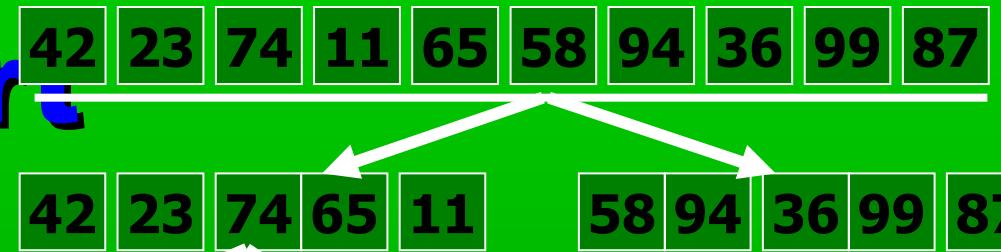
## 2.2 Merge sort (Sắp xếp trộn)

Bài toán: SX dãy A[1..n] theo thứ tự tăng

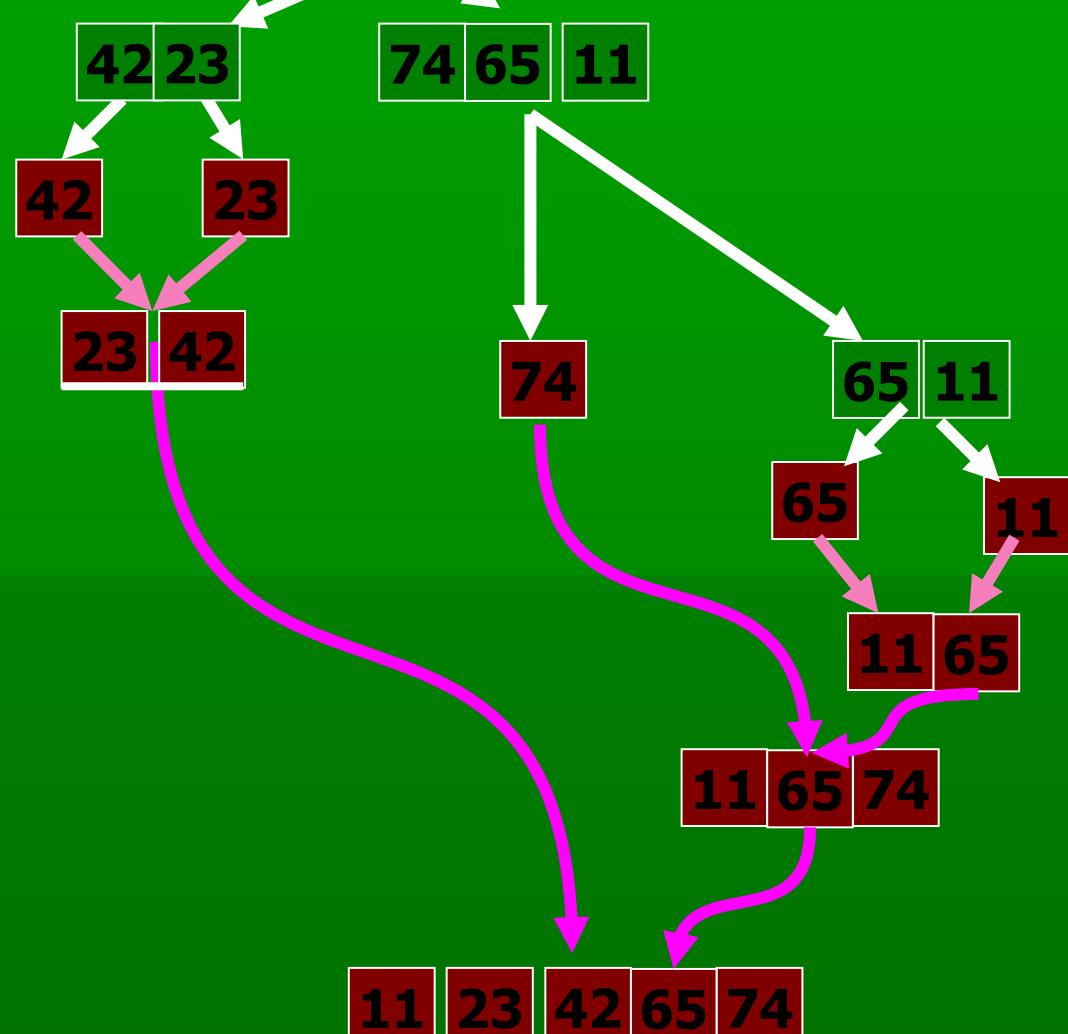
Ý tưởng:

- ❑ Nếu A nhỏ thì SX A[1..n] bằng các pp đơn giản
- ❑ Nếu A lớn thì:
  - Chia A thành 2 mảng con U và V
  - SX U và V độc lập theo thuật toán chia để trị
  - Trộn U và V để được mảng A theo thứ tự

## 2.2 Merge sort



- Ví dụ



## 2.2 Merge sort (tt)

Procedure Merge-Sort(A, i, j)

**if**  $i < j$  **then** //Mảng chứa ít nhất 2 phần tử

$k = [(i+j)/2];$

        Merge-Sort(A, i, k);

        Merge-Sort(A, k+1, j);

        Merge(A, i, k, j);

## 2.2 Merge sort (tt)

Procedure Merge (A, i, k, j)

```
m:=i;          //chỉ số của mảng A[i..k]
n:=k+1;         //chỉ số của mảng A[k+1..j]
h:=i;          //chỉ số của mảng C[i..j] để lưu kết quả trộn
while (m<=k) and (n<=j) do
begin  if A[m]<A[n] then  begin  C[h]:=A[m]; m:=m+1; end
       else  begin  C[h]:=A[n]; n:=n+1; end;
       h:=h+1;
end;
if m>k then      //mảng A[i..k] hết
   for l:=n to j do  begin  C[h]:=A[l]; h:=h+1;  end;
if n>j then      //mảng A[k+1..j] hết
   for l:=m to k do  begin  C[h]:=A[l]; h:=h+1;  end;
A:=C;           //gán mảng C về cho A
```

## 2.2 Merge sort (tt)

- Đánh giá độ phức tạp:
  - ❑ Xem lại chương 1

## 2.3 QuickSort

- Bài toán: SX A[1..n] tăng dần
- Ý tưởng: chia phạm vi cần sắp xếp thành 2 phần nhờ một phần tử đặc biệt gọi là mốc (pivot)
  - ❑ Giả sử phần tử mốc có giá trị là p, khi đó mảng được phân hoạch thành 2 phần:
    - Mọi phần tử đứng trước p đều  $\leq p$
    - Mọi phần tử đứng sau p đều  $> p$
  - ❑ Khi đó ta chỉ cần SX độc lập từng phần của mảng
- **Procedure QuickSort(A, i, j);**  
    **If**  $j-i$  **đủ nhỏ then** SX trực tiếp  
**Else**  
        **Với**  $i < j$   
            **Pivot(A, i, j, p);**  
            **QuickSort(A, i, p-1);**  
            **QuickSort(A, p+1, j);**

## 2.3 QuickSort (tt)

**Procedure Pivot(A, i, j, p)**

```
p:= A[i]; //chọn phần tử đầu làm giá trị mốc  
left:=i+1; right:=j;  
Repeat  
    while(A[left] <= p) and(left<=right) left++;  
    while(A[right] > p) and(left <= right) right--;  
    if(left <right) then Swap(A[left], A[right]);  
Until left>right;  
Swap(A[i], A[right]);  
p:=right;
```

## 2.4 Hoán đổi 2 phần của một dãy số

### ➤ Bài toán:

- ❑ Cho một dãy số nguyên  $A[1..n]$  gồm n phần tử.  
Hãy chuyển m ( $m \leq n$ ) phần tử đầu tiên của dãy với phần còn lại của dãy ( $n-m$  phần tử) (không dùng một mảng phụ).
- ❑ Chẳng hạn:  $n=8$  với  $a[1..8] = (1, 2, 3, 4, 5, 6, 7, 8)$ 
  - Nếu  $m=3$  thì kết quả là:  $a[1..8] = (4, 5, 6, 7, 8, 1, 2, 3)$
  - Nếu  $m=4$  thì kết quả là:  $a[1..8] = (5, 6, 7, 8, 1, 2, 3, 4)$
  - Nếu  $m=5$  thì kết quả là:  $a[1..8] = (6, 7, 8, 1, 2, 3, 4, 5)$

## 2.4 Hoán đổi 2 phần của một dãy số

➤ Ý tưởng:

Chia bài toán thành 2 bài toán con:

- ❑ Bài toán thứ nhất: Hoán đổi 2 dãy con có độ dài bằng nhau
  - Cụ thể là hoán đổi nửa số phần tử đầu và cuối của dãy cho nhau bằng cách đổi chỗ từng cặp tương ứng
- ❑ Bài toán thứ hai: cùng dạng như bài toán 1 nhưng kích thước nhỏ hơn
  - Có thể gọi đệ quy bài toán 1

## 2.4 Hoán đổi 2 phần của một dãy số

➤ Mô tả 1 thuật toán:

- ❑ Nếu  $m = n-m$  : Hoán đổi các phần tử của 2 nửa mảng có độ dài bằng nhau
- ❑ Nếu  $m \neq n-m$  :
  - Nếu  $m < n-m$ :
    - ❑ Hoán đổi  $m$  phần tử đầu với  $m$  phần tử cuối của phần còn lại.
    - ❑ Trong mảng  $a[1.. n-m]$ , hoán đổi  $m$  phần tử đầu với phần còn lại.
  - Nếu  $m > n-m$ :
    - ❑ Hoán đổi  $n-m$  phần tử đầu với  $n-m$  phần tử cuối của phần còn lại.
    - ❑ Trong mảng  $a[n-m+1.. n]$ , hoán đổi  $n-m$  phần tử cuối với phần tử đầu còn lại.

## 2.4 Hoán đổi 2 phần của một dãy số

➤ Mô tả 2 thuật toán chi tiết hơn:

Hoandoi(a, n, m)

{    i = m;    j = n-m;    m = m+1;

Trong khi (i # j)

{ Nếu (i >j) thì

{              Traodoi(a, m-i, m, j);  
                  i = i - j;

}

Ngược lại {            j = j - i;

                  Traodoi(a, m-i, m+j, i);

}

}

Traodoi(a, m-i, m, i);        //khi i=j

}

## 2.4 Hoán đổi 2 phần của một dãy số

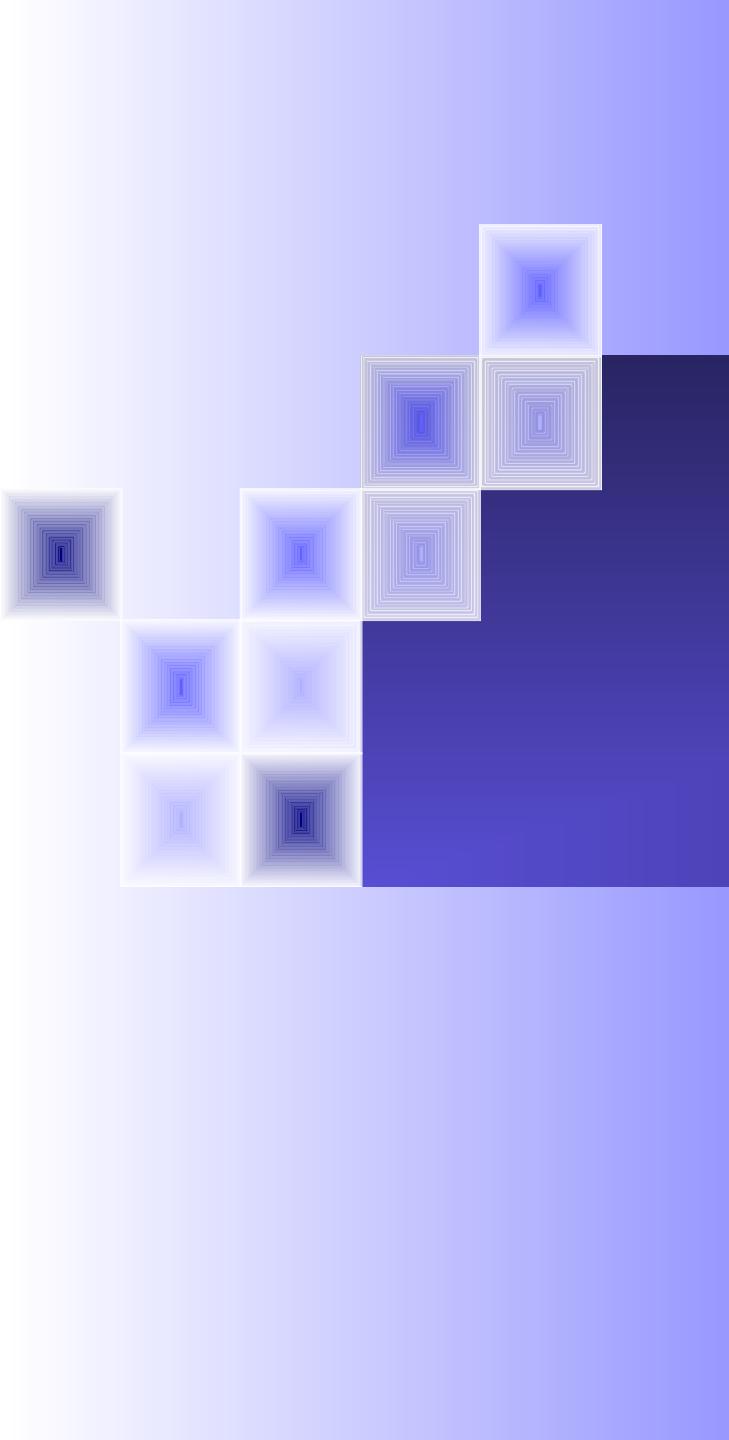
- Mô tả 2 thuật toán chi tiết hơn:

// Thuật toán Traodoi

Traodoi(a, i, j, m)

{ với mọi  $k = 0$  đến  $k = m-1$  thì

Đổi chỗ ( $a[i + k], a[j + k]$ );



# QUY HOẠCH ĐỘNG

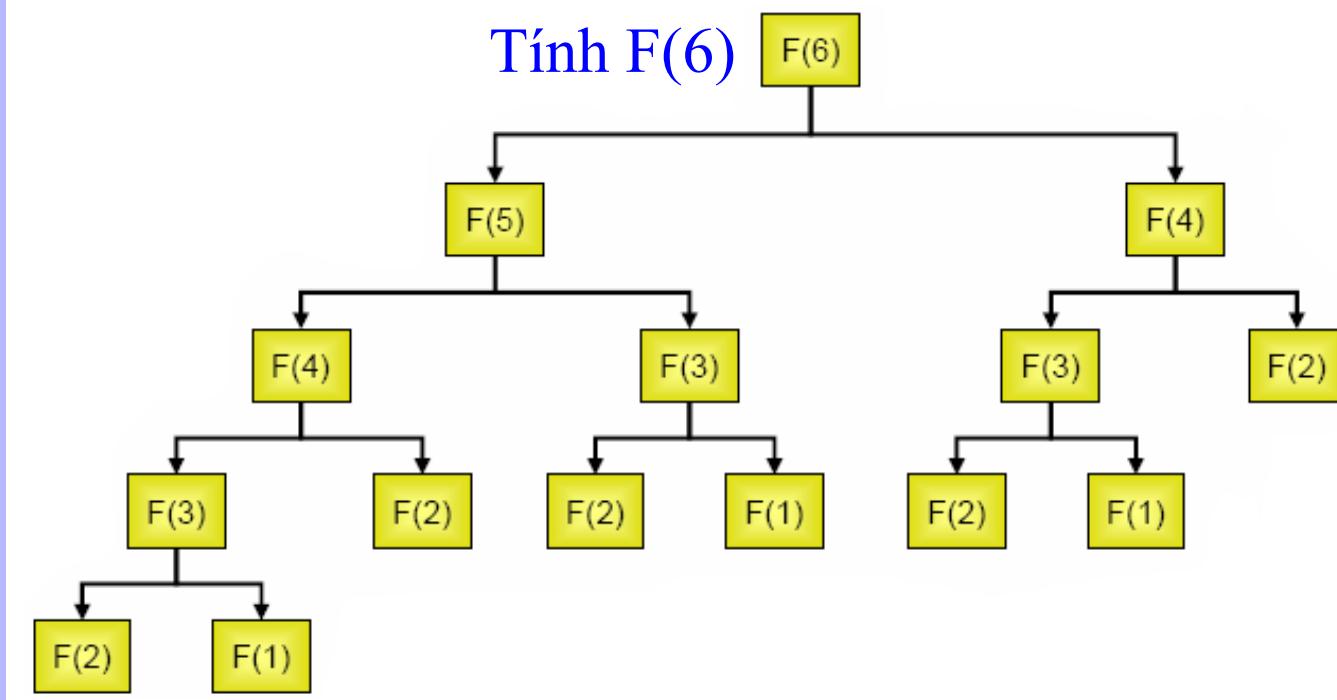
*(Dynamic Programming)*

# 1. Giới thiệu

➤ Số Fibonacci thứ n được tính như sau:

```
Function F(n: Integer): Integer;  
Begin  
    if (n = 0) or (n=1) then F := 1  
    else F := F(n - 2)+F(n - 1);  
End;
```

# 1. Giới thiệu (tt)



➤ Cách khác:

$F_0$	$F_1$	$F_2$	$F_3$	$F_4$	$F_5$	$F_6$
1	1	2	3	5	8	13

# 1. Giới thiệu (tt)

- Tư tưởng của giải thuật Qui hoạch động:
  - ❑ Tương tự như Chia để trị
  - ❑ Ở đây, bắt đầu từ bài toán nhỏ làm cơ sở để giải các bài toán lớn hơn
  - ❑ Lời giải các bài toán nhỏ thường được lưu lại (khác với CDT)
- Là kỹ thuật tiếp cận từ dưới lên
- Thường được áp dụng giải các bài toán tối ưu

# 1. Giới thiệu (tt)

## ➤ Các bước giải bài toán Qui hoạch động

- ❑ Xây dựng *hàm QHD* (hàm mối quan hệ giữa bài toán hiện tại với bài toán trước đó)
- ❑ Lập bảng lưu giá trị của các bài toán con
- ❑ Tính các giá trị ban đầu ứng với các trường hợp đơn giản
- ❑ Tính các giá trị còn lại cho đến khi nhận được giá trị cần tìm

# 1. Giới thiệu (tt) – Tính $C(n, k)$

- Ví dụ: Tính  $C(n,k)$  bằng kỹ thuật đệ qui  
Function  $C(n,k)$ 

```
If (k=0) or (k=n) then return 1  
Else return C(n-1,k)+C(n-1,k-1).
```
- Phương pháp trên đơn giản nhưng không hiệu quả vì tính lại rất nhiều bài toán con (giống bài toán Fibonacci)
- Cách tính khác: Dùng mảng 2 chiều C để lưu giá trị  $C(i, j)$  tại các ô  $(i, j)$  của mảng

# 1. Giới thiệu (tt) – Tính $C(n, k)$

➤ Dùng mảng 2 chiều C để lưu giá trị  $C(i, j)$

	0	1	2	3	...	k-1	k
1	1	1					
2	1	2	1				
..	1						
n-1	1					$C(n-1, k-1)$	$C(n-1, k)$
n	1						$C(n, k)$

Function  $C2(n, k)$

For  $j:=0$  to 1 do  $C[1,j]:=1;$

For  $i:=2$  to  $n$  do  $C[i,0]:=1;$

For  $i:=2$  to  $n$  do *{tính dòng thứ i của bảng}*

For  $j:=1$  to  $\min(i, k)$  do  $C[i,j]:=C[i-1,j]+C[i-1,j-1];$

## 2. Một số bài toán

- Bài toán cái túi nguyên
- Bài toán đổi tiền
- Bài toán phân hoạch
- Bài toán nhân nhiều ma trận
- Bài toán dãy con dài nhất

## 2.1 Cái túi nguyên

➢ Bài toán: Có  $n$  loại đồ vật (mỗi loại có SL không hạn chế) có giá trị và trọng lượng (là các số nguyên) khác nhau cần chọn để bỏ vào một cái túi có thể đựng được trọng lượng tối đa là  $g$ .

Yêu cầu: Cần chọn các đồ vật sao cho *tổng giá trị* các đồ vật trong túi là *lớn nhất*.

## 2.1 Cái túi nguyên (tt)

- Theo kỹ thuật QHĐ: ta sẽ tính phương án tốt nhất cho các trọng lượng túi từ 1 đến g
  - ❑ Gọi  $C_{ij}$  là giá trị lớn nhất của các đồ vật có thể đặt vào túi có trọng lượng j mà chỉ dùng các đồ vật từ 1 đến i
  - ❑ Giá trị cần tìm là  $C_{n,g}$
  - ❑ Gọi m là mảng khối lượng các đồ vật, val là mảng các giá trị tương ứng các đồ vật
  - ❑ Khi đó,  $C_{l,k} = (k \text{ div } m[1]) * \text{val}[1]$
- Tìm công thức truy hồi:  $C_{ij} = ?$

## 2.1 Cái túi nguyên (tt)

- Tìm công thức truy hồi:  $C_{i,j}=?$ 
  - Nhận xét:
    - Khi xét thêm đồ vật thứ i, ta có 2 khả năng lựa chọn
      - Hoặc vẫn sử dụng i-1 đồ vật cũ mà không dùng đồ vật thứ i
      - Hoặc sử dụng ít nhất một đồ vật loại i
    - $C_{i,j} = \max(C_{i-1,j}; C_{i,j-m[i]} + val[i])$

## 2.1 Cái túi nguyên (tt)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1																		
2																		
3																		
4																		
5																		

## 2.1 Cái túi nguyên (tt)

```
Procedure Caitui1;
    For k:=1 to g do
        c[k]:= (m[1] div k)*val[1];
    For i:=2 to n do
        For j:=1 to g do
            If (j-m[i]>=0) then
                If c[j]< c[j-m[i]]+val[i] then
                    c[j]:= c[j-m[i]]+val[i]
```

## 2.2 Bài toán đổi tiền

- *Bài toán:* Giả sử có  $n$  loại tiền giấy, mỗi loại có số tờ không giới hạn. Cần trả số tiền là  $S$  với số tờ là ít nhất
- Áp dụng kỹ thuật Qui hoạch động:
  - ❑ Ta giả sử  $T[1..n]$  là mảng ứng với các loại tiền, tức là  $T[i]$  là giá trị của loại tiền thứ  $i$
  - ❑  $c[i,j]$  là số tờ ít nhất dùng để trả số tiền  $j$  mà chỉ dùng các loại tiền  $T[1], T[2], \dots, T[i]$ .
  - ❑  $C[i,j] = +\infty$  nếu không tìm được phương án trả tiền
  - ❑  $C[n,S]$  là giá trị cần tìm.

## 2.2 Bài toán đổi tiền (tt)

- ... Áp dụng kỹ thuật Qui hoạch động
  - ❑ Tìm công thức truy hồi để tính  $c[i,j]$ :
    - Trả số tiền  $j$  sử dụng các loại tiền  $T[1], T[2], \dots, T[i-1]$
    - Hoặc, trả số tiền  $j$  có sử dụng ít nhất một tờ tiền  $T[i]$
  - Như vậy,  $c[i,j] = \min(c[i-1,j], c[i,j-T[i]])+1)$
- ❑ Trường hợp cơ sở:

$$c_{1j} = \begin{cases} j \text{ div } T[1] & \text{nếu } j \bmod T[1] = 0 \\ +\infty & \text{nếu } j \bmod T[1] \neq 0 \end{cases}$$

## 2.2 Bài toán đổi tiền (tt)

- Ví dụ: ta có các loại tiền sau:  $T[1]=2$ ,  $T[2]=5$ ,  $T[3]=7$ ,  $T[4]=10$ ,  $T[5]=12$  và số tiền phải trả là 17. Ta có bảng sau:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	$\infty$	$\infty$	1	$\infty$	2	$\infty$	3	$\infty$	4	$\infty$	5	$\infty$	6	$\infty$	7	$\infty$	8	$\infty$
2																		
3																		
4																		
5																		

## 2.2 Bài toán đổi tiền (tt)

➤ Mô tả thuật toán

```
Procedure Doitien(i,s);
    For j:=1 to s do
        If j mod T[1]=0 then c[j]:=j div T[1]
        Else c[j]:= +∞;
    For m:=2 to i do
        For j:=1 to s do
            If T[m]<j then
                c[j]:=min(c[j],c[j-T[m]]+1)
    Return c[s];
```

## 2.3 Bài toán dãy con dài nhất

- *Bài toán:* Giả sử có một dãy số nguyên  $a$  có  $n$  phần tử là  $a_1, a_2, \dots, a_n$ . Hãy tìm một dãy con tăng có nhiều phần tử nhất trong dãy.
- *Đặc trưng bài toán:*
  - ❑ Các phần tử trong dãy con kết quả chỉ được xuất hiện một lần.
  - ❑ Thứ tự các phần tử trong dãy kết quả phải được giữ nguyên trình tự so với dãy ban đầu.

## 2.3 Bài toán dây con dài nhất

### ➤ Áp dụng kỹ thuật Quy hoạch động:

Tính từ phần tử đầu tiên đến phần tử cuối cùng của dây.

- ❑ Vì độ dài dây con phụ thuộc vào độ dài của dây ban đầu  
-> Bảng phương án là bảng một chiều (mảng một chiều)
- ❑ Gọi  $L(i)$  là độ dài của dây con dài nhất, các phần tử được lấy trong miền từ  $a_1$  đến  $a_i$  ( $a_i$  : phần tử cuối cùng)
  
- ❑ Xây dựng công thức truy hồi để tính  $L(i)$ :
  - $L(1) = 1$  (vì chỉ có 1 phần tử thì độ dài của nó là 1)
  - $L(i) = \max (1, L(j) + 1) \quad (\forall j < i \text{ và } a[j] \leq a[i])$

## 2.3 Bài toán dãy con dài nhất

- Giải thích công thức truy hồi  $L(i)$ :
  - ❑ Dãy ban đầu chỉ có 1 phần tử  $L(1) = 1$
  - ❑ Trường hợp ghép thêm một phần tử vào trong dãy con thì phải có 1 phần tử  $j$  đứng trước  $i$  thỏa mãn  $a[j] \leq a[i]$ 
    - Tức là, dãy con dài nhất có phần tử cuối cùng là  $a[j]$  có  $L(j)$  phần tử
    - Khi ghép thêm phần tử  $a[i]$  nào thì phần tử cuối cùng của dãy là  $a[i]$  có  $(L(j) + 1)$  phần tử

## 2.3 Bài toán dây con dài nhất

➤ Mô tả thuật toán xây dựng mảng độ dài L(i):

- ❑ Input: Dãy  $a[1], a[2], \dots, a[n]$
- ❑ Output: Mảng L (tượng trưng cho bảng phương án)
- ❑ Algorithm:

$L[1] = 1;$

For ( $i = 2 ; i \leq n ; i++$ )

{  $L[i] = 1;$

    For ( $j = 1 ; j \leq i-1 ; j++$ )

        if ( $(a[j] \leq a[i]) \&\& (L[i] < L[j] + 1)$ )

$L[i] = L[j] + 1;$

}

## 2.3 Bài toán dây con dài nhất

- Xây dựng dây kết quả kq chứa vị trí các phần tử được chọn bằng cách truy vết từ bảng phương án.

Giả mă:

```
i = 1;  
For ( j=2 ; j <= n; j++)  
    if (L[i] < L[j])      i = j ;  
For (k= L[i] ; k>=1 ; k--)  
{  kq[k] = i ;  
    for ( j=1 ; j <= i ; j++)  
        if ((a[j] <= a[i]) && (L[i] = L[j] + 1))  
            i = j ;  
        break;  
}
```

## 2.3 Bài toán dây con dài nhất

➤ *Ví dụ minh họa cụ thể:*

Cho mảng a có 20 phần tử sau. Hãy tìm dây con tăng dài nhất có trong mảng a.

1 3 5 8 11 6 4 10 15 17 19 11 3 17 10 15 14 10 1 20

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a(i)	1	3	5	8	11	6	4	10	15	17	19	11	3	17	10	15	14	10	1	20

-> Ta được bảng phương án sau:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L(i)	1	2	3	4	5	4	3	5	6	7	8	6	3	8	6	7	7	7	2	9

## 2.3 Bài toán dây con dài nhất

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
L(i)	1	2	3	4	5	4	3	5	6	7	8	6	3	8	6	7	7	7	2	9

- ☐ Tiếp tục xây dựng được bảng truy vết từ bảng phương án L(i) (Bảng kq lưu chỉ số của phần tử được chọn)

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Kq (i)	1	2	3	4	5	9	10	11	20	0	0	0	0	0	0	0	0	0	0	0

- ☐ Các phần tử trong dây con kết quả là:

1    3    5    8    11    15    17    19    20

## 2.4 Bài toán phân tích số

➤ *Bài toán:* Cho số tự nhiên  $n \leq 100$ . Hãy cho biết có bao nhiêu cách phân tích số  $n$  thành tổng của các số nguyên dương  $\leq n$ .

Các cách phân tích là hoán vị của nhau chỉ tính là 1 cách.

Ví dụ:  $n = 7$  ta có:  $n = 1 + 1 + 2 + 3$

$$n = 1 + 2 + 3 + 1$$

$$n = 2 + 1 + 3 + 1$$

$$n = 2 + 1 + 1 + 3$$

$$n = 3 + 1 + 3 + 1$$

...

=> chỉ tính là 1 cách

## 2.4 Bài toán phân tích số

➤ Áp dụng kỹ thuật Quy hoạch động:

- ❑ Gọi  $F[m, v]$  là số cách phân tích số  $v$  thành tổng các số nguyên dương  $\leq m$ .
- ❑ Phân tích số  $v$  thành tổng các số nguyên dương  $\leq m$  chia 2 trường hợp:
  - TH1: Không có số  $m$  trong phép phân tích  
 $\Rightarrow F[m, v] =$  số cách phân tích số  $v$  thành tổng các số nguyên dương  $< m$ , tức là  $F[m-1, v]$
  - TH2: Có chứa ít nhất một số  $m$  trong phép phân tích  
Nếu bỏ đi số  $m$  thì sẽ được cách phân tích số  $(v-m)$  thành tổng các số nguyên dương  $\leq m$ , tức là  $F[m, v-m]$   
 $\Rightarrow F[m, v] = F[m-1, v] + F[m, v-m]$

(chỉ đúng khi không tính lặp lại các hoán vị của một cách)

## 2.4 Bài toán phân tích số

- ❑ Khi  $m > v$  : chỉ xảy ra TH 1
- ❑ Khi  $m \leq v$  : có thể 2 TH

⇒ Công thức truy hồi:

$$F[m, v] = \begin{cases} F[m-1, v] & \text{nếu } m > v \\ F[m-1, v] + F[m, v-m] & \text{nếu } m \leq v \end{cases}$$

⇒ Số cách phân tích  $n$  thành tổng các số nguyên  $\leq n$  chính là  $F[n, n]$

## 2.4 Bài toán phân tích số

➤ Xây dựng bảng lưu số cách phân tích số n: Giả sử n = 7

$m \backslash v$	0	1	2	3	4	5	6	7
0	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	4	4
3	1	1	2	3	4	5	7	8
4	1	1	2	3	5	6	9	11
5	1	1	2	3	5	7	10	13
6	1	1	2	3	5	7	11	14
7	1	1	2	3	5	7	11	15

## 2.4 Bài toán phân tích số

Kiểm tra:  $n=7$

1, m = 1:

1 1 1 1 1 1 1      1 cách

2, m = 2:

1 1 1 1 1 2 }      1 cách

1 1 1 2 2 }      2 cách

1 2 2 2 }      2 cách

3, m = 3:

1 1 1 1 3 }      3 cách

1 3 3 }      3 cách

2 2 3 }      3 cách

1 1 2 3 }      3 cách

4, m = 4:

1 1 1 4 }      3 cách

1 2 4 }      3 cách

3 4 }      3 cách

5, m = 5:

1 1 5 }      3 cách

2 5 }      2 cách

6, m = 6:

1 6 }      1 cách

7, m = 7:

7 }      1 cách

---

15 cách

## 2.4 Bài toán phân tích số

➤ *Thuật toán:*

```
Function    Phan_tich;
    F[0, 0] := 1;
    for v:=1 to n do F[0, v] := 0;
    for m:=1 to n do
        for v:=0 to n do
            if m > v then F[m, v] := F[m-1, v]
            else F[m, v] := F[m-1, v] + F[m, v-m];
    Phan_tich := F[n, n];
```

## 2.5 Bài toán phân hoạch

### ➤ *Bài toán:*

Giả sử có 20 file cần lưu trữ vào 2 đĩa, kích thước mỗi file được tính bằng Megabyte. Tổng kích thước các file là 200 MB. Có 2 đĩa để lưu trữ, mỗi đĩa có dung lượng 100 MB.

### ➤ *Phát biểu lại bài toán:*

Cho  $n$  số nguyên dương  $x_1, \dots, x_n$  và một số tự nhiên  $T$ , cần xác định trong dãy đã cho có thể tìm được các số có tổng bằng  $T$  hay không?

Trong bài toán ban đầu  $n=20$ ,  $T=100$ .

## 2.5 Bài toán phân hoạch (tt)

- Có thể giải quyết bài toán này bằng đệ qui:
  - ❑ Nếu tìm được các số có tổng bằng T thì  $x_n$  có thể nằm trong các số này hoặc không.
  - ❑ Vì vậy câu trả lời là được nếu như trong các số  $x_1, \dots, x_{n-1}$  tìm được các số có tổng bằng  $T$  hoặc bằng  $T - x_n$ .

```
Function PH(X,n,T) :boolean;
    If T<0 then PH := False
    Else
        If T=0 then PH := true
        Else PH:= PH(X,n-1,T) or PH(X,n-1,T-X[n]);
```

- ❑ Nếu X có n phần tử thì cần tới  $2^n$  lời gọi đệ qui

## 2.5 Bài toán phân hoạch (tt)

➤ Giải bằng PP Qui hoạch động:

- Gọi  $M_{i,j}$  là khẳng định đúng ( $M_{i,j} = \text{True}$ ) nếu có thể tìm được trong  $x_1, \dots, x_i$  các số có tổng bằng  $j$  và sai ( $M_{i,j} = \text{False}$ ) nếu ngược lại
- Giá trị cần tính là  $M_{n,T}$
- Trường hợp cơ sở:
  - $M_{i,0} = \text{True}$  với mọi  $i$
  - $M_{1,j} = X[1] = j$
- Công thức QHĐ:
  - $M_{i,j} = M_{i-1,j}$  nếu  $x_i > j$  và
  - $M_{i,j} = M_{i-1,j} \text{ or } M_{i-1,j-x_i}$  nếu  $x_i \leq j$ .

## 2.5 Bài toán phân hoạch (tt)

➤ Ví dụ:  $X=[6,3,5,9,2]$ .  $T=13$ .

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1														
2														
3														
4														
5														

## 2.5 Bài toán phân hoạch (tt)

- ...Giải bằng PP Qui hoạch động:

```
Procedure PH2 (X,n,T) ;  
    For i:=1 to n do  
        M[i,0]:=true;  
    For j:=1 to T do  
        M[1,j]:=X[1]=j;  
  
    For i:=2 to n do  
        for j:=1 to T do  
            if X[i]>j then M[i,j]:=M[i-1,j]  
            else M[i,j]:= M[i-1,j] or M[i-1,j-X[i]];  
Return M[n,T];
```

## 2.5 Bài toán phân hoạch (tt)

- ...Giải bằng PP Qui hoạch động: dùng mảng 1 chiều

```
Procedure PH3 (X ,n ,T) ;  
    M[0]:=true;  
    for i:=2 to n do  
        for j:= X[i] to T do  
            M[j]:= M[j] or M[j-X[i]];  
return M[T];
```



# THUẬT TOÁN THAM LAM

## Greedy Algorithms

# 1. Giới thiệu (tt)

- Sử dụng để tìm lời giải tối ưu
- Trong tình huống tổng quát ta có:
  - ❑ Một tập các **đối tượng đề cử** (có thể sẽ được chọn để xây dựng lời giải)
  - ❑ Một tập các đối tượng **đã được lựa chọn**
  - ❑ Một hàm để kiểm tra xem một tập các đối tượng đề cử có **cung cấp lời giải** cho bài toán hay không (không nhất thiết tối ưu)
  - ❑ Một hàm để kiểm tra xem một tập đối tượng đề cử có là **triển vọng** hay không
  - ❑ Một **hàm chọn** để xác định ứng viên có triển vọng nhất
  - ❑ Một **hàm lượng giá** (hàm đích) cho giá trị của một lời giải (để tối ưu hóa)

# 1. Giới thiệu (tt)

- *Ví dụ 1:* Giả sử có các loại tiền giấy có giá trị lần lượt là 1, 5, 10, 25. Cần trả số tiền  $n$  với số tờ ít nhất.
- ❑ *Các đối tượng đề cử:* tập hữu hạn các đồng tiền thuộc các loại tiền đã cho, mỗi loại có ít nhất một tờ.
  - ❑ *Một lời giải:* tổng giá trị của các tờ tiền đã chọn bằng  $n$ .
  - ❑ *Tập có triển vọng:* tổng giá trị các tờ tiền trong tập hợp đã chọn không vượt quá  $n$ .
  - ❑ *Hàm chọn:* chọn tờ tiền có giá trị cao nhất còn lại trong tập đề cử.
  - ❑ *Hàm lượng giá:* số tờ tiền đã sử dụng trong lời giải.

# 1. Giới thiệu (tt)

- Các bước giải bài toán tối ưu
  - ❑ Khởi đầu tập S các đối tượng để cử đã lựa chọn là tập rỗng
  - ❑ Tại mỗi bước, ta thử thêm vào một đối tượng tốt nhất còn lại (nhờ hàm chọn)
    - Nếu tập mới không có triển vọng, bỏ đối tượng này đi (không xét lại nữa), chọn đối tượng khác
    - Ngược lại, đối tượng mới này xếp vào tập S
    - Kiểm tra xem tập S có phải là lời giải không
- Khi một thuật toán tham lam hoạt động đúng, lời giải đầu tiên tìm được luôn là lời giải tối ưu.

# 1. Giới thiệu (tt)

Function Greedy(C:tập hợp)

{C là tập hợp tất cả các đối tượng để cử}

S:= $\emptyset$  {S là tập chứa lời giải được xây dựng từng bước}

While (S chưa là lời giải) and (C $\neq\emptyset$ ) do

x:= một phần tử của C làm tối đa hàm chọn

C:=C\{x}

If (S $\cup$ {x}) là có triển vọng then S:=S $\cup$ {x}

If S là lời giải then return S

Else “Không có lời giải”

# 1. Giới thiệu (tt)

- Thuật toán Tham lam không phải lúc nào cũng cho lời giải tối ưu
- *Ví dụ 2:* Giả sử có các loại tiền giấy có giá trị lần lượt là 1, 5, 10, 12, 25. Cần trả số tiền  $n$  với số tờ ít nhất.
  - ❑ Giả sử số tiền cần trả là 20.
  - ❑ Theo thuật toán tham lam các tờ tiền sẽ được chọn lần lượt như sau: 12, 5, 1, 1, 1 và cho ta một phương án không tối ưu.
- *Ví dụ 3:* Giả sử có các loại tiền giấy có giá trị lần lượt là 2, 5, 10, 12, 25. Cần trả số tiền  $n$  với số tờ ít nhất.
  - ❑ Giả sử số tiền cần trả là 15. Theo thuật toán tham lam các tờ tiền sẽ được chọn lần lượt như sau: 12, 2 và không dẫn tới lời giải.

## 2.1 Bài toán Tìm đường đi ngắn nhất

- Xét đồ thị  $G=(V,E)$  là đơn đồ thị có trọng số không âm,  $V=\{1, 2, \dots, n\}$ ,  $v$  là một đỉnh thuộc  $V$ .
- Bài toán đặt ra là tìm đường đi ngắn nhất từ  $v$  tới tất cả các đỉnh còn lại của đồ thị

## 2.1 Bài toán Tìm đường đi ngắn nhất

- Theo kỹ thuật Tham lam, ta gọi:
  - ❑ C là tập các đỉnh của đồ thị
  - ❑ S là tập các đỉnh đã chọn, lúc đầu  $S=\text{rỗng}$
  - ❑ Hàm chọn: chọn một đỉnh trong C sao cho khoảng cách từ đó đến v nhỏ nhất
  - ❑ Giả sử đồ thị được cho bởi ma trận trọng số L với  $L[i,j]=\infty$  nếu không có cạnh  $(i,j)$

## 2.1 Bài toán Tìm đường đi ngắn nhất

Procedure Dijkstra(L)

{Kết quả là mảng  $D[2..n]$ }

C:= {2,3,..,n}

S:= {1}

For i:=2 to n do  $D[i]:=L[1,i]$

{Lặp tham lam}

Repeat n-1 lần

v:= phần tử của C có  $D[v]$  nhỏ nhất

C:=C \ {v}; S:= S $\cup$  {v}

For w thuộc C do

$D[w]:= \min(D[w], D[v]+L[v,w])$

## 2.2 Bài toán lập lịch

➤ *Bài toán:*

- ❑ Một người phục vụ phải phục vụ cho  $n$  khách hàng.
- ❑ Khách hàng thứ  $i$  cần phục vụ trong thời gian  $t[i]$ ,  $1 \leq i \leq n$ .
- ❑ Tìm một trình tự phục vụ sao cho tổng thời gian chờ đợi và thời gian được phục vụ của tất cả các khách hàng (gọi là *thời gian hệ thống*) là nhỏ nhất.

## 2.2 Bài toán lập lịch

Ví dụ: Giả sử có 3 khách hàng với  $t[1]=5$ ,  $t[2]=10$ ,  $t[3]=3$ .

Thứ tự	Tổng thời gian hệ thống T	
1 2 3:	$5+(5+10)+(5+10+3)$	=38
1 3 2:	$5+(5+3)+(5+3+10)$	=31
2 1 3:	$10+(10+5)+(10+5+3)$	=43
2 3 1:	$10+(10+3)+(10+3+5)$	=41
3 1 2:	$3+(3+5)+(3+5+10)$	=29    ← tối ưu
3 2 1:	$3+(3+10)+(3+10+5)$	=34

## 2.2 Bài toán lập lịch

- Ta sẽ xây dựng thuật toán theo từng bước
- Giả sử ta đã xếp lịch cho các khách hàng  $i_1, i_2, \dots, i_m$
- Khi xếp lịch cho khách hàng  $j$  thì  $T$  tăng thêm một lượng
$$t[i_1] + t[i_2] + \dots + t[i_m] + t[j]$$
- Như vậy, để tối thiểu  $T$  ta sẽ tối thiểu  $t[j]$
- Để tối thiểu  $t[j]$  ta chỉ cần chọn khách hàng có thời gian phục vụ bé nhất

## 2.2 Bài toán lập lịch

➤ Procedure Lichpv;

{ $C$  là tập khách hàng,  $C=\{1, 2, \dots, n\}$ ,  $L$  là lịch phục vụ}

$i:=1;$

While  $C \neq \emptyset$  do

$x :=$  khách hàng trong  $C$  có tx nhỏ nhất

$L[i]:=x;$

$i:=i+1;$

$C:=C \setminus \{x\}$

## 2.3 Bài toán sơn màu đồ thị

➤ *Bài toán:*

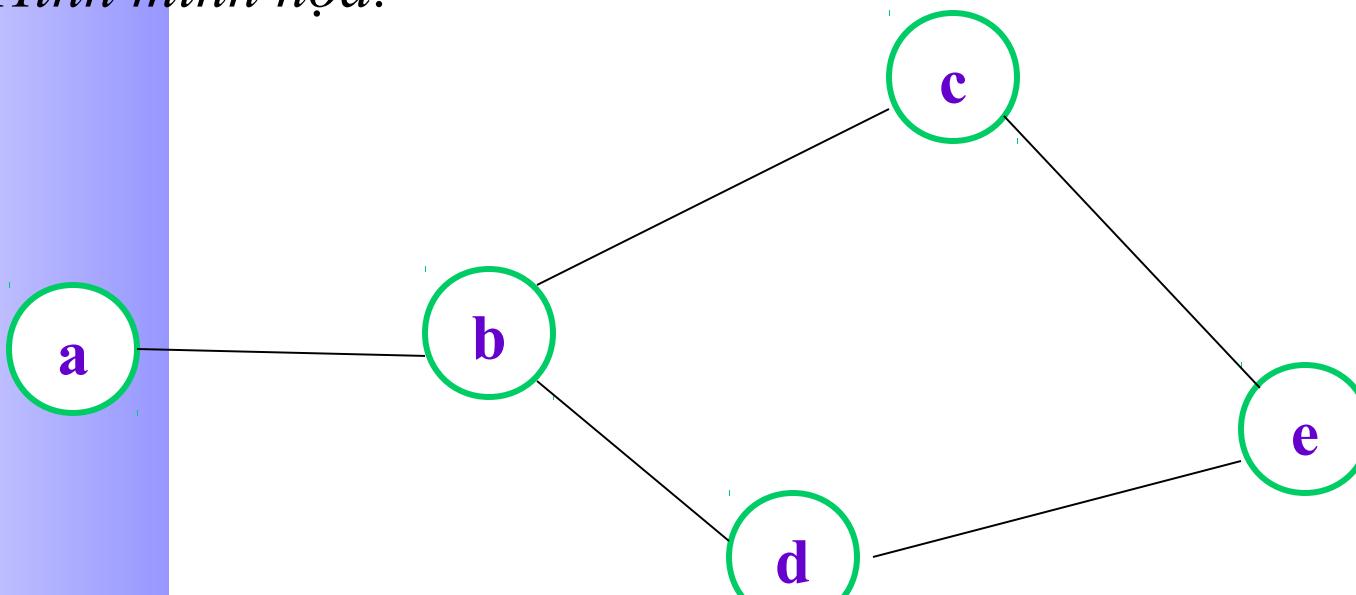
- ❑ Giả sử  $G = (V, E)$  là một đồ thị không định hướng.
- ❑ Yêu cầu: Cần sơn màu các đỉnh đồ thị sao cho 2 đỉnh kề nhau được sơn bởi 2 màu khác nhau và sử dụng số màu ít nhất có thể được.

## 2.3 Bài toán sơn màu đồ thị

- Sử dụng chiến lược tham lam như sau:
  - ❑ Dùng 1 màu chọn 1 đỉnh chưa sơn và sơn đỉnh đó.
  - ❑ Sau đó, với mỗi đỉnh  $i$  chưa sơn:
    - Nếu đỉnh  $i$  không kề với các đỉnh đã được sơn bởi màu đang sử dụng thì sơn đỉnh đó bởi màu đó
  - ❑ Tiếp tục cho đến khi không còn đỉnh nào có thể sơn được màu đó thì ta dùng màu mới để sơn và áp dụng cách sơn như trên.
  - ❑ Lặp lại cho đến khi sơn hết các đỉnh của đồ thị

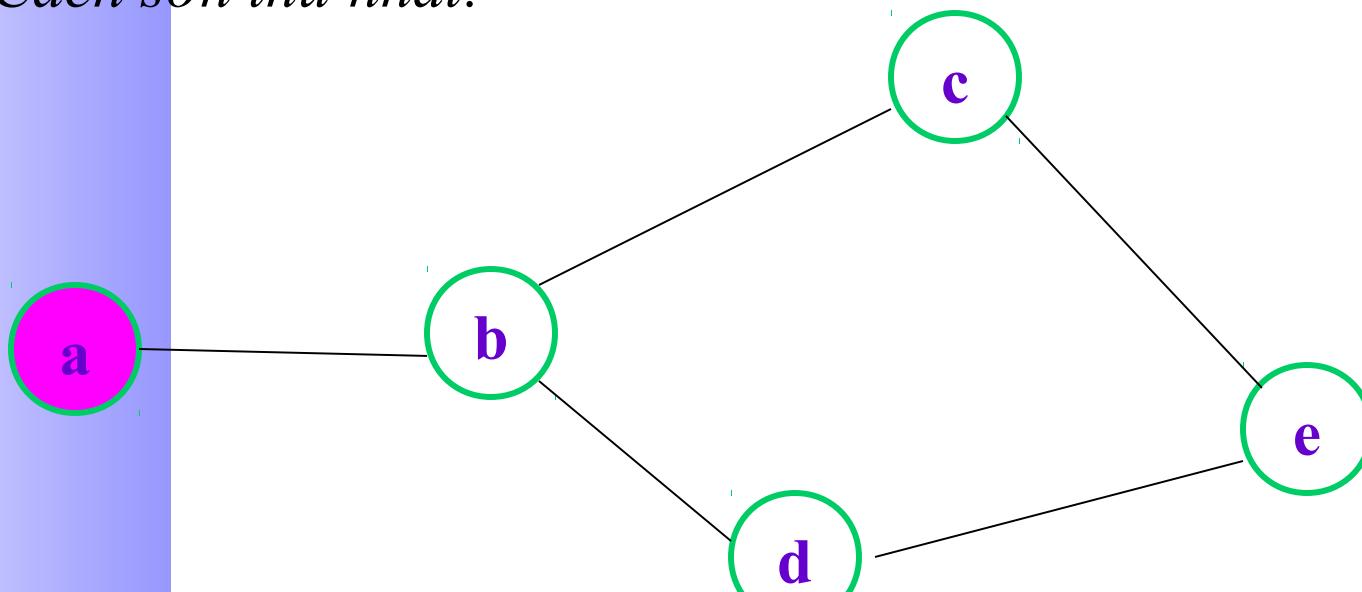
## 2.3 Bài toán sơn màu đồ thị

➤ *Hình minh họa:*



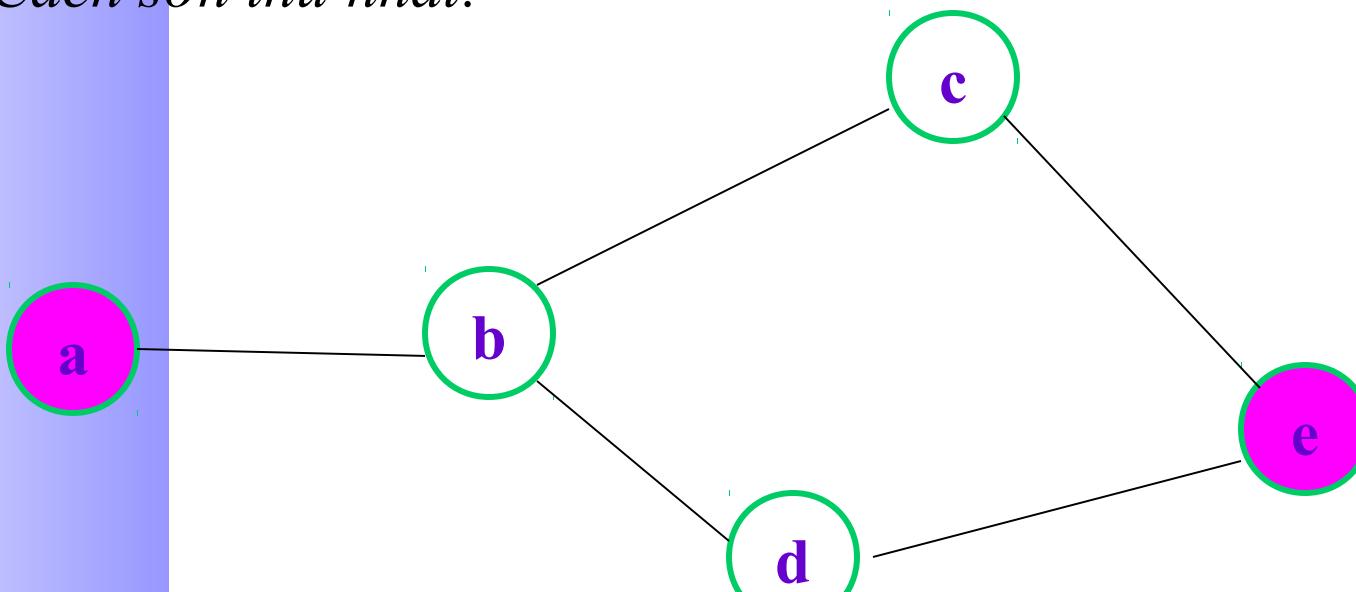
## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ nhất:*



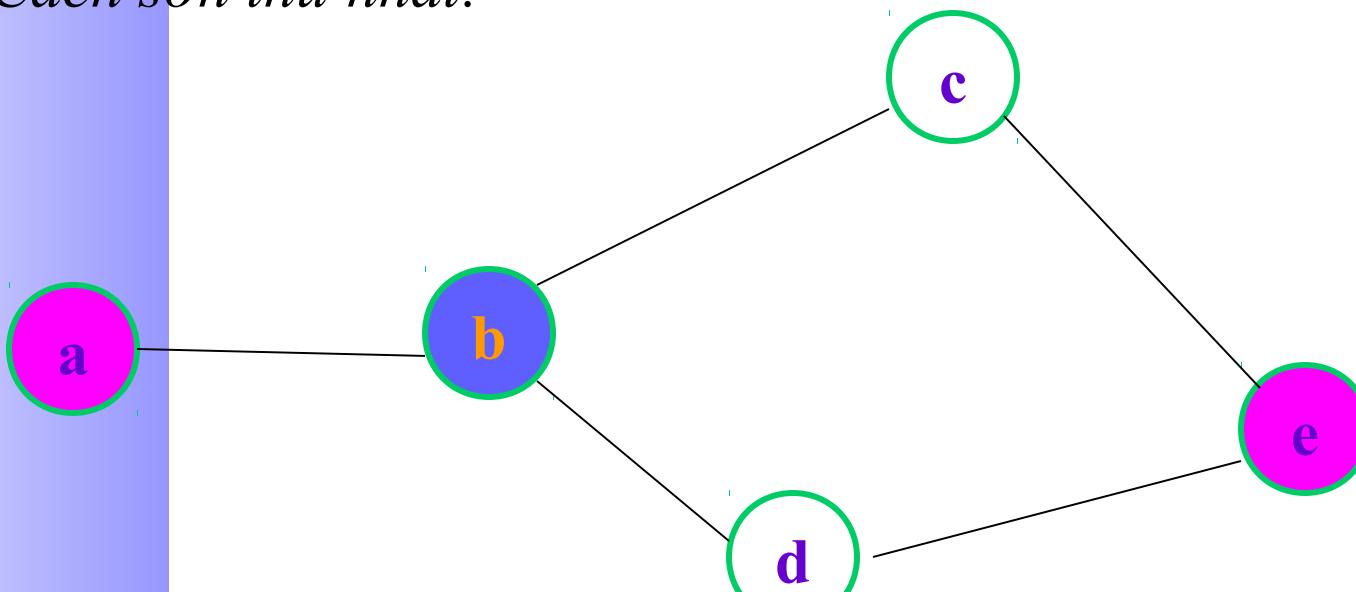
## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ nhất:*



## 2.3 Bài toán sơn màu đồ thị

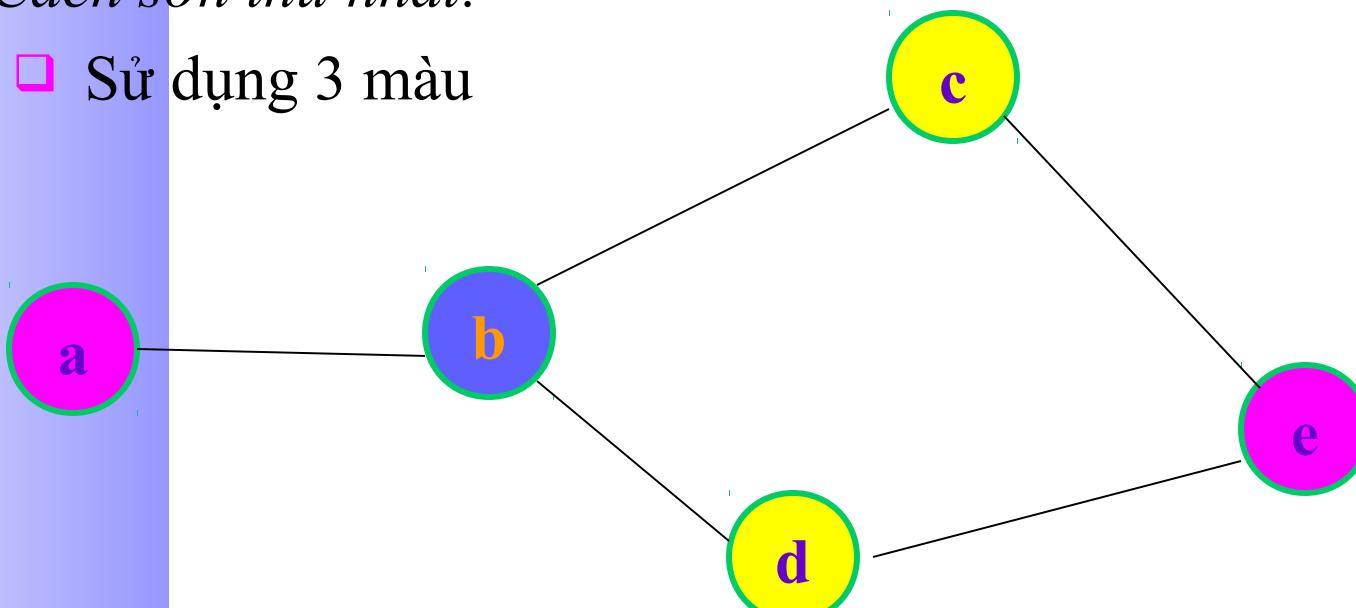
➤ *Cách sơn thứ nhất:*



## 2.3 Bài toán sơn màu đồ thị

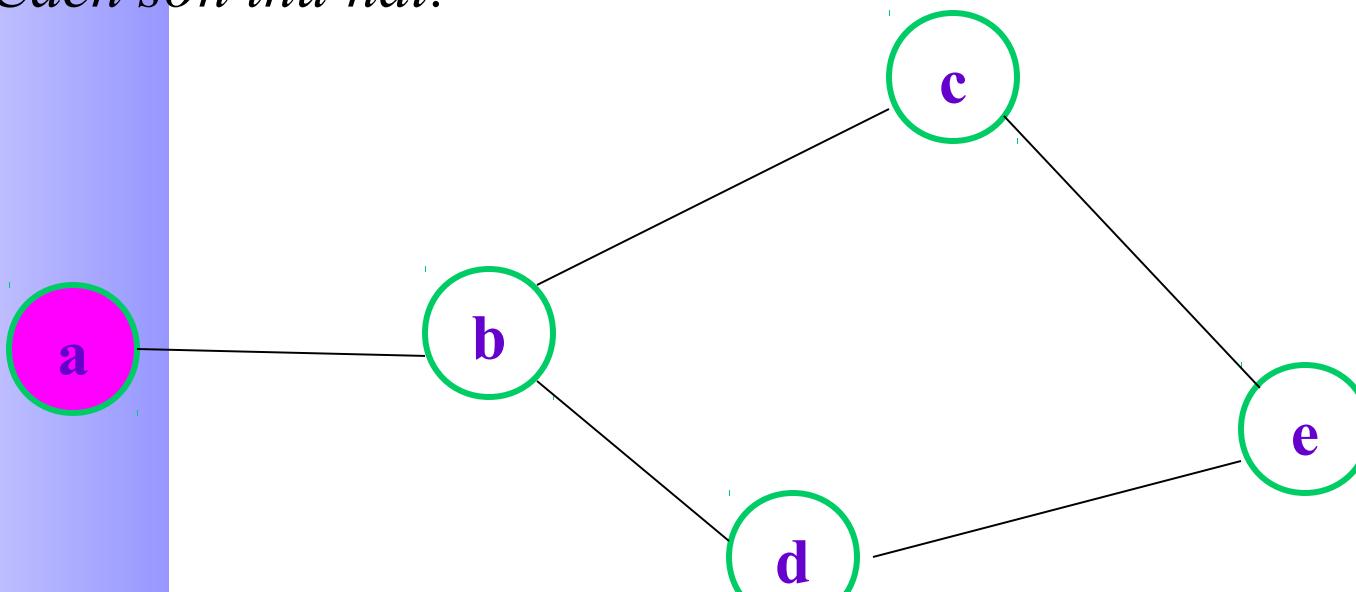
➤ *Cách sơn thứ nhất:*

- ❑ Sử dụng 3 màu



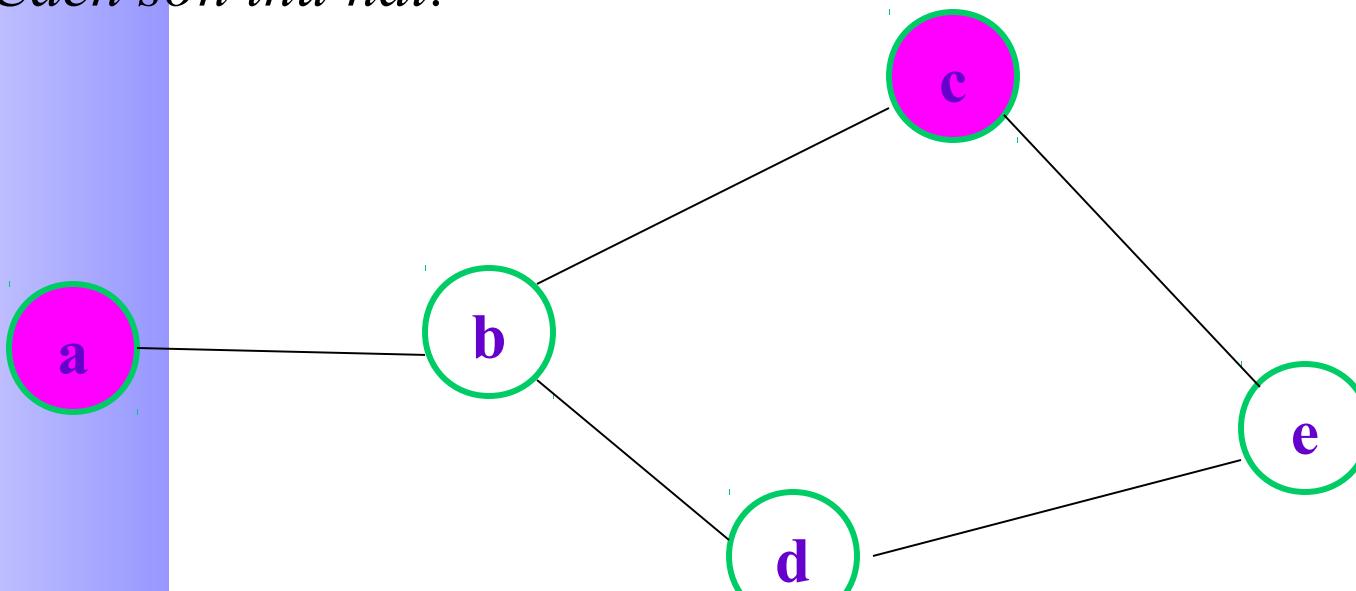
## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ hai:*



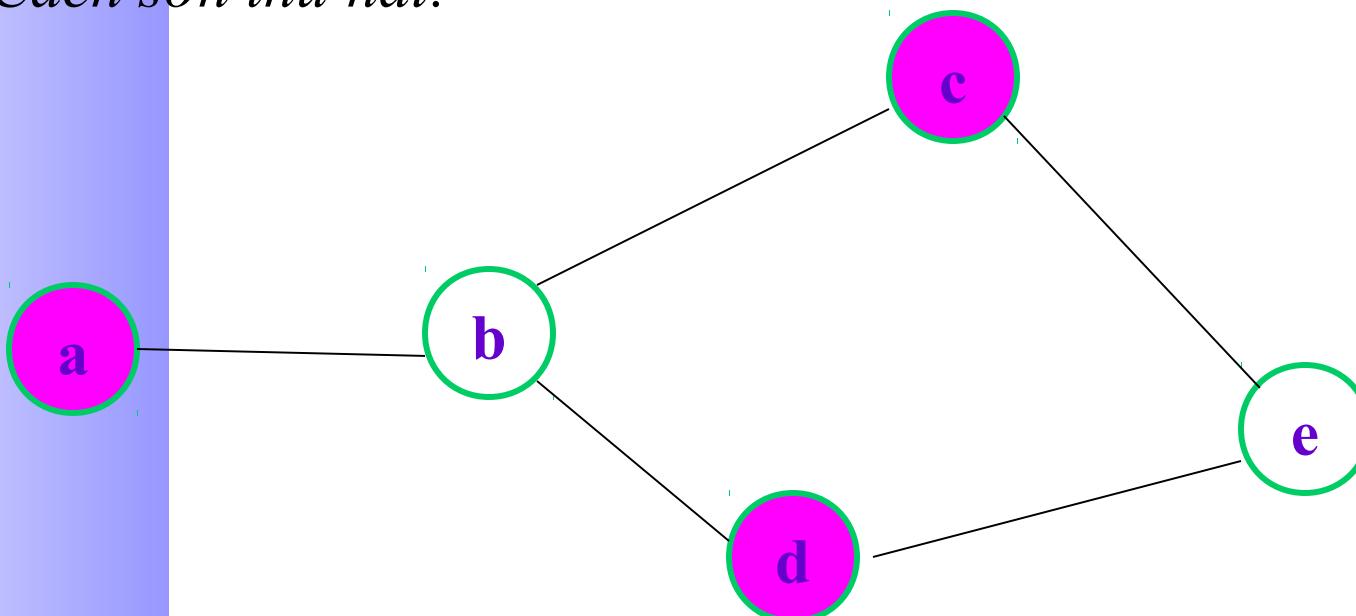
## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ hai:*



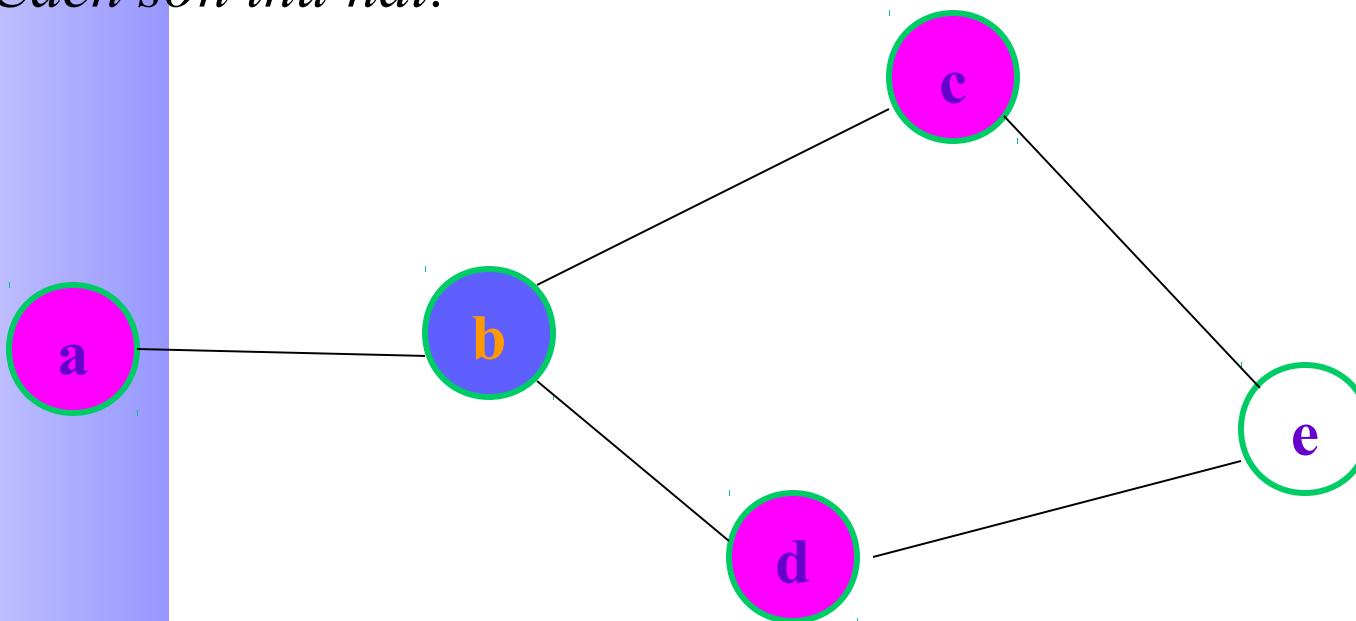
## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ hai:*



## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ hai:*

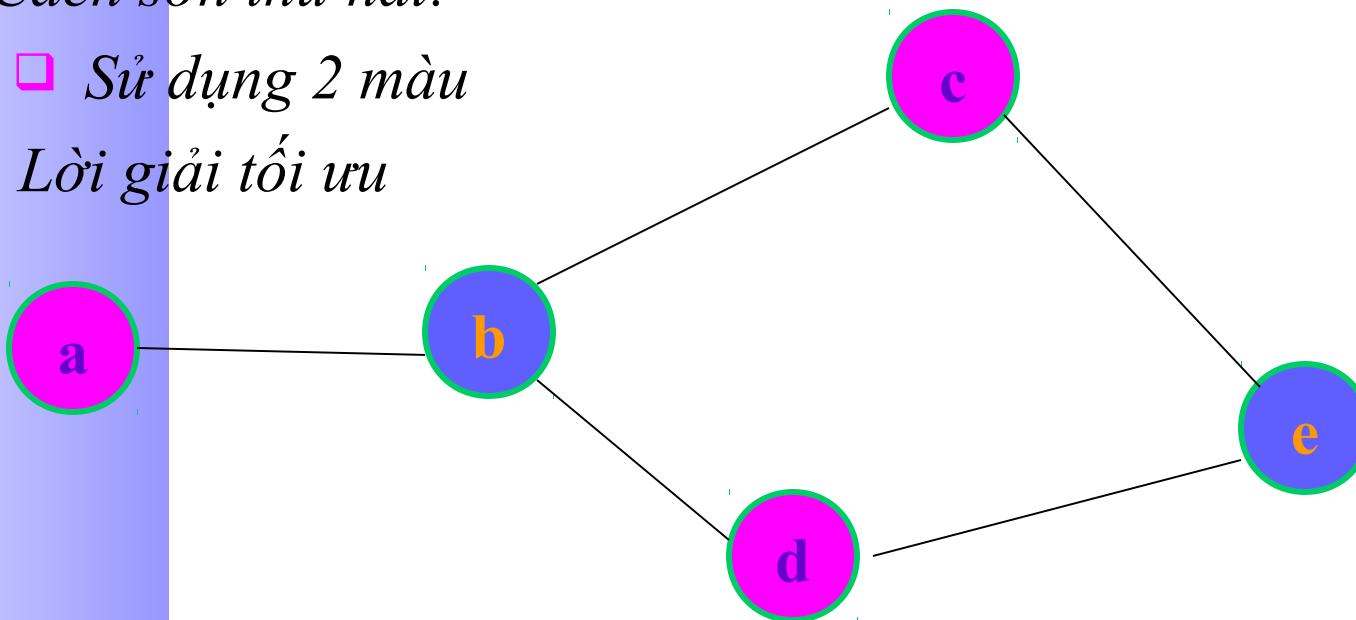


## 2.3 Bài toán sơn màu đồ thị

➤ *Cách sơn thứ hai:*

❑ Sử dụng 2 màu

=> *Lời giải tối ưu*



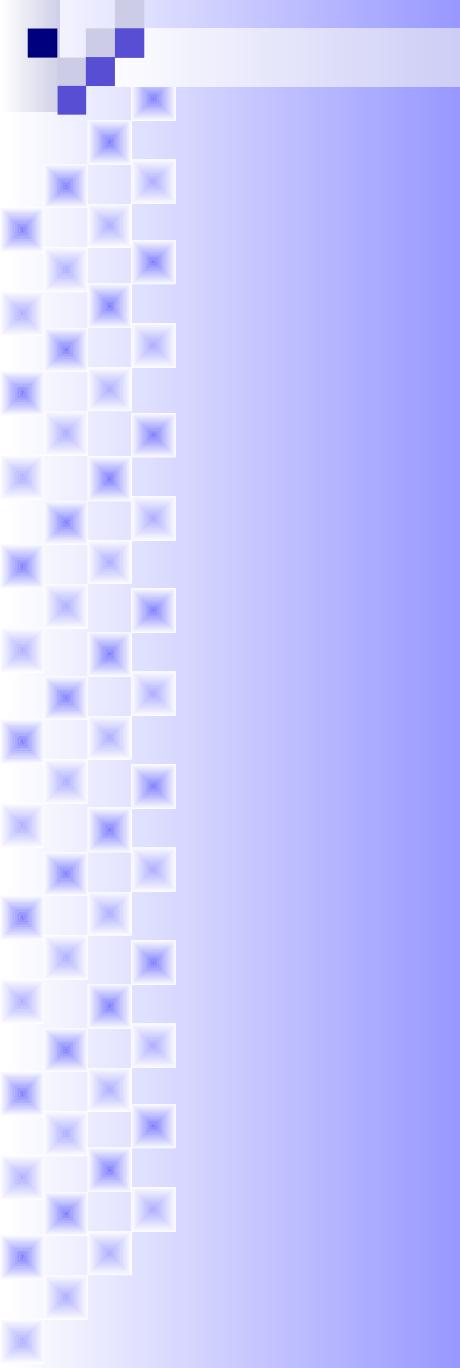
## 2.3 Bài toán sơn màu đồ thị

➤ Thủ tục:

- Giả sử  $V$  là tập các đỉnh của đồ thị  
 $V_0$  là tập các đỉnh chưa được sơn màu  
 $V_1$  là tập các đỉnh được sơn bởi màu mới
- Mã hóa các màu bởi số nguyên  $k = 1, 2, 3, \dots$

## 2.3 Bài toán sơn màu đồ thị

```
Procedure Coloring;  
begin  
    k:=0;      V0:=Ø;  
    while V0 <> Ø do  
        begin  
            k:= k + 1;  
            chọn x∈ V0 và sơn x bởi màu k;  
            V1:= {x};  
            for mỗi v∈ V0 do  
                if v không kề mọi w∈ V1 then  
                    begin  
                        sơn v bởi màu k;  
                        V1 := V1 ∪ {v};  
                    end;  
            V0:=V0 - V1;  
        end;  
    end;
```



**Thank you!**