Giải thuật Tìm kiếm (Search algorithm)

Lecturer: Duc-Hieu Tran

Title: MSc. Computer Science

Nội dung

- Khái quát về tìm kiếm
- Tìm kiếm tuyến tính
- Tìm kiếm nhị phân
- So sánh đánh giá
- Kết luận

Khái quát

❖Giới thiệu

- **Tìm kiếm là một thao tác rất phổ biến trong cuộc sống hàng ngày**
  - **Tìm kiếm tài liệu, tìm kiếm nội dung, tìm kiếm đối tượng,…**
  - **Tìm kiếm hồ sơ, tìm kiếm dữ liệu, tìm kiếm tin tức,…**
- **Ví dụ**
  - **Trong Database: tìm kiếm 1 customer, tìm 1 bank account, tìm 1 record,…**
  - **Internet: Google search, Binq search, CocCoc search,…**
  - **…**

Khái quát

❖Giới thiệu

- **Trong hầu hết các hệ thống phần mềm**
  - **Tìm kiếm là thao tác bắt buộc phải có**
  - **Cho phép quản lý và tra cứu dữ liệu**
  - **Tìm kiếm luôn phải được thực hiện một cách nhanh chóng và hiệu quả**

Khái quát

❖Giới thiệu

- **Có nhiều phương pháp tìm kiếm**
  - Tìm kiếm tuần tự / tuyến tính (Sequential / Linear Search)
  - Tìm kiếm nhị phân (Binary Search)
  - Tìm kiếm trên từ điển
  - …
- **Mục tiêu bài học**
  - Tìm hiểu về 2 thuật toán tìm kiếm cơ bản
  - Đánh giá để lựa chọn thuật toán phù hợp khi áp dụng

Tìm kiếm
Tuyến tính

Tìm kiếm tuyến tính

❖Bài toán

❑INPUT

- Một danh sách/mảng A = [a_0, a_1, …, a_{n-1}] có n phần tử dữ liệu
- Giá trị *x*

❑PROCESS

- Tìm vị trí xuất hiện của *x* trong A ⇔ Tìm chỉ số i sao cho a[i] = *x*

❑OUTPUT

- Vị trí xuất hiện đầu tiên của *x* nếu *x* ∈ A
- Trả về -1 nếu *x* ∉ A

Tìm kiếm tuyến tính

❖Linear / Sequential Search

❑Phương pháp

- Vét cạn (exhaustive)
- Lính canh (sentinel)
- Hướng thiết kế từ dưới lên (Bottom-up Design)

❑Cấu trúc dữ liệu áp dụng

- Mảng một chiều / nhiều chiều
- Danh sách liên kết đơn

Phương pháp Vét cạn

❖**Ý tưởng**: Duyệt qua từng phần tử của mảng A = [a_0, a_1, …, a_{n-1}] cho đến khi gặp được a[i] = *x* hoặc không còn phần tử nào để duyệt

❖**Thuật giải**

Bước 1: khởi tạo i = 0

Bước 2: So sánh A[i] với *x*

Nếu (A[i] == *x*) ⇒ trả về i ⇒ Dừng

Ngược lại, đi tới bước 3

Bước 3: Tăng i = i + 1

Nếu (i == n) ⇒ hết mảng & không tìm thấy. Trả về -1 ⇒ Dừng

Ngược lại, quay về bước 2

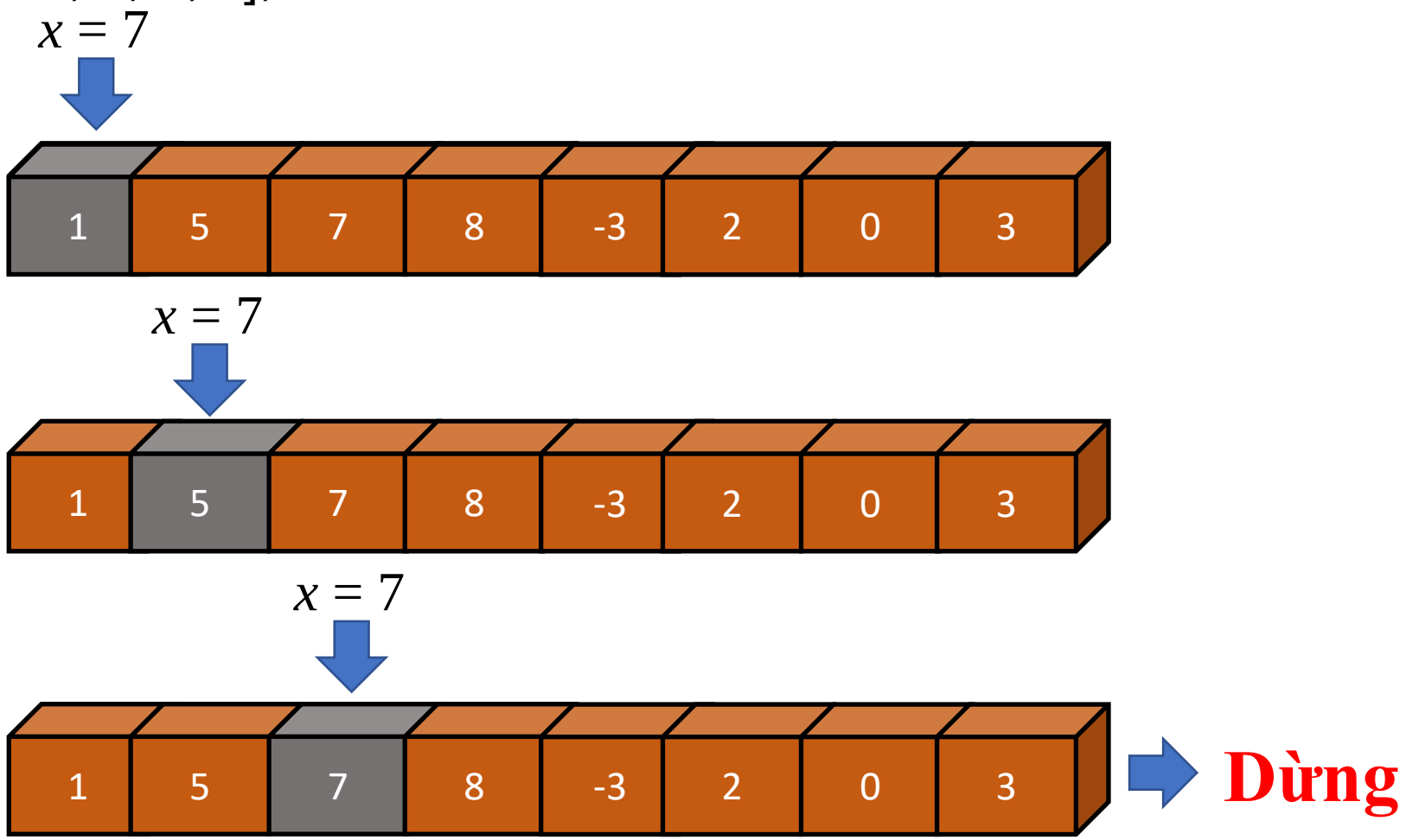
Phương pháp Vét cạn

❖Java

```
public static int Find(int x, int[] A, int n) {
	int result = -1;
	for (int i=0; i<n; i++) {
		if (A[i] == x) {
			result = i;
			break;
		}
	}
	return result;
}
```

Phương pháp Vét cạn

❖**Ví dụ:** A=[1, 5, 7, 8, -3, 2, 0, 3], x = 7



Phương pháp Vét cạn

❖Độ phức tạp giải thuật

- Phép toán chính là phép so sánh
- Trường hợp tốt nhất **a[0] = *x*** ⇒ tốn 1 phép so sánh ⇔ O(1)
- Trường hợp xấu nhất **a[n-1] = *x*** ⇒ tốn n phép so sánh ⇔ O(n)
- Trường hợp trung bình **a[n/2] = *x*** ⇒ tốn (n+1)/2 phép so sánh ⇔ O(n)

Phương pháp Vét cạn

❖Độ phức tạp giải thuật

➢ Trường hợp trung bình

$$f(n) = \frac{1 + 2 + 3 + \ldots + n}{n} = \frac{n(n+1)/2}{n} = \frac{n+1}{2}$$

Ta có:

$f(n) = \frac{1}{2}n + \frac{1}{2} \Leftrightarrow f(n) \leq c.g(n)$ với $g(n) = n$ và $\forall c > 0$ và $n \geq 0$

Vậy tiệm cận trên của f(n) là O(n) hay độ phức tạp của giải thuật trong trường hợp trung bình là O(n)

Phương pháp Vét cạn

❖Độ phức tạp giải thuật

➢ Trường hợp xấu nhất

Ta có:

$f(n) = n \leq c.n \Leftrightarrow f(n) \leq c.g(n)$ với $g(n) = n$ và $\forall c > 0$ và $n \geq 0$

Vậy tiệm cận trên của f(n) là O(n) hay độ phức tạp của giải thuật là O(n)

Phương pháp Lính canh

❖Nhận xét

- Trong thuật toán vét cạn, ta luôn phải kiểm tra điều kiện kết thúc mảng ở mỗi bước lặp
- Có thể bỏ qua điều kiện kiểm tra này bằng cách thêm vào một phần tử "lính canh"
- "Lính canh" là phần tử có giá trị bằng với *x* và được thêm vào cuối mảng

Phương pháp Lính canh

❖**Ý tưởng**: Duyệt qua từng phần tử của mảng A = [a_0, a_1, …, a_{n-1}, x] cho đến khi gặp được a[i] = x

❖**Thuật giải**

Bước 1: khởi tạo i = 0;

Bước 2: So sánh A[i] với x

Nếu (A[i] == x) ⇒ trả về i; đi tới bước 3

Ngược lại, tăng i = i + 1, quay lại bước 2

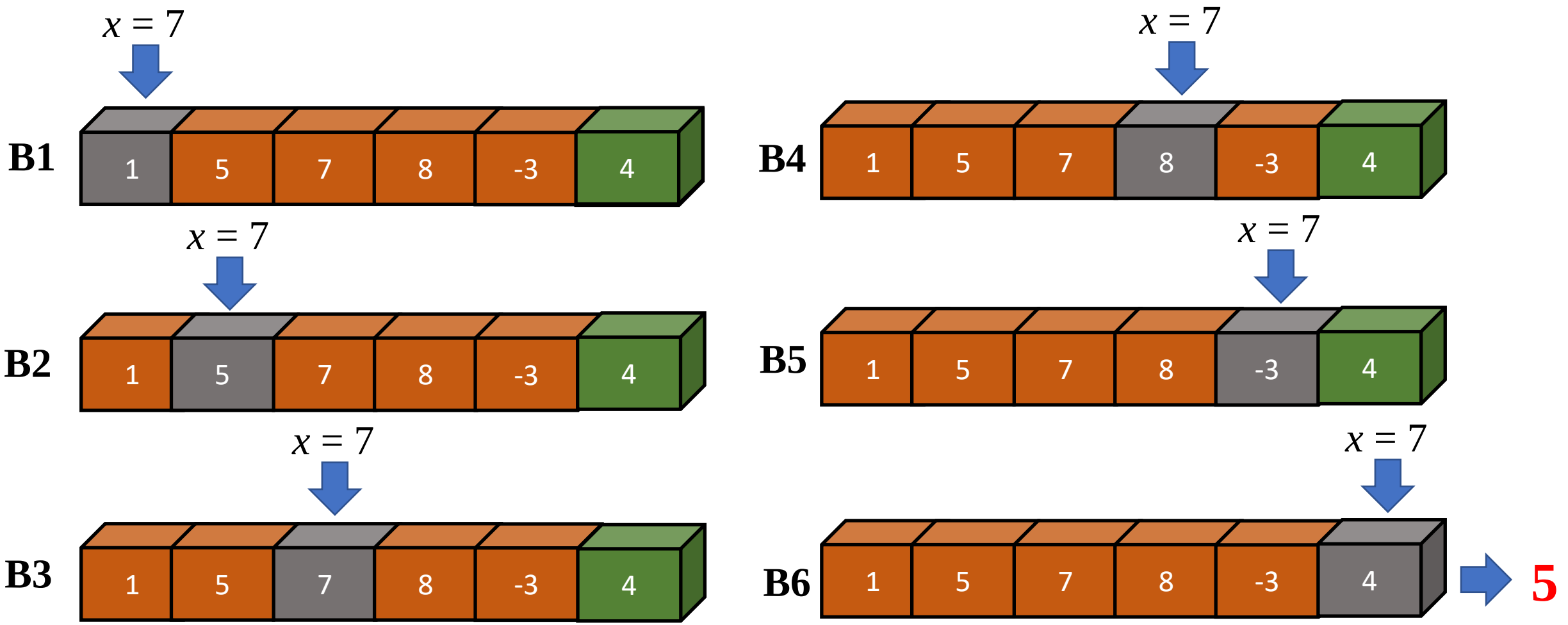
Bước 3: So sánh i với n

Nếu i < n ⇒ i là vị trí của x trong A

Nếu i ≥ n ⇒ không tìm thấy x trong A. Trả về -1;

Phương pháp Lính canh

❖**Ví dụ**: A=[1, 5, 7, 8, -3], $x = 4$



Phương pháp Lính canh

❖Độ phức tạp giải thuật

- Phép toán chính là phép so sánh
- Trường hợp tốt nhất **a[0] = *x*** ⇒ tốn 1 phép so sánh ⇔ O(1)
- Trường hợp xấu nhất **a[n] = *x*** ⇒ tốn n+1 phép so sánh ⇔ O(n)
- Trường hợp trung bình **a[(n+1)/2] = *x*** ⇒ tốn (n+2)/2 phép so sánh ⇔ O(n)

Phương pháp Lính canh

❖**Xem xét thời gian chạy của giải thuật**

- Số phép so sánh ít hơn so với vét cạn
- Thực nghiệm cho thấy, trong trường hợp n lớn, thời gian tìm kiếm giảm khi dùng phương pháp lính canh
- **Ví dụ**: với n = 15000: nhanh hơn phương pháp vét cạn khoảng 20%

Tìm kiếm
Nhị phân

Tìm kiếm Nhị phân

❖Binary Search

❑Phương pháp

- Chia để trị (Divide-and-conquer)
- Thiết kế từ trên xuống (Top-down Design)

❑Cấu trúc dữ liệu áp dụng

- Mảng một chiều / nhiều chiều **đã sắp xếp thứ tự** (tăng dần hoặc giảm dần)
- Danh sách liên kết đơn **đã sắp xếp thứ tự** (tăng dần hoặc giảm dần)

Tìm kiếm Nhị phân

❖Ý tưởng

- Giả sử mảng A = [a_0, a_1,…, a_{n-1}] đã có thứ tự tăng dần ($a_0 \le a_1 \le \ldots \le a_{n-1}$)
- So sánh *x* với phần tử chính giữa mảng ($a_{n/2}$)
- Nếu *x* là phần tử giữa thì dừng
- Nếu *x* nhỏ hơn phần tử giữa ⇒ Xem xét mảng A[a_0, a_1,…,$a_{n/2-1}$]
- Nếu *x* lớn hơn phần tử giữa ⇒ Xem xét mảng A[$a_{n/2+1}$, $a_{n/2+2}$,…,a_{n-1}]
- Lặp lại cho tới khi tìm được *x* hoặc không thể chia mảng ra được nữa

Tìm kiếm Nhị phân

❖Thuật toán

Bước 1: khởi tạo left = 0 và right = n-1;

Bước 2: Nếu left ≤ right, thực hiện:

Đặt mid = (left + right) / 2;

So sánh *x* và A[mid]

Nếu *x* < A[mid], gán right = mid – 1;

Nếu *x* > A[mid], gán left = mid + 1;

Nếu *x* = A[mid] ⇒ trả về mid và kết thúc;

Quay lại bước 2

(Giải thuật sẽ kết thúc khi tìm thấy *x* hoặc left > right)

Tìm kiếm Nhị phân

❖Java – kĩ thuật đệ qui

```
public static int Find(int x, int[] A, int left, int right) {
        if (left > right) return -1;
        else {
                int mid = (left + right) / 2;
                if (x < A[mid]) return Find(x, A, left, mid-1);
                else if (x > A[mid]) return Find(x, A, mid+1, right);
                else return mid; // Tìm thấy x tại vị trí mid
        }
}
```

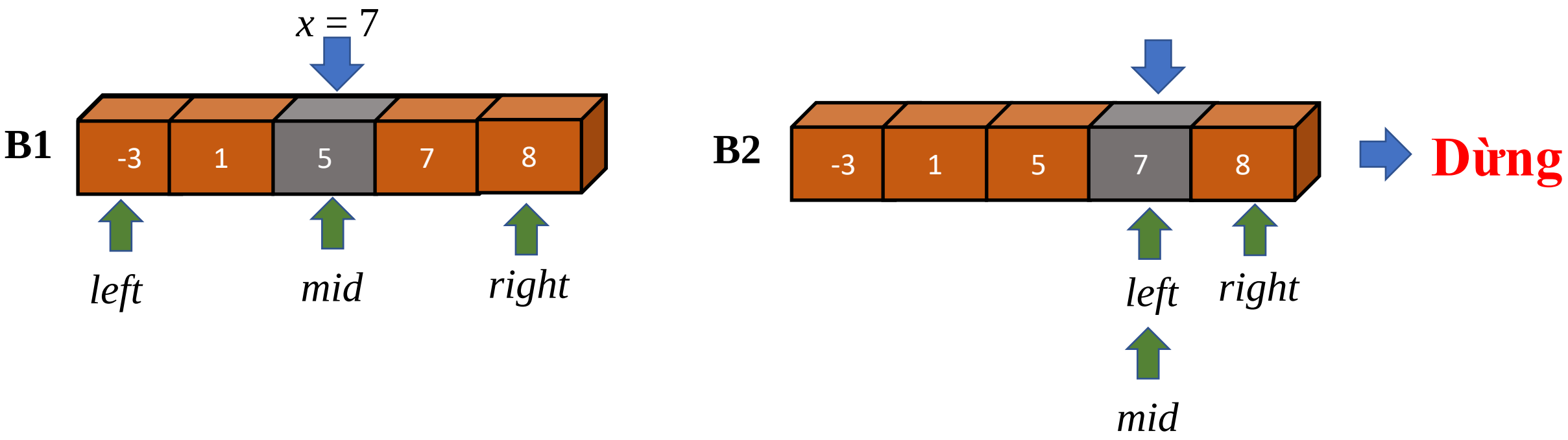
Tìm kiếm Nhị phân

❖Java – kĩ thuật vòng lặp (khử đệ qui)

```
public static int Find(int x, int[] A, int left, int right) {
    int mid;
    while (left <= right) {
        mid = (left + right) / 2;
        if (x == A[mid]) return mid;
        if (x < A[mid]) right = mid - 1;
        if (x > A[mid]) left = mid + 1;
    }
    return -1;
}
```

Tìm kiếm Nhị phân

❖**Ví dụ:** A=[-3, 1, 5, 7, 8], $x = 7$



Tìm kiếm Nhị phân

❖**Ví dụ:** A=[1, 3, 4, 5, 8, 15, 17, 22, 25, 30], x = 5

(*cách trình bày bài thi lý thuyết trên giấy*)

Lần lặp	First	Last	First>Last	Mid	A[Mid]	x = A[Mid]	x < A[Mid]	x > A[Mid]

Tìm kiếm Nhị phân

❖**Ví dụ:** A=[1, 3, 4, 5, 8, 15, 17, 22, 25, 30], $x = 5$

(*cách trình bày bài thi lý thuyết trên giấy*)

Lần lặp	First	Last	First>Last	Mid	A[Mid]	x = A[Mid]	x < A[Mid]	x > A[Mid]
Ban đầu	0	9	False	4	8	False	True	False
1	0	3	False	1	3	False	False	True
2	2	3	False	2	4	False	False	True
3	3	3	False	3	5	True		

⇒ Tìm thấy x tại vị trí thứ 3

Như vậy, sau 3 lần lặp (đệ quy) thuật toán kết thúc

Tìm kiếm Nhị phân

❖Độ phức tạp giải thuật

- Kích thước của mảng giảm khoảng ½ sau mỗi lần xét
- Gọi k là số lần phân chia mảng, ta được:

$$2^{k-1} < n < 2^k \Leftrightarrow k-1 < \log_2 n < k \Rightarrow k = 1 + \lfloor \log_2 n \rfloor$$

- Số lần thực hiện vòng lặp là khoảng k lần, mỗi vòng lặp thực hiện 1 phép so sánh

Tìm kiếm Nhị phân

❖Độ phức tạp giải thuật

- k là số lần lặp của giải thuật
- Trường hợp tốt nhất k=1 ⇔ *x* là phần tử chính giữa của mảng
- Trường hợp xấu nhất k=1 + $\lfloor \log_2 n \rfloor$ ⇔ *x* không thuộc mảng hoặc *x* là phần tử cuối cùng của mảng
- Trường hợp trung bình k = (1 + $\lfloor \log_2 n \rfloor$) / 2 ⇔ *x* không phải là phần tử đầu tiên hoặc cuối cùng của mảng

Tìm kiếm Nhị phân

❖Độ phức tạp giải thuật

- Trường hợp tốt nhất k=1: O(1)
- Trường hợp xấu nhất k=1 + $\lfloor \log_2 n \rfloor$: O($\log_2 n$)
- Trường hợp trung bình k = (1 + $\lfloor \log_2 n \rfloor$) / 2: O($\log_2 n$)

So sánh hiệu suất

❖So sánh trường hợp xấu nhất của 2 thuật toán

Kích thước mảng	Trường hợp xấu nhất	
	Tìm kiếm tuần tự	Tìm kiếm nhị phân
100.000	100.000	16
200.000	200.000	17
400.000	400.000	18
800.000	800.000	19
1.600.000	1.600.000	20

Kết luận

- Việc tìm kiếm trong trường hợp tổng quát thường được áp dụng với giải thuật tìm kiếm tuần tự
- Nếu dữ liệu đầu vào là một mảng đã có thứ tự thì nên dùng thuật toán tìm kiếm nhị phân
- Tìm kiếm nhị phân dùng kết quả của phép so sánh để thu hẹp vùng tìm kiếm kế tiếp
- Hiệu suất tìm kiếm nhị phân là một hàm logarit theo số phần tử của mảng

Bài tập

❖Cho một mảng số nguyên có các phần tử:

-9 -9 -5 -2 0 3 7 7 10 15

a/ Tính số lần so sánh để tìm ra phần tử *x* = -9 bằng phương pháp

a.1. Tìm kiếm tuyến tính (tuần tự)

a.2. Tìm kiếm nhị phân

Nhận xét và so sánh 2 phương pháp tìm kiếm trên trong t/h tổng quát (*x* có giá trị bất kì)

b/ Hãy cho biết trong t/h tìm kiếm nhị phân, phần tử -9 nào sẽ được tìm thấy ? (thứ 1 hay thứ 2)

Hỏi & Đáp

"Formal education will make you a living;
self-education will make you a fortune"

Bài học kế tiếp

Bài toán sắp xếp

- Giải thuật sắp xếp cơ bản
- Giải thuật sắp xếp cải tiến

Tài liệu tham khảo

➢Tài liệu môn học

- [1] Michael T. Goodrich, Roberto Tamassia, Data Structures & Algorithms in Java (6^{th} Edition)
- [2] Trần Hạnh Nhi, Dương Anh Đức, Cấu trúc dữ liệu & giải thuật, Khoa CNTT, trường ĐH KHTN ĐHQG TpHCM

➢Tài liệu tham khảo thêm

- [3] Thomas H. Cormen et al., 2009, Introduction to Algorithms, 3^{rd} Edition, ebook.
- [4] Hoàng M. L., 2002, Cấu trúc dữ liệu và giải thuật, ĐHSP Hà Nội.