

# **Bài 12.**

## **Các thuật toán sắp xếp nhanh $O(n \log n)$**

- ◆ **Sắp xếp nhanh – Quick sort**
- ◆ **Sắp xếp trộn - Merge sort**
- ◆ **Vun đống – Heap sort**

# Chia và trị - Divide and conquer

- ◆ Chia và trị là phương pháp thiết kế thuật toán theo kiểu:
  - **Phân chia**: Chia dữ liệu đầu vào  $S$  của bài toán thành 2 tập con rời nhau  $S_1$  và  $S_2$
  - **Đệ qui**: Giải bài toán với dữ liệu vào là các tập con  $S_1$  và  $S_2$
  - **Trị**: kết hợp các kết quả của  $S_1$  và  $S_2$  thành kết quả của  $S$
- ◆ Trường hợp cơ sở cho thuật toán đệ qui ở đây là các bài toán có kích thước 0 hoặc 1

# Sắp xếp nhanh – Quick sort

## ◆ Ý tưởng (sử dụng phương pháp chia và trị):

- Thực hiện phân hoạch dãy S cần sắp thành 3 dãy S1, S2, S3.  
Trong đó:
  - S<sub>2</sub> chỉ có một phần tử, tất cả các phần tử của dãy S3 đều > phần tử của dãy S2.
  - Tất cả các phần tử của dãy S1 đều  $\leq$  phần tử của dãy S2
  - Dãy S1, S3 có thể là rỗng
- Tiếp tục phân hoạch dãy S1 và S3 độc lập theo nguyên tắc trên đến khi dãy cần thực hiện phân hoạch chỉ có một phần tử thì dừng lại. Khi đó ta được dãy các phần tử được sắp.

# Thuật toán sắp xếp Quick sort

- ♦ Từ ý tưởng của thuật toán, ta có thể dễ dàng xây dựng thuật toán sắp xếp dưới dạng đệ qui như sau:

**Algorithm** *QuickSort (array A, i, j);*

**Input:** Dãy các phần tử  $A[i], \dots, A[j]$  và hai số nguyên  $i, j$

**Output:** Dãy  $A[i], \dots, A[j]$  được sắp.

**if**  $i < j$  **then**

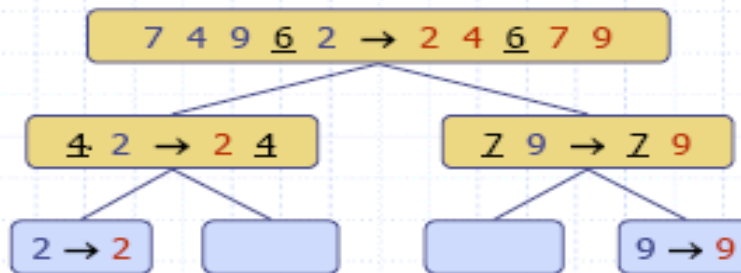
    Partition (A, i, j, k); //k lấy chỉ số của phần tử làm S2

    Quicksort (A, i, k-1);

    Quicksort (A, k+1, j);

# Ví dụ

## Quick-Sort



Quick-Sort

1

**Vấn đề đặt ra ở đây là  
phân hoạch dãy  $S$  như thế  
nào?**

# Thuật toán phân hoạch

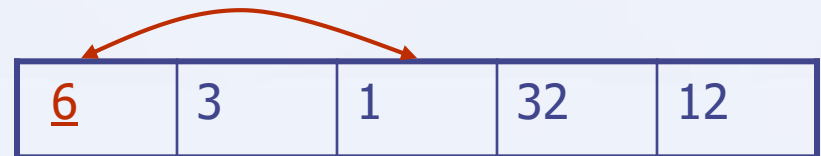
- Chọn một phần tử bất kỳ của dãy làm dãy S2 (phần tử này được gọi là phần tử chốt -pivot).
- Thực hiện chuyển các phần tử có khóa  $\leq$  phần tử chốt về bên trái và các phần tử  $>$  phần tử chốt về bên phải, sau đó đặt phần tử chốt về đúng vị trí của nó trong dãy.



<u>6</u>	12	32	1	3
----------	----	----	---	---



<u>6</u>	3	32	1	12
----------	---	----	---	----



<u>6</u>	3	1	32	12
----------	---	---	----	----



1	3	<u>6</u>	32	12
---	---	----------	----	----

Sau khi phân hoạch

# Chú ý

- Phần tử chốt có thể được chọn là một phần tử bất kỳ của dãy.
  - Phần tử chốt có thể chọn là phần tử đầu hoặc giữa hoặc cuối dãy.
  - Tốt nhất là chọn phần tử chốt mà nó làm cho việc phân hoạch thành hai dãy  $S_1$  và  $S_3$  có số phần tử xấp xỉ bằng nhau.



# Thuật toán

- Phân hoạch dãy gồm các phần tử  $A[i], \dots, A[j]$
- Chọn phần tử **đầu** dãy làm chốt
- Sử dụng 2 biến **left** và **right**:
  - left chạy từ trái sang phải bắt đầu từ  $i$ .
  - right chạy từ phải sang trái bắt đầu từ  $j$
  - Biến left được tăng cho tới khi  $A[\text{left}].\text{Key} > A[i].\text{Key}$  hoặc  $\text{left} > \text{right}$
  - Biến right được giảm cho tới khi  $A[\text{right}].\text{Key} \leq A[i].\text{Key}$
  - Nếu  $\text{left} < \text{right}$  thì ta đổi  $A[\text{left}]$  và  $A[\text{right}]$
  - Quá trình trên được lặp lại cho tới khi nào  $\text{left} > \text{right}$
  - Cuối cùng trao đổi  $A[i]$  và  $A[\text{right}]$

# Ví dụ phân hoạch

10	3	24	1	4	21	54	5
i							j

?

# Thuật toán phân hoạch

**Algorithm** *Partition (Array A, i, j, &right )*

**Input:** Dãy các phần tử  $A[i], \dots, A[j]$ , 2 số nguyên  $i, j$

**Output:** Dãy  $A[i], \dots, A[j]$  được phân hoạch, right là chỉ số của phần tử làm S2.

$p \leftarrow A[i];$

$left \leftarrow i; right \leftarrow j;$

**while** (  $left < right$  )

**while**(  $A[left].Key \leq p.Key \ \&\& \ left \leq right$  )

$left \leftarrow left + 1;$

**while**(  $A[right].Key > p.Key$  )  $right \leftarrow right - 1;$

**if**  $left < right$  **then**

**SWAP**( $A[left], A[right]$ );

**if**  $i \neq right$  **then**

$A[i] \leftarrow A[right];$

$A[right] \leftarrow p;$

# Ví dụ Sắp xếp dãy số

A= ... 

10	3	24	1	4	21	54	5
----	---	----	---	---	----	----	---

 ...  
i=1 j=8

?

# Mô tả quá trình Sắp xếp

Quicksort(A,1, 8)

10	3	24	1	4	21	54	5
----	---	----	---	---	----	----	---

i=1 j=8

i < j partition(A,1,8,k)

4	3	5	1	10	21	54	24
---	---	---	---	----	----	----	----

i=1 k=5 j=8

Quicksort(A,1, 4)

4	3	5	1	10	21	54	24
---	---	---	---	----	----	----	----

i=1 j=4

i < j partition(A,1,4,k)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=1 k=3 j=4

Quicksort(A,1, 2)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=1 j=2

i<j partition(A,1,2,k)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=k=1 j=2

Quicksort(A,2, 2)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=j=2

Quicksort(A,4, 4)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=j=4

Quicksort(A,6, 8)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=6

j=8

i<j partition(A,6,8,k)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

i=k=6

j=8

Quicksort(A,6,5)

1	3	4	5	10	21	54	24
---	---	---	---	----	----	----	----

j=5 i=6

Quicksort(A,7,8)  
i<j Partition(A,7,8,k)

1	3	4	5	10	21	24	54
---	---	---	---	----	----	----	----

i=7 k=j=8

Quicksort(A,7,7)

1	3	4	5	10	21	24	54
---	---	---	---	----	----	----	----

i=7 j=7

Quicksort(A,9,8)

1	3	4	5	10	21	24	54
---	---	---	---	----	----	----	----

i=9 j=8

# Thời gian chạy

- Thủ tục partition kiểm tra tất cả các phần tử trong mảng nhiều nhất một lần, vì vậy nó mất thời gian tối đa là  $O(n)$ .
- Thủ tục partition sẽ chia phần mảng được sắp thành 2 phần.
- Mặt khác cần chia liên tiếp  $n$  phần tử thành hai phần thì số lần chia nhiều nhất là  $\log_2 n$  lần.
- Vậy thời gian chạy của thuật toán QuickSort là  $O(n \log n)$



# Thuật toán MergeSort

## Ý tưởng:

- Giả sử ta có hai dãy  $A[i], \dots, A[k]$  và  $A[k+1], \dots, A[j]$  và hai dãy này đã được sắp.
- Thực hiện trộn hai dãy trên để được dãy  $A[i], \dots, A[j]$  cũng được sắp
- Do hai dãy  $A[i], \dots, A[k]$  và dãy  $A[k+1], \dots, A[j]$  đã được sắp nên việc trộn hai dãy thành một dãy được sắp là rất đơn giản.
- Vậy trộn như thế nào?

# Ví dụ: Trộn hai dãy sau

A

...	1	3	24	4	21	54	...
	i		k	k+1		j	

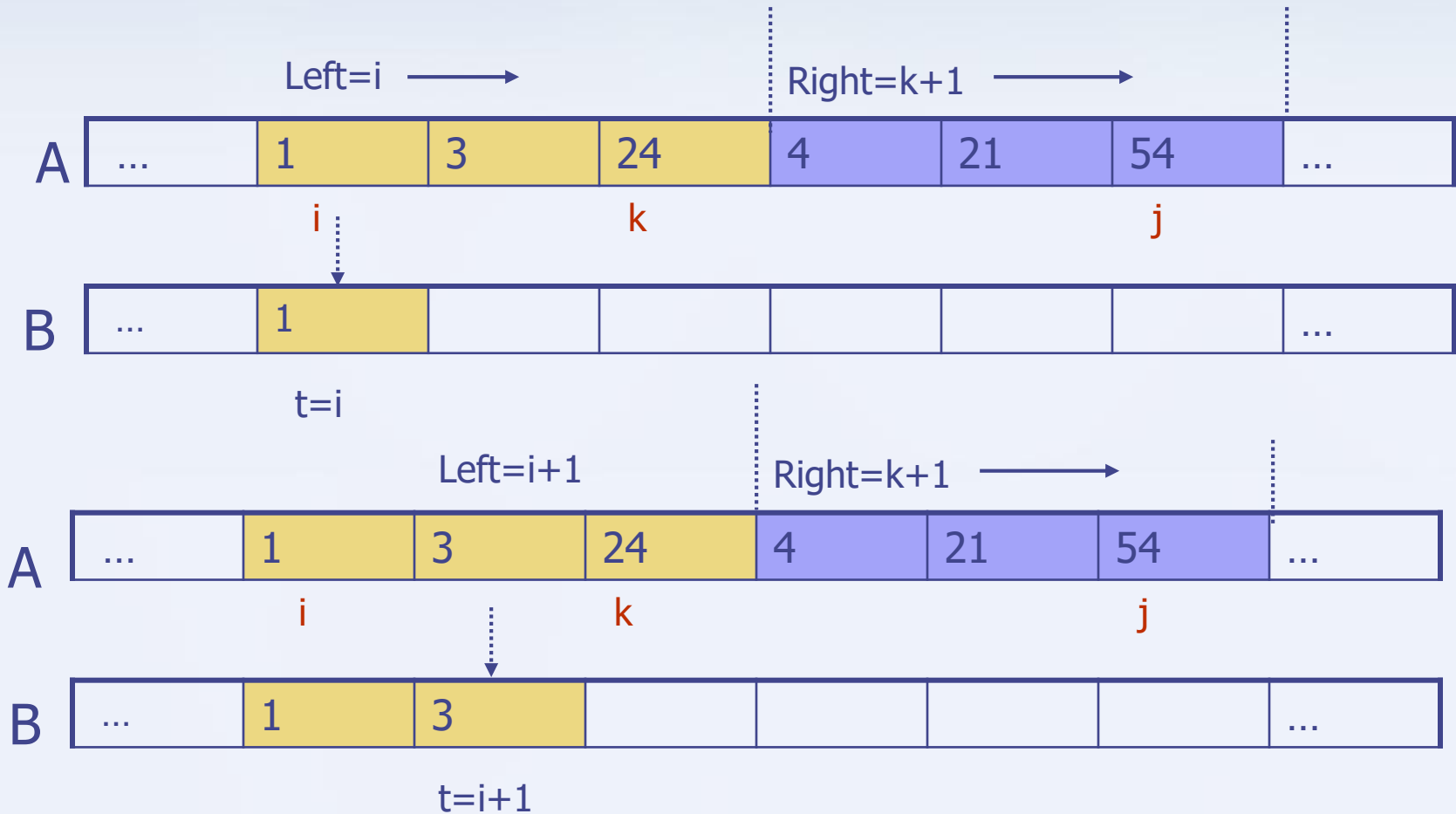
# Thuật toán trộn

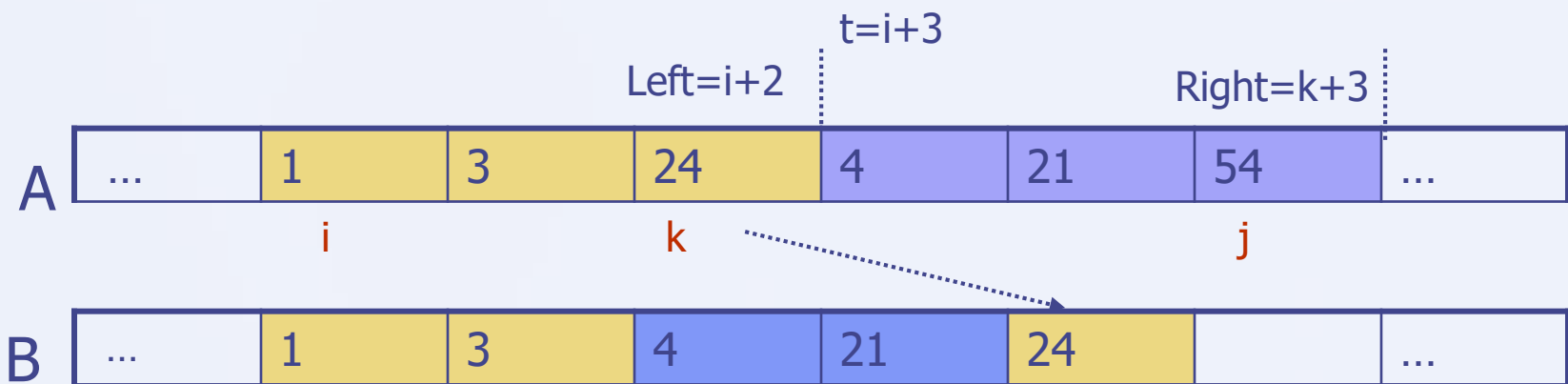
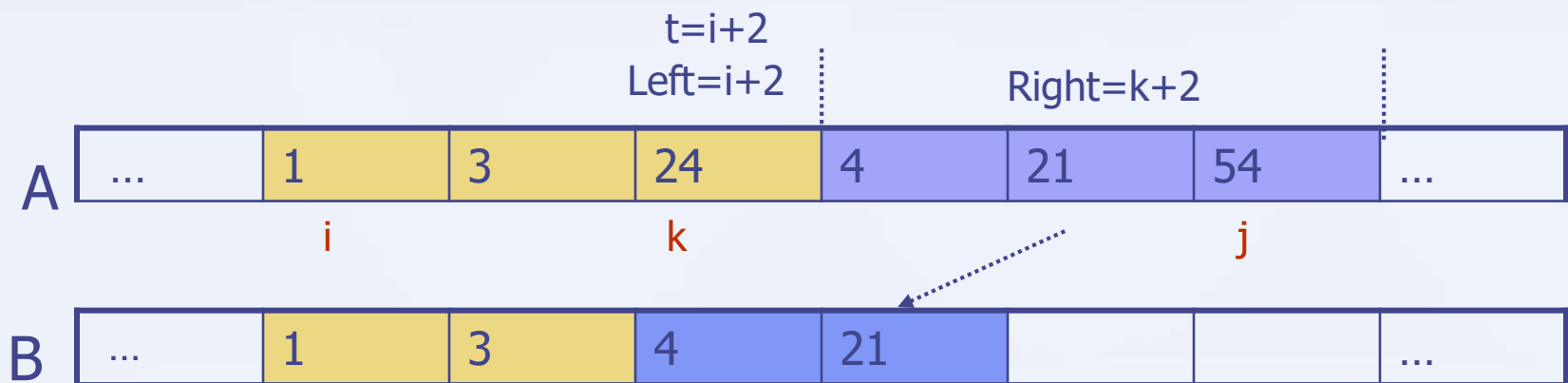
- Sử dụng hai biến *left*, *right*, *t* và sử dụng mảng phụ *B[i],...,B[j]*. *left* xuất phát từ *i*, *right* xuất phát từ *k+1*, *t* xuất phát từ *i* trên mảng phụ *B*.
- Nếu  $A[\textit{left}].\textit{key} < A[\textit{right}].\textit{key}$  thì  $B[t] \leftarrow A[\textit{left}]$ ,  $t \leftarrow t+1$  và  $\textit{left} \leftarrow \textit{left}+1$
- Nếu  $A[\textit{left}].\textit{key} \geq A[\textit{right}].\textit{key}$  thì  $B[t] \leftarrow A[\textit{right}]$ ,  $t \leftarrow t+1$  và  $\textit{right} \leftarrow \textit{right}+1$
- Quá trình trên được thực hiện cho đến khi  $\textit{left} > k$  hoặc  $\textit{right} > j$  thì dừng lại.

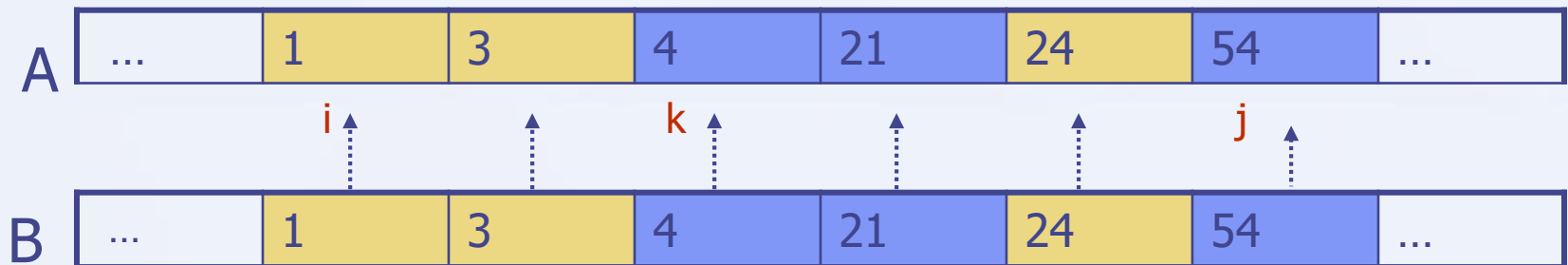
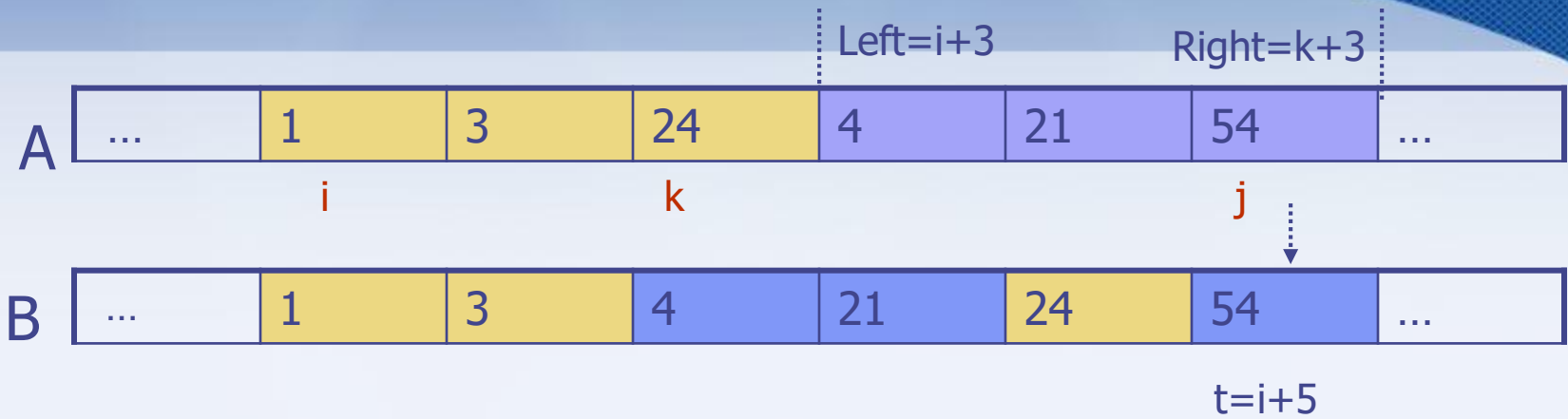
# Thuật toán trộn (tiếp)

- Nếu  $left > k$  thì  $B[t] \leftarrow A[right], \dots, B[j] \leftarrow A[j]$ .
- Nếu  $right > j$  thì  $B[t] \leftarrow A[left], B[t+1] \leftarrow A[left+1], \dots, B[t+k-left] \leftarrow A[k]$ .
- Gán  $A[i] \leftarrow B[i], \dots, A[j] \leftarrow B[j]$

# Quá trình trộn dãy







# Thuật toán giả mã

**Algorithm** *Merge(array A, int i, int k, int j)*

**Input:** Hai dãy  $A[i], \dots, A[k]$  và  $A[k+1], \dots, A[j]$  đã được sắp và các số nguyên  $i, j$

**Output:** Dãy  $A[i], \dots, A[j]$  cũng được sắp

$\text{left} \leftarrow i; \text{right} \leftarrow k+1; t \leftarrow i;$

**While**  $(\text{left} \leq k)$  and  $(\text{right} \leq j)$  **do**

**if**  $A[\text{left}].\text{key} < A[\text{right}].\text{key}$  **then**

$B[t] \leftarrow A[\text{left}];$

$\text{left} \leftarrow \text{left} + 1;$

$t \leftarrow t + 1;$

**else**

$B[t] \leftarrow A[\text{right}];$

$\text{right} \leftarrow \text{right} + 1;$

$t \leftarrow t + 1 ;$                    //kết thúc while

**If**  $\text{left} > k$  **then**

**for**  $r \leftarrow \text{right}$  **to**  $j$  **do**

$B[t] \leftarrow A[r];$

$t++;$

**else**

**for**  $r \leftarrow \text{left}$  **to**  $k$  **do**

$B[t] \leftarrow A[r];$

$t++;$

**for**  $r \leftarrow i$  **to**  $j$  **do**

$A[r] \leftarrow B[r] ;$



# Thuật toán

- Để sắp xếp dãy  $A[1], \dots, A[n]$  ta thực hiện như sau:
- Chia dãy trên thành hai dãy:  $A[1], \dots, A[k]$  và dãy  $A[k+1], \dots, A[n]$ , trong đó  $k = (n+1)/2$
- Thực hiện sắp xếp 2 dãy  $A[1], \dots, A[k]$  và  $A[k+1], \dots, A[n]$  độc lập cũng theo thuật toán Mergesort.
- Thực hiện trộn hai dãy:  $A[1], \dots, A[k]$  và dãy  $A[k+1], \dots, A[n]$  để được dãy  $A[1], \dots, A[n]$  cũng được sắp

# Thuật toán giả mã

**Algorithm** *Mergesort(array A, int i, int j)*

**Input:** Dãy các phần tử  $A[i], \dots, A[j]$

**Output:** Dãy  $A[i], \dots, A[j]$  được sắp.

**if**  $i < j$  **then**

$k \leftarrow (i+j)/2;$

Mergesort(A, i, k);

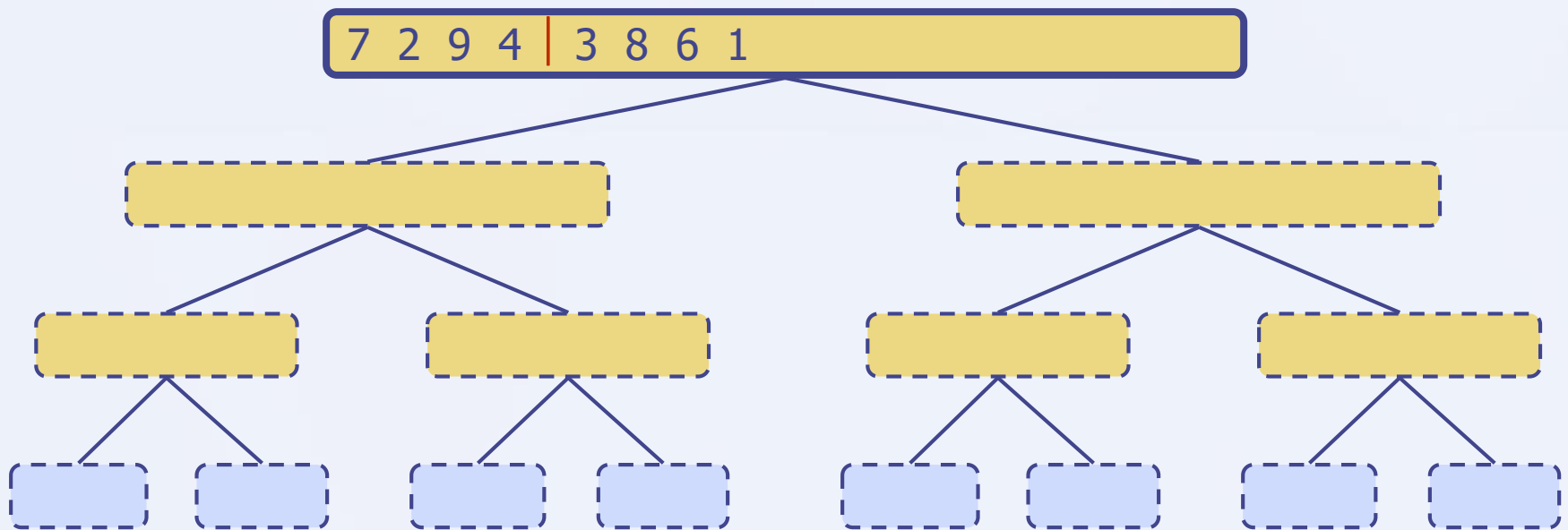
Mergesort(A, k+1, j);

Merge(A, i, k, j);

# Mô tả quá trình thực hiện sắp xếp

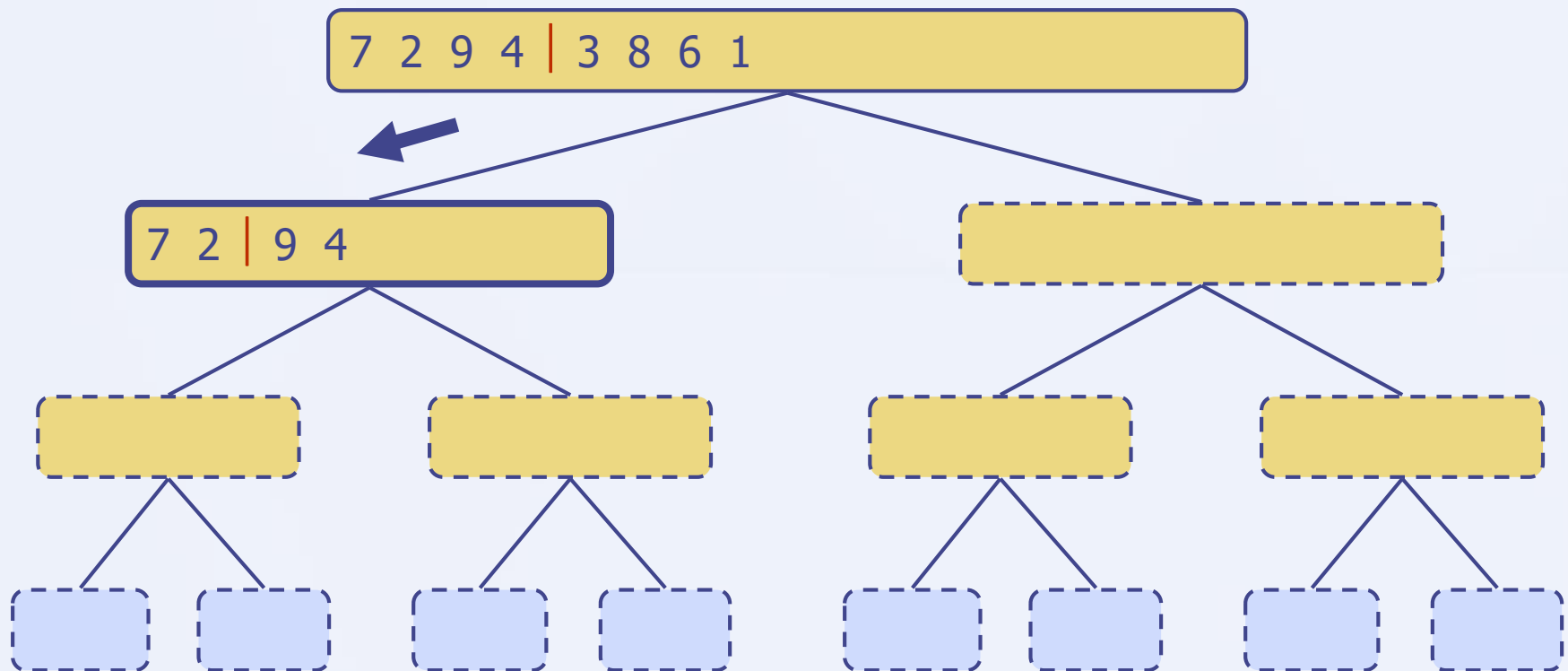
❖ Ví dụ sắp xếp dãy:  $A = 7\ 2\ 9\ 4\ 3\ 8\ 6\ 1$

- Gọi thủ tục MergeSort( $A, 1, 8$ ), chia đôi dãy



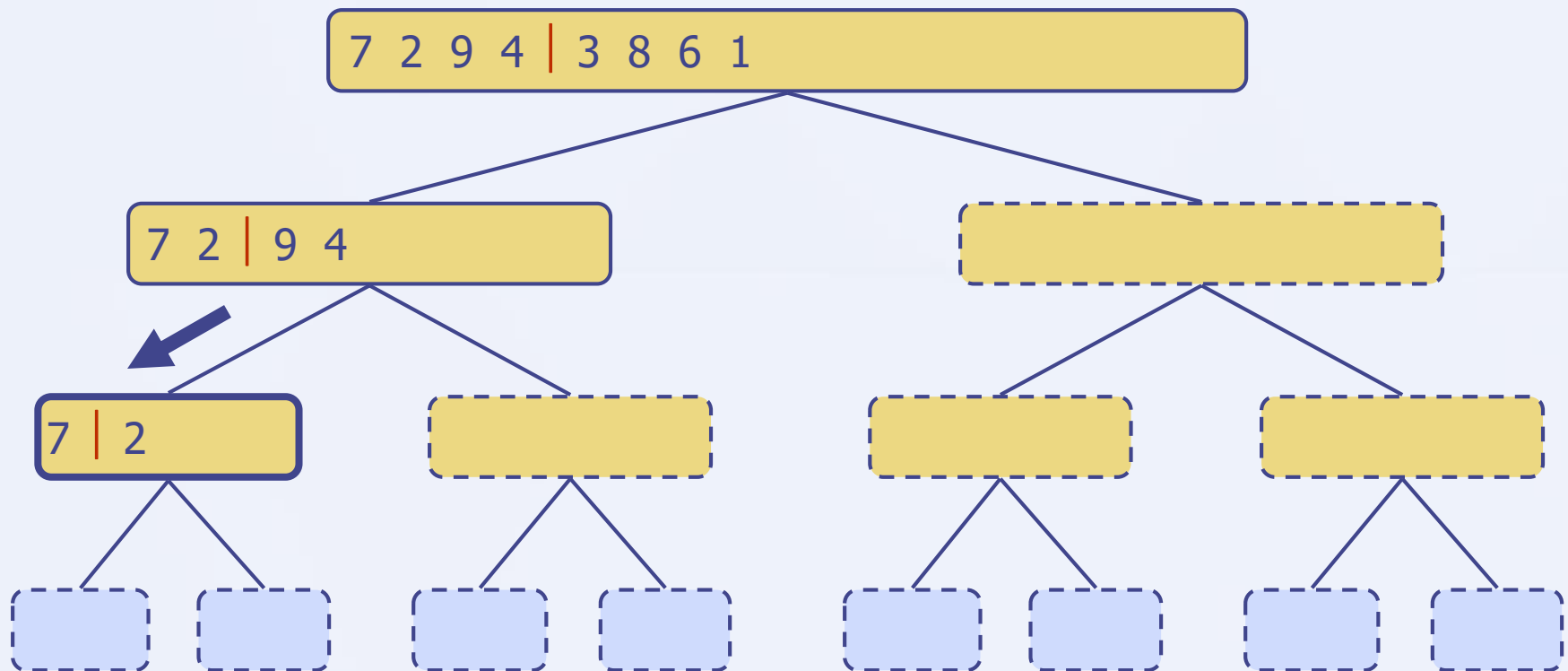
# Tiếp

❖ Gọi đệ qui và phân chia Mergesort(A,1,4)



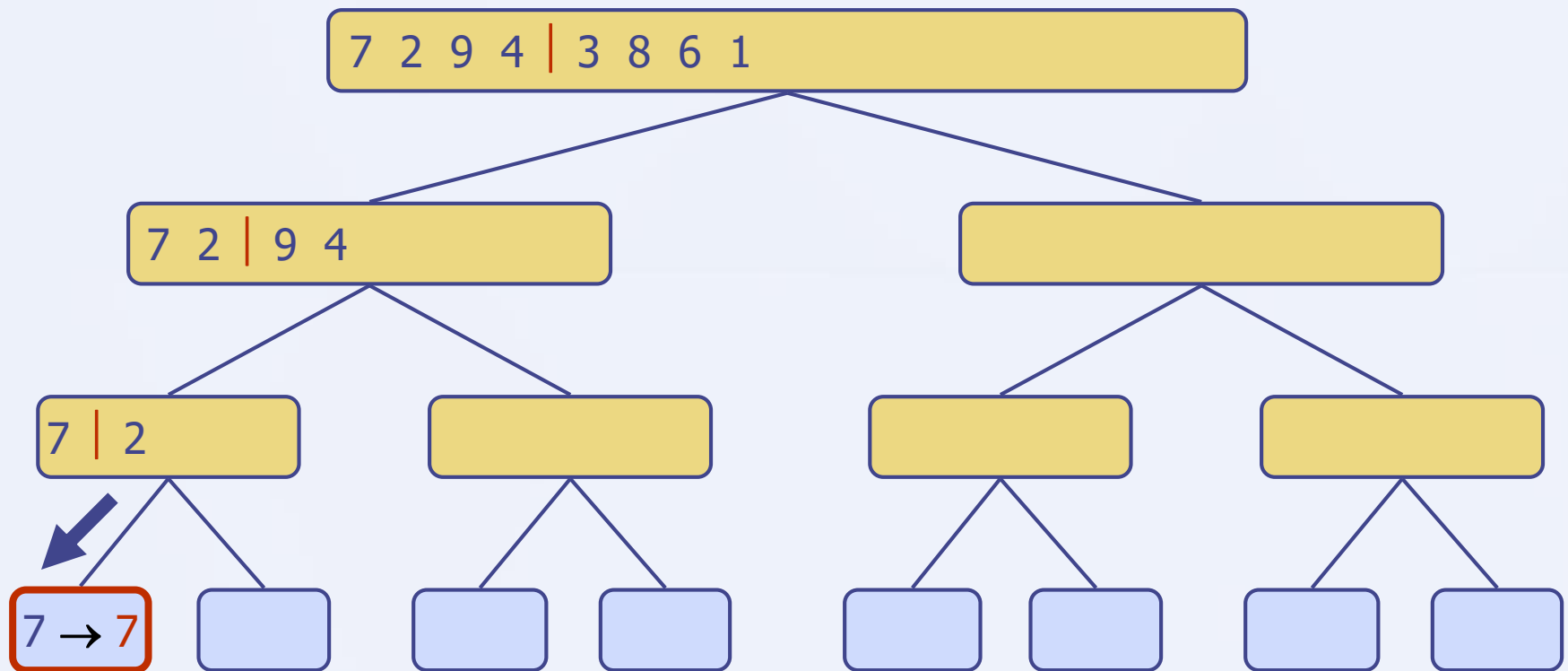
# Tiếp

❖ Gọi đệ qui và phân chia Mergesort(A,1,2)



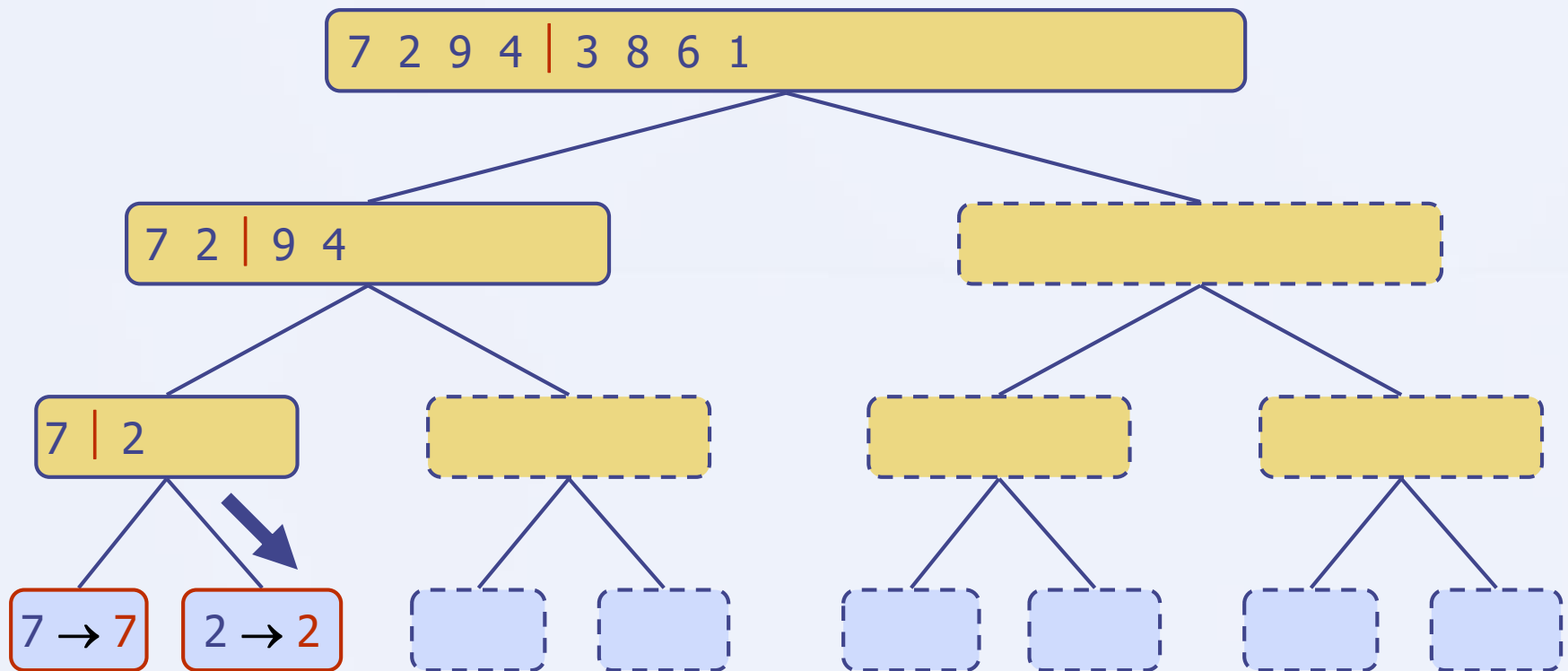
# Tiếp

- ❖ Gọi đệ qui Mergesort(A,1,1), đây là trường hợp cơ sở



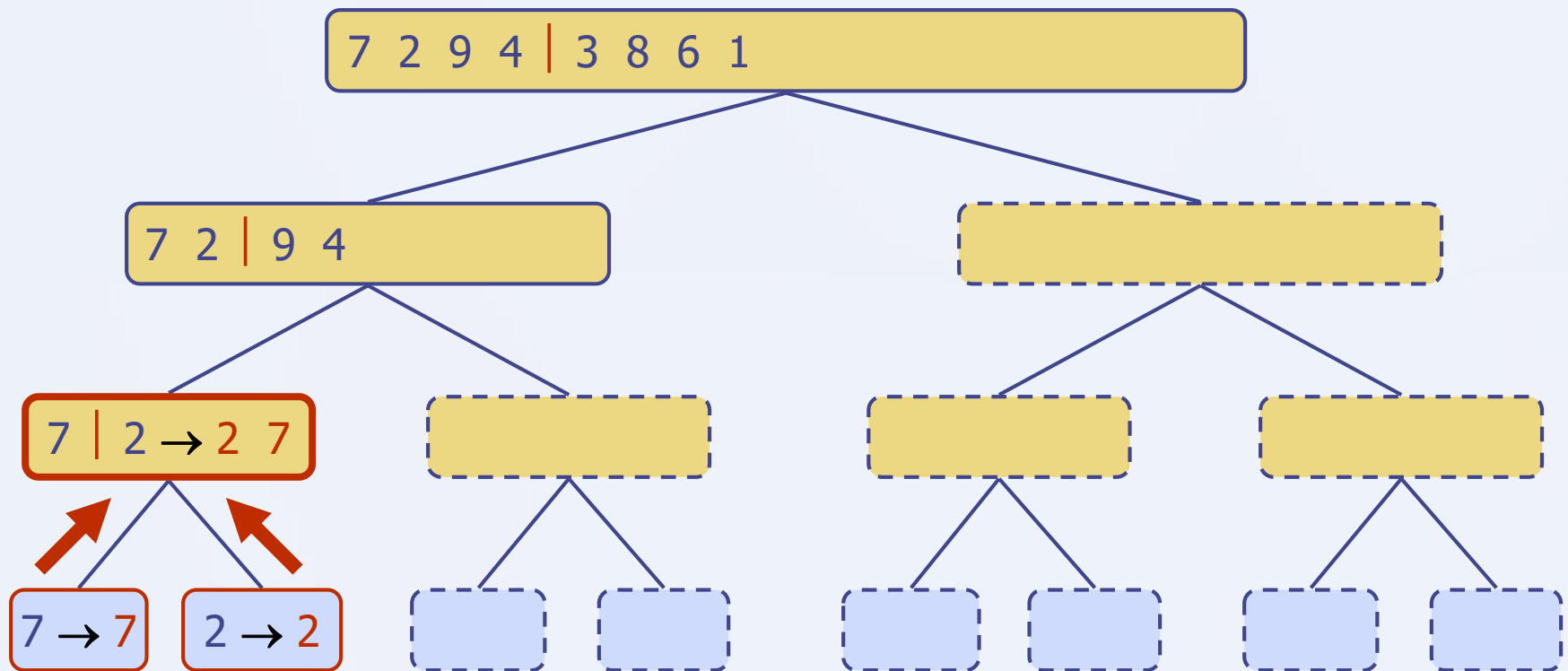
# Tiếp

- ❖ Gọi đệ qui Mergesort(A,2,2), đây là trường hợp cơ sở



# Tiếp

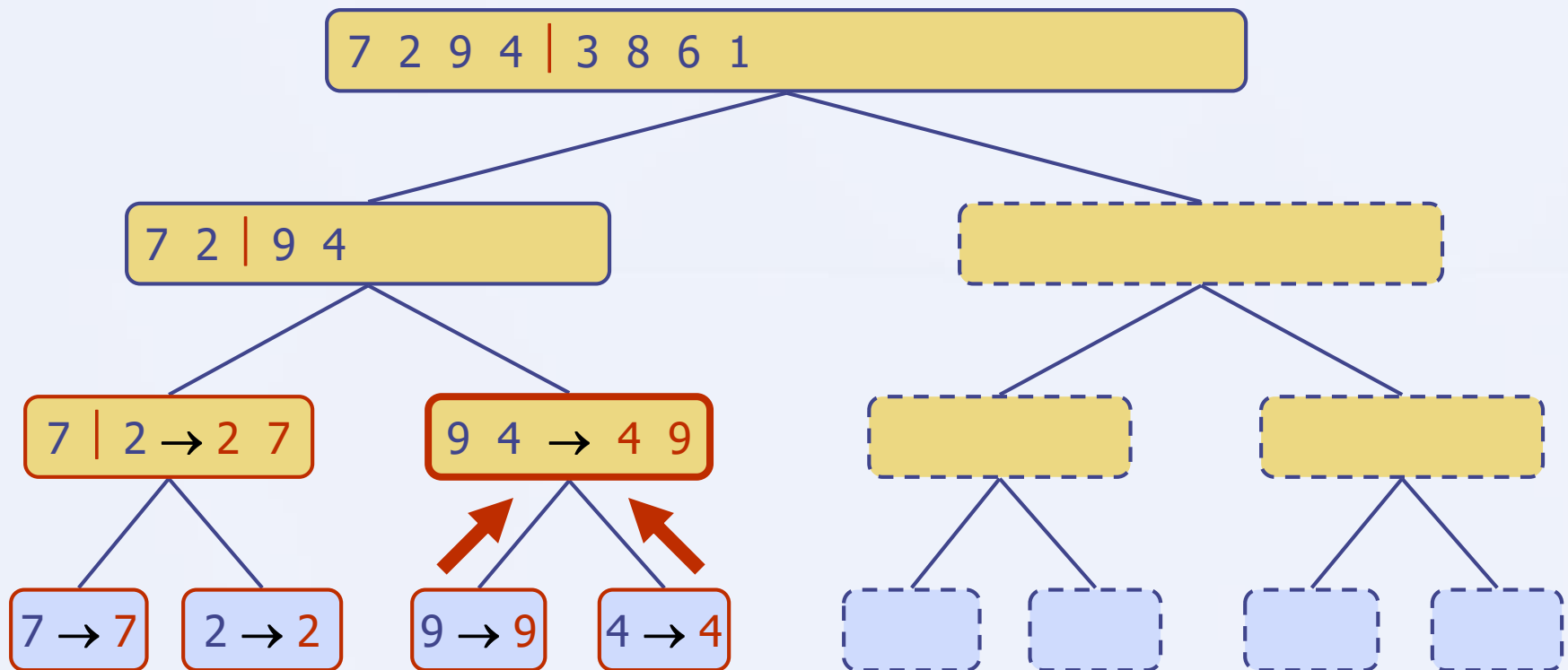
❖ Trộn merge(A,1,1,2)





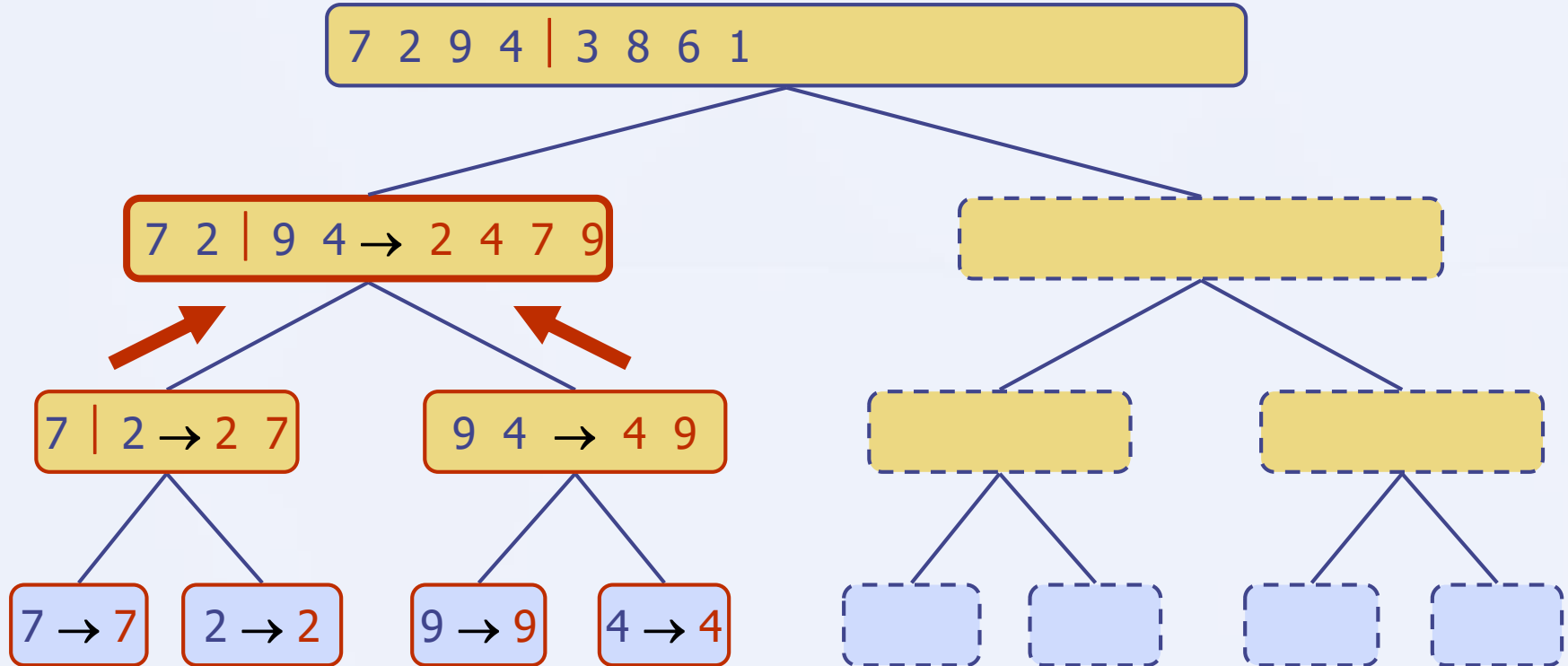
# Execution Example (cont.)

- ❖ Gọi đệ qui Mergesort(A,3,3), Mergesort(A,4,4) và trộn merge(A,3,3,4)



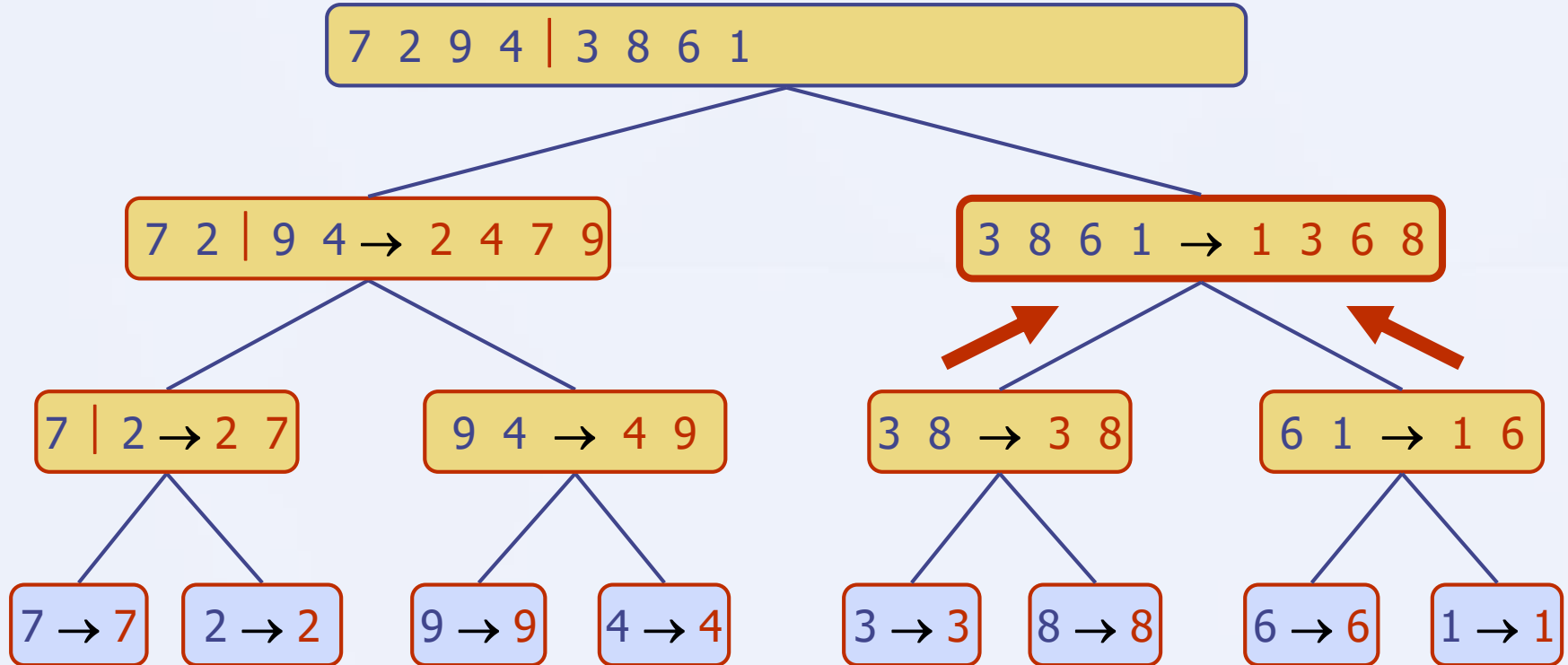
# Tiếp

❖ Trộn merge(A,1,2,4)



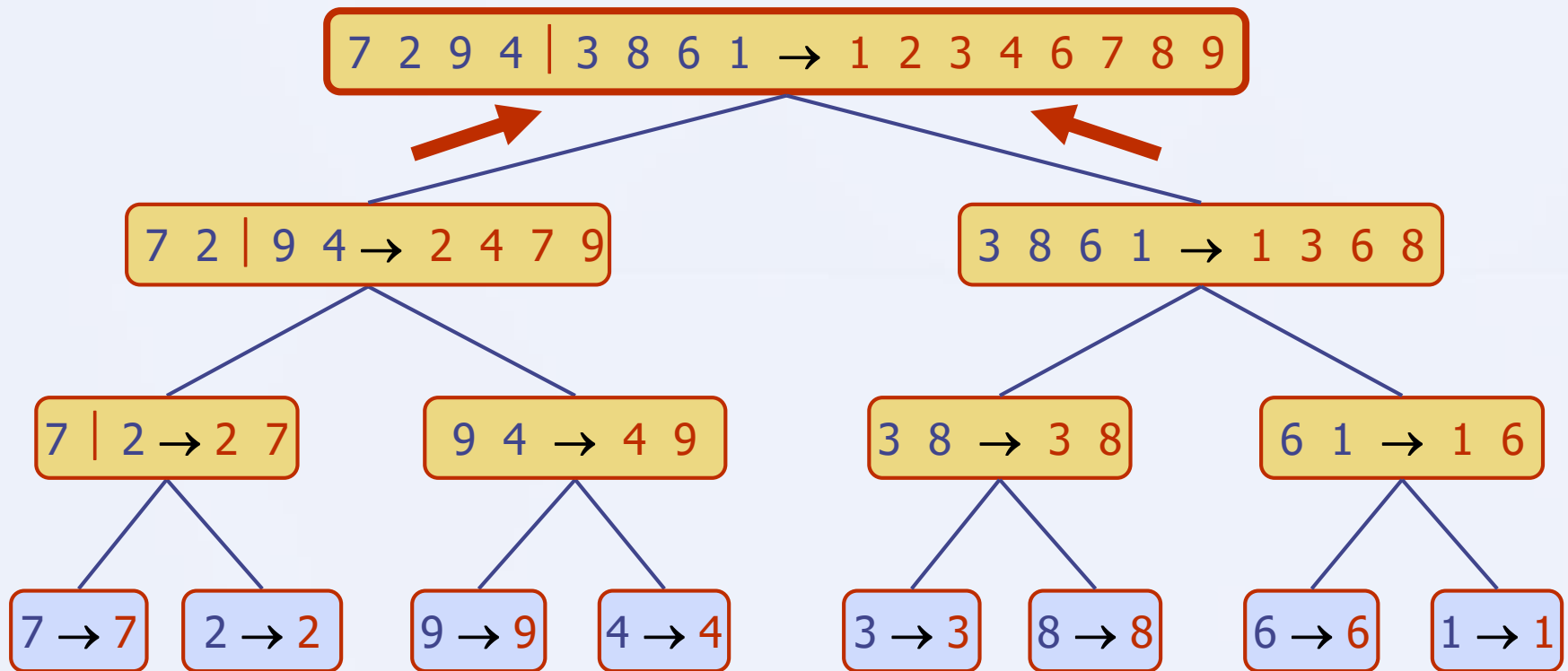
# Tiếp

❖ Tương tự như trên với nửa bên phải của dãy



# Tiếp

- ❖ Trộn hai nửa dãy thành dãy được sắp merge(A, 1, 4, 8)



# Thời gian chạy của thuật toán

- ◆ Chiều cao  $h$  của cây merge-sort là  $O(\log n)$ 
  - Tại mỗi bước gọi đệ qui ta chia dãy cần sắp thành hai phần,
- ◆ Thời tổng thời gian làm việc trên các nút ở mức  $i$  nhiều nhất là  $O(n)$ 
  - Chúng ta chia và trộn  $2^i$  chuỗi có kích thước là  $n/2^i$
  - Chúng ta gọi  $2^i+1$  lần đệ qui
- ◆ Vì vậy, tổng thời gian chạy của thuật toán mergesort là  $O(n \log n)$

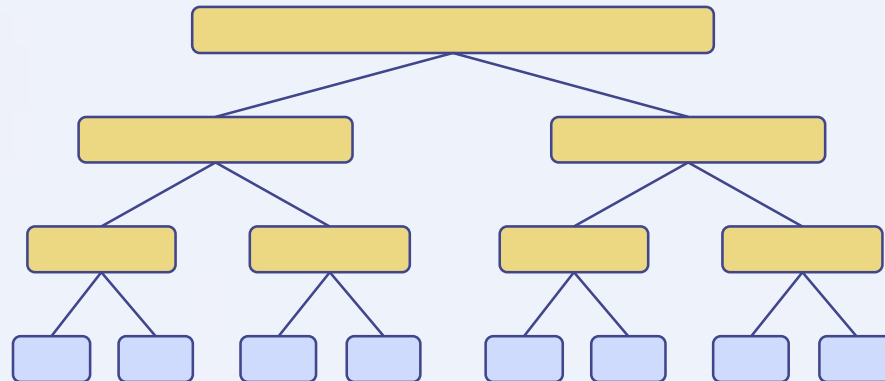
ĐSâu	#dãy	size
------	------	------

0	1	$n$
---	---	-----

1	2	$n/2$
---	---	-------

$i$	$2^i$	$n/2^i$
-----	-------	---------

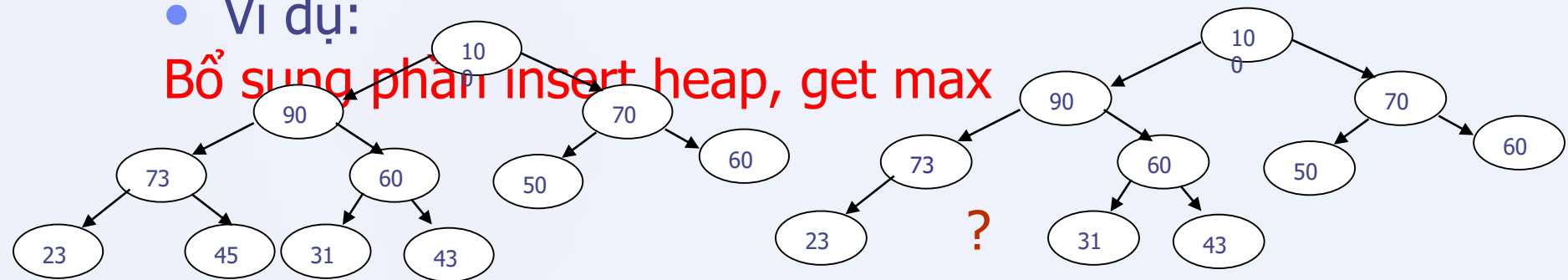
...	...	...
-----	-----	-----



# Cây Heap và Thuật toán sắp xếp vun đống Heapsort

- Cây heap (đống) là một cây nhị phân được sắp xếp theo khóa của các nút với các tính chất sau:
  - Giá trị khóa của nút gốc  $\geq$  giá trị khóa của hai con
  - Tất cả các mức đều đầy trừ mức thấp nhất có thể thiếu một số nút
  - Các nút lá phải xuất hiện liên tiếp từ trái qua phải
- Như vậy nút gốc có giá trị khóa lớn nhất
- Ví dụ:

Bổ sung phần insert heap, get max



Cây Heap

Không phải cây Heap

# Mảng biểu diễn cây heap

- Mảng  $A[1], \dots, A[n]$  là mảng biểu diễn cây heap nếu:
  - $A[i] \geq A[2i]$  và  $A[i] \geq A[2i+1]$  với  $i=1..n/2$
  - Tức là nút  $A[i]$  có cha là  $A[i/2]$ , con trái  $A[2i]$  và con phải  $A[2i+1]$
- Như vậy phần tử đầu của mảng có giá trị lớn nhất
- Ví dụ:

A	100	90	70	73	60	50	60	23	45	31	43
---	-----	----	----	----	----	----	----	----	----	----	----

$$A[1] \geq A[2], A[1] \geq A[3]$$

$$A[2] \geq A[4], A[2] \geq A[5]$$

$$A[3] \geq A[6], A[3] \geq A[7]$$

$$A[4] \geq A[8], A[4] \geq A[9]$$

$$A[5] \geq A[10], A[5] \geq A[11]$$

# Thuật toán sắp xếp vun đống

## Ý tưởng:

- Tạo mảng  $A[1], \dots, A[n]$  biểu diễn cây Heap.
- Tráo đổi phần tử  $A[1]$  với phần tử  $A[n]$ .
- Tạo mảng  $A[1], \dots, A[n-1]$  biểu diễn cây heap
- Tráo đổi phần tử  $A[1]$  với phần tử  $A[n-1]$ .
- Lặp lại quá trình trên đến khi mảng chỉ còn 1 phần tử



# Tạo đồng



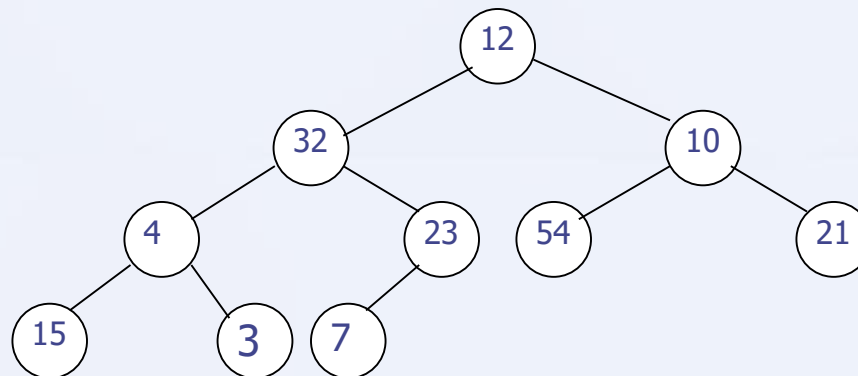
# Tạo mảng biểu diễn cây heap

- Theo tính chất của mảng biểu diễn cây Heap thì các phần tử từ  $n/2+1$  đến  $n$  (là các nút lá) không cần điều kiện ràng buộc. Vì vậy ta thực coi các phần tử này đã thỏa mãn điều kiện cây heap.
- Ta thực hiện:
  - Bổ sung phần tử  $n/2$  vào  $A[n/2+1], \dots, A[n]$  để được mảng gồm  $A[n/2], \dots, A[n]$  thỏa mãn kiện
  - Bổ sung phần tử  $n/2-1$  vào  $A[n/2], \dots, A[n]$  để được mảng gồm  $A[n/2-1], \dots, A[n]$  thỏa mãn kiện
  - Và cứ tiếp tục làm như vậy cho đến khi bổ sung phần tử  $A[1]$  vào  $A[2], \dots, A[n]$  để được mảng gồm  $A[1], \dots, A[n]$  thỏa mãn điều kiện

# Ví dụ

Cho mảng như dưới đây, hãy biến đổi mảng để được mảng thỏa mãn tính chất mảng biểu diễn cây heap

12	32	10	4	23	54	21	15	3	7
----	----	----	---	----	----	----	----	---	---



Cây tương ứng với mảng

# Mô tả trên mảng: $N=10$

12	32	10	4	23	54	21	15	3	7
----	----	----	---	----	----	----	----	---	---

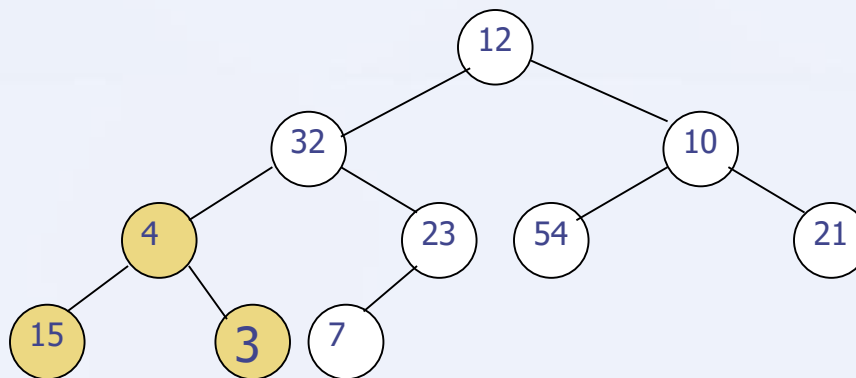
$i=5$

12	32	10	4	23	54	21	15	3	7
----	----	----	---	----	----	----	----	---	---

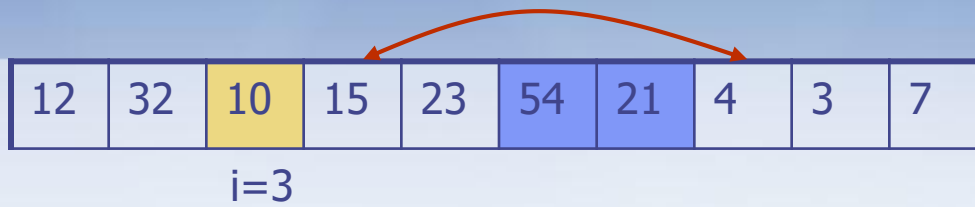
$i=4$

- Đổi chỗ  $A[5]$  và  $A[10]$  nếu  $A[5] < A[10]$

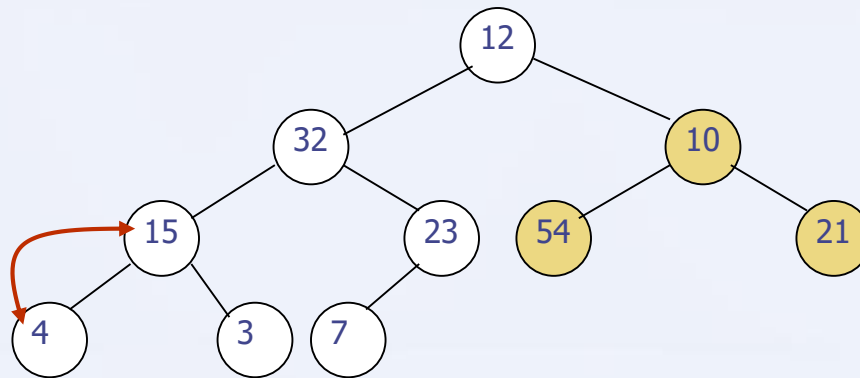
- Tính  $\max(A[8], A[9])$ . Nếu  $A[4] < \max$  thì đổi chỗ  $A[4]$  với phần tử đạt max



Cây tương ứng với mảng



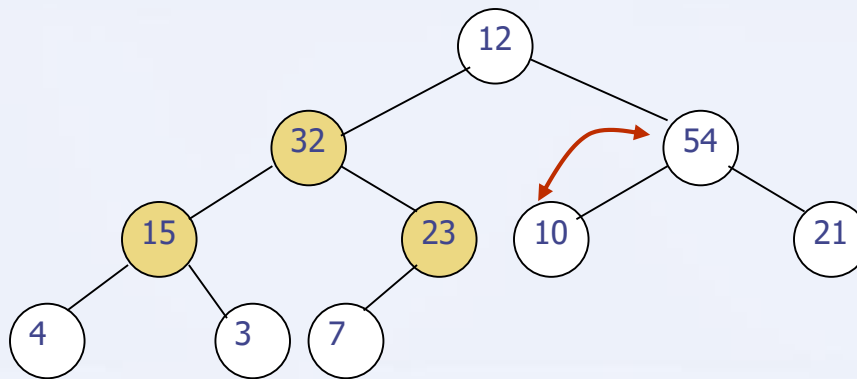
- Tính  $\max(A[6], A[7])$ . Nếu  $A[3] < \max$  thì đổi chỗ  $A[3]$  với phần tử đạt max



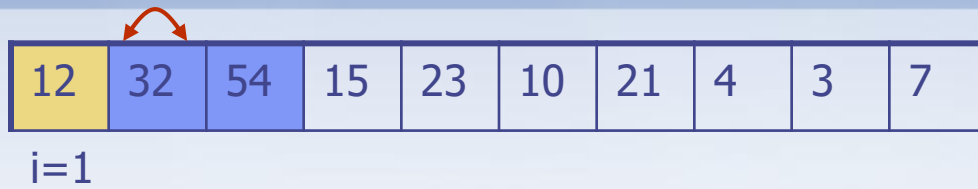
Cây tương ứng với mảng



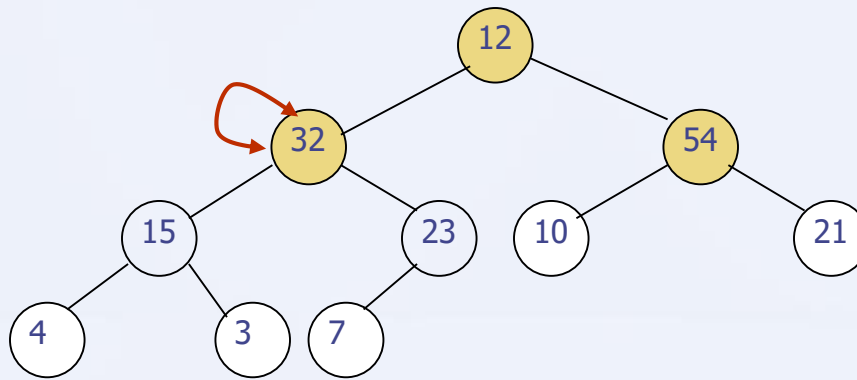
- Tính  $\max(A[4], A[5])$ . Nếu  $A[2] < \max$  thì đổi chỗ  $A[2]$  với phần tử đạt max



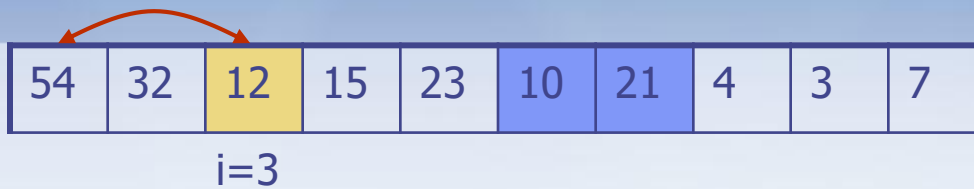
Cây tương ứng với mảng



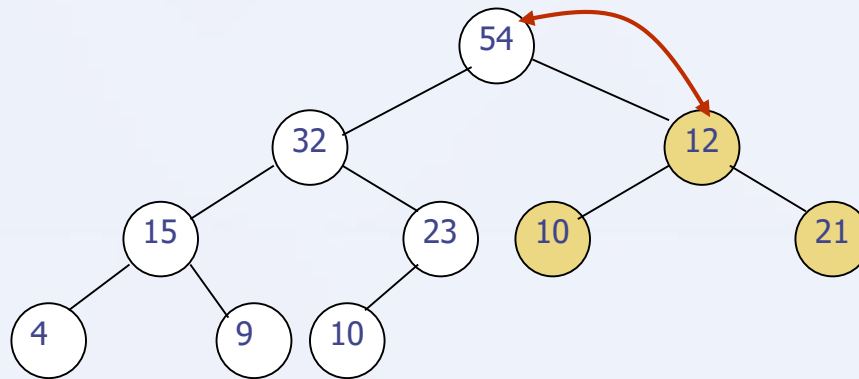
- Tính  $\max(A[2], A[3])$ . Nếu  $A[1] < \max$  thì đổi chỗ  $A[1]$  với phần tử đạt max



Cây tương ứng với mảng

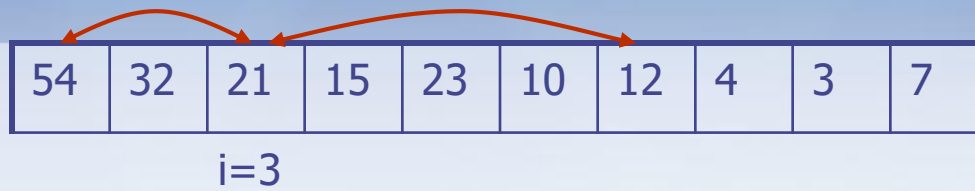


- Tính  $\max(A[6], A[7])$ . Nếu  $A[3] < \max$  thì đổi chỗ  $A[3]$  với phần tử đạt max

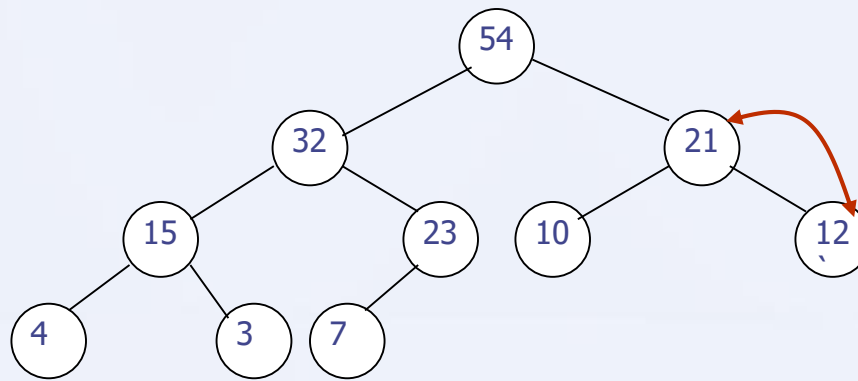


Cây tương ứng với mảng





- Tính  $\max(A[6], A[7])$ . Nếu  $A[3] < \max$  thì đổi chỗ  $A[3]$  với phần tử đạt max



Cây heap

## Thuật toán bổ sung một phần tử để tạo mảng biểu diễn cây heap

**Algorithm** *Pushdown (Array A, i, n);*

**Input:** số nguyên  $i, n$ , mảng  $A[i], \dots, A[n]$ , trong đó  $A[i+1], \dots, A[n]$  thỏa mãn tính chất cây heap

**Output:** Mảng  $A[i], \dots, A[n]$  thỏa mãn tính chất cây heap

$j \leftarrow i; kt \leftarrow 0;$

**while** ( $j \leq n/2$ ) and ( $kt=0$ ) **do**

**if**  $2*j = n$  **then**

$max \leftarrow 2*j;$

**else**

**if**  $A[2*j].key \leq A[2*j+1].key$  **then**

$max \leftarrow 2*j+1$

**else**

$max \leftarrow 2*j;$

**if**  $A[j].key < A[max].key$  **then**

        swap ( $A[j], A[max]$ );

$j \leftarrow max;$

**else**

$kt \leftarrow 1;$

# Thuật toán sắp xếp vun đống

**Algorithm** *Heapsort(Array A, n);*

**Input:** Mảng A có n phần tử và số nguyên n

**Output:** Mảng A được sắp theo thứ tự tăng dần của thuộc tính khóa

**for**  $i \leftarrow n/2$  **downto** 1 **do**

    Pushdown(A, i, n);

**for**  $i \leftarrow n$  **downto** 2 **do**

    swap(A[1], A[i]);

    Pushdown(A, 1, i-1);

# Ví dụ:

Mô tả quá trình sắp xếp của dãy số

12 43 11 34 23 43 12 435

# Thời gian chạy

- Thời gian thực hiện thủ tục Pushdown.
  - Là t/g thực hiện của vòng lặp while.
  - Gọi  $k$  là số lần lặp, ta có  $i \cdot 2^k \leq n$  hay  $k \leq \log_2(n/i)$ .
  - T/g thực hiện hàm Pushdown ( $A, i, n$ ) là  $O(\log(n/i))$
- Xét thủ tục HeapSort
  - Vòng lặp for đầu có số lần lặp là  $n/2$
  - Mỗi lần gọi hàm Pushdown 1 lần. Do đó t/g thực hiện là  $O(\log_2 n)$ .
  - Tương tự, vòng lặp for thứ 2 có số lần lặp là  $n-1$ .  $O(n \log_2 n)$ .
  - Vì vậy t/g thực hiện HeapSort là  $O(n \log_2 n)$ .

# Hết

# Bài tập

Xây dựng các thủ tục sắp xếp theo 6 phương pháp đã học