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

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

Loop Invariant

InsertionSort

0 b.length
pre: b | ? |

0 b.length
post: b | đã sắp xếp |

0 i b.length
inv: b | đã sx? | |

b[0..i-1] đã sắp xếp

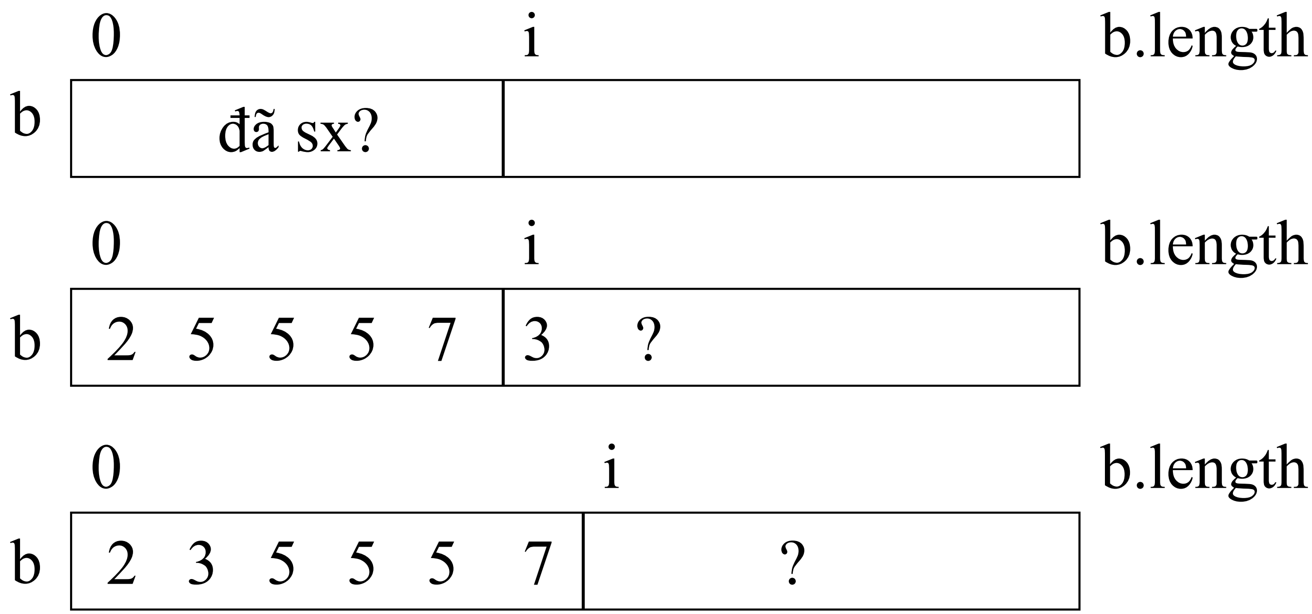
0 i b.length
inv: b | đã xử lý? | |

b[0..i-1] đã xử lý

Mỗi vòng lặp, i = i+1: Giữ inv đúng?

inv:

e.g.



Xử lý trong mỗi vòng lặp?

inv:

	0	i	b.length
b	đã sx?		

e.g.

	0	i	b.length
b	2 5 5 5 7	3 ?	

Loop body (inv đúng trước và sau)

2 5 5 5 **3**	**7** ?
2 5 5 **3** **5**	7 ?
2 5 **3** **5** 5	7 ?
2 **3** **5** 5 5	7 ?

Đẩy b[i] vào vị trí được sắp xếp trong b[0..i], sau đó tăng i

Thời gian tỉ lệ với số lần đổi chỗ

		b.length
b	2 3 5 5 5 7	?

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 tạp trường hợp tồi nhất	Bộ nhớ	Không xáo trộn?
Insertion Sort	$O(n^2)$. $O(n^2)$	$O(1)$	Không xáo trộn

SelectionSort

pre: b

0	b.length
?	

post: b

0	b.length
đã sx	

inv: b

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

Giữ invariant đúng?

e.g.: b

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

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 tạp trường hợp tối 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

Quicksort

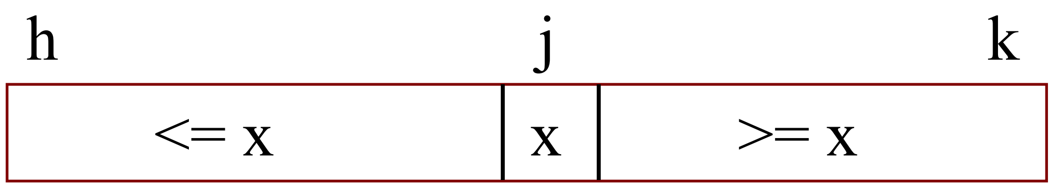
Thuật toán phân đoạn

x là khóa

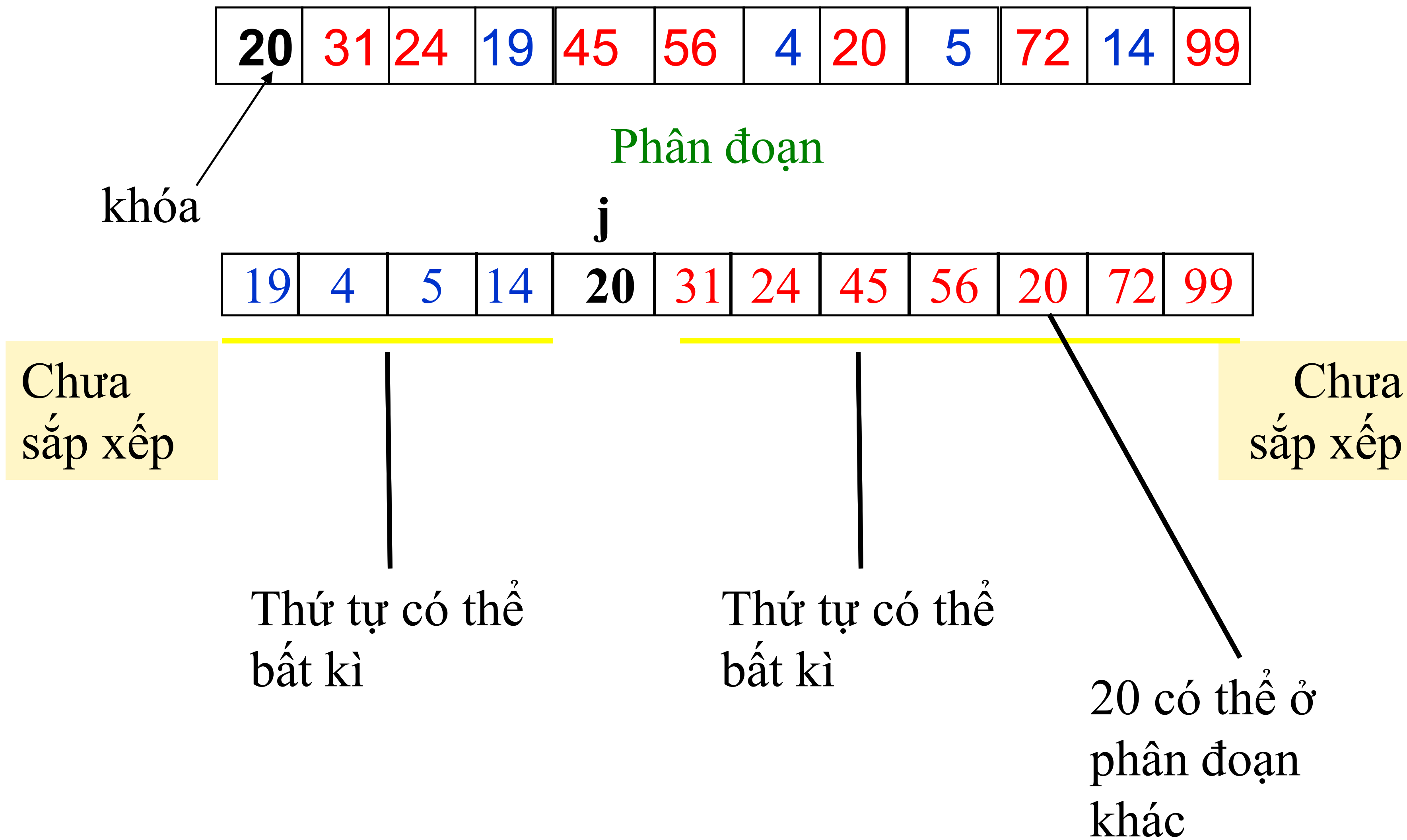
pre:

Đổi chỗ tới khi :

post:



Thuật toán phân đoạn



Thuật toán phân đoạn

pre:

	h	h+1 … k
b	x	?

post:

	h	j	k
b	<= x	x	>= x

Kết hợp pre và post để xác định invariant

	h	j	t	k
b	<= x	x	?	>= x

invariant cần ít nhất 4 phần

Thuật toán phân đoạn

	h	j		t k
b	<= x	x	?	>= x

```
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

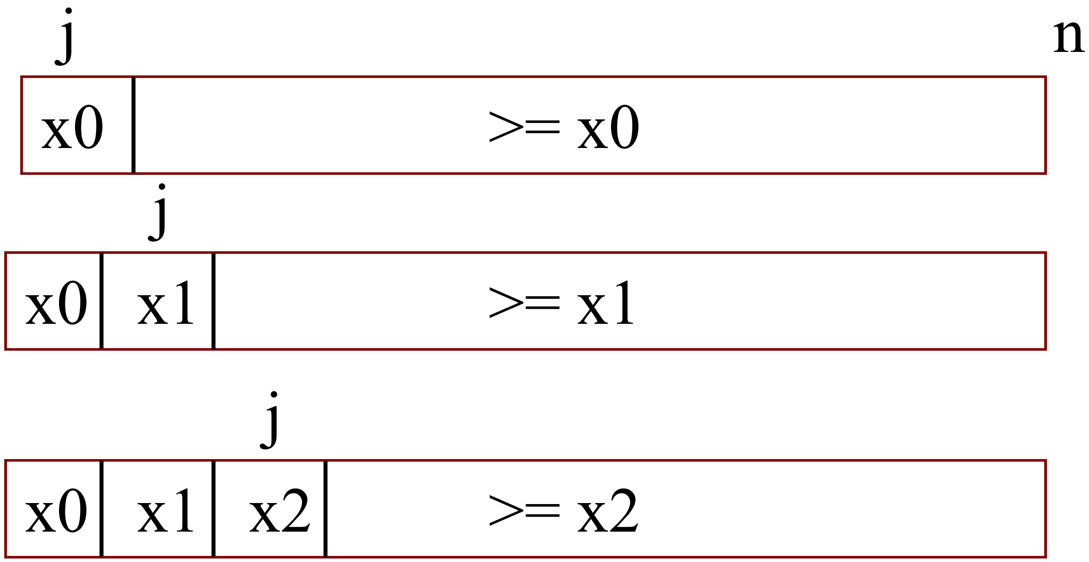
```
/** Sắp xếp b[h..k]. */
public static void QS(int[] b, int h, int k) {
   if (b[h..k] có 1 phần tử) return;
    int j=  partition(b, h, k);
      // Có b[h..j–1] <= b[j] <= b[j+1..k]

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

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

Base case

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

h	j	k
<= x	x	>= x

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



Phân đoạn tại 0

Phân đoạn tại 1

Phân đoạn tại 2

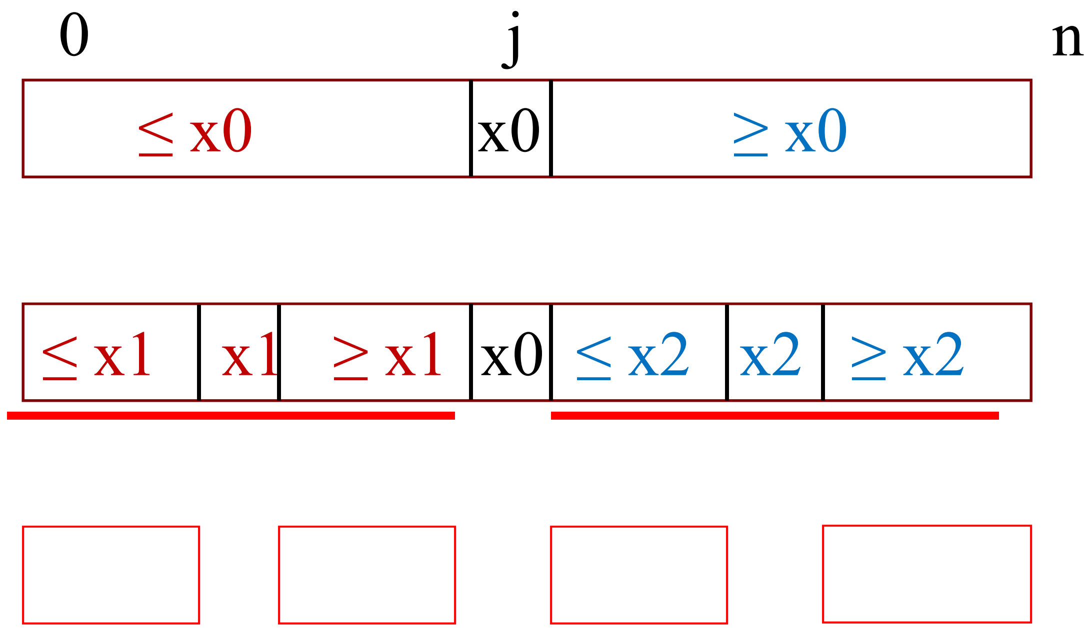
```
/** Sắp xếp b[h..k]. */
public static void QS(int[] b, int h, int
k) {
   if (b[h..k] has < 2 elements) return;
   int j=  partition(b, h, k);
   QS(b, h, j-1);     QS(b, j+1, k);
```

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 <= n/2

Độ sâu 2. Phân 4 đoạn độ dài <= 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).