

# Cấu trúc dữ liệu và giải thuật

**TS. Phạm Tuấn Minh**

Khoa Công nghệ Thông tin, Đại học Phenikaa

[minh.phamtuan@phenikaa-uni.edu.vn](mailto:minh.phamtuan@phenikaa-uni.edu.vn)

<https://sites.google.com/site/phamtuanminh/>

## Chương 6: Một số vấn đề nâng cao

# Loop Invariant

1-3

## InsertionSort

**pre:** b 

0	b.length
?	

**post:** b 

0	b.length
đã sắp xếp	

**inv:** b 

0	i	b.length
đã sx?		

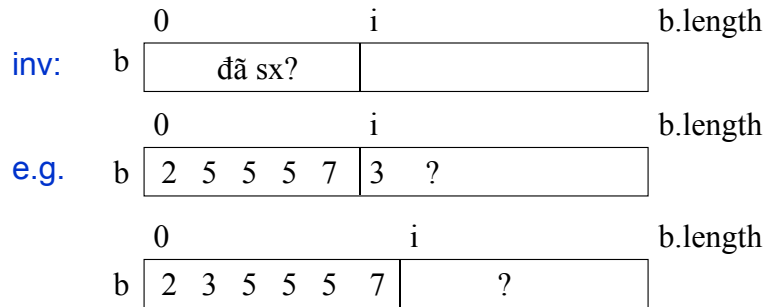
  
b[0..i-1] đã sắp xếp

**inv:** b 

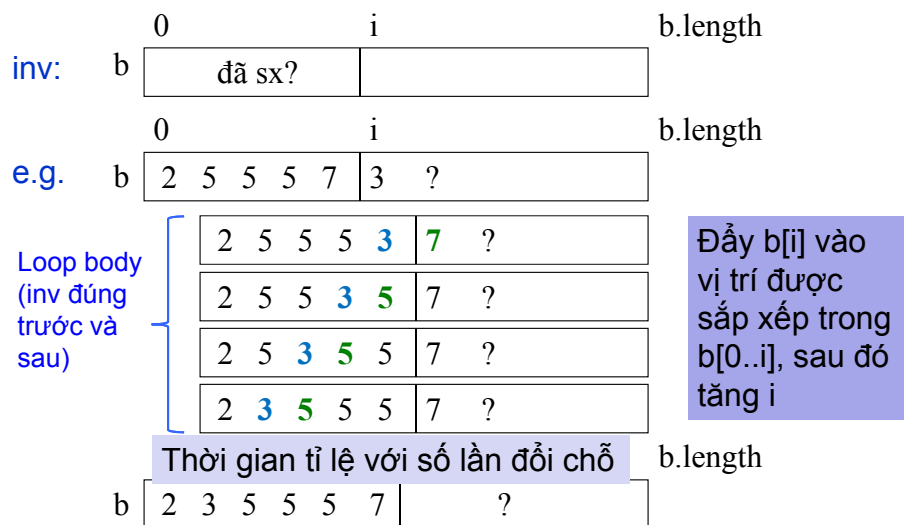
0	i	b.length
đã xử lý?		

  
b[0..i-1] đã xử lý

Mỗi vòng lặp,  $i = i+1$ : Giữ inv đúng?



Xử lý trong mỗi vòng lặp?



## Insertion Sort

```
// sắp xếp mảng số nguyên, b[]  
// inv: b[0..i-1] đã sắp xếp  
for (int i= 0; i < b.length; i= i+1) {  
    // Đẩy b[i] vào vị trí được sắp xếp  
    // trong b[0..i]  
  
}
```

## Insertion Sort

```
// sắp xếp mảng số nguyên, b[]  
// inv: b[0..i-1] đã sắp xếp  
for (int i= 0; i < b.length; i= i+1) {  
    // Đẩy b[i] vào vị trí được sắp xếp  
    // trong b[0..i]  
  
    int k= i;  
    while (k > 0 && b[k] < b[k-1]) {  
        <Đổi chỗ b[k] và b[k-1]>  
        k= k-1;  
    }  
}
```

invariant P: b[0..i] được sắp xếp, **riêng** b[k] có thể < b[k-1]

k			i	
2	5	3	5	5
7	?			

ví dụ

bắt đầu?

dừng?

tiến triển?

duy trì bất biến?

## Insertion Sort

```
// sắp xếp mảng số nguyên, b[]  
// inv: b[0..i-1] đã sắp xếp  
for (int i= 0; i < b.length; i= i+1) {  
    // Đẩy b[i] vào vị trí được sắp xếp  
    // trong b[0..i]  
}
```

`n = b.length`

- Worst-case:  $O(n^2)$  (reverse-sorted input)
- Best-case:  $O(n)$  (sorted input)
- Expected case:  $O(n^2)$

## Insertion Sort: Không xáo trộn

```
// sắp xếp mảng số nguyên, b[]  
// inv: b[0..i-1] đã sắp xếp  
for (int i= 0; i < b.length; i= i+1) {  
    // Đẩy b[i] vào vị trí được sắp xếp  
    // trong b[0..i]  
}
```

Một thuật toán sắp xếp gọi là không xáo trộn (stable) nếu hai giá trị bằng nhau giữ nguyên vị trí tương đối.

Ban đầu: (3 **7**, 2 8, **7**, 6)

Sắp xếp không xáo trộn (2, 3, 6, **7**, **7**, 8)

Sắp xếp xáo trộn (2, 3, 6, **7**, **7**, 8)

## Hiệu năng

Algorithm	Độ phức tạp trường hợp trung bình và tập trường hợp tồi nhất	Bộ nhớ	Không xáo trộn?
Insertion Sort	$O(n^2)$ . $O(n^2)$	$O(1)$	Không xáo trộn

11

## SelectionSort

pre: b 0      ?      b.length      post: b 0      đã sx      b.length

inv: b 0      đã sx <= b[i..]      i      >= b[0..i-1]      b.length

Giữ invariant đúng?

e.g.: b 0      1 2 3 4 5 6      i      9 9 9 7 8 6 9      b.length

Tăng i lên 1 và inv đúng chỉ khi b[i] là min của b[i..]

## SelectionSort

```
// sắp xếp b[]  
// inv: b[0..i-1] đã sắp xếp và  
//      b[0..i-1] <= b[i..]  
for (int i = 0; i < b.length; i = i+1) {  
    int m = index of min of b[i..];  
    <Đổi chỗ b[i] và b[m]>  
}
```

$n = b.length$

- Worst-case  $O(n^2)$
- Best-case  $O(n^2)$
- Expected-case  $O(n^2)$

0                      i                      length  
b   đã sx, giá trị nhỏ hơn   |   giá trị lớn hơn

Mỗi lần lặp, đổi chỗ giá trị min của đoạn này và b[i]

## SelectionSort: Xáo trộn

```
// sắp xếp b[]  
// inv: b[0..i-1] đã sắp xếp và  
//      b[0..i-1] <= b[i..]  
for (int i = 0; i < b.length; i = i+1) {  
    int m = index of min of b[i..];  
    <Đổi chỗ b[i] và b[m]>  
}
```

Đổi chỗ b[i] với giá trị nhỏ nhất của b[i..], 3 đổi chỗ với 8 => thay đổi vị trí tương đối của hai giá trị 8

0                      i                      length  
b   đã sx, giá trị nhỏ hơn   |   8 7 8 3 5 6

## Hiệu năng

Algorithm	Độ phức tạp trường hợp trung bình và tập trường hợp tồi nhất		Bộ nhớ	Không xáo trộn?
Insertion Sort	$O(n^2)$ .	$O(n^2)$	$O(1)$	Không xáo trộn
Selection Sort	$O(n^2)$ .	$O(n^2)$	$O(1)$	Xáo trộn

15

## Quicksort



## Thuật toán phân đoạn

pre: 

h	h+1	k
x	?	

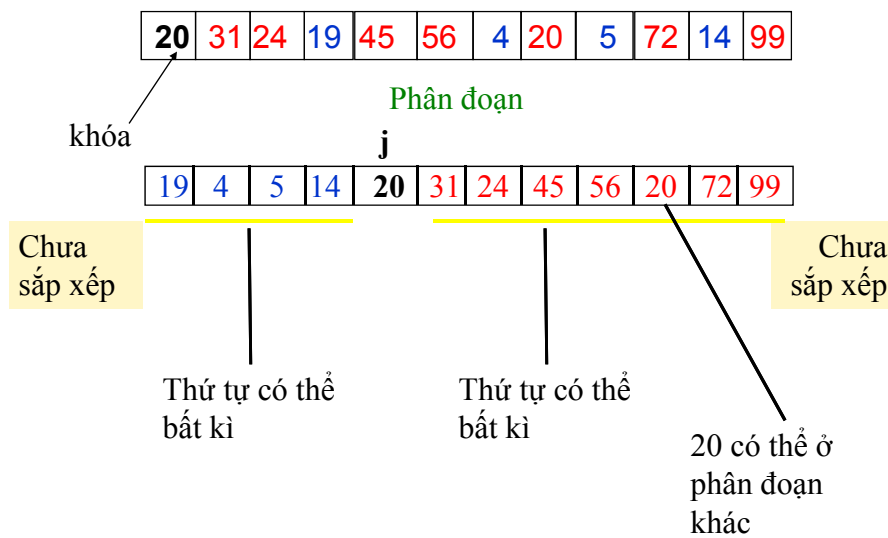
x là khóa

Đổi chỗ tới khi :

post: 

h	j	k
$\leq x$	x	$\geq x$

## Thuật toán phân đoạn



## Thuật toán phân đoạn

pre:  $b$   $\begin{array}{|c|c|} \hline x & ? \\ \hline \end{array}$   $\begin{array}{c} h \quad h+1 \quad k \end{array}$

post:  $b$   $\begin{array}{|c|c|c|} \hline \leq x & x & \geq x \\ \hline \end{array}$   $\begin{array}{c} h \quad j \quad k \end{array}$

Kết hợp pre và post để xác định invariant

$b$   $\begin{array}{|c|c|c|c|} \hline \leq x & x & ? & \geq x \\ \hline \end{array}$   $\begin{array}{c} h \quad j \quad t \quad k \end{array}$

invariant  
cần ít  
nhất 4  
phần

## Thuật toán phân đoạn

$b$   $\begin{array}{|c|c|c|c|} \hline \leq x & x & ? & \geq x \\ \hline \end{array}$   $\begin{array}{c} h \quad j \quad t \quad k \end{array}$

```
j = h; t = k;
while (j < t) {
    if (b[j+1] <= b[j]) {
        Đổi chỗ b[j+1], b[j]; j = j+1;
    } else {
        Đổi chỗ b[j+1], b[t]; t = t-1;
    }
}
```

Khởi tạo,  $j = h$  và  $t = k$ , sơ đồ giống sơ đồ bắt đầu

Kết thúc, khi  $j = t$ , phần “?” rỗng, sơ đồ thành sơ đồ kết quả

Thời gian tuyến tính:  $O(k+1-h)$

## QuickSort

**/\*\* Sắp xếp b[h..k]. \*/**

**public static void QS(int[] b, int h, int k) {**

**if (b[h..k] có 1 phần tử) return;**

Base case

**int j= partition(b, h, k);**

**// Có b[h..j-1] <= b[j] <= b[j+1..k]**

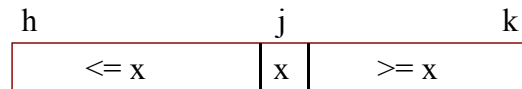
**// Sắp xếp b[h..j-1] and b[j+1..k]**

**QS(b, h, j-1);**

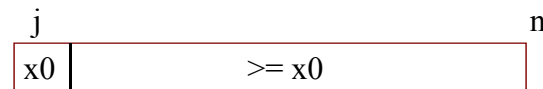
**QS(b, j+1, k);**

**}**

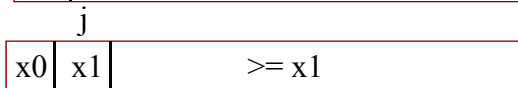
Hàm thực hiện thuật toán phân đoạn và trả về vị trí j của khóa



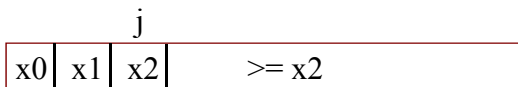
## Trường hợp tồi nhất: Khóa luôn là giá trị nhỏ nhất



Phân đoạn tại 0



Phân đoạn tại 1



Phân đoạn tại 2

**/\*\* Sắp xếp b[h..k]. \*/**

**public static void QS(int[] b, int h, int k) {**

**if (b[h..k] has < 2 elements) return;**

**int j= partition(b, h, k);**

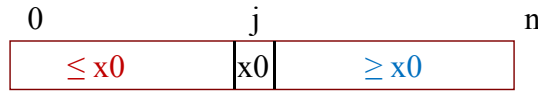
**QS(b, h, j-1); QS(b, j+1, k);**

Độ sâu đệ quy:  
 $O(n)$

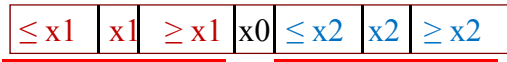
Xử lý tại độ sâu  
i:  $O(n-i)$

$O(n*n)$

## Trường hợp tốt nhất: Khóa luôn là giá trị ở giữa



Độ sâu 0. Phân đoạn  $n$



Độ sâu 1. Phân 2 đoạn độ dài  $\leq n/2$



Độ sâu 2. Phân 4 đoạn độ dài  $\leq n/4$

Độ sâu tối đa:  $O(\log n)$ . Thời gian phân đoạn tại mỗi độ sâu  $O(n)$

Tổng thời gian:  $O(n \log n)$ .

## Cấu trúc dữ liệu và giải thuật

- Nội dung bài giảng được biên soạn bởi TS. Phạm Tuấn Minh.