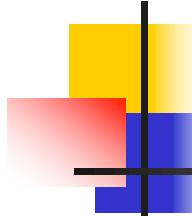


Bài 3:

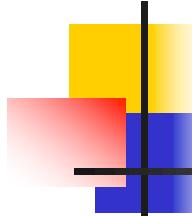
CÁC THUẬT TOÁN TÌM KIẾM- SẮP XẾP

GV Nguyễn Thị Quỳnh Như- Email:
ntqnhu@hou.edu.vn



Nội dung bài học

- Tìm kiếm:
 - Định nghĩa tìm kiếm
 - Các phương pháp tìm kiếm
 - Tìm kiếm tuyến tính
 - Tìm kiếm nhị phân
- Sắp xếp
 - Định nghĩa sắp xếp
 - Các phương pháp sắp xếp
 - Sắp xếp chọn trực tiếp- Selection Sort
 - Sắp xếp chèn trực tiếp- Insertion Sort
 - Sắp xếp nổi bọt- Bubble Sort
 - Sắp xếp nhanh- Quick Sort
 - Sắp xếp vun đống- Heap Sort
 - Sắp xếp trộn – Merge Sort

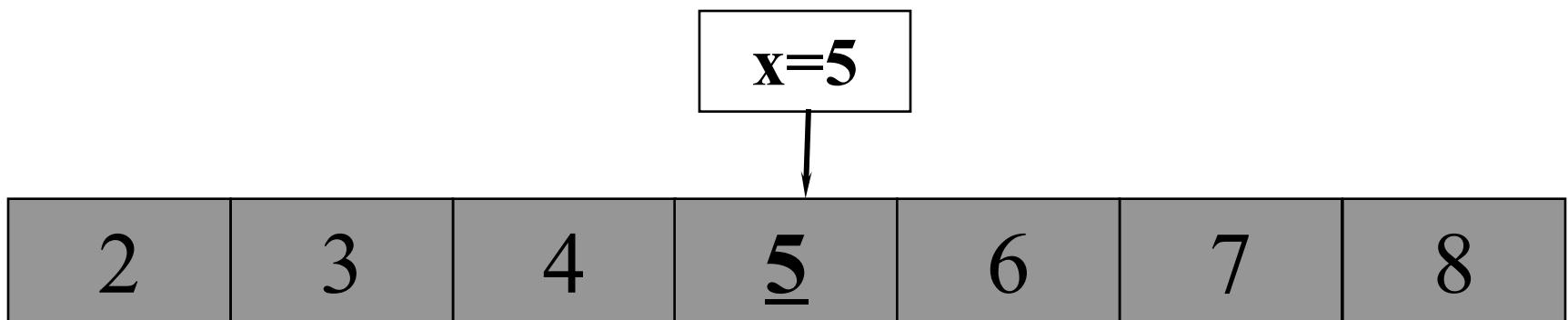


I. Tìm kiếm:

- **Định nghĩa:** Tìm kiếm là kỹ thuật tìm xem liệu một phần tử có khoá x bất kỳ có mặt trong một dãy các phần tử đã cho hay không?
- Tuỳ theo vị trí lưu trữ dữ liệu cần tìm kiếm mà ta có tìm kiếm trong và tìm kiếm ngoài
- Trong phạm vi chương trình ta nghiên cứu tìm kiếm trong với 2 phương pháp:
 - Tìm kiếm tuyến tính
 - Tìm kiếm nhị phân

Ví dụ

- **Bài toán:** cho dãy a gồm n phần tử, yêu cầu tìm kiếm xem trong dãy a có phần tử x hay không? (*n phần tử nguyên*)



1. Thuật toán tìm kiếm tuyến tính

■ Ý tưởng

- Lần lượt duyệt từ đầu mảng đến cuối mảng, mỗi lần so sánh phần tử cần tìm x với phần tử tương ứng của mảng
- Quá trình duyệt liên tục đến khi tìm thấy hoặc duyệt hết mảng.

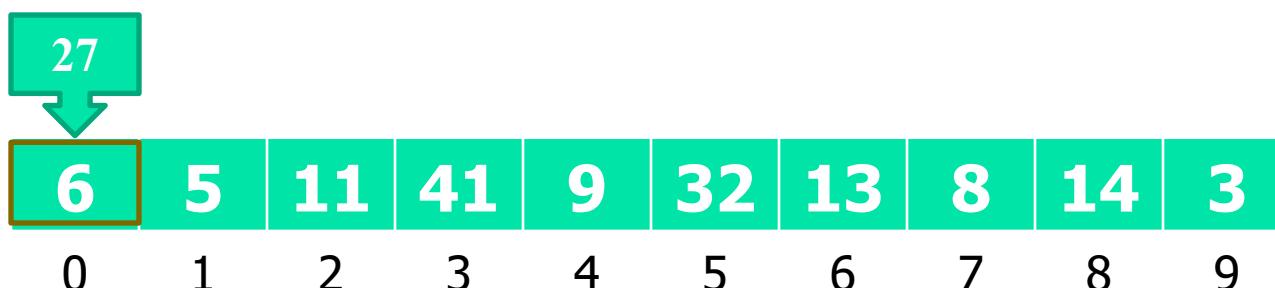
Minh họa thuật toán

■ Minh họa tìm $x = 9$



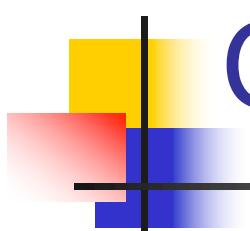
Đã tìm thấy tại vị trí 4

■ Minh họa tìm $x = 27$



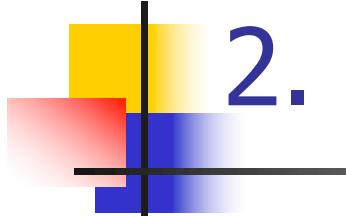
Đã hết mảng

Không tìm thấy



Cài đặt thuật toán:

- Định nghĩa hàm **LineSearch** tìm kiếm phần tử x trong mảng a có n phần tử. Kết quả hàm trả lại vị trí đầu tiên tìm thấy phần tử x hoặc trả lại giá trị -1 nếu không tìm thấy.
- Sinh viên tự cài đặt thuật toán



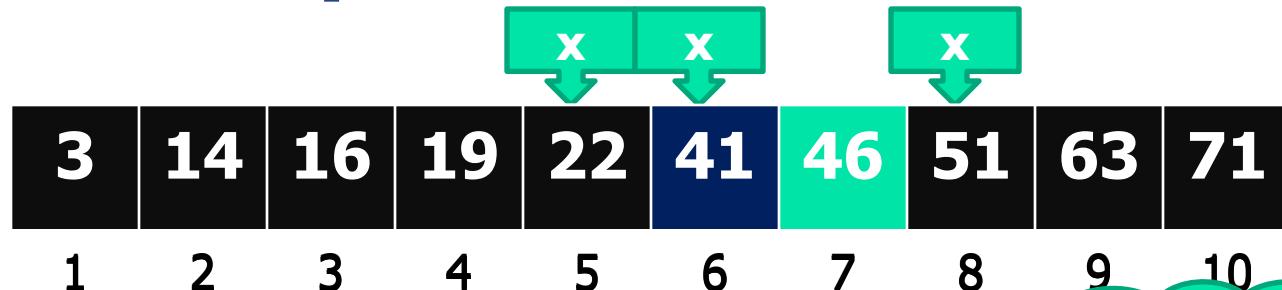
2. Thuật toán tìm kiếm nhị phân

- Ý tưởng:
 - Xét dãy đã có thứ tự (tăng hoặc giảm)
 - Giải thuật tìm kiếm nhị phân dựa vào nguyên tắc: So sánh giá trị x với phần tử giữa của dãy tìm kiếm hiện hành. Dựa vào giá trị này sẽ quyết định dãy tìm kiếm mới là nửa sau hay nửa trước của dãy hiện hành.

- Giả sử dãy tăng dần $a_{i-1} \leq a_i \leq a_{i+1}$
 $\Leftrightarrow a_1 \leq a_2 \leq a_3 \leq \dots \leq a_{i-1} \leq a_i \leq a_{i+1} \leq \dots \leq a_{n-1} \leq a_n$
- Khi đó với một phần tử a_i bất kỳ là phần tử giữa, ta có:
 - \Leftrightarrow Nếu $x > a_i \Leftrightarrow x$ chỉ có thể nằm trong khoảng $[a_{i+1} \dots a_n]$
 - \Leftrightarrow Nếu $x < a_i \Leftrightarrow x$ chỉ có thể nằm trong khoảng $[a_1 \dots a_{i-1}]$

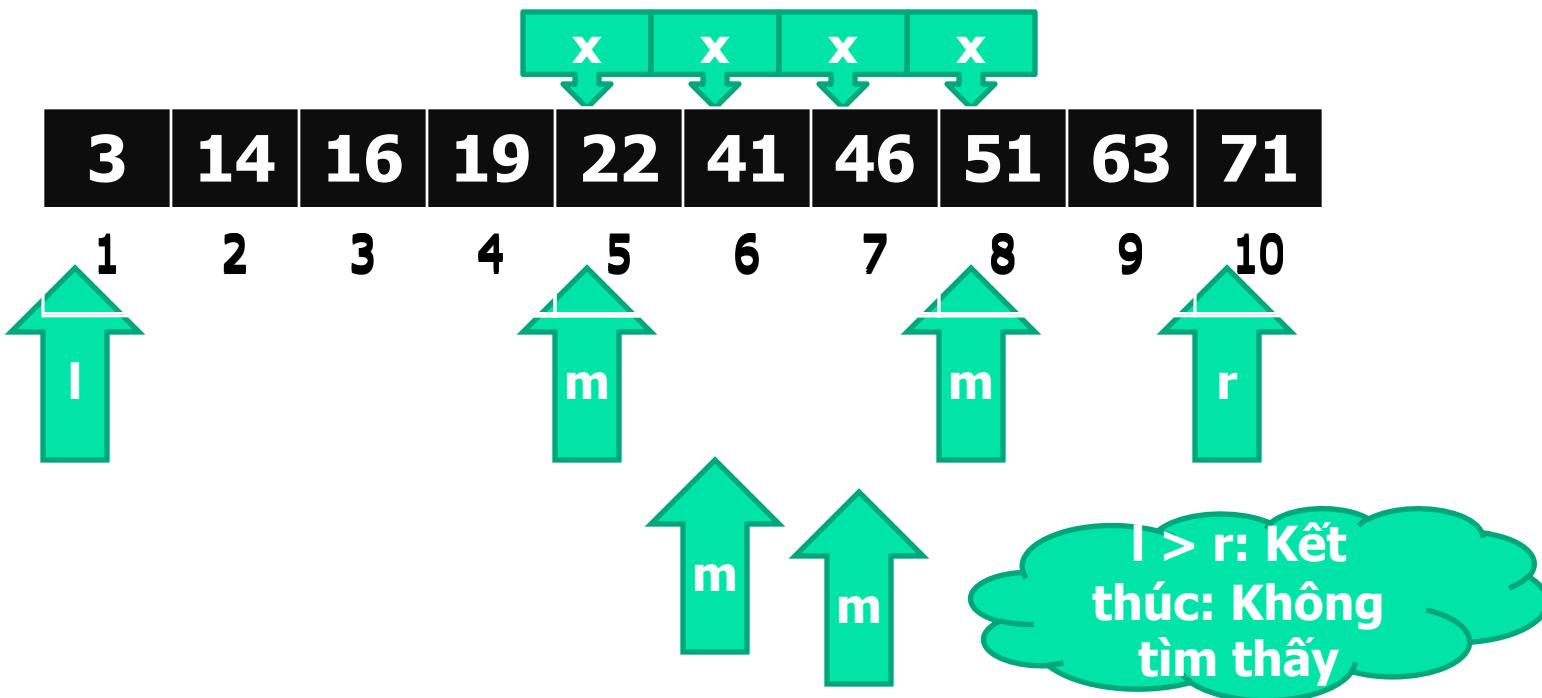
Minh họa thuật toán

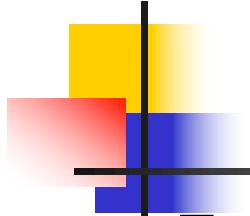
Minh họa tìm $x = 41$



Tìm thấy x
tại vị trí 6

Minh họa tìm $x = 45$





Mô tả thuật toán:

■ Input:

x – là giá trị cần tìm

n – số phần tử mảng

a – mảng chứa các phần tử đã được sắp xếp tăng dần

left, right – là chỉ số đầu và chỉ số cuối của mảng thực hiện tìm

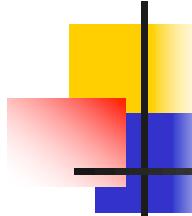
middle – chỉ số giữa của mảng

■ Output:

vị trí chứa phần tử x (nếu tìm thấy)

Các bước thực hiện thuật toán

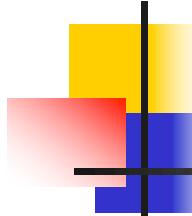
- Bước 1: Khởi đầu tìm kiếm trên tất cả các phần tử của dãy: $\text{left}=0$, $\text{right}=n-1$
- Bước 2: Tính $\text{middle}=(\text{left}+\text{right})/2$ (lấy phần nguyên)
- So sánh: $a[\text{middle}]$ với x :
 - Nếu $a[\text{middle}]=x$: Tìm thấy \Rightarrow dừng thuật toán
 - Nếu $a[\text{middle}]>x$: tiếp tục tìm x trong dãy con mới với $\text{right}=\text{middle}-1$ (tìm trong nửa đầu dãy)
 - Nếu $a[\text{middle}]<x$: tiếp tục tìm x trong dãy con mới với $\text{left}=\text{middle}+1$ (tìm trong nửa sau dãy)
- Bước 3:
 - Nếu $\text{left}<=\text{right}$: dãy còn phần tử, tiếp tục quay lại bước 2 để tìm kiếm tiếp
 - Ngược lại: Dãy hiện hành hết phần tử, dừng thuật toán.



Cài đặt thuật toán:

- Định nghĩa hàm **BinarySearch** tìm kiếm phần tử x trong mảng a có n phần tử. Kết quả hàm trả lại vị trí đầu tiên tìm thấy phần tử x hoặc trả lại giá trị -1 nếu không tìm thấy. Yêu cầu mảng a đã được sắp xếp tăng dần

```
int BinarySearch( int a[ ], int n, int x )
{
    int left=0, right=n-1;
    int middle;
    do
    {
        middle = (left+right)/2;
        if (x == a[middle]) break;
        else
            if (x<a[middle]) right= middle - 1;
            else left = middle + 1;
    } while ( left <= right );
    if ( left <= right )    return middle;
    else return -1; //ko tìm thấy phần tử x
}
```



Ví dụ minh họa

- Cho dãy số a:

2 3 4 5 6 7 8

- Tìm xem phần tử $x=8$ có trong dãy không?
- Yêu cầu chạy từng bước

left = 0; right = 6 => middle = 3

x=8



left = 4; right = 6 => middle = 5

x=8

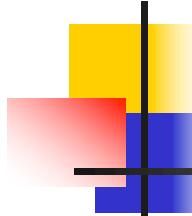


left = 6; right = 6 => middle = 6

x=8



Dùng do **left = right** => tìm thấy phần tử x trong dãy



Bài tập áp dụng:

- Tìm phần tử $x = 6$ trong dãy sau:

9 3 7 4 6 8 2

- Tìm phần tử $x = 4$ trong dãy sau:

1 4 6 8 9 10

Yêu cầu:

Áp dụng cả hai thuật toán tìm kiếm

II. Sắp xếp

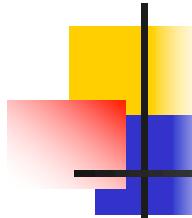
- Định nghĩa
- Các thuật toán sắp xếp:
 - Selection Sort
 - Insertion Sort
 - Quick Sort
 - Heap Sort
 - Bubble Sort
 - Merge Sort

1. Định nghĩa

- Cho một dãy số $a=\{a_1, a_2, \dots, a_n\}$ bất kỳ.
- Sắp xếp là quá trình bố trí lại các phần tử của một tập đối tượng theo một thứ tự nào đó.
- Sắp xếp trong (sắp xếp nội) là phương pháp sắp xếp thực hiện trực tiếp trên dữ liệu được lưu trữ trong vùng nhớ chính của máy tính (Sắp xếp trên mảng).

2. Các thuật toán cơ bản

- Sắp xếp chọn trực tiếp- Selection Sort
- Sắp xếp chèn trực tiếp- Insertion Sort
- Sắp xếp nổi bọt- Bubble Sort
- Sắp xếp nhanh- Quick Sort
- Sắp xếp vun đống- Heap Sort
- Sắp xếp trộn – Merge Sort



Bài toán

- Cho dãy a gồm n phần tử $\{a_1, a_2, \dots, a_n\}$.
Sắp xếp các phần tử theo thứ tự tăng dần
- Input:
 - n: số phần tử của mảng
 - $a=\{a_1, a_2, \dots, a_n\}$: Mảng chứa các phần tử bất kỳ
- Output: Mảng được sắp xếp tăng dần.

1. Sắp xếp chọn trực tiếp Selection Sort

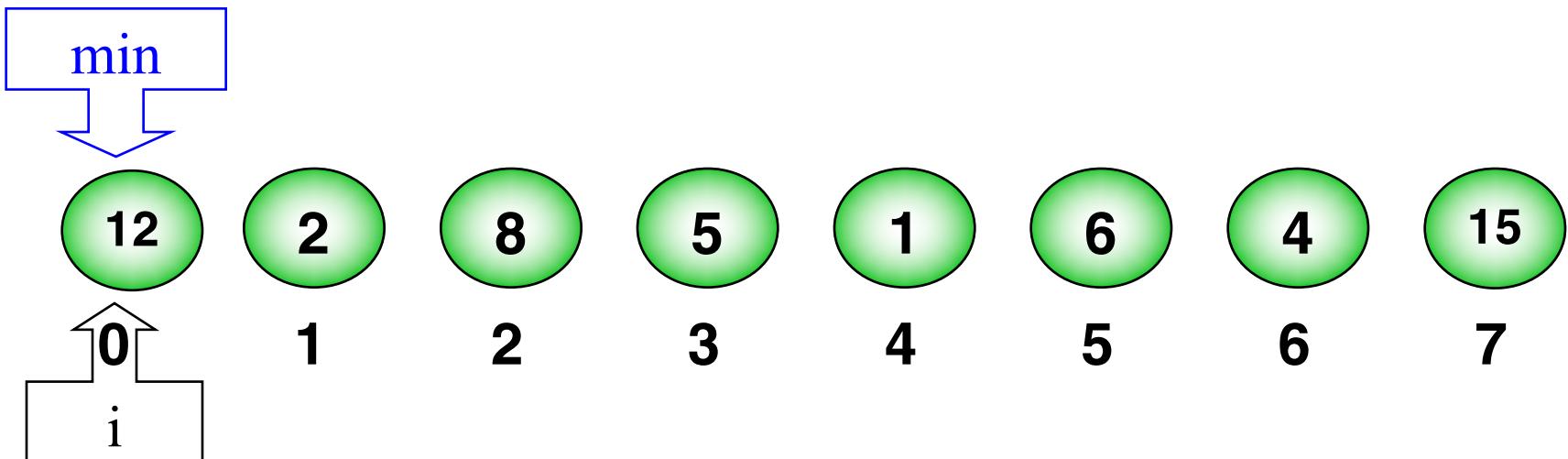
■ Ý tưởng:

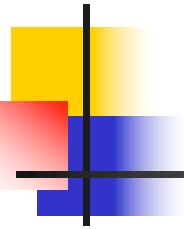
- Tại mỗi bước chọn ra phần tử nhỏ nhất trong số phần tử chưa xét và đưa vào vị trí thích hợp, cố định phần tử này không xét lại nữa.
- Dãy ban đầu có n phần tử, thuật toán thực hiện $n-1$ lần để đưa phần tử nhỏ nhất trong dãy hiện hành về vị trí đúng ở đầu dãy

Minh họa thuật toán Selection Sort

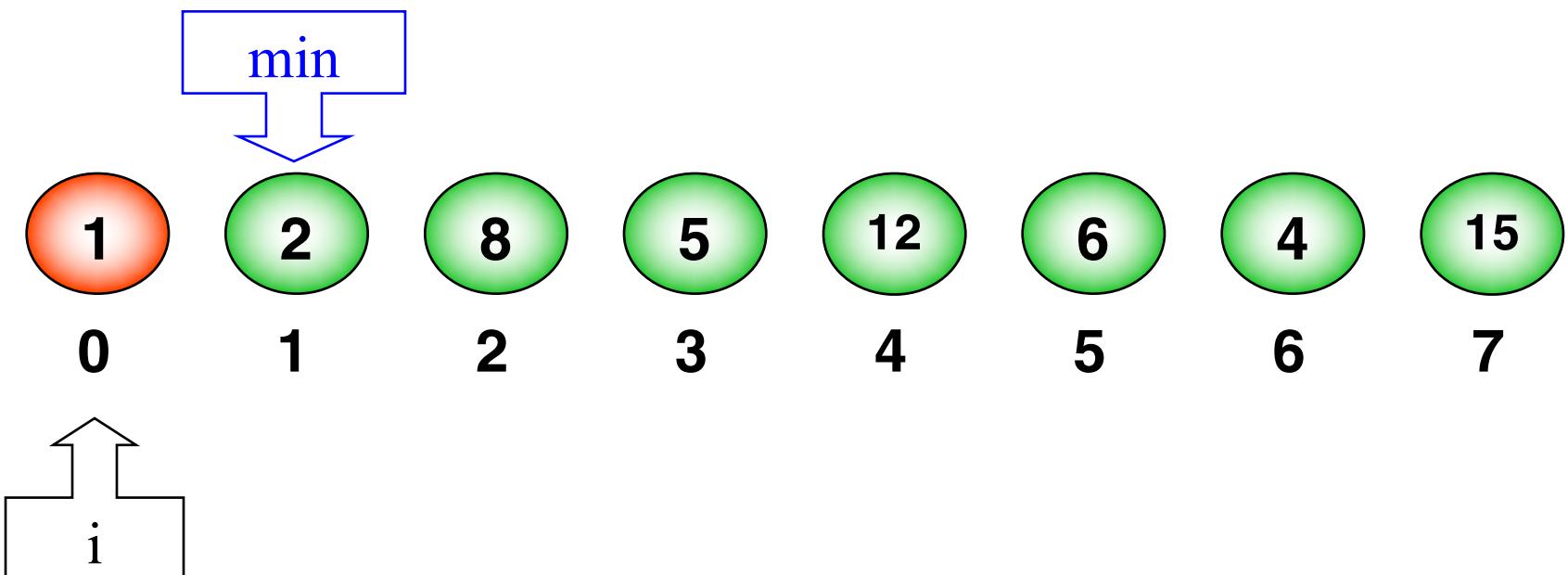
Tìm vị trí chứa giá trị nhỏ nhất tron

Hoandoi(a[0], a[4])

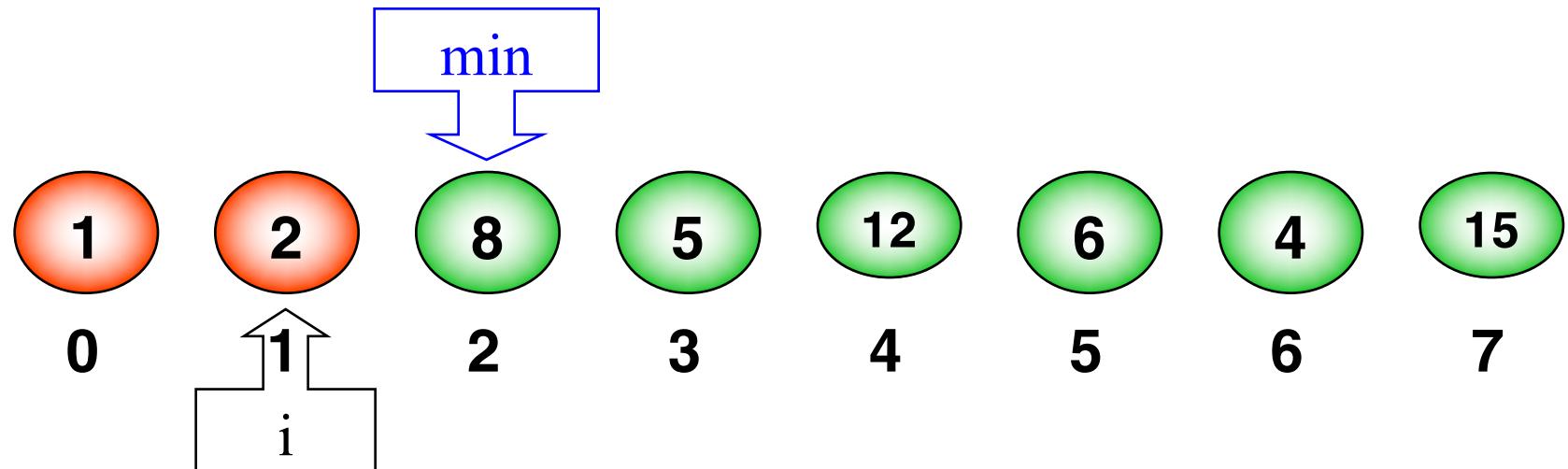




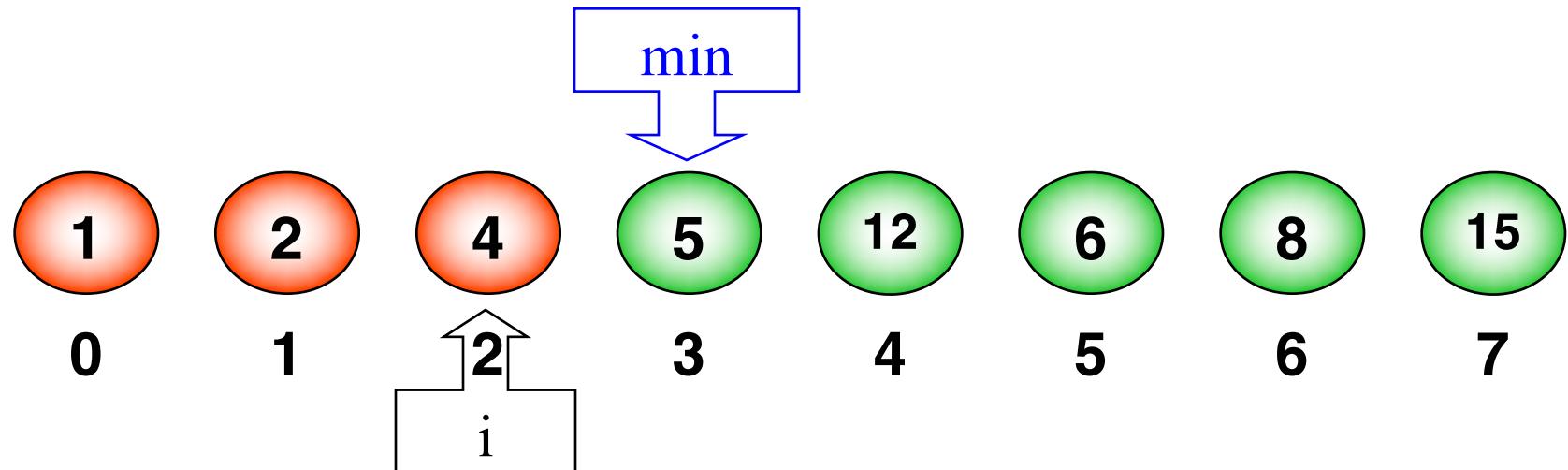
Tìm vị trí chứa giá trị nhỏ nhất trong **Hoandoi(a[1], a[1])**



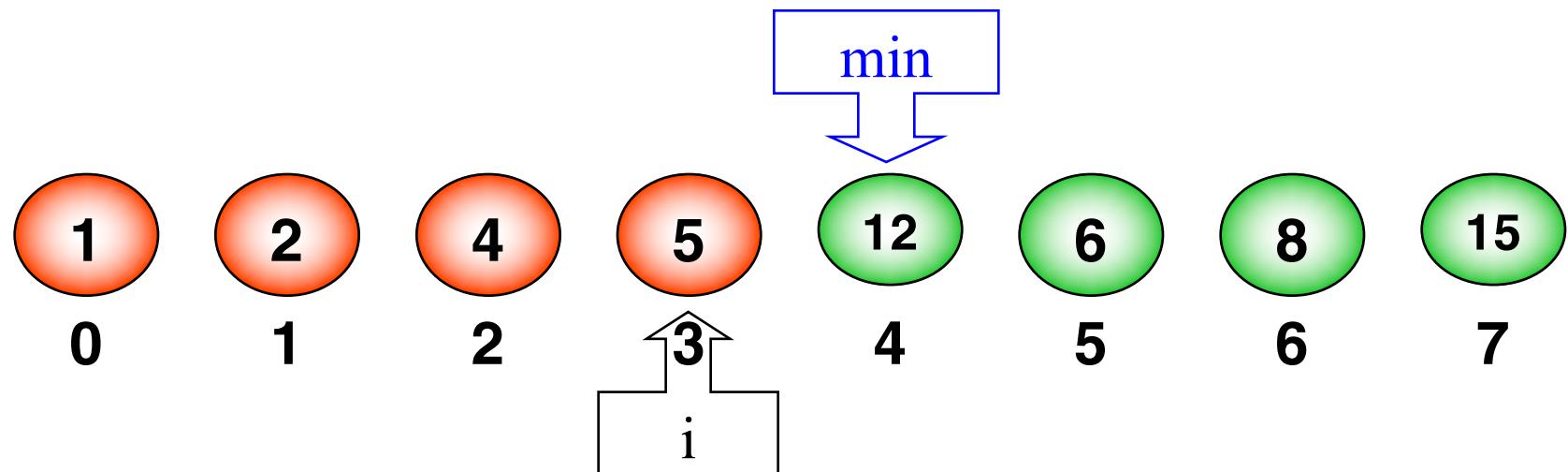
Tìm vị trí chứa giá trị nhỏ nhất trong **Hoandoi(a[2], a[6])**



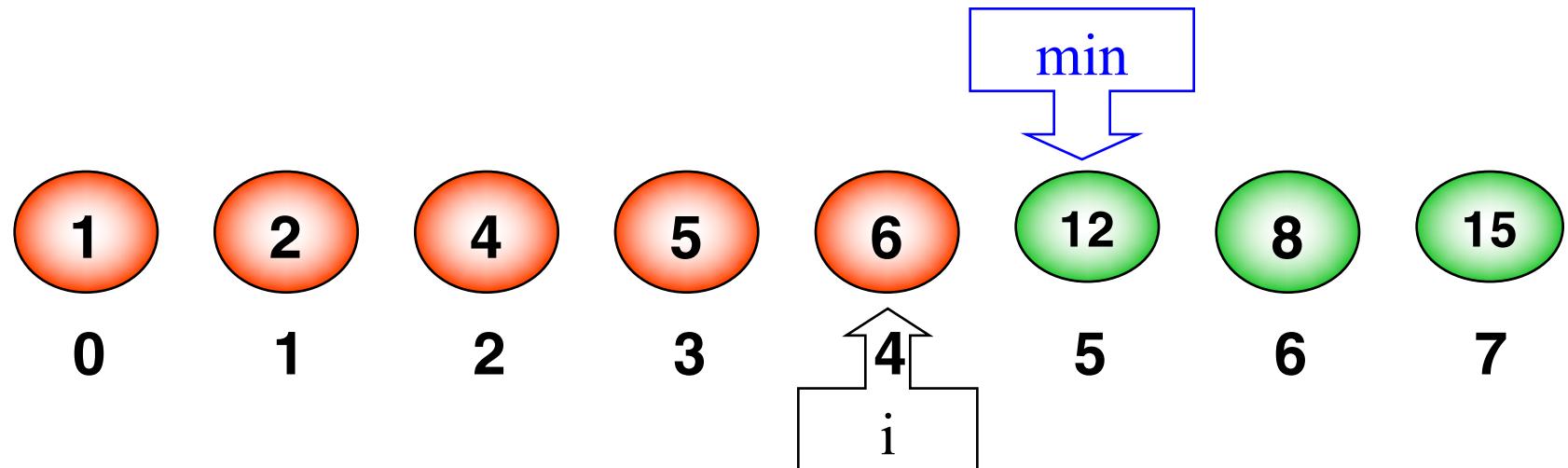
Tìm vị trí chứa giá trị nhỏ nhất trong **Hoandoi(a[3], a[3])**



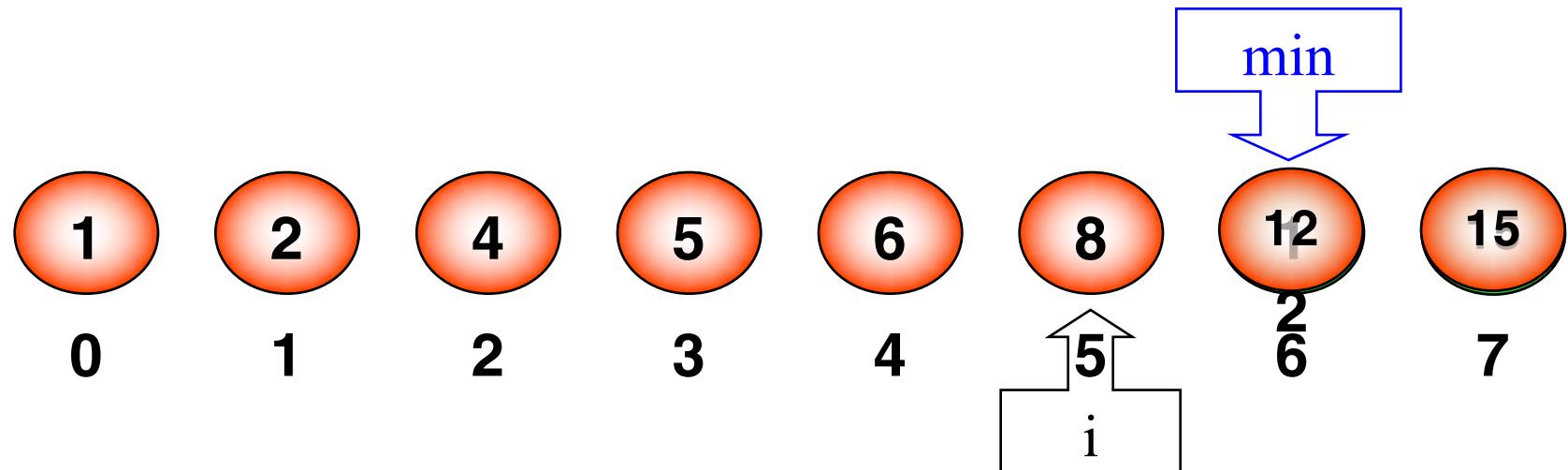
Tìm vị trí chứa giá trị nhỏ nhất trong **Hoandoi (a[4], a[5])**

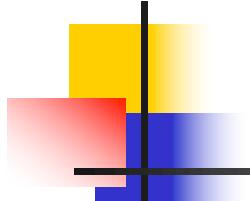


Tìm vị trí chứa giá trị nhỏ nhất tro **Hoandoi(a[5], a[6])**



Tìm vị trí chứa giá trị nhỏ nhất trong đoạn(6, 7)





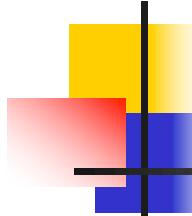
Mô tả thuật toán:

- Bước 1: $i = 0$ *(bắt đầu từ vị trí đầu tiên)*
- Bước 2: tìm chỉ số phần tử **min** nhỏ nhất trong dãy hiện hành từ **a[i]** đến **a[n-1]**
- Bước 3: Hoán vị **a[i]** với **a[min]**
- Bước 4:
 - nếu $i < n-1$ thì $i = i+1$ và lặp lại bước 2
 - ngược lại thì $n-1$ phần tử đã được sắp xếp => Dừng thuật toán

Cài đặt thuật toán:

```
void SelectionSort( int a[ ], int n )
{
    int min, i, j, tg;
    for( i=0 ; i<n-1 ; i++ )
    {
        min = i;
        for( j=i+1 ; j<n ; j++ )
            if ( a[j] < a[min] ) min=j;

        if( min != i ) //Hoán đổi a[min] với a[i]
        {
            tg=a[min] ; a[min]=a[i] ; a[i]=tg;
        }
    }
}
```



Bài tập áp dụng

- Cho dãy số sau:

5 6 2 2 10 12 9 10 9 3

- Yêu cầu sắp xếp dãy tăng dần bằng thuật toán **SelectionSort**

Mô tả bảng

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	6	<u>2</u>	2	10	12	9	10	9	3
2	6	5	<u>2</u>	10	12	9	10	9	3
	2	5	6	10	12	9	10	9	<u>3</u>
		3	6	10	12	9	10	9	<u>5</u>
			5	10	12	9	10	9	<u>6</u>
				6	12	<u>9</u>	10	9	10
					9	12	10	<u>9</u>	10
						9	<u>10</u>	12	10
							10	12	<u>10</u>
								10	12
<u>2</u>	2	3	5	6	9	9	10	10	12

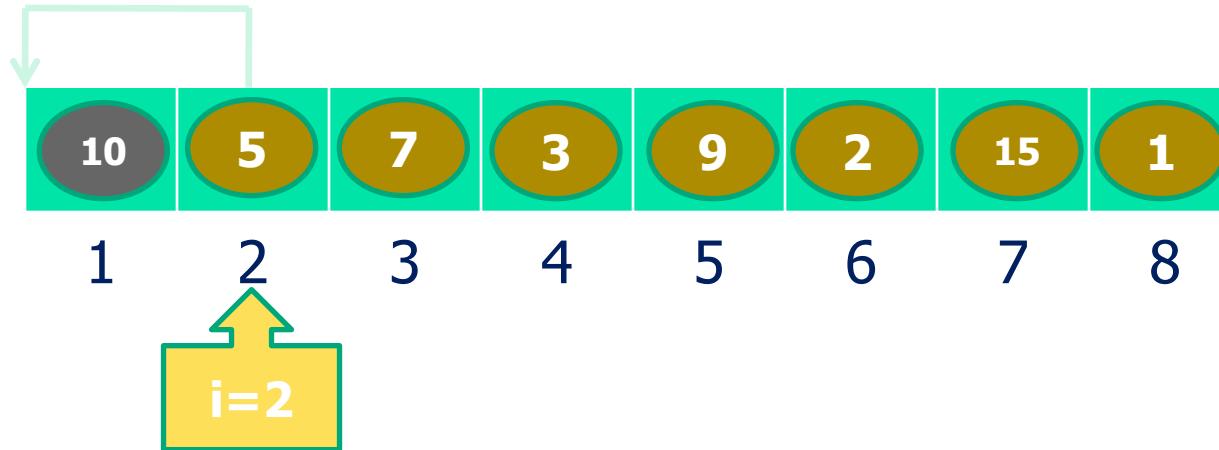
2. Sắp xếp chèn trực tiếp- Insertion sort

■ Ý tưởng:

- Cho dãy ban đầu: a_1, a_2, \dots, a_n .
- Ta có thể xem đoạn a_1 đã được sắp, sau đó thêm a_2 vào đoạn a_1 sẽ có đoạn a_1, a_2 được sắp. Tiếp tục thêm cho đến khi thêm a_n vào dãy a_1, a_2, \dots, a_{n-1} ta có dãy a_1, a_2, \dots, a_n được sắp.

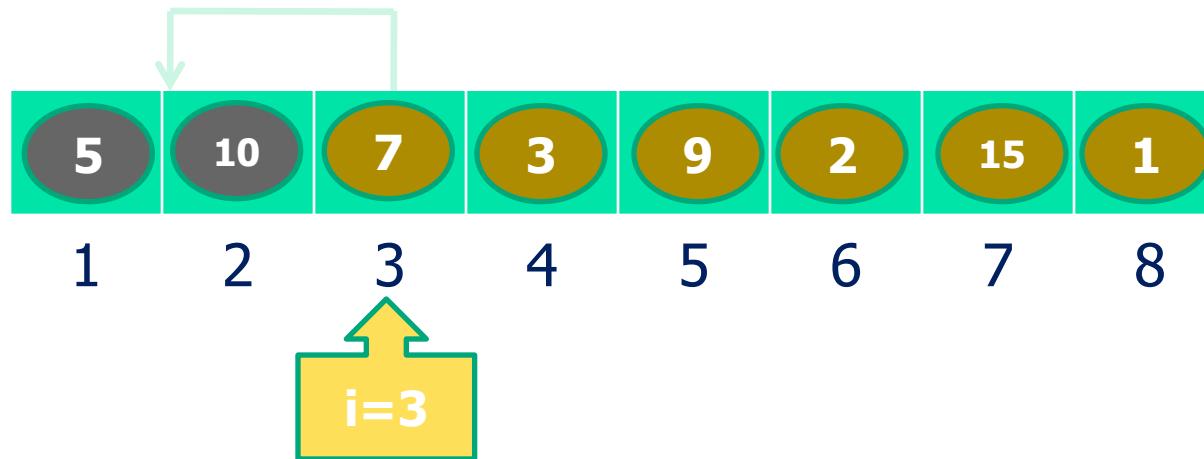
Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 2



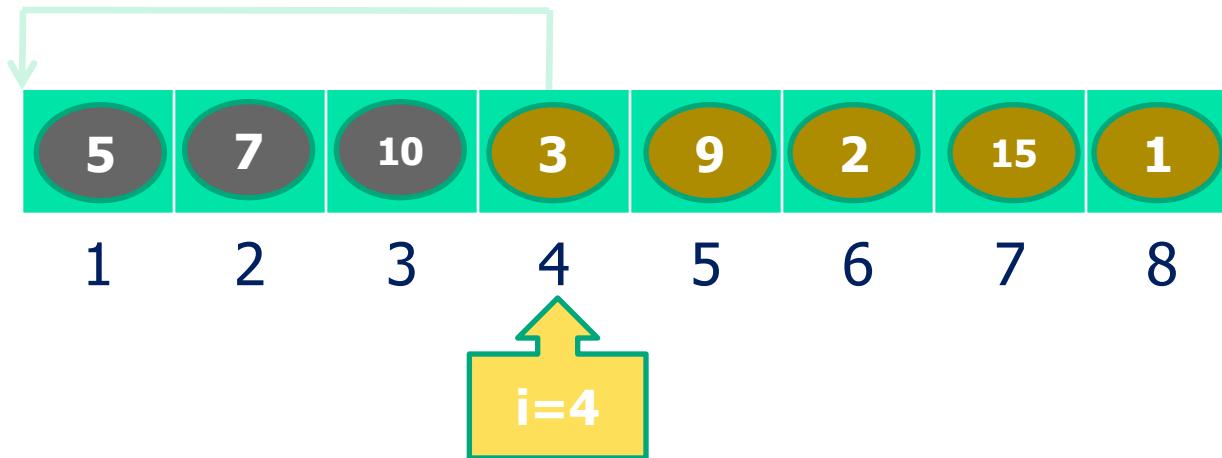
Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 3



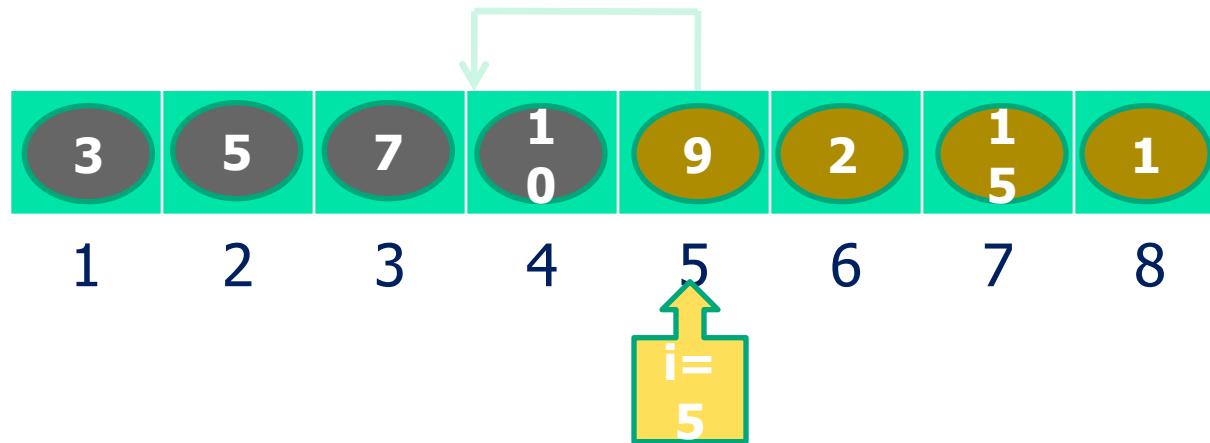
Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 4

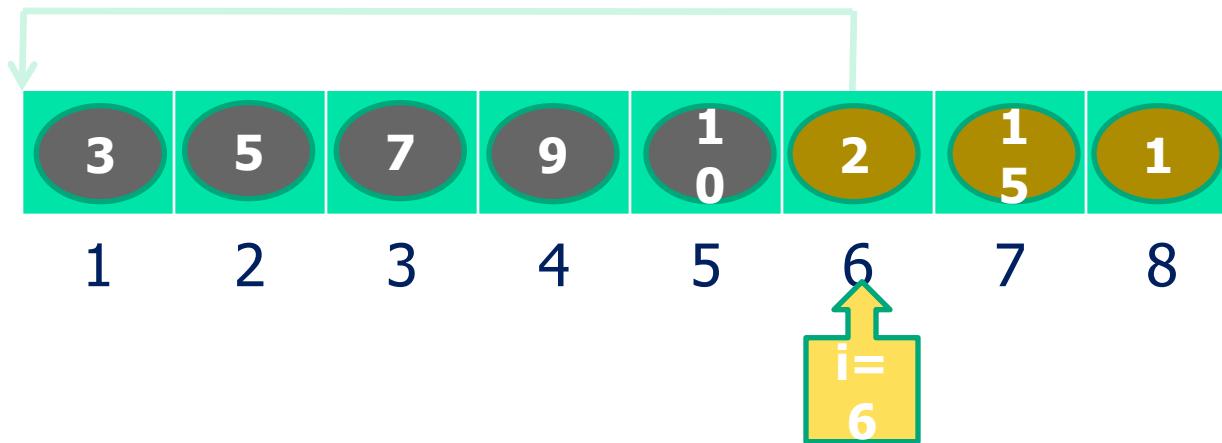


Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 5

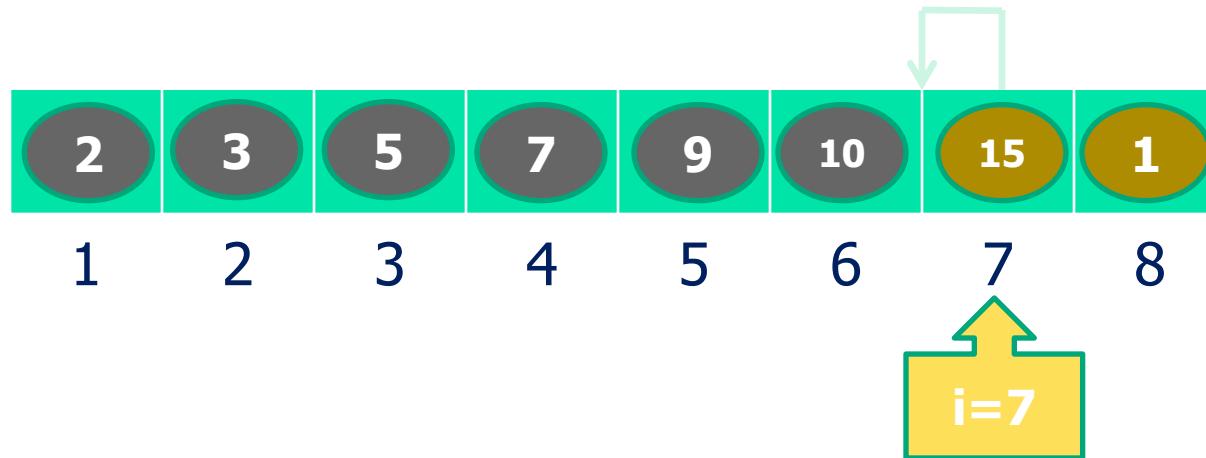


Tìm vị trí chèn cho phần tử thứ 6



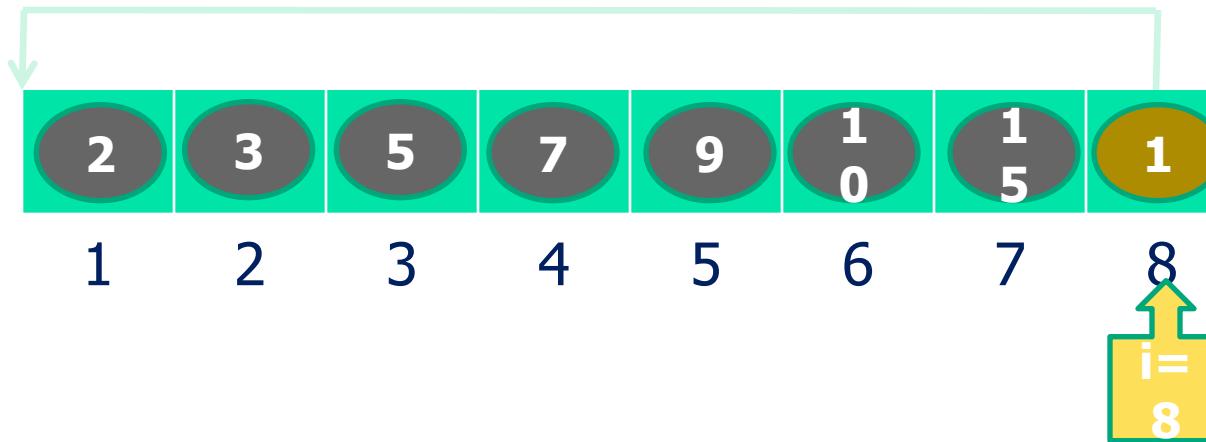
Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 7



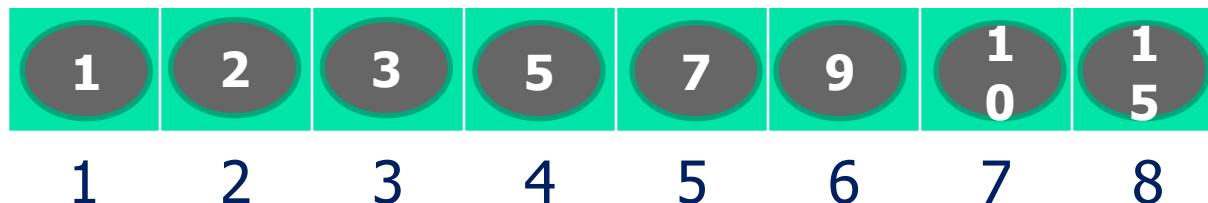
Minh họa thuật toán

Tìm vị trí chèn cho phần tử thứ 8

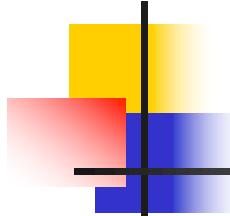


Minh họa thuật toán

Kết thúc



i=9



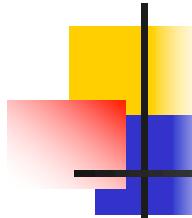
Mô tả thuật toán

- Bước 1: $i = 1$ // giả sử $a[0]$ đã được sắp xếp
- Bước 2: $x = a[i]$, tìm vị trí **pos** thích hợp trong đoạn từ $a[0]$ đến $a[i-1]$ để chèn $a[i]$ vào
- Bước 3: Dịch chuyển các phần tử từ **$a[pos]$** đến **$a[i-1]$** sang phải một vị trí để được vị trí chèn $a[i]$ vào
- Bước 4: chèn **$a[i]$** vào vị trí **pos** vừa tìm được bằng cách gán **$a[pos] = x$**
- Bước 5: $i = i+1$
 - Nếu $i < n \Rightarrow$ lặp lại bước 2
 - Ngược lại \Rightarrow Dừng thuật toán

Cài đặt thuật toán

```
void InsertionSort( int a[ ], int n )  
{int pos,i,x;  
for(i=1;i<n;i++)  
{  
    x=a[i];  
    pos=i-1;  
    while( (pos>=0) && (a[pos]>x) )  
        //tìm vị trí chèn a[i]  
    {  
        a[pos+1]=a[pos];  
        pos--;  
    }  
    a[pos+1]=x;          //chèn x vào dãy  
}  
}
```

Vừa tìm vị trí để chèn vừa dịch chuyển các phần tử ra cuối dãy một vị trí



Bài tập áp dụng

- Cho dãy số sau:

5 6 2 2 10 12 9 10 9 3

- Yêu cầu sắp xếp dãy tăng dần bằng thuật toán InsertionSort

Insertion

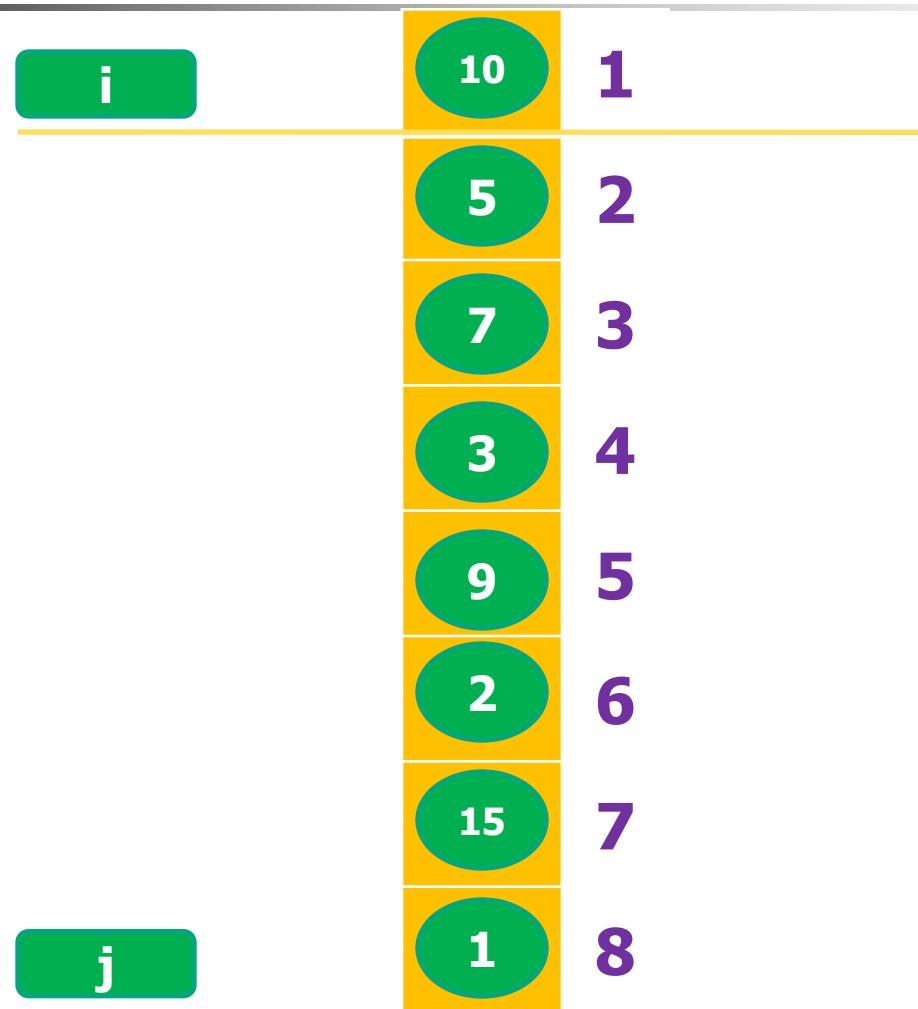
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	6	2	2	10	12	9	10	9	3
5	6								
<u>2</u>	5	6							
2	<u>2</u>	5	6						
2	2	5	6	<u>10</u>					
2	2	5	6	10	<u>12</u>				
2	2	5	6	<u>9</u>	10	12			
2	2	5	6	9	10	<u>10</u>	12		
2	2	5	6	9	<u>9</u>	10	10	12	
2	2	<u>3</u>	5	6	9	9	10	10	12

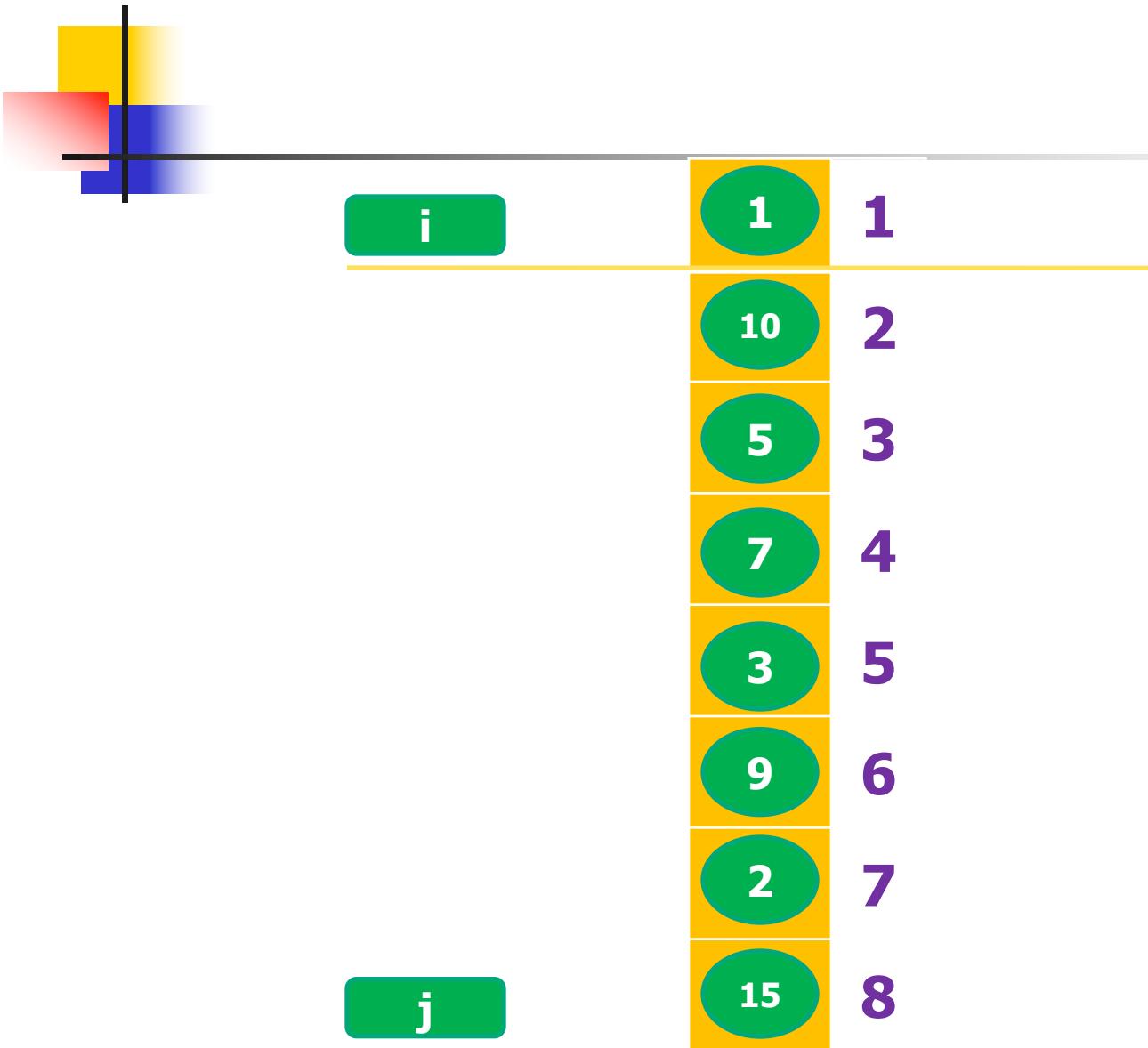
3. Sắp xếp nổi bọt-BubbleSort

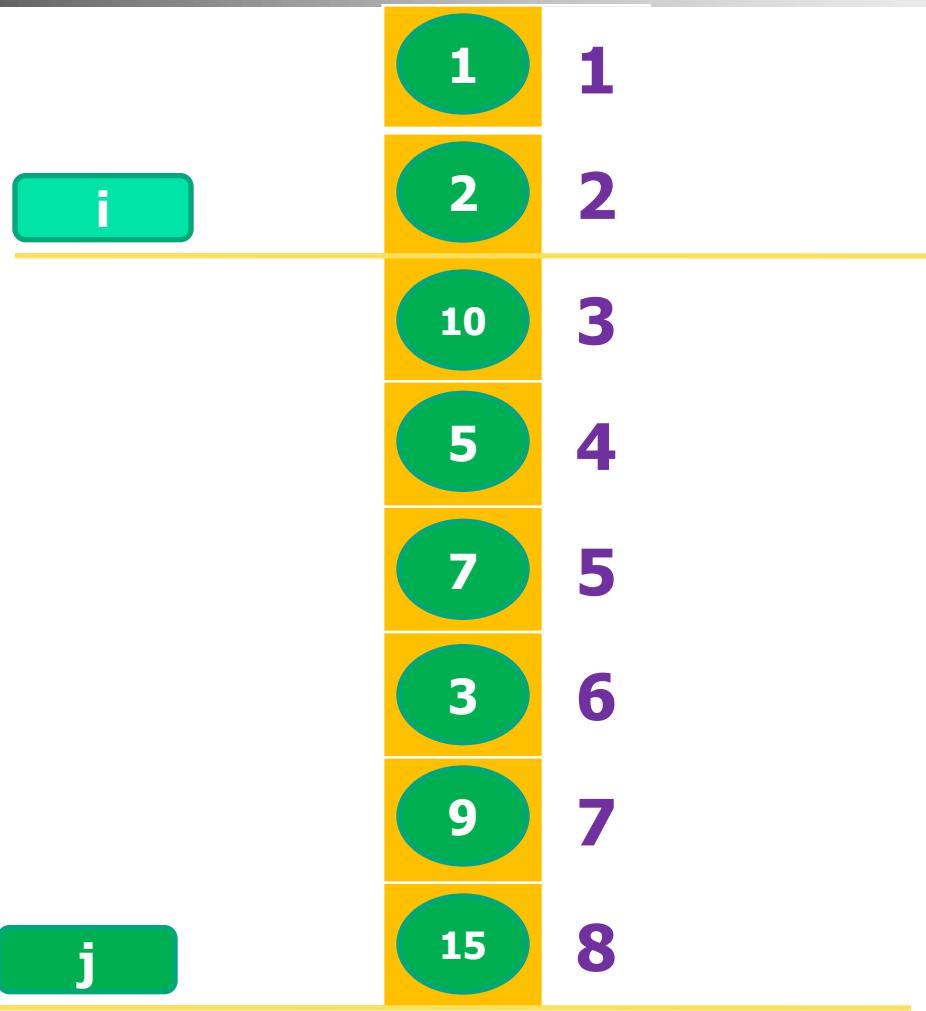
■ Ý tưởng:

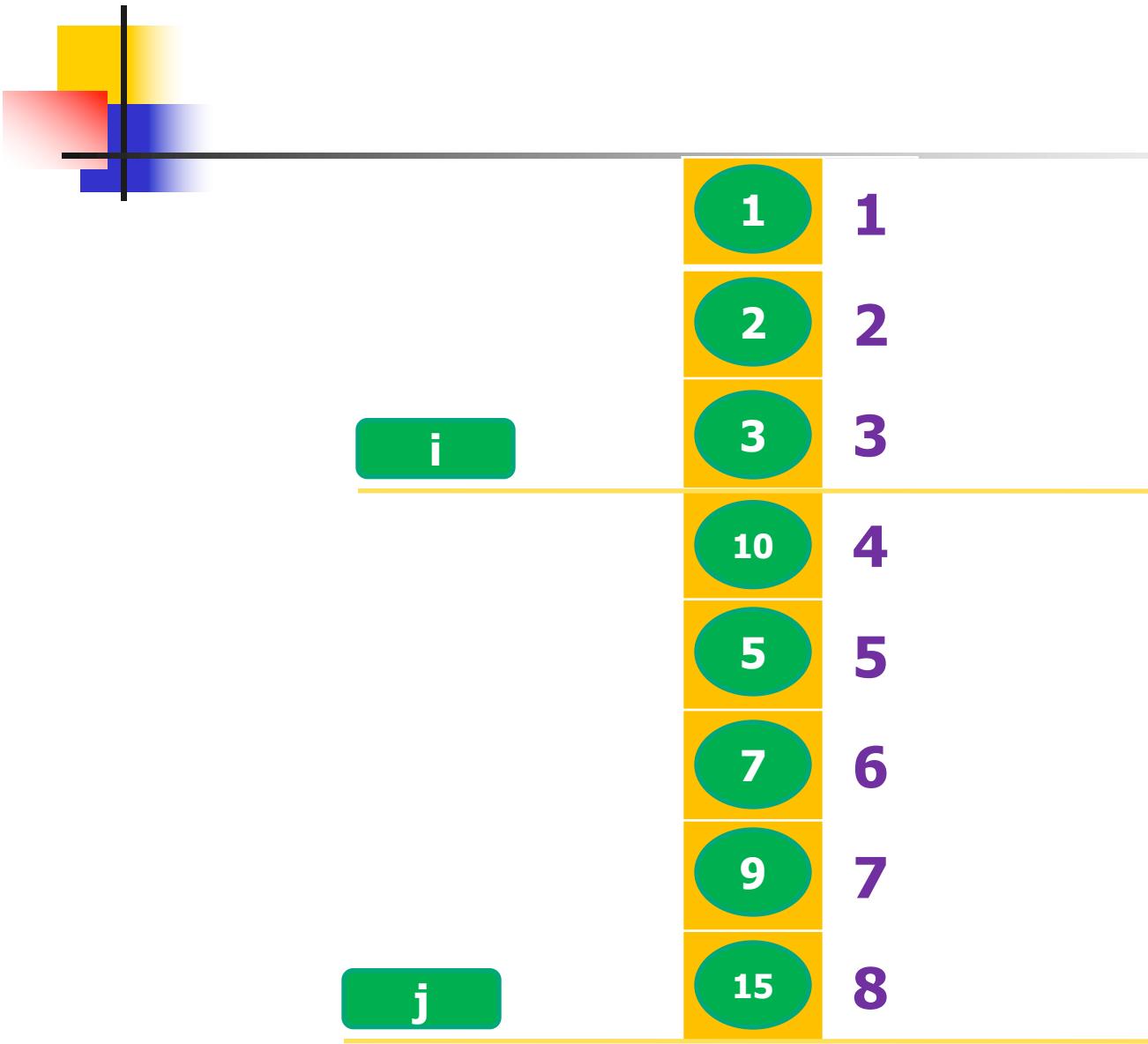
- Xuất phát từ cuối dãy, tiến hành đổi chỗ cặp phần tử đứng cạnh nhau trong danh sách để đưa phần tử nhỏ hơn về vị trí thấp hơn trong danh sách. Quá trình đổi chỗ dừng khi phần tử nhỏ hơn được đưa về đầu dãy hiện hành.
- Lặp lại quá trình này với dãy mới, không kể phần tử đầu dãy hiện hành, cho đến khi dãy hiện hành không còn phần tử nào nữa.

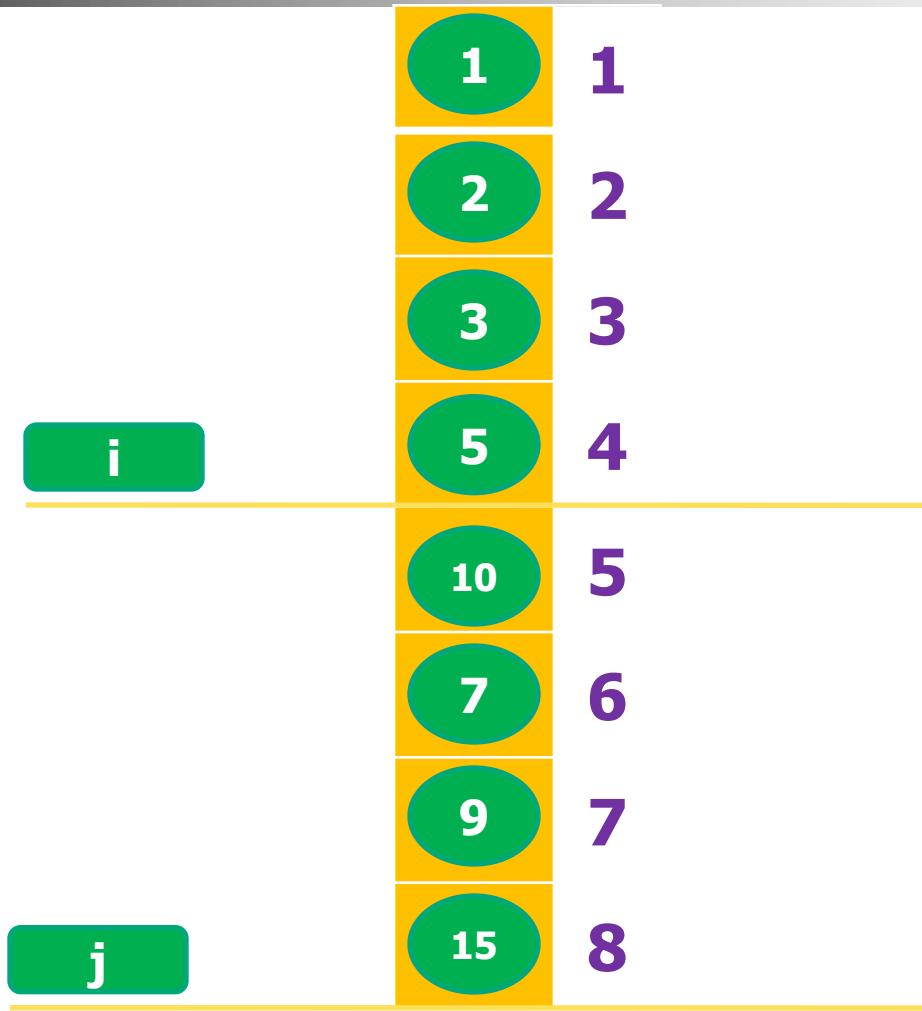
b. Minh Họa Thuật Toán Nối Bọt

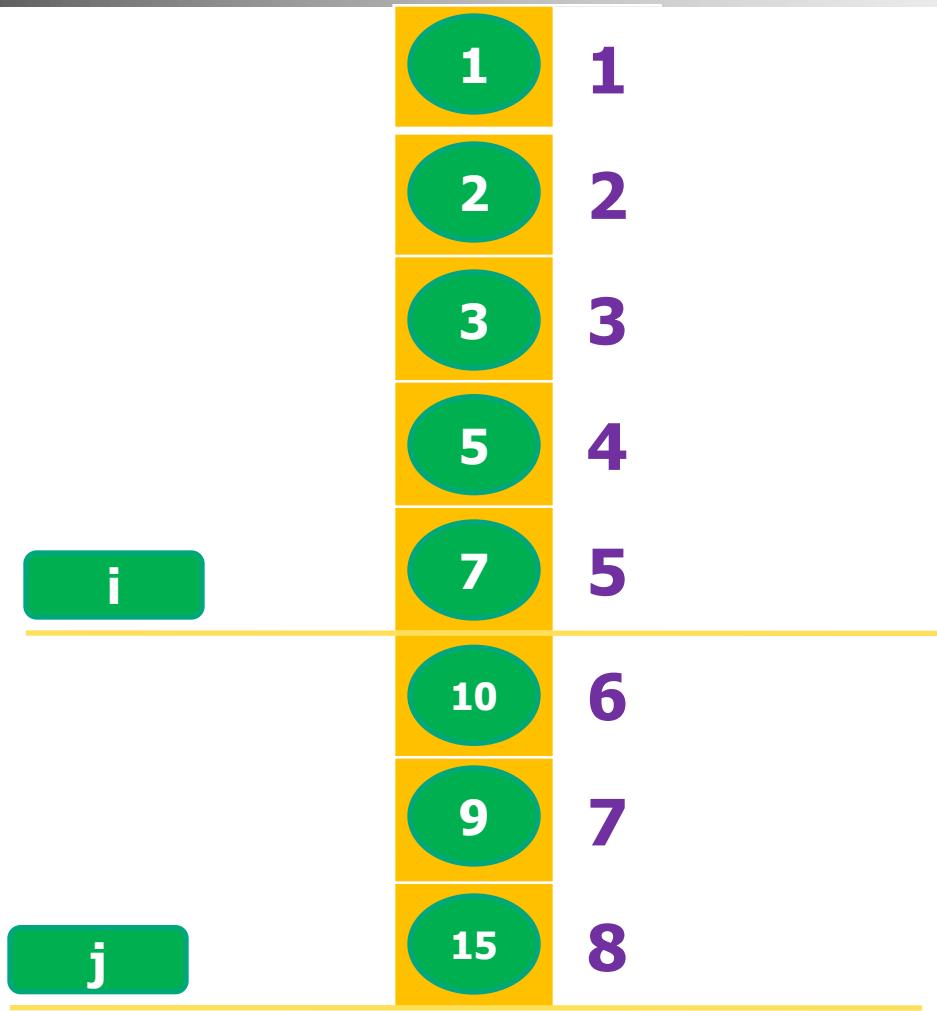


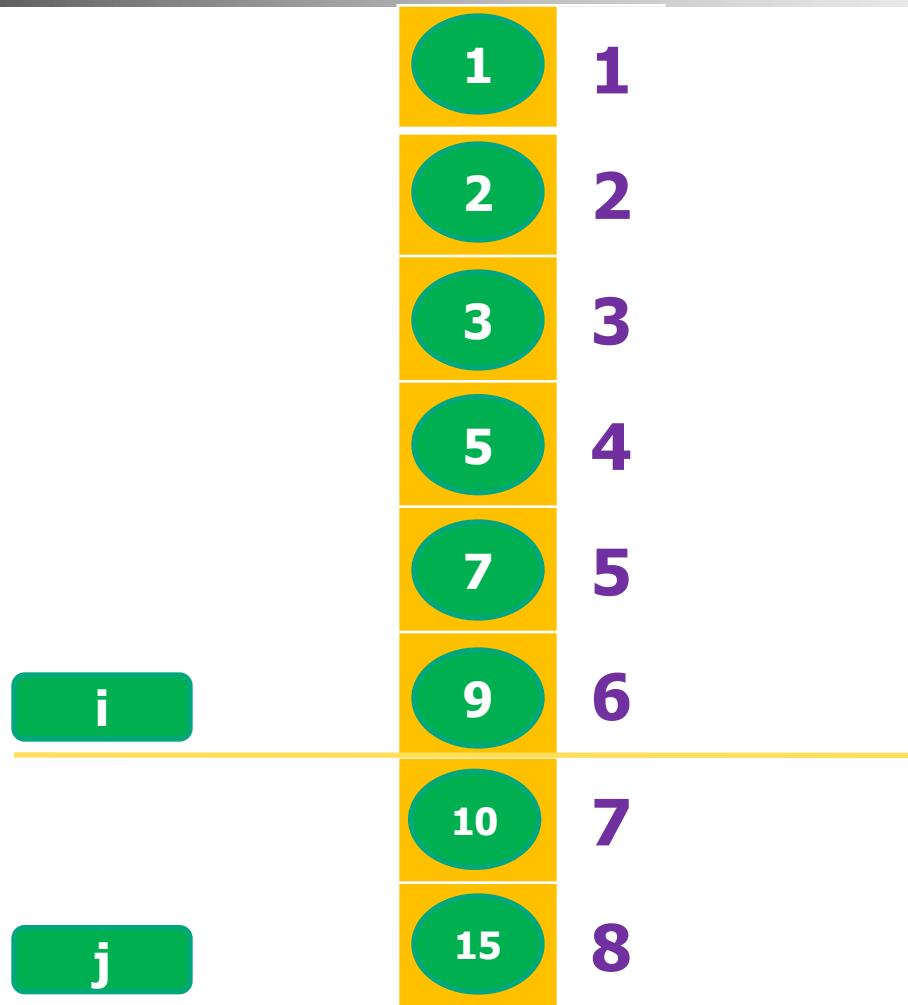


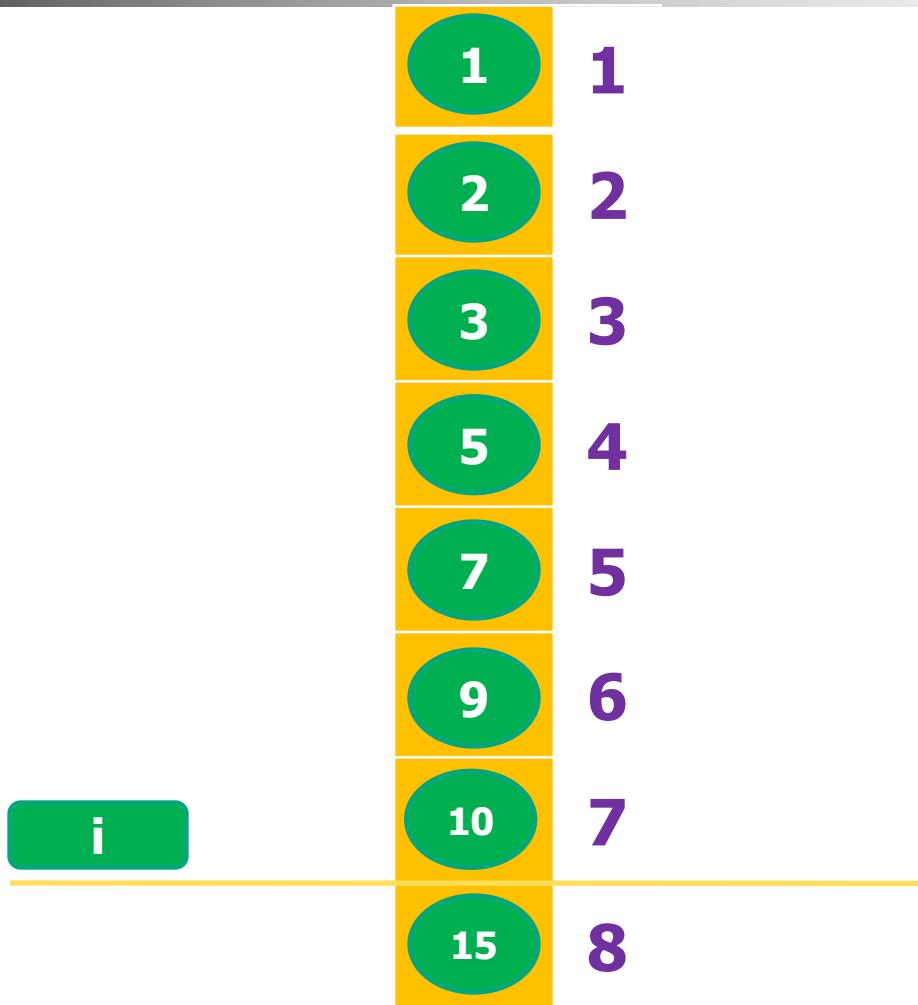


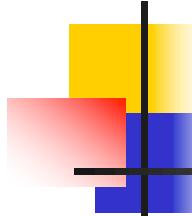






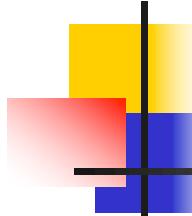






Mô tả thuật toán

- Bước 1: $i = 0$
- Bước 2: $j = n - 1$ // duyệt từ cuối đến ptử thứ i
- Trong khi $j > i$ thực hiện
 - Nếu $a[j] < a[j-1]$ thì hoán đổi hai phần tử
 - $j = j - 1$
- Bước 3: $i = i + 1$
 - Nếu $i > n-1 \Rightarrow$ Hết dãy và dừng thuật toán
 - Ngược lại lặp lại bước 2



Bài tập áp dụng

- Cho dãy số sau:

5 6 2 2 10 12 9 10 9 3

- Yêu cầu sắp xếp dãy tăng dần bằng thuật toán BubbleSort

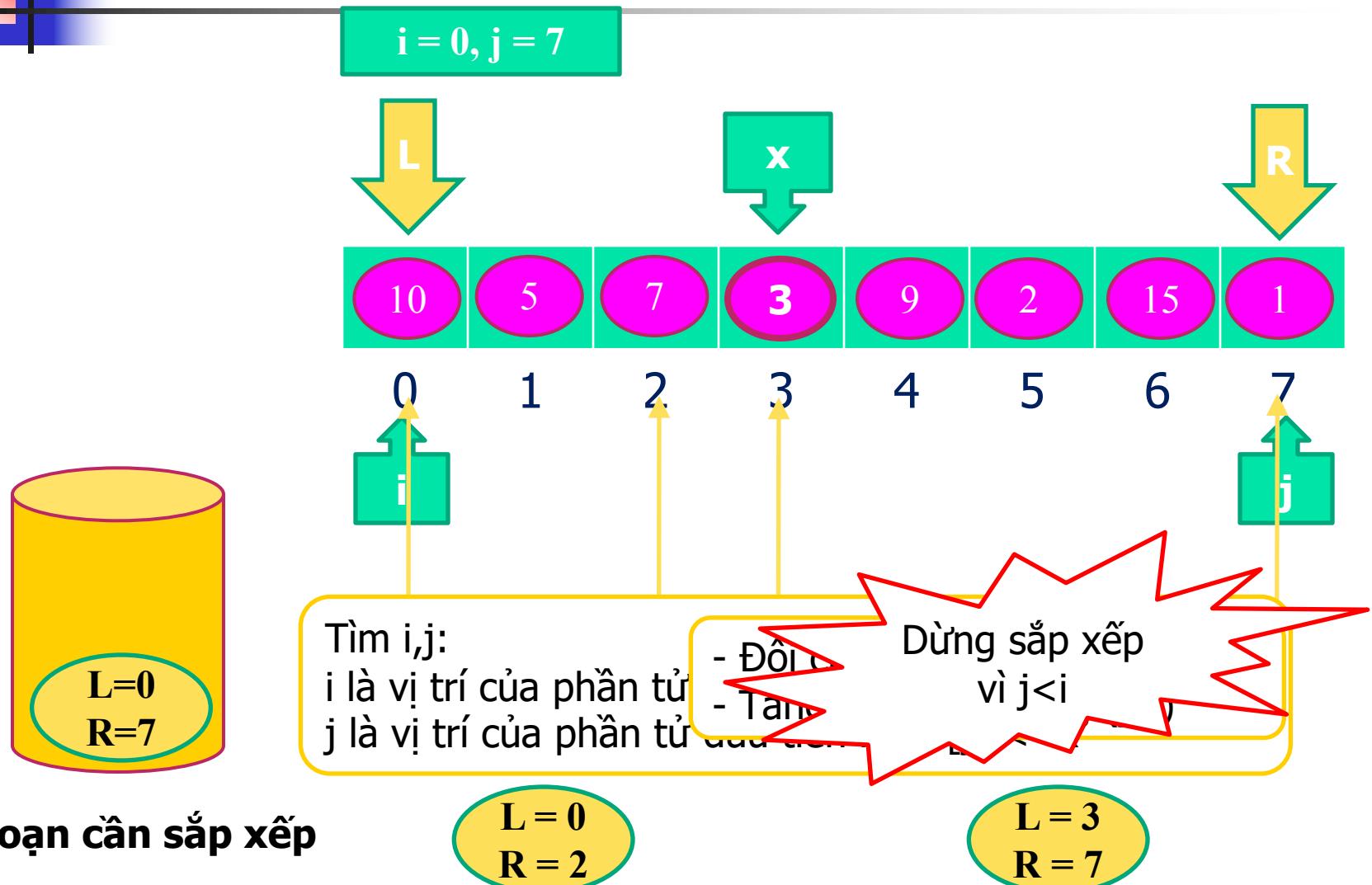
Thuật toán Bubble Sort

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
5	6	2	2	10	12	9	10	9	3
2	5	6	2	3	10	12	9	10	9
	2	5	6	3	9	10	12	9	10
		3	5	6	9	9	10	12	10
			5	6	9	9	10	10	12
				6	9	9	10	10	12
					9	9	10	10	12
						9	10	10	12
							10	10	12
								10	12
2	2	3	5	6	9	9	10	10	12 ⁵⁹

4. Sắp xếp nhanh - QuickSort

- Ý tưởng: Chọn một phần tử ngẫu nhiên nào đó của dãy làm “chốt”, mọi phần tử nhỏ hơn chốt được xếp vào vị trí trước chốt, mọi phần tử lớn hơn chốt được xếp vào vị trí sau chốt.

Minh họa thuật toán

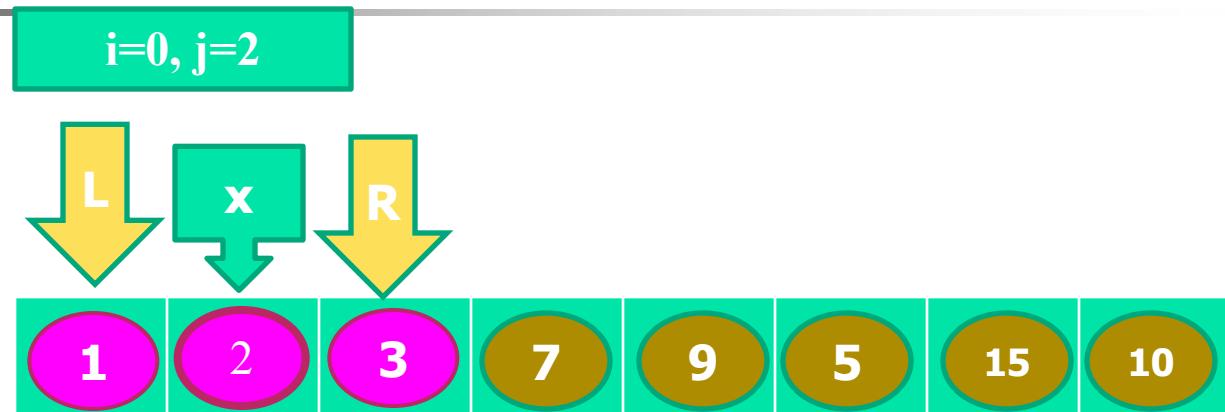
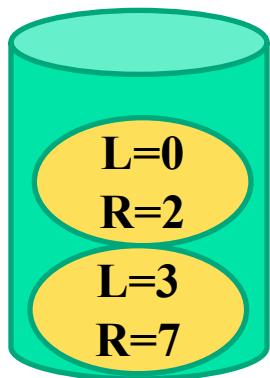


Đoạn cần sắp xếp

$i < j$, ta có đoạn cần sắp xếp mới với $L=0, R=2$

$L = 3$
 $R = 7$

$i < r$, ta có đoạn cần sắp xếp mới với $L=3, R=7$



1 2 3 4 5 6 7 8

i

j

- Đổi chỗ $a[i], a[j]$
- Tăng i ($i++$), giảm j ($j--$)

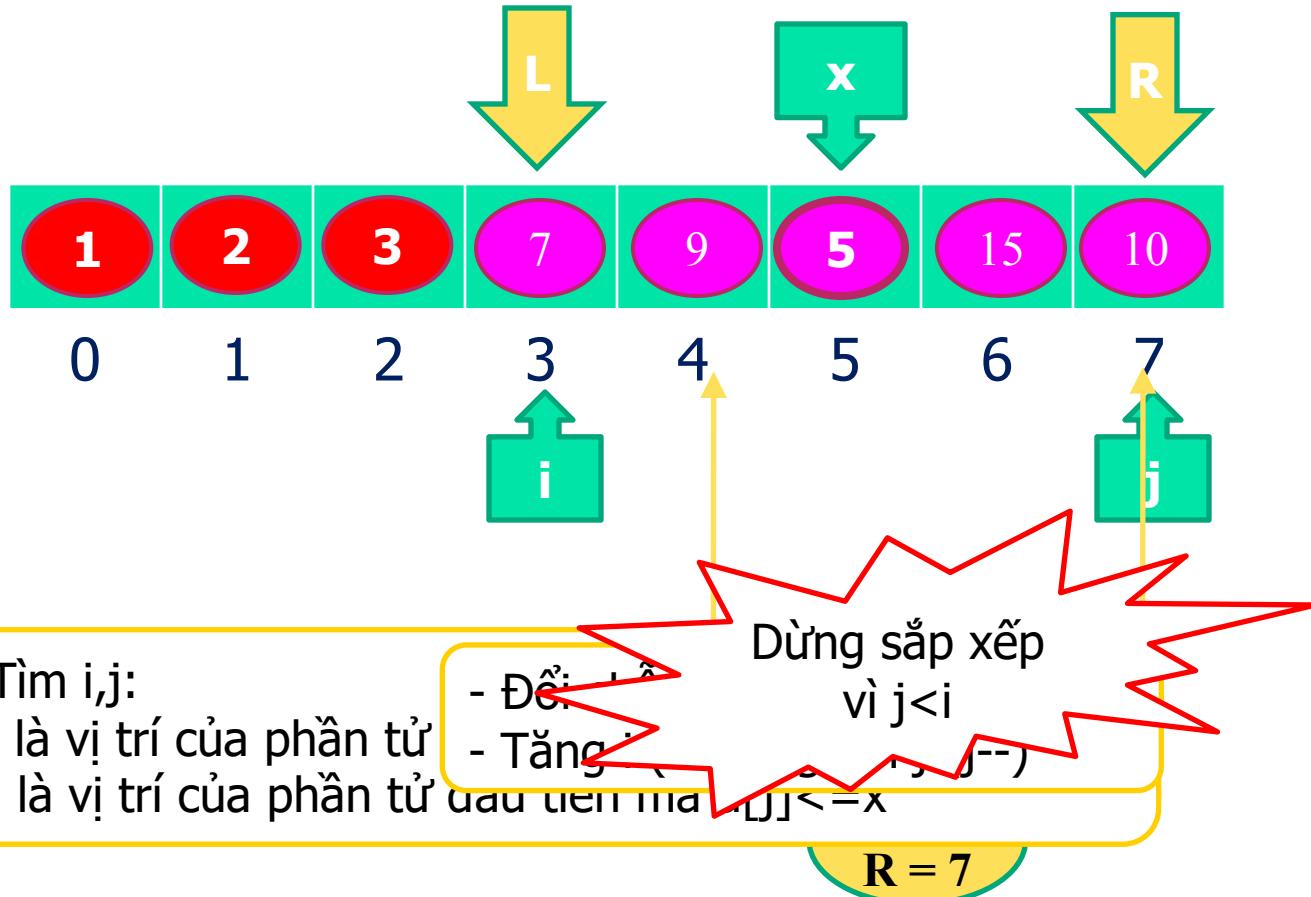
Dừng sắp xếp
vì $j < i$

Tìm i, j :
 i là vị trí của phần tử đầu tiên mà $a[i] >= x$
 j là vị trí của phần tử đầu tiên mà $a[j] <= x$

Đoạn cân sắp xếp

Minh họa thuật toán

$i = 3, j = 7$



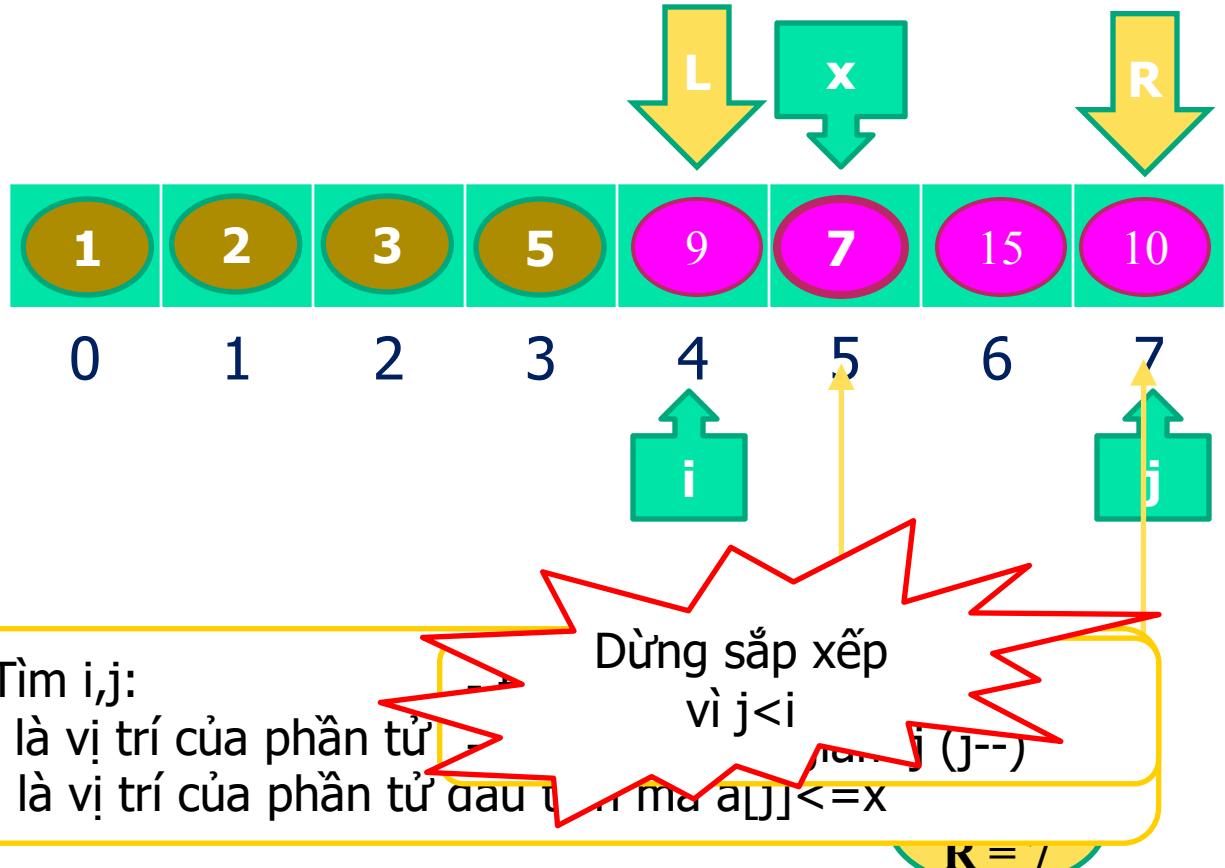
Đoạn cần sắp xếp

GV Nguyễn Thị Quỳnh Như- Email: ntqnhu@hou.edu.vn

$i < r$, ta có đoạn cần sắp xếp mới
với $L=4, R=7$

Minh họa thuật toán

$i = 4, j = 7$



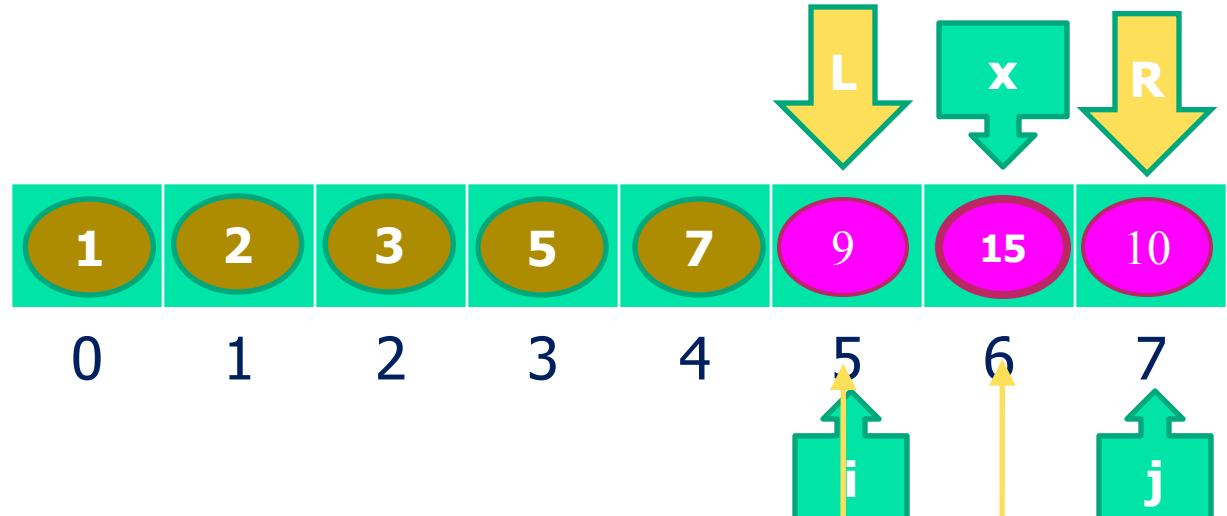
Đoạn cần sắp xếp

$i < r$, ta có đoạn cần sắp xếp mới

GV Nguyễn Thị Quỳnh Như- Email: ntqnhu@hou.edu.vn

Minh họa thuật toán

$i = 5, j = 7$



Tìm i, j :
i là vị trí của phần tử
j là vị trí của phần tử đầu tiên

- Đón
- Tạo

Dừng sắp xếp
vì $j < i$

Đoạn cần sắp xếp

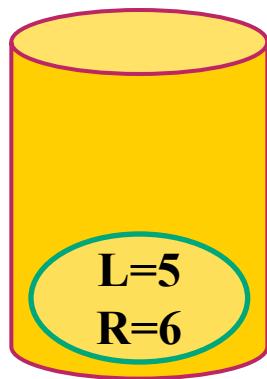
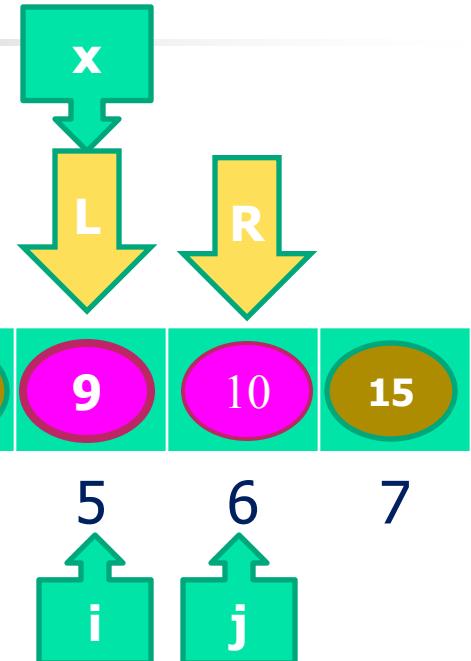
$i < r$, ta có đoạn cần sắp xếp mới

GV Nguyễn Thị Quỳnh Như- Email: ntqnhu@hou.edu.vn

với $L=5, R=6$

Minh họa thuật toán

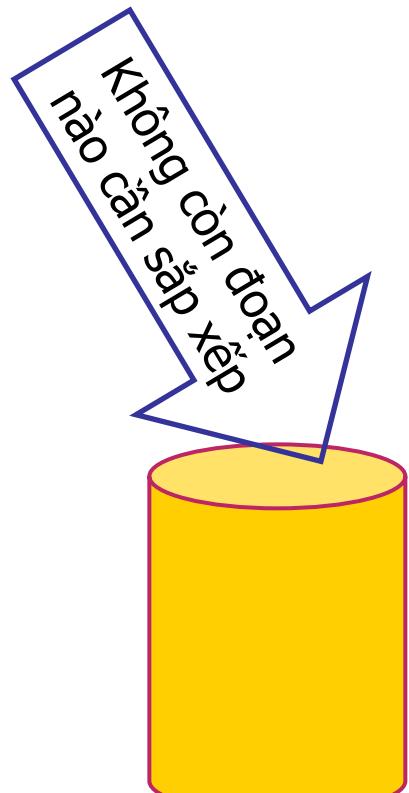
$i = 5, j = 6$



Dừng sắp xếp
vì $j < i$
- Tăng i ($i++$), giảm j ($j--$)
L là vị trí
j là vị trí của phần tử đầu tiên mà $a[j] \leq x$

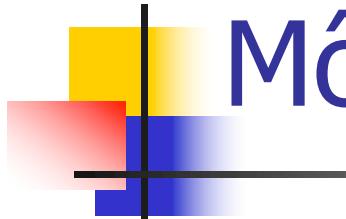
Đoạn cần sắp xếp

Minh họa thuật toán



Kết thúc!

Đoạn cần sắp xếp



Mô tả thuật toán

- Được thực hiện gồm hai bước:
 - Phân hoạch bài toán
 - Sắp xếp trên từng phân hoạch
- **Thuật toán:** Cho dãy a_L, a_{L+1}, \dots, a_R

Bước 1: Phân hoạch dãy $a_L \dots a_R$ thành các dãy con:

- Dãy con 1: $a_L \dots a_j < x$
- Dãy con 2: $a_{j+1} \dots a_{i-1} = x$
- Dãy con 3: $a_i \dots a_R > x$

Bước 2: Sắp xếp trên từng phân hoạch

- Nếu ($L < j$) Phân hoạch dãy $a_L \dots a_j$
- Nếu ($i < R$) Phân hoạch dãy $a_i \dots a_R$

Phân hoạch bài toán

Bước 1: Phân hoạch dãy (a_1, \dots, a_n), với $L=1$, $R=n$

- Bước 1.1:
 - Chọn tùy ý phần tử chốt $x=a[k]$ trong dãy a_1, a_2, \dots, a_n (thường chọn chốt là phần tử giữa $k=(l+r)/2$)
 - $i=l$, $j=r$;
- Bước 1.2: Phát hiện và điều chỉnh các phần tử $a[i]$ và $a[j]$ sai vị trí:
 - Chừng nào $a[i] < x$ thì $i++$
 - Chừng nào $a[j] > x$ thì $j--$
 - Nếu $i \leq j$ thì:
 - đổi chỗ $a[i]$ với $a[j]$.
 - $i++, j--$
- Bước 1.3: Nếu $i < j$ thì quay lại bước 1.2, ngược lại thì sang bước 2

■ Bước 2:

- Nếu ($L < j$): quay lại bước 1, Phân hoạch dãy $a_L \dots a_j$, với $R=j$
- Nếu ($i < R$): quay lại bước 1, Phân hoạch dãy $a_i \dots a_R$, với $L=i$

■ Bước 3: Dừng thuật toán

Cài đặt thuật toán

```
void QuickSort( int a[ ], int L , int R )  
{  
    int i,j,x;  
    x = a[ (L+R)/2 ];  
    i = L; j = R;  
    do  
    {  
        while ( a[i] < x ) i++;  
        while ( a[j] > x ) j--;  
        if ( i <= j )  
        {  
            Hoanvi( a[i] , a[j] );  
            i++; j--;  
        }  
    } while(i<=j);  
    if (L<j) QuickSort(a,L,j);  
    if (i<R) QuickSort(a,i,R);  
}
```

Bài tập áp dụng

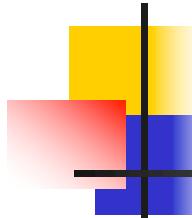
- Cho dãy số a:

12 2 8 5 1 6 4 15

- Sắp xếp dãy tăng dần theo thuật toán Quick Sort.

5. Sắp xếp kiểu vun đống

- Đống là một dạng cây nhị phân hoàn chỉnh đặc biệt mà giá trị lưu tại mọi nút nhánh đều lớn hơn hay bằng giá trị lưu trong hai nút con của nó
- Gọi dãy khóa $k[1..n]$ là biểu diễn của một cây nhị phân hoàn chỉnh. Ta có
 - $k[i]$ lưu giá trị nút thứ i .
 - Nút con của nút thứ i là nút thứ $2i$ và $2i+1$.
 - Nút cha của nút thứ i là nút $i \text{ div } 2$
- Để sắp xếp theo giải thuật này ta cần thực hiện 2 bước:
 - Tạo đống từ dãy ban đầu
 - Sắp xếp bằng cách vun lại thành đống từ đống hiện thời sau khi đã loại được gốc của đống trước đó .



Nhận xét

- Cây nhị phân 1 nút là 1 đống
- Để vun một nhánh cây gốc r thành đống, ta xem hai nhánh con của r (nhánh có gốc $2r$ và $2r+1$) là đống rồi.
- Gọi h là chiều cao của cây, các nút ở mức h (nút lá) đã là một đống, ta vun lên để những nút ở mức $h-1$ cũng là gốc của 1 đống , tiếp tục cho đến khi vun đống nút gốc, mức 1.

Thuật toán vun đống

■ Ý tưởng:

- Ban đầu dãy khóa $k[1, \dots, n]$ được vun từ dưới lên để biểu diễn thành một đống, khi đó khóa $k[1]$ ứng với nút gốc của đống là khóa lớn nhất, ta đảo khóa $k[1]$ với khóa $k[n]$.
- Dãy khóa mới là dãy $k[1, n-1]$. Tiếp tục vun đống, đảo khóa cho đến khi đống chỉ còn lại 1 nút.

Thuật toán vun đống với cây gốc r

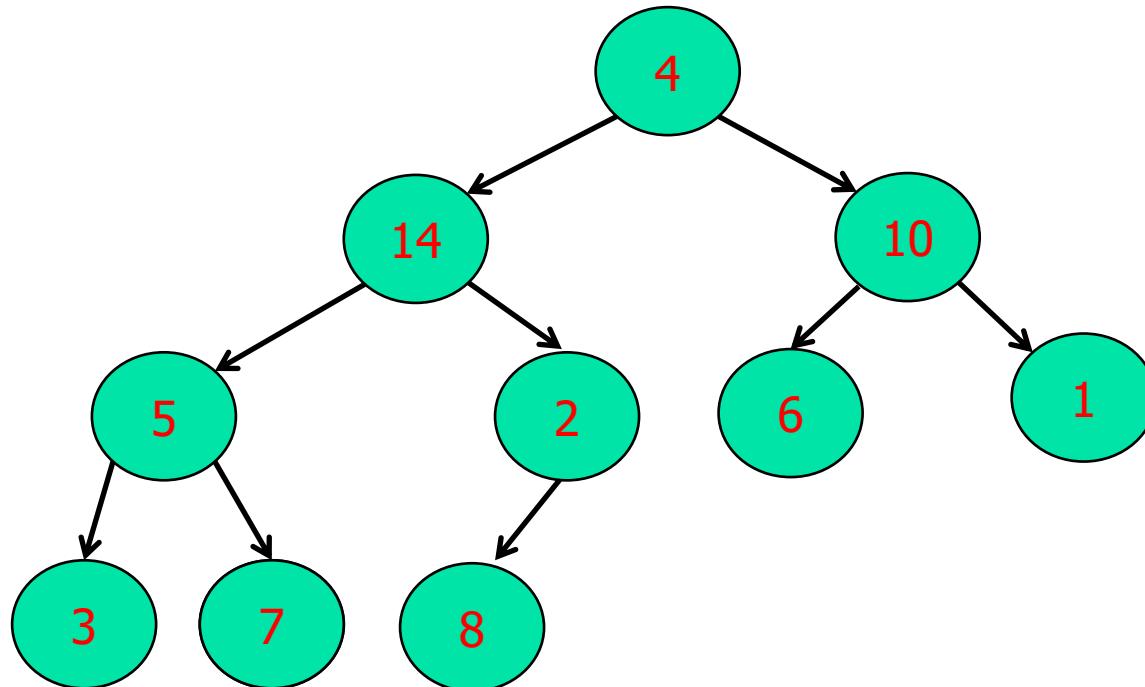
- Gọi r là gốc của cây cần vun đống.
- Ta có 2 nhánh con của r phải được vun đống rồi
- Thuật toán thực hiện như sau:
 - Bước 1: Giả sử cây có n nút, có gốc r chứa khóa V.
 - Bước 2:
 - Từ r ta đi tới nút con chứa giá trị lớn nhất trong 2 nút con cho đến khi gặp nút con c mà mọi nút con của c đều chứa giá trị $<=V$.
 - Trên đường đi từ r đến c, ta đẩy giá trị nút con lên nút cha, đặt giá trị V vào c.
 - Bước 3: Đảo khóa nút r với khóa của nút cuối cùng trong cây (nút thứ n với cây có n nút).
 - Bước 4: $n=n-1$, quay lại bước 1 cho đến khi $n=1$.

Thuật toán tạo đống

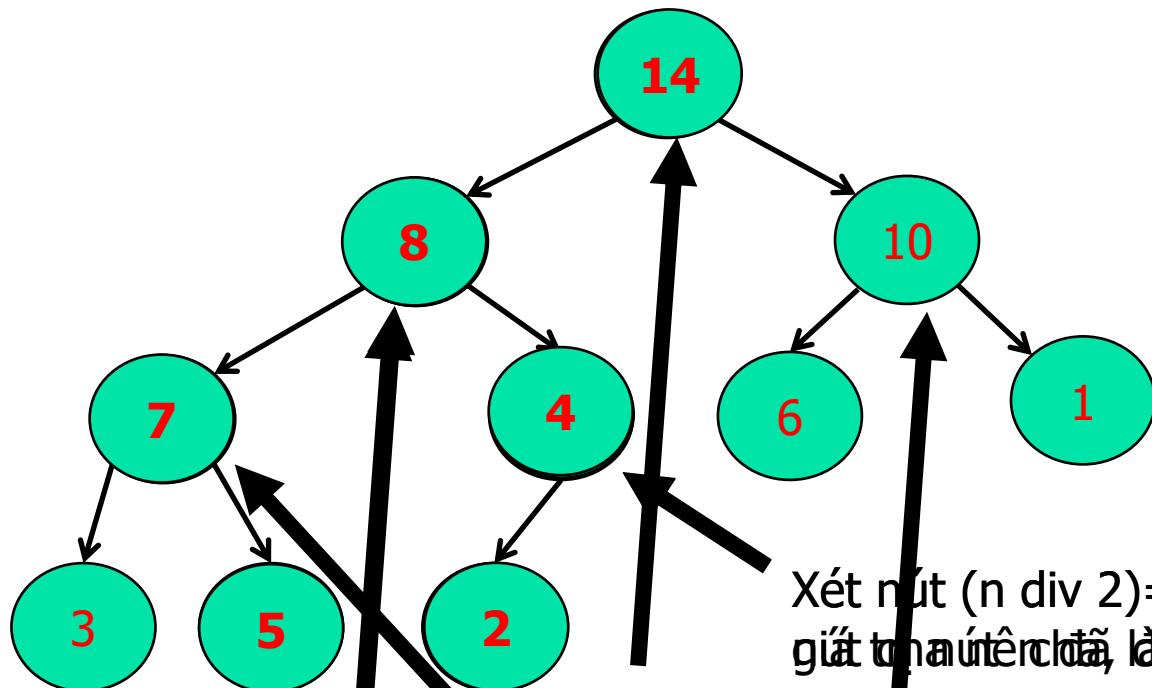
```
void adjust(int i, int n)
//vun đống với gốc là i
{ int key;
key=k[i]; j=2*i;//khoa nut cha ban dau
while (j<=n)
{ if ((j<n) && (k[j]<k[j+1])) j++;
if (key >=k[j]) {break;}
//khoa nut cha lon hon nut con
k[i]=k[j];
// khoa cha thay bang khoa con
i=j; j=j*2;
} k[i]=key;
//khoa cha moi thay bang khoa ban dau
}
```

Tạo đống từ dãy ban đầu

- Cho dãy sau: 4, 14, 10, 5, 2, 6, 1, 3, 7, 8.
Hãy sắp xếp dãy theo thứ tự tăng dần.
- Cây nhị phân tương ứng:



Vun đống cho cây ban đầu



Các nút lá đã là một đống

Cần vun đống cho các nút không phải là nút lá theo hướng từ dưới lên

Các nút không phải là lá là các nút có thứ tự từ 1 đến ($n \div 2$)

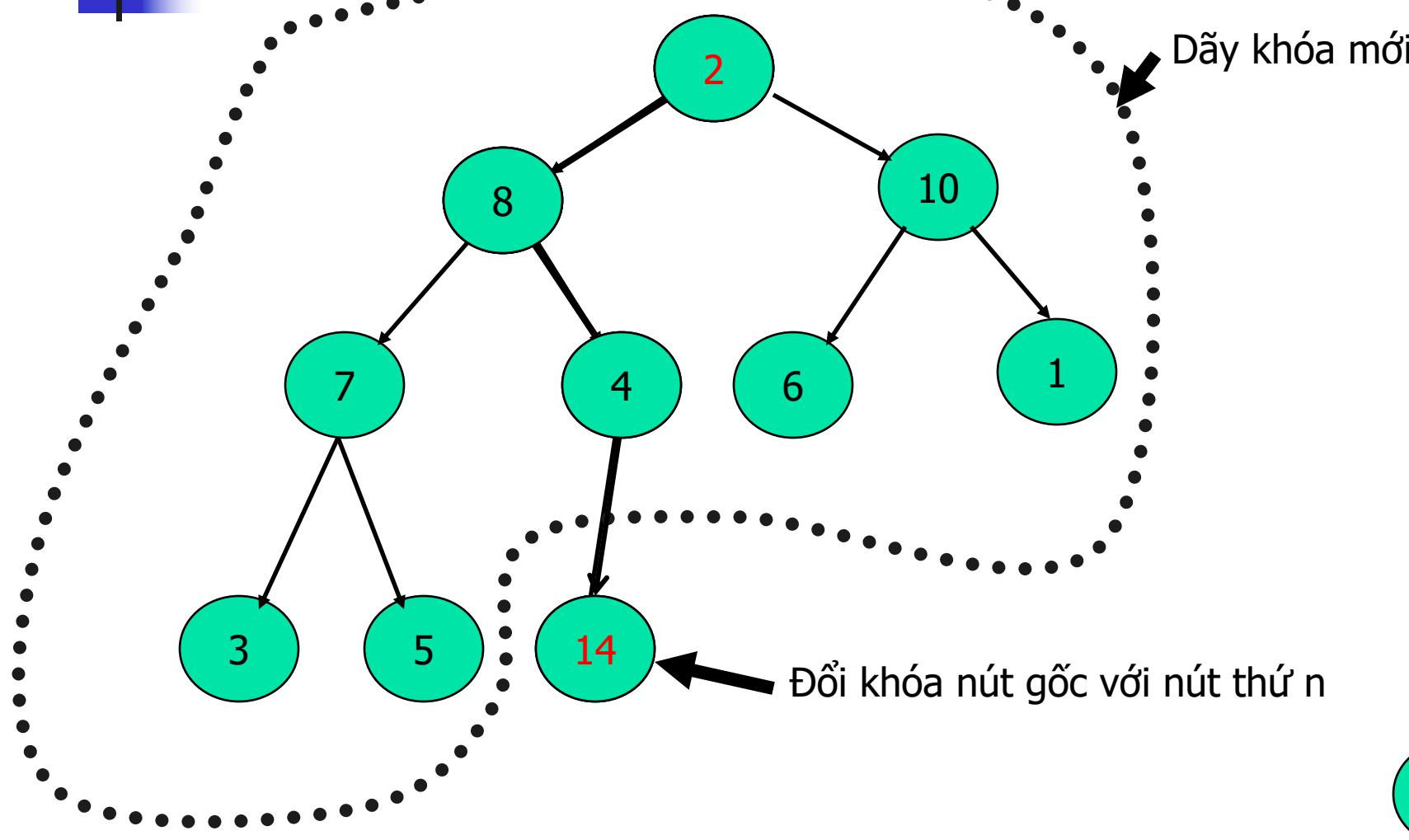
Xét nút $(n \div 2) = 5$, ta có nút con ~~lớn hơn~~ ~~giữa~~ ~~tùy~~ ~~nhân~~ ~~lần~~ ~~nhất~~ ~~tối~~ ~~đỗng~~ nút cha

Xét nút $(n \div 2)-1 = 4$, ta có nút con lớn hơn

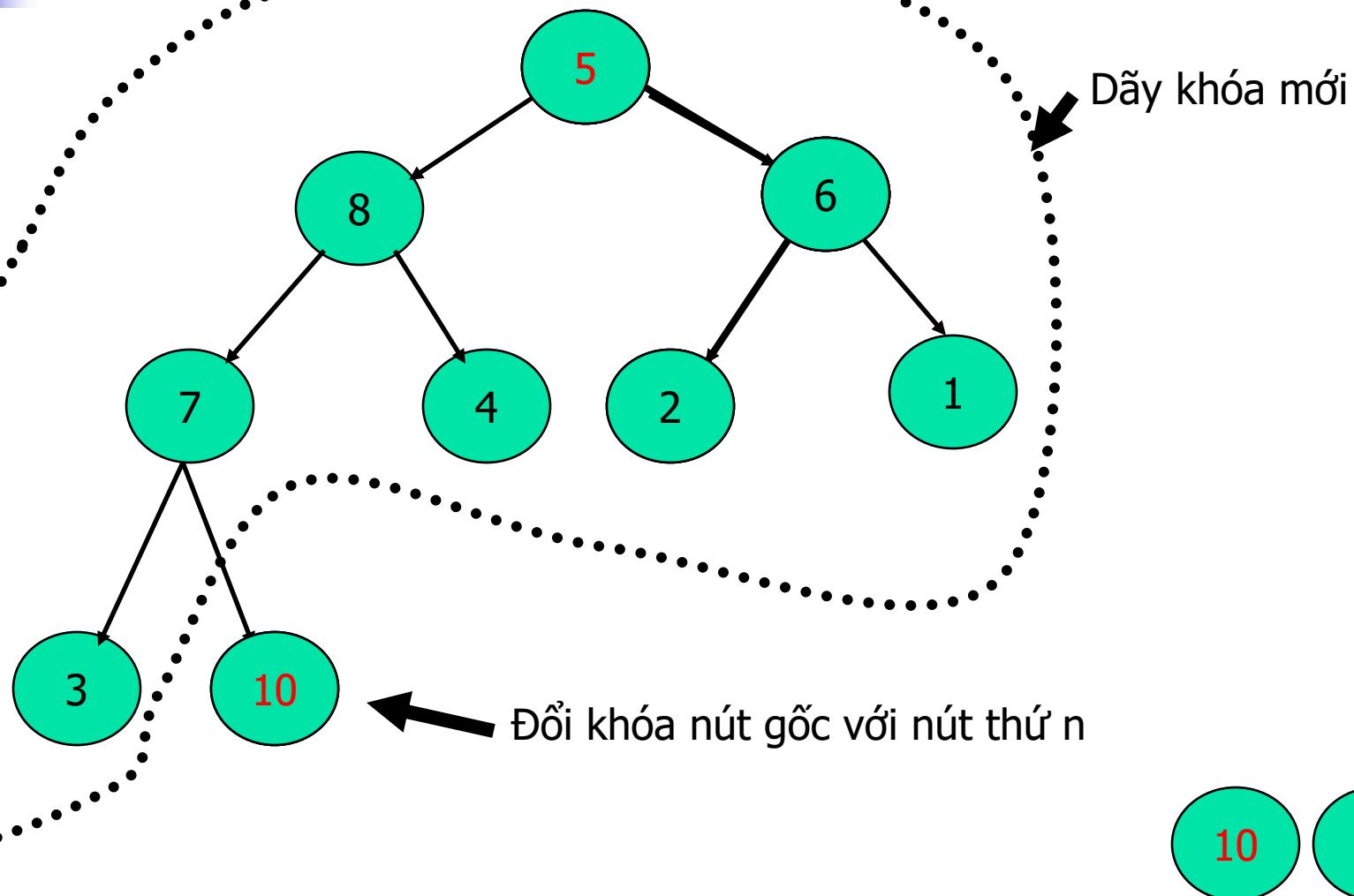
Xét nút $(n \div 2)-2 = 3$, có các nút con lớn hơn nút cha nên ta đổi chỗ Xét nút $(n \div 2)-3 = 2$, có các nút con lớn nhất là $n \div 2$ mứt cha

Xét nút $(n \div 2)-2 = 3$, có các nút con lớn hơn nút cha nên ta đổi chỗ Xét nút $(n \div 2)-3 = 2$, có các nút con lớn nhất trong 2 nút
đảo nút cha với nút con có giá trị lớn nhất trong 2 nút
nên đã là một đống Xét nút $(n \div 2)-3 = 2$, có các nút con
đảo nút cha với nút con có giá trị lớn nhất trong 2 nút
nên đã là một đống

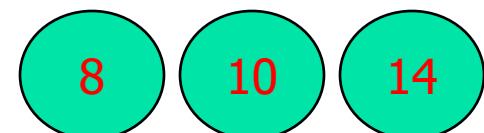
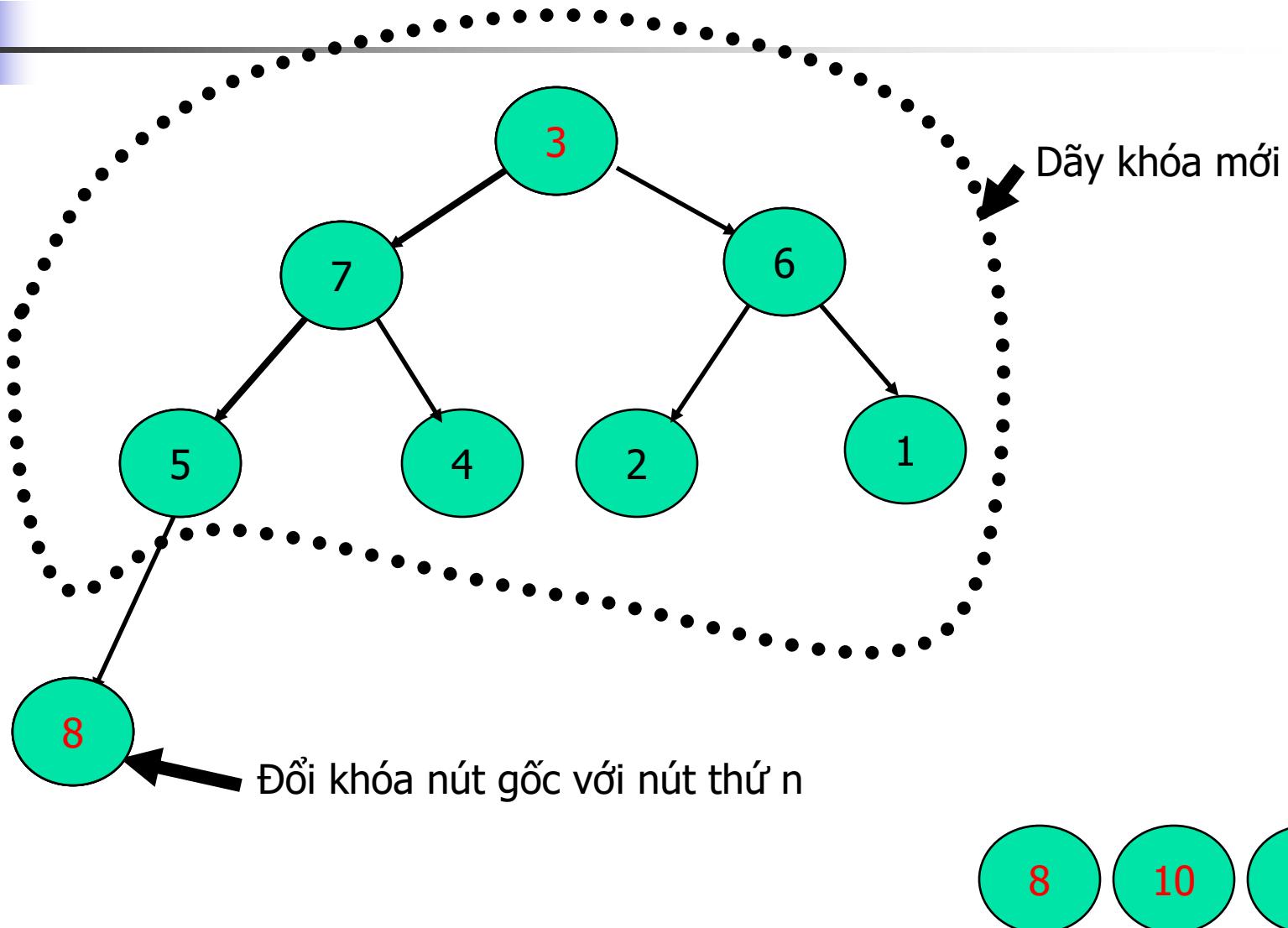
Minh họa vun đống gốc r với v=4



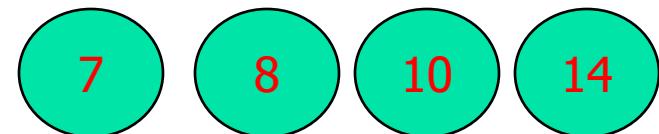
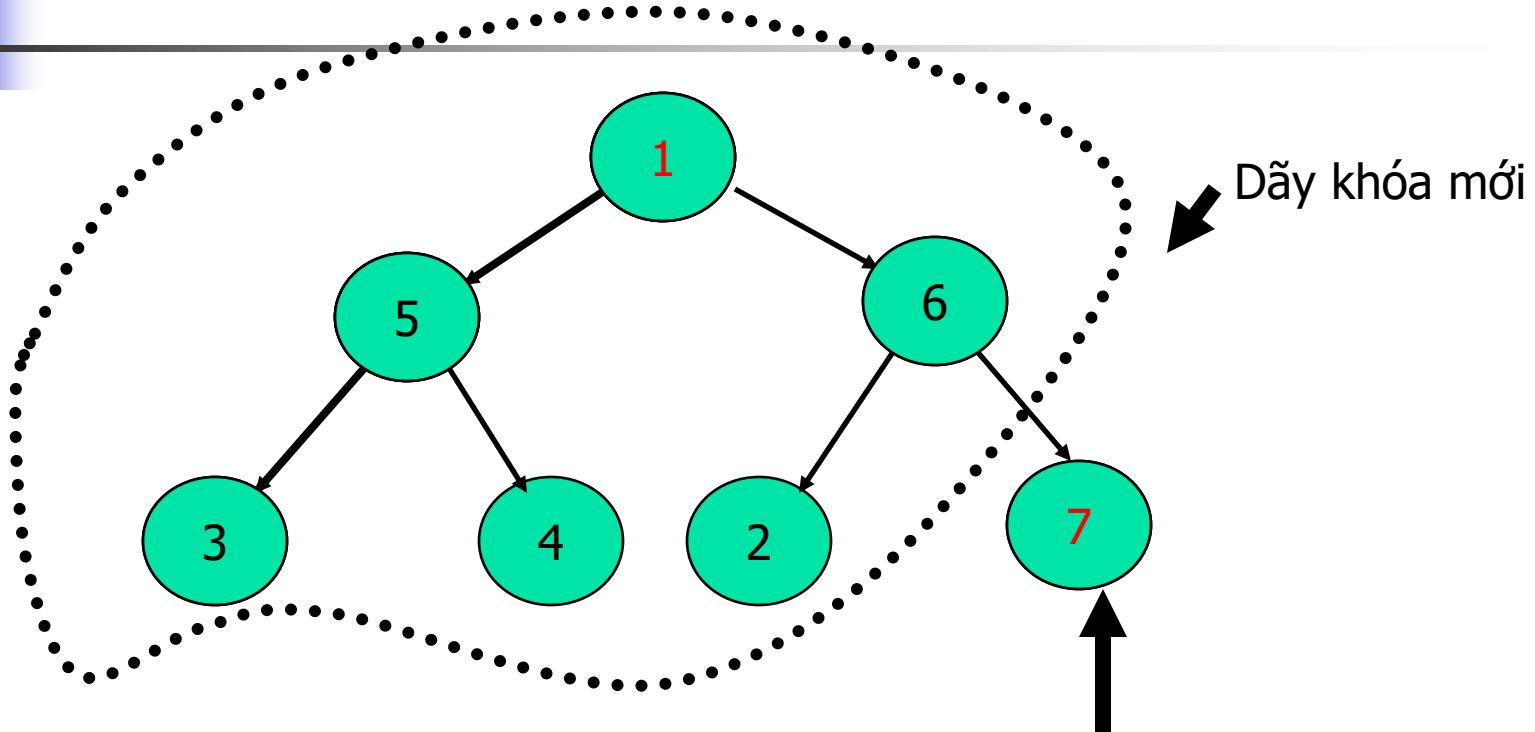
Minh họa vun đống gốc r với $v=2$



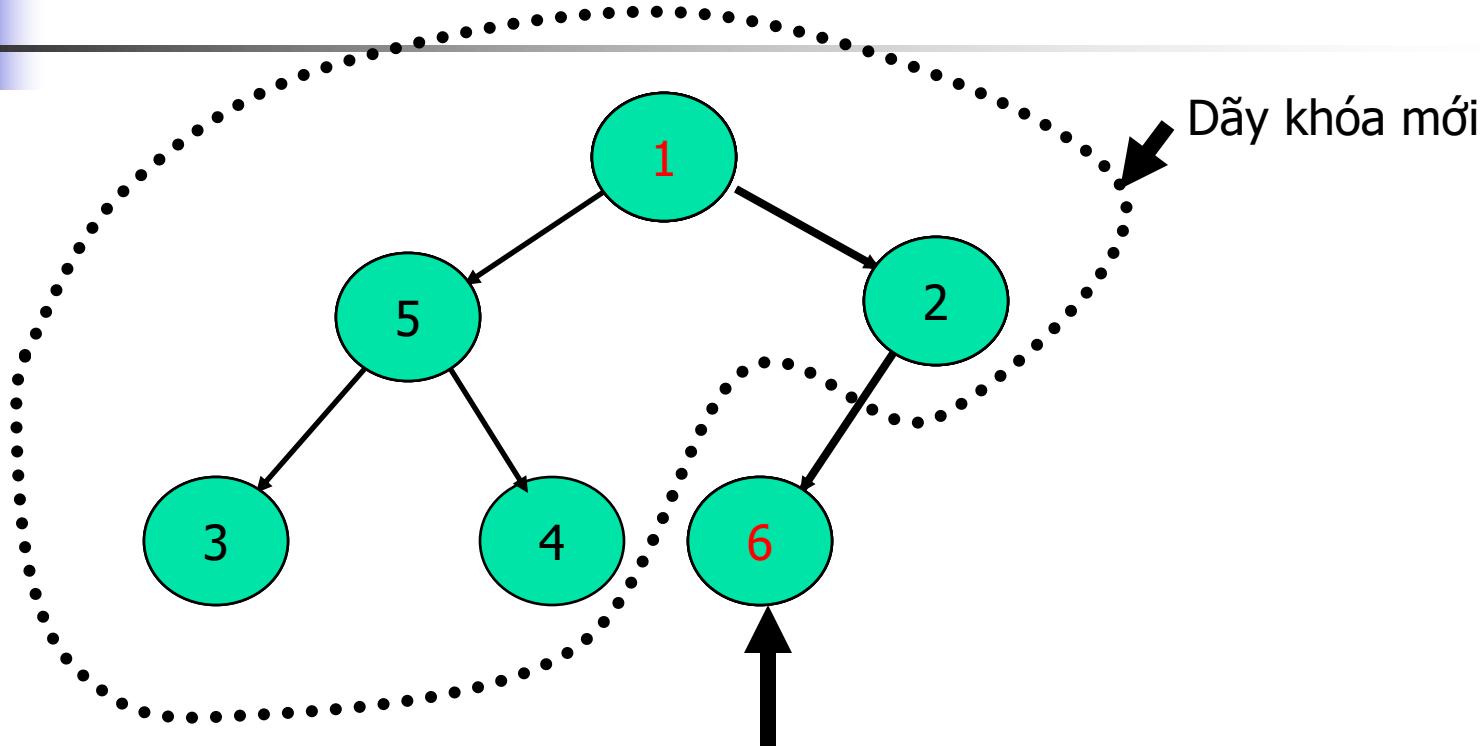
Minh họa vun đống gốc r với v=5



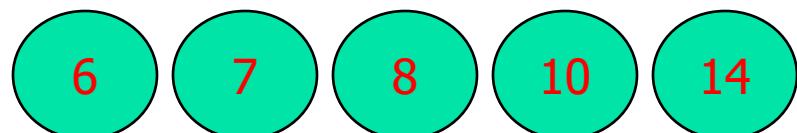
Minh họa vun đống gốc r với v=3



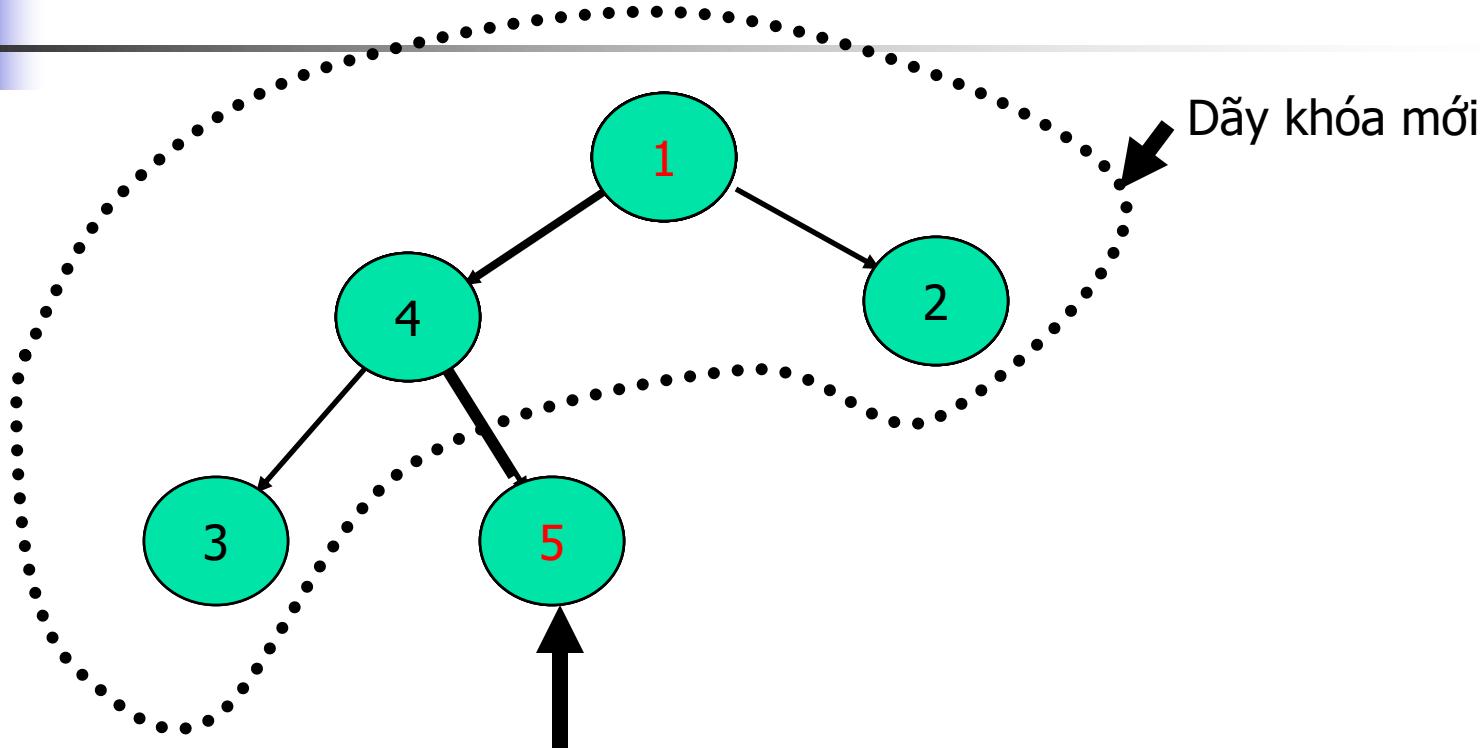
Minh họa vun đống gốc r với $v=1$



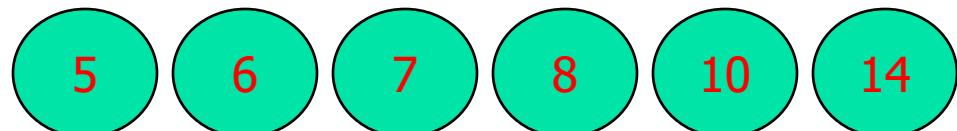
Đổi khóa nút gốc với nút thứ n



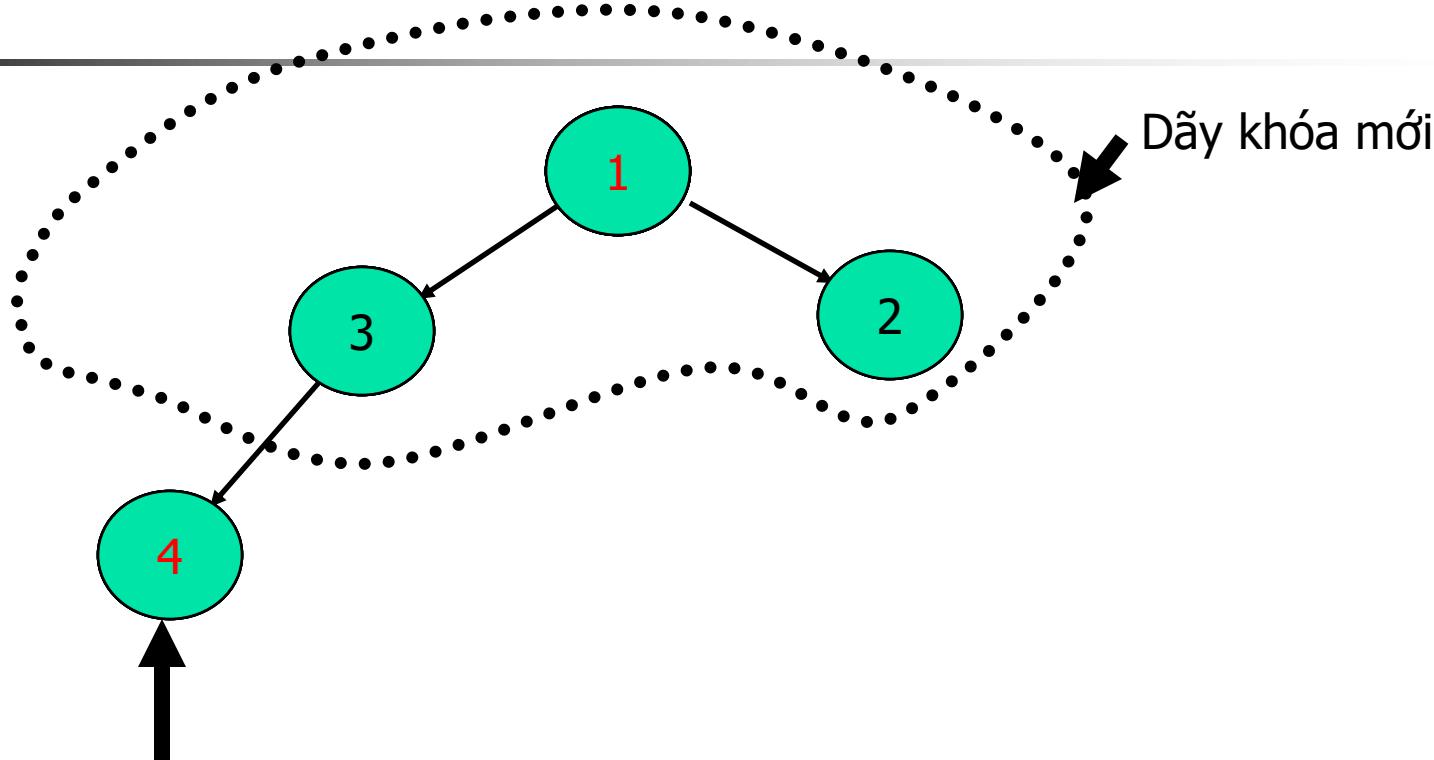
Minh họa vun đống gốc r với $v=1$



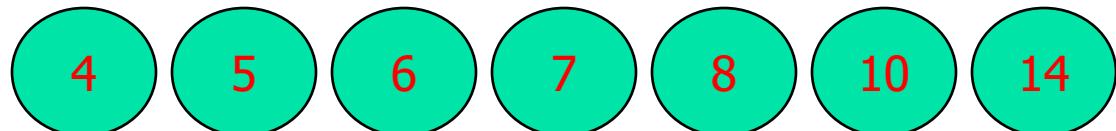
Đổi khóa nút gốc với nút thứ n



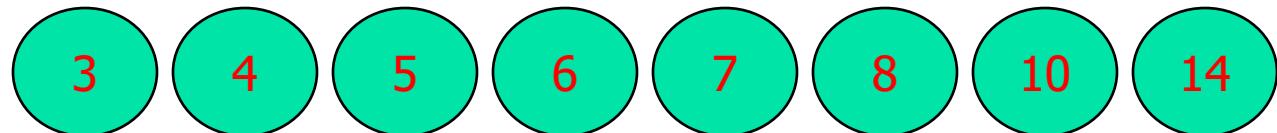
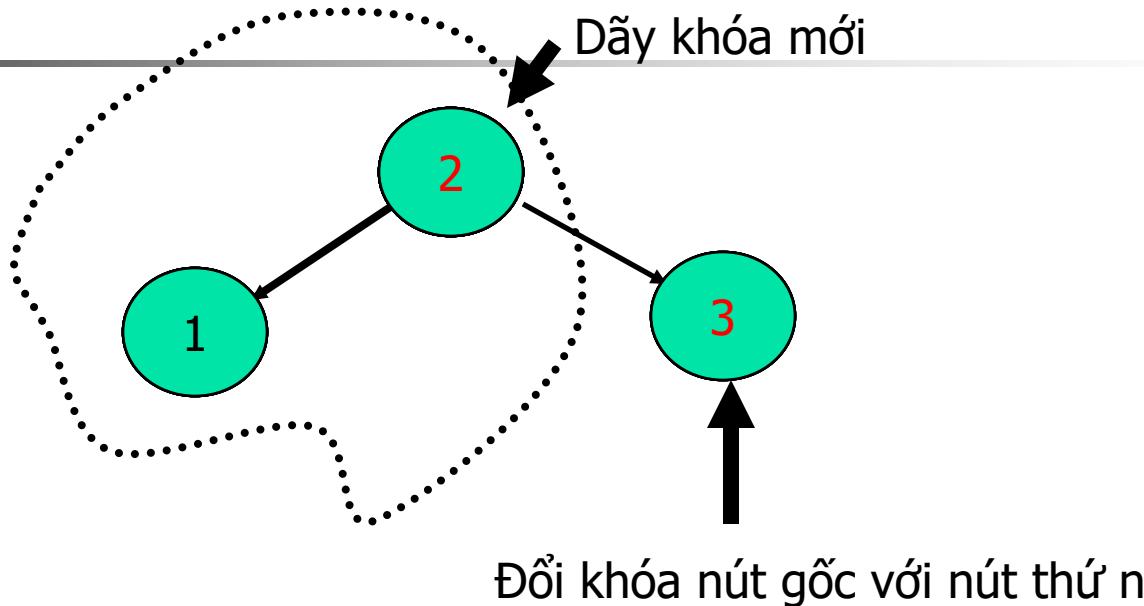
Minh họa vun đống gốc r với 4



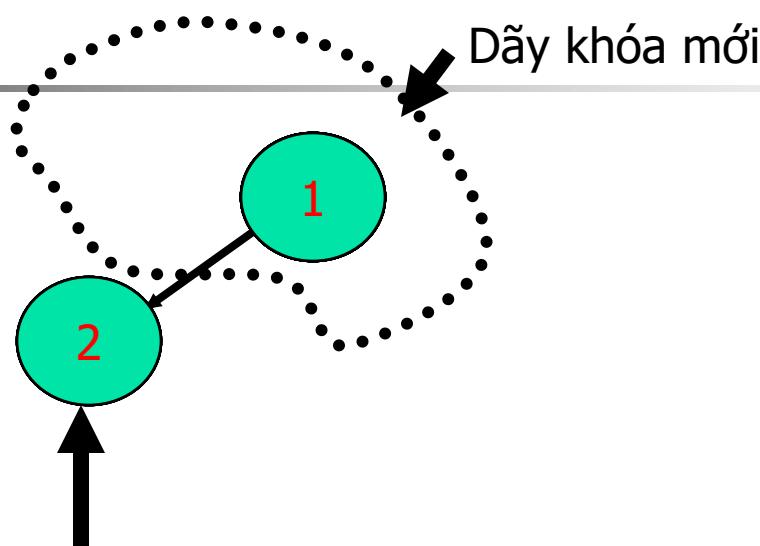
Đổi khóa nút gốc với nút thứ n



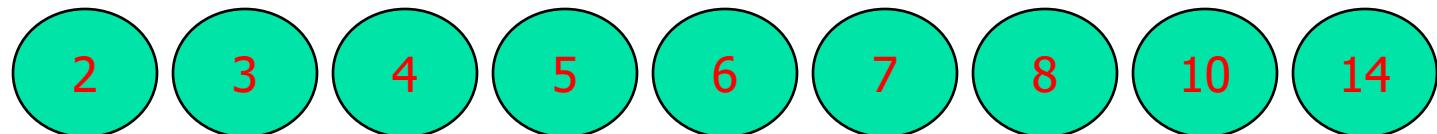
Minh họa vun đống gốc r với $v=1$



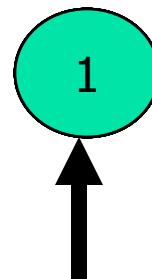
Minh họa vun đống gốc r với $v=1$



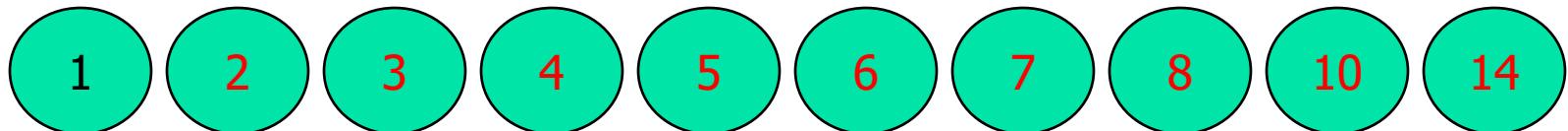
Đổi khóa nút gốc với nút thứ n

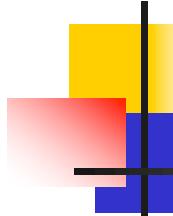


Minh họa vun đống gốc r với $v=1$



Dãy chỉ còn 1 khóa, dừng



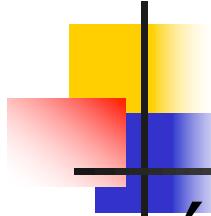


Bài tập:

- Cho 1 danh sách 7 số nguyên : {7, 12, 9, 5, 15, 6, 11}. Hãy mô tả các bước thực hiện sắp xếp dãy số tăng dần theo giải thuật Vun đống (Heap sort)

6. Sắp xếp trộn MergeSort

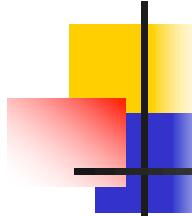
- Phép trộn 2 đường là phép hợp nhất hai dãy khóa đã sắp xếp để ghép lại thành một dãy khóa có kích thước bằng tổng của hai dãy khóa ban đầu
- Dãy khóa mới tạo thành cũng có thứ tự sắp xếp



Phép trộn hai đường

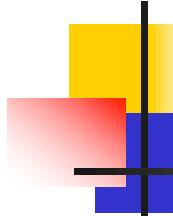
■ Ý tưởng:

- So sánh hai khóa đứng đầu hai dãy, chọn ra khóa nhỏ nhất, đưa vào vị trí thích hợp trong miền sắp xếp (dãy khóa phụ có kích thước bằng tổng kích thước 2 dãy khóa ban đầu).
- Loại khóa vừa xét ra khỏi dãy khóa chứa nó
- Quá trình này tiếp tục cho đến khi một trong hai dãy khóa đã cạn. Chuyển toàn bộ dãy khóa còn lại vào miền sắp xếp.



Ví dụ: Phép trộn hai đường

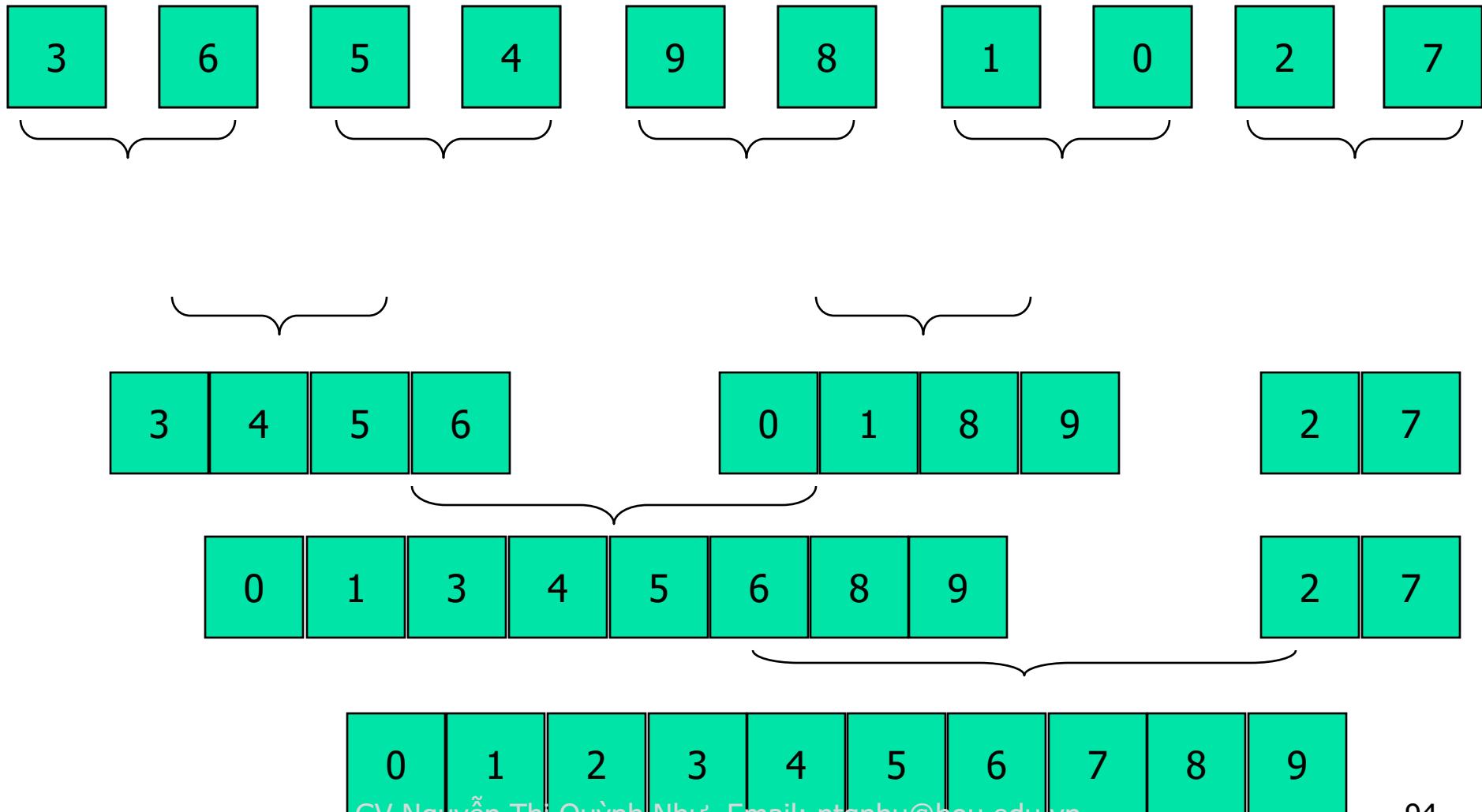
Dãy 1	Dãy 2	Khóa nhỏ nhất trong dãy	Miền sắp xếp
{1,3,10,11}	{2,4,9}	1	{1}
{3,10,11}	{2,4,9}	2	{1,2}
{3,10,11}	{4,9}	3	{1,2,3}
{10,11}	{4,9}	4	{1,2,3,4}
{10,11}	{9}	9	{1,2,3,4,9}
{10,11}	{}	Dãy 2 hết, đưa tất cả các phần tử trong dãy 1 vào miền sắp xếp	{1,2,3,4,9,10,11}

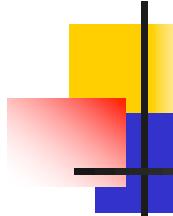


Sắp xếp trộn 2 đường trực tiếp

- Ta có thể coi mỗi khóa trong dãy khóa $k[1,..n]$ là một mạch độ dài 1.
- Rõ ràng các mạch có độ dài 1 đã được sắp xếp.
- Nếu trộn hai mạch độ dài 1 liên tiếp sẽ có được mạch độ dài 2, tiếp tục trộn hai mạch có độ dài 2 sẽ được mạch có độ dài 4,...
- Số mạch giảm dần sau mỗi lần trộn
- Quá trình trộn dừng khi chỉ còn 1 mạch

Ví dụ: sắp xếp trộn





Bài tập:

- Cho 1 danh sách 7 số nguyên : {7, 12, 9, 5, 15, 6, 11}. Hãy mô tả các bước thực hiện sắp xếp dãy số tăng dần theo giải thuật sắp xếp trộn (Merge sort)