

BUỔI 02: TÌM KIẾM & SẮP XẾP

LÝ THUYẾT

CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

IT003.N210

Giảng viên: Ths Đặng Văn Em

Email: vanem@uit.edu.vn

Điện thoại: 0966661006



TÌM KIẾM & SẮP XẾP



1. Cấu trúc dữ liệu vector.
2. Xác định và phát biểu bài toán tìm kiếm và sắp xếp
3. Các giải thuật tìm kiếm và sắp xếp
4. Ưu điểm và hạn chế của thuật toán tìm kiếm và sắp xếp
5. Triển khai cài đặt C++
6. Bài tập chương 2



1. CẤU TRÚC DỮ LIỆU VECTOR.

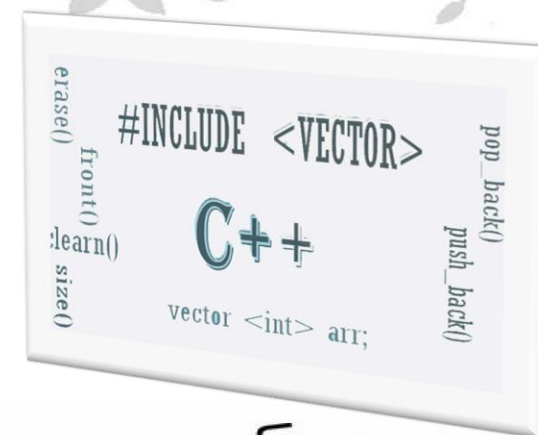
Bài toán minh họa:

Nhập một danh sách số nguyên dương A với số phần tử không biết trước. Thao tác nhập kết thúc khi phần tử nhập vào có giá trị $A_i \leq 0$.

- ❖ In ra màn hình danh sách A, vị trí i của phần tử có giá trị k (k được nhập từ bàn phím).
- ❖ In ra 5 giá trị lớn nhất của A. Nếu không tìm thấy thì đặt i=-1.

Ví dụ: A = {1,2,8,3,7,4,6,10,9,21}, k=4.

- ❖ Kết quả: 1, 2, 8, 3, 7, 4, 6, 10, 9, 21
- ❖ Kết quả: 5 21, 10, 9, 8, 7



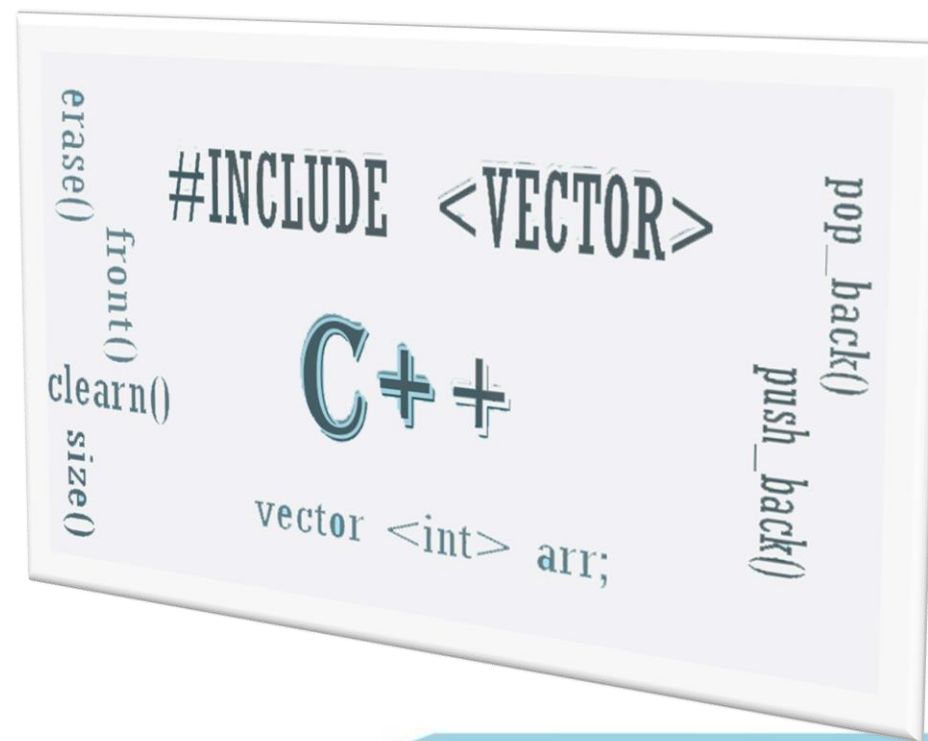
1. CẤU TRÚC DỮ LIỆU VECTOR.

Bài toán minh họa:

Bài toán có thể được giải quyết bằng cách sử dụng thư viện **<vector>** và **<algorithm>** như sau.

```
#include <iostream>
#include <vector>
#include <algorithm>

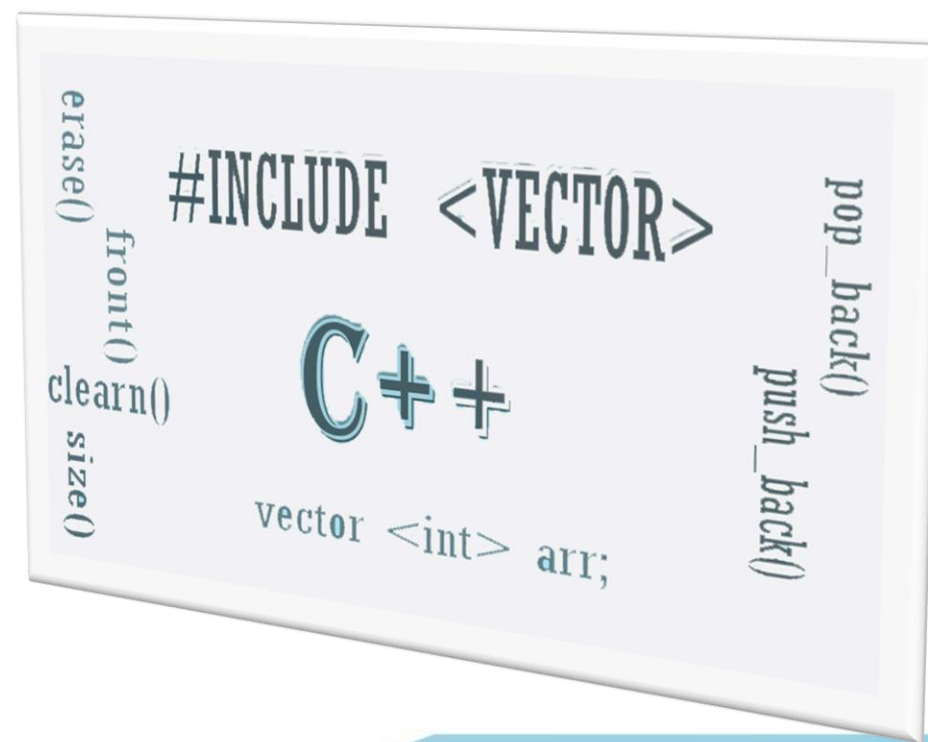
using namespace std;
void NhapDS (vector<int> &);
void InDS (vector<int>);
void Top5 (vector<int>);
int TimK (vector<int>, int);
```



1. CẤU TRÚC DỮ LIỆU VECTOR.

Bài toán minh họa: sử dụng `<vector>` và `<algorithm>`.

```
int main()
{
    vector<int> A;
    int k;
    NhapDS(A);
    cout << "Danh sach da nhap: ";
    InDS(A);
    cout << "Gia tri can tim k = ";
    cin >> k;
    cout << TimK(A, k) << "\t";
    Top5(A);
    return 0;
}
```

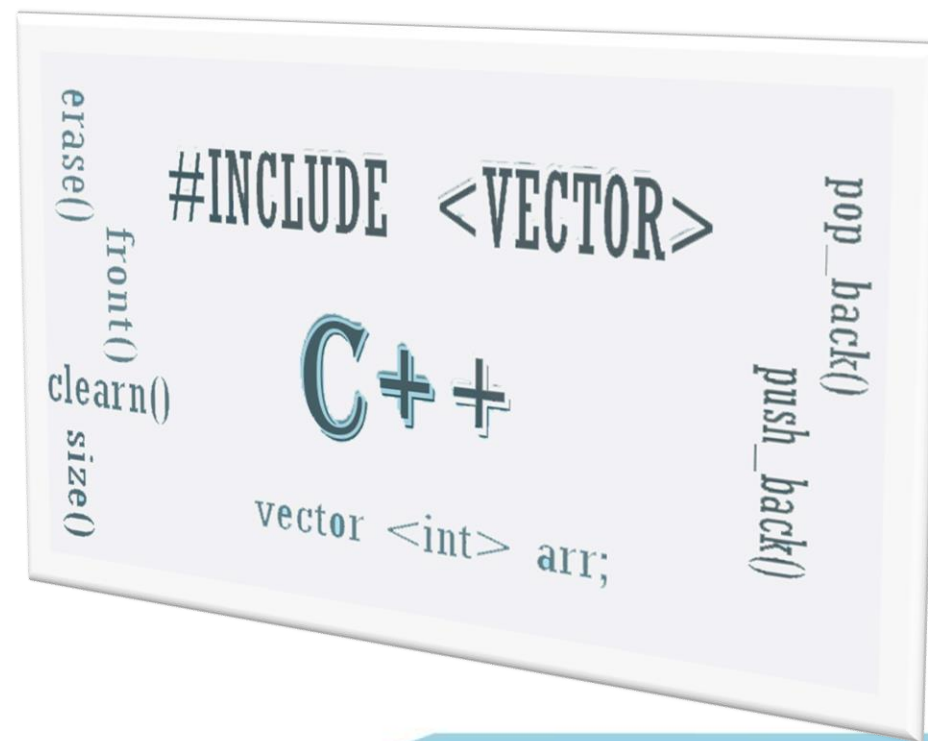


1. CẤU TRÚC DỮ LIỆU VECTOR.

Bài toán minh họa: sử dụng `<vector>` và `<algorithm>`.

```
void NhapDS (vector<int> &v)
{
    int tmp;
    cout << "Nhap danh sach" << endl;
    cin >> tmp;
    while (tmp > 0) {
        v.push_back(tmp);
        cin >> tmp;
    }
}

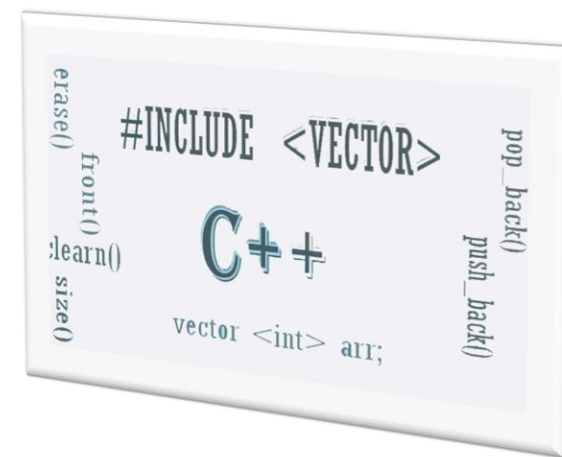
void InDS (vector<int> v)
{
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << ' ';
    cout << endl;
}
```



1. CẤU TRÚC DỮ LIỆU VECTOR.

Bài toán minh họa: sử dụng `<vector>` và `<algorithm>`.

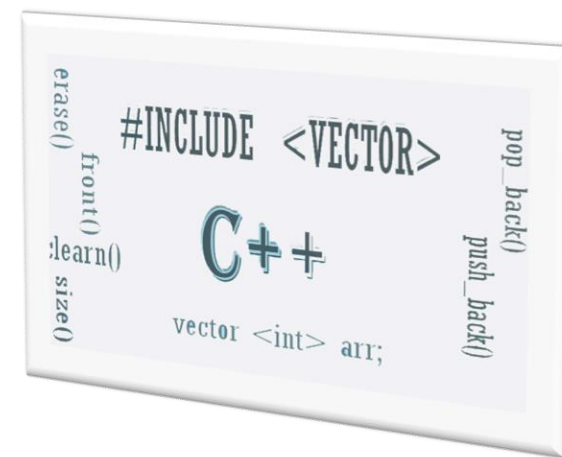
```
void Top5 (vector<int> v)
{
    sort(v.begin(), v.end());
    if (v.size() < 5) {
        cout << "DS khong co du 5 phan tu" << endl;
        return;
    }
    for (vector<int>::iterator i = v.end() - 1;
        i > v.end() - 6; i--)
        cout << *i << ' ';
}
```



1. CẤU TRÚC DỮ LIỆU VECTOR.

Bài toán minh họa: sử dụng `<vector>` và `<algorithm>`.

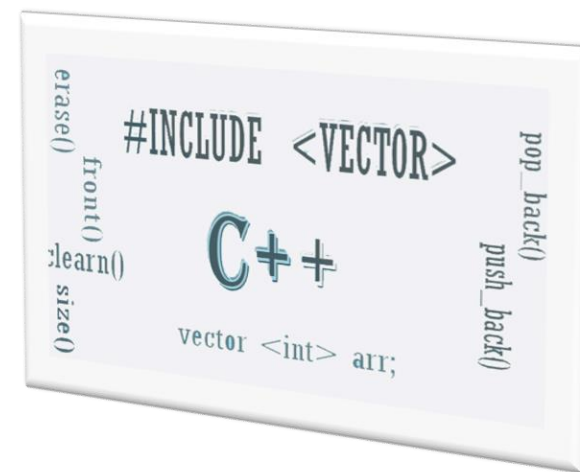
```
int TimK(vector<int> v, int k)
{
    vector<int>::iterator i;
    i = find(v.begin(), v.end(), k);
    if (i != v.end())
        return i - v.begin();
    return -1;
}
```



1. CẤU TRÚC DỮ LIỆU VECTOR.

STL (*Standard Template Library*) là gì?

Standard Template Library viết tắt là **STL**. Đây là một tập hợp rất nhiều hàm thường chủ yếu được dùng để lưu trữ và xử lý dữ liệu. STL là một thư viện các thuật toán và vòng lặp. STL được phát triển là nhằm để tái sử dụng các mã được viết và kiểm nghiệm sẵn nhằm tiết kiệm thời gian, công sức.

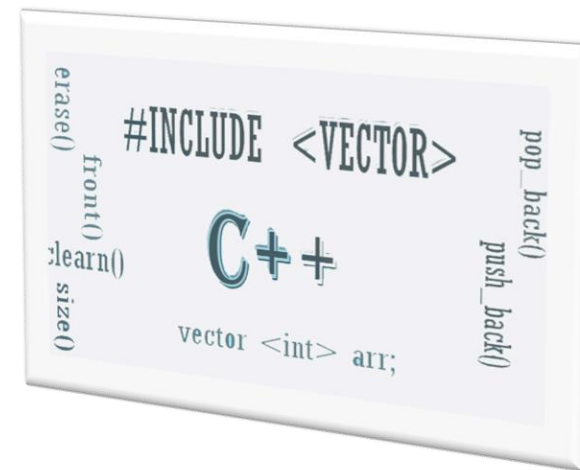


1. CẤU TRÚC DỮ LIỆU VECTOR.

Vector trong C++ là gì?

Vector: Vector C++ là một mảng có khả năng “động”, tức có khả năng **tự động thay đổi kích thước** nếu một phần tử bị xóa hay được chèn vào thêm. Đồng nghĩa với việc vùng chứa sẽ tự xử lý việc lưu trữ. Biến kiểu **vector** được khai báo như sau:

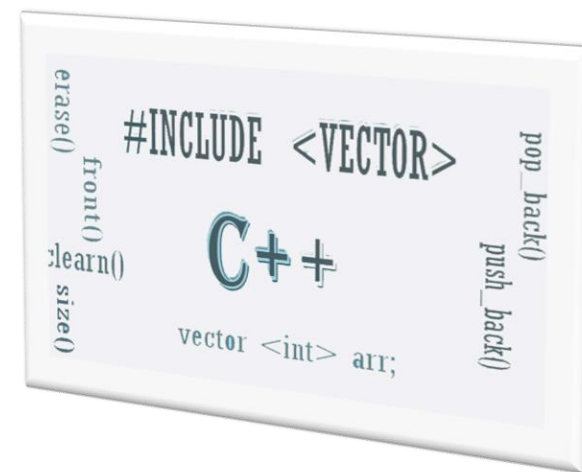
vector<kiểu> tên_biến; *vector <int> array;*



1. CẤU TRÚC DỮ LIỆU VECTOR.

Ưu điểm vector trong C++ là gì?

- ❖ Không cần khai báo kích thước của mảng như int A[100]..., vector có khả năng tự động nâng kích thước lên.
- ❖ Thêm 1 phần tử vào vector đã đầy thì vector sẽ tự động tăng kích thước của nó lên để tạo chỗ chứa cho giá trị mới này.

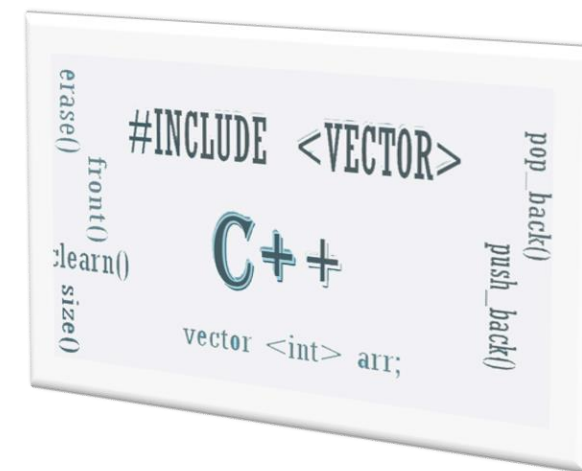


1. CẤU TRÚC DỮ LIỆU VECTOR.

Ưu điểm vector trong C++ là gì?

- ❖ Có khả năng cho bạn biết số lượng các phần tử mà bạn đang lưu trong nó.
- ❖ Trong vector, bạn hoàn toàn có thể dùng số phần tử âm ví dụ A[-6], A[-5], điều này rất tiện trong việc cài đặt các giải thuật khác

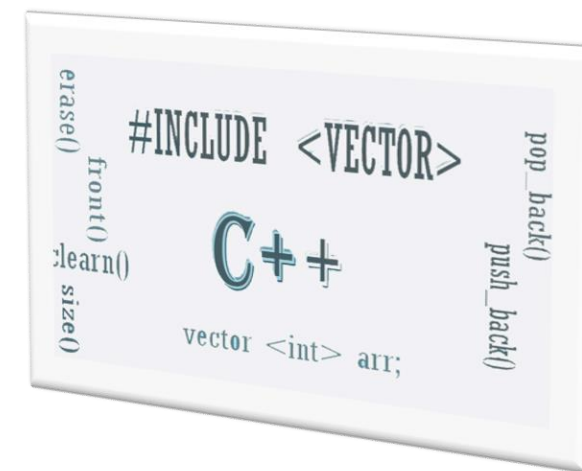
Ví dụ: `vector<kiểu>::iterator`



1. CẤU TRÚC DỮ LIỆU VECTOR.

Vector cung cấp nhiều chức năng hữu ích khác nhau:

- ❖ **Modifiers:**
- ❖ **Iterators**
- ❖ **Capacity**
- ❖ **Element access**



1. CẤU TRÚC DỮ LIỆU VECTOR.



Modifiers:

- ❖ **push_back():** hàm này được sử dụng để đẩy một phần tử trong vector về sau. Nếu kiểu đối tượng được truyền trong tham số không giống với kiểu của vector thì chúng sẽ bị ném bỏ.
- ❖ **assign():** ghi đè các giá trị mới bằng cách thay thế giá trị cũ
- ❖ **pop_back():** sử dụng để làm giảm kích thước của vector xuống 1 phần tử.
- ❖ **insert():** sử dụng để chèn phần tử mới trước vị trí được trả bởi vòng lặp
- ❖ **erase():** sử dụng để xóa các phần tử khỏi vòng lặp.
- ❖ **swap():** dùng để hoán đổi giá trị/ nội dung của Vector có cùng kiểu, không nhất thiết cùng kích thước.
- ❖ **clear():** dùng để loại bỏ các phần tử của vùng chứa Vector

1. CẤU TRÚC DỮ LIỆU VECTOR.



Iterators

- ❖ **begin()** Trả về biến iterator trỏ đến phần tử đầu của vector
- ❖ **end()** Trả về biến iterator trỏ đến vị trí sau phần tử cuối của vector
- ❖ **size()** Trả về số phần tử của vector
- ❖ **push_back()** Thêm một phần tử vào cuối vector

Yêu cầu sinh viên tìm hiểu thêm các phương thức khác trong slide tham khảo thư viện STL

1. CẤU TRÚC DỮ LIỆU VECTOR.



Capacity

- ❖ **size()**: hàm này trả về số lượng phần tử trong vector.
- ❖ **max_size()**: hàm sử dụng để trả về số phần tử tối đa vector có thể giữ được.
- ❖ **capacity()**: hàm được dùng để trả về kích thước không gian lưu trữ của vector được cấp bằng đồ thị số.
- ❖ **resize()**: hàm được sử dụng để chứa các phần tử “n”. Nếu kích thước hiện tại của vector lớn hơn n, các phần tử phía sau n sẽ bị xóa khỏi vector
- ❖ **empty()**: nếu giá trị trả về của hàm là true, vector của bạn đang trống. Nếu giá trị trả về là false, vector của bạn không trống

1. CẤU TRÚC DỮ LIỆU VECTOR.

Ví dụ:

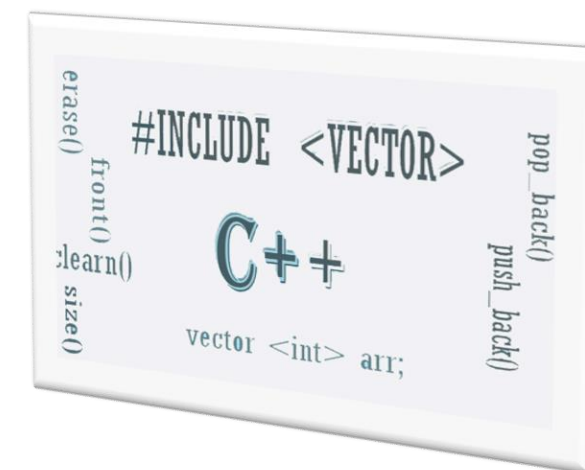
Vectors in C++

Vector is sequence container (same as dynamic array) which resizes itself automatically.

Diagram illustrating the structure of a C++ vector. The vector contains elements 1, 3, 7, 4, 5. The `begin()` method points to the first element (1). The `at(2)` method points to the element at index 2 (7). The `end()` method points to the first empty space after the last element (5). The `size()` method indicates the number of elements currently in the vector (5). The `capacity()` method indicates the total number of elements the vector can hold (8).

Containers

- `<array>`
- `<deque>`
- `<forward_list>`
- `<list>`
- `<map>`
- `<multimap>`
- `<multiset>`
- `<ordered_map>`
- `<ordered_set>`
- `<set>`
- `<stack>`
- `<string_view>`
- `<string>`
- `<vector>`



1. CẤU TRÚC DỮ LIỆU VECTOR.



Tìm kiếm với đối tượng vector?

Các phần tử trong một biến vector có thể được tìm kiếm nhờ hàm `find()` (được định nghĩa trong `<algorithm>`). Hàm `find()` được sử dụng như sau:

```
vector<kiểu> A;
```

```
vector<kiểu>::iterator i;
```

```
int k;
```

```
...
```

```
i = find(A.begin(), A.end(), k);
```

Kết quả của hàm `find()` là một biến kiểu iterator trỏ tới phần tử cần tìm của vector hoặc trả về `end()` nếu không tìm thấy

1. CẤU TRÚC DỮ LIỆU VECTOR.



Sắp xếp với đối tượng vector?

Có thể sắp xếp các phần tử trong một biến kiểu vector bằng hàm `sort()` (được định nghĩa trong `<algorithm>`). Hàm `sort()` được sử dụng như sau:

`vector<kiểu> A;`

Sắp xếp theo thứ tự tăng dần:

`sort(A.begin(), A.end());`

Sắp xếp theo thứ tự giảm dần:

`sort(A.begin(), A.end(), greater<kiểu>());`

Yêu cầu sinh viên tìm hiểu các biến thể của các hàm `find()` và `sort()` trong slide tham khảo STL

2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.

Tìm kiếm & sắp xếp:

- ❖ Nhu cầu tìm kiếm sắp xếp.
- ❖ Các giải thuật tìm kiếm.
- ❖ Các giải thuật sắp xếp.
- ❖ Cấu trúc hàng đợi ưu tiên.
- ❖ Bài tập chương.

WHY



2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.



Nhu cầu tìm kiếm Google:

← → ↻ google.com/search?source=hp&ei=IESdXr3OFpmGyAPymKroCA&q=cấu+trúc+dữ+liệu+và+giải+thuật&oq=Cấu+trúc+dữ&gs_lcp=CgZwc3ktYWIQAR... ☆ | 📄 | 📁 | 👤

Google X | 🗣️ 🔍 | 📁 | 👤

[All](#) [Images](#) [Videos](#) [News](#) [More](#) Settings Tools


About 6,390,000 results (0.42 seconds)

cuuduongthancong.com > cau-truc-d... ▾ [Translate this page](#)
[PDF]Cấu Trúc Dữ Liệu Và Giải Thuật - giáo trình, bài giảng ...
Sách tham khảo: [1] "Data Structures and Algorithms in C++", A. Drozdek, Thomson Learning Inc., 2005. [2] "C/C++: How to Program", 7th Ed. – Paul Deitel and ...

vietjack.com > cau-truc-du-lieu-va-gi... ▾ [Translate this page](#)
Cấu trúc dữ liệu và giải thuật (Data Structure and Algorithms)
Cấu trúc dữ liệu và giải thuật (Data Structure and Algorithms) - Học Cấu trúc dữ liệu & giải thuật với ngôn ngữ C, C++ và Java theo các bước cơ bản tới nâng ...
[Cấu trúc dữ liệu là gì](#) · [Giải thuật là gì](#) · [Giải thuật tiệm cận](#) · [Cấu trúc dữ liệu cây](#)

nguyenvanhieu.vn > cau-truc-du-lieu ▾ [Translate this page](#)
Cấu trúc dữ liệu - Lập Trình Không Khó
Các cấu trúc dữ liệu và các giải thuật được xem như là 2 yếu tố quan trọng nhất trong lập

See cấu trúc d... Sponsored ⓘ

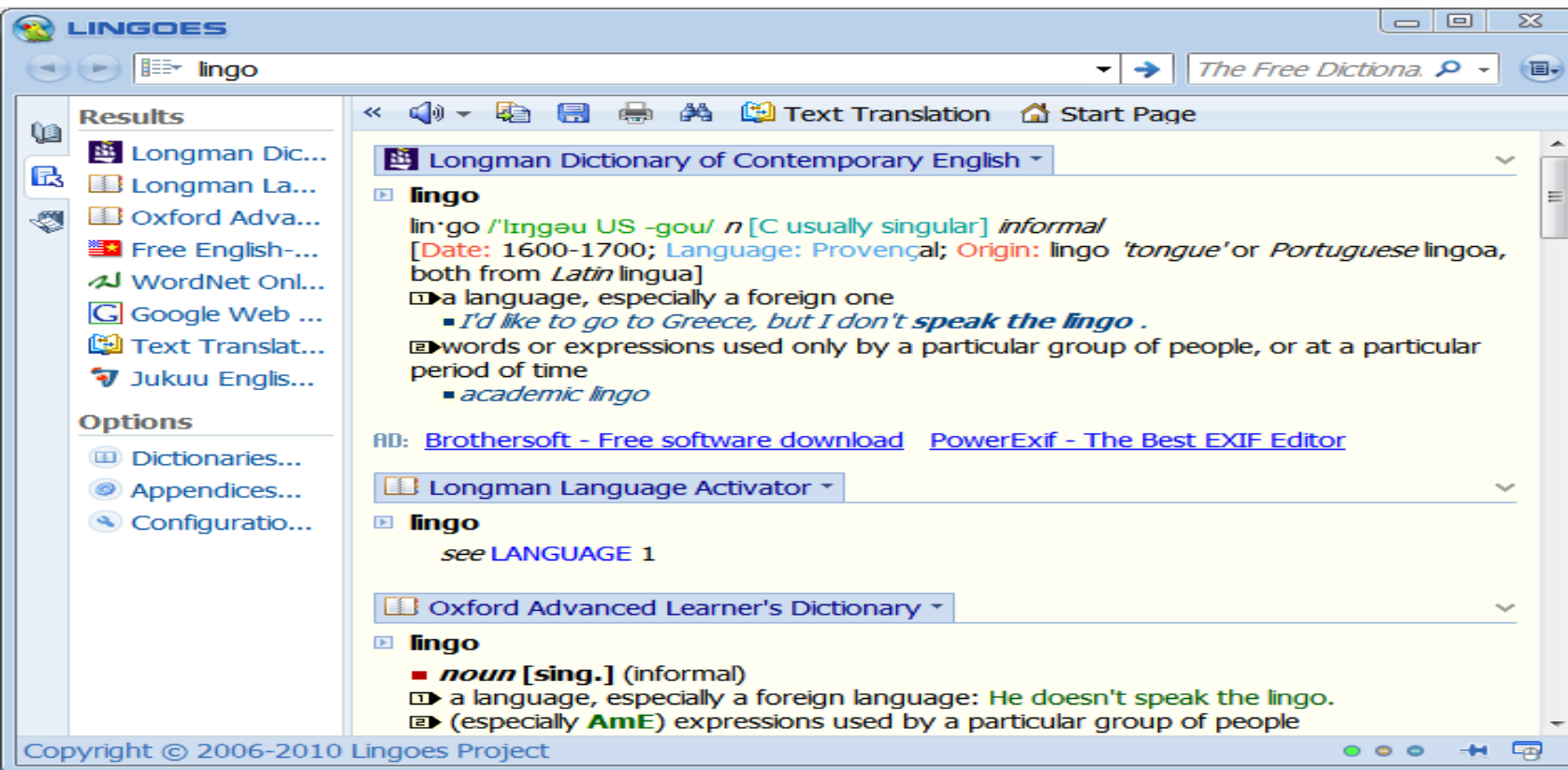
 **Cấu Trúc Dữ Liệu Và Thuật Toán Phân Tích Và Cài Đặt Trên CC ...**
₫55,000
tiki.vn

→ More on Google



2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.

Tìm kiếm Từ điển:




2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.



Truy vấn dữ liệu:



ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
THƯ VIỆN



TRANG CHỦ | GIỚI THIỆU | DỊCH VỤ | TRA CỨU OPAC | TÀI NGUYÊN | LIÊN HỆ

Tra cứu thư mục trực tuyến

Bắt kỳ ▾ mạng xã hội

Q

⋮

Tên thư viện	Số biểu ghi	Trạng thái
» Thư viện Trường Đại học Công nghệ Thông tin ĐHQG-HCM	28	Kết nối thành công

#	Mô tả	Tác giả chính	Chọn
1	Phân tích mạng xã hội và ứng dụng : Giáo trình / Đỗ Phúc .- HCM : Đại học Quốc gia TP.HCM , 2017 005.7 Đ 450 PH	Đỗ, Phúc	<input type="checkbox"/>
2	Mạng xã hội địa điểm trên Android Nguyễn Thành Luân, , TS. Nguyễn Anh Tuấn , , 2012	Nguyễn Thành Luân	<input type="checkbox"/>

SÁCH THEO KHO

Kho đọc

959

Kho mượn - Giáo trình

66

Kho mượn - Tham Khảo

88

Khóa luận

4

Luận văn

12

Ưu tiên

5

MƯỢN NHIỀU

OUTCOMES Outcomes Pre-



2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.



Soạn thảo và Tra cứu văn bản:

The screenshot shows the Wikipedia page for 'Data structure'. The page title is 'Data structure' and it is part of the 'Wikipedia:Administration § Data structure and development' section. The page content includes a warning box stating: 'This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. Find sources: "Data structure" – news · newspapers · books · scholar · JSTOR (January 2017) (Learn how and when to remove this template message)'. Below the warning box, the text reads: 'In computer science, a **data structure** is a data organization, management and storage format that enables **efficient** access and modification. [1][2][3] More precisely, a **data**'.

keys	hash function	buckets
John Smith		00
		01 521-8976



2. XÁC ĐỊNH VÀ PHÁT BIỂU BÀI TOÁN TÌM KIẾM VÀ SẮP XẾP.



KẾT XUẤT DỮ LIỆU:

- ❖ Sắp xếp các mục từ cho từ điển.
- ❖ Sắp xếp danh sách trong các báo cáo tổng hợp.
- ❖ Tổ chức dữ liệu của một đối tượng cho phù hợp.
- ❖ Sắp xếp để thiết lập thứ tự cho danh sách.

LÀM TĂNG HIỆU QUẢ TÌM KIẾM



3. CÁC GIẢI THUẬT TÌM KIẾM:

Phát biểu bài toán tìm kiếm:

- ❖ Cho danh sách $A[]$ có n phần tử $a_0, a_1, a_2, \dots, a_{n-1}$.
- ❖ Để đơn giản trong việc trình bày giải thuật ta dùng mảng một chiều $A[]$ để lưu danh sách các phần tử nói trên trong bộ nhớ chính.
- ❖ Tìm phần tử có giá trị khóa là X trong A . Nếu $A[i]$ có giá trị khóa là X thì trả về chỉ số i .



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính(Linear search):

- ❖ Từ khóa: **Linear search**
- ❖ Điều kiện: Danh sách $A = \{a_0, a_1, \dots, a_{n-1}\}$ chưa có thứ tự
- ❖ Phân tích: Không có thông tin nào ngoài thông tin có được khi so sánh **X** với giá trị khóa của **A[i]**.
- ❖ Ý tưởng: So sánh **X** lần lượt với phần tử thứ 0, thứ 1, thứ 2...của mảng A cho đến khi gặp được khóa **X** cần tìm, hoặc tìm hết mảng mà không thấy.

Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]



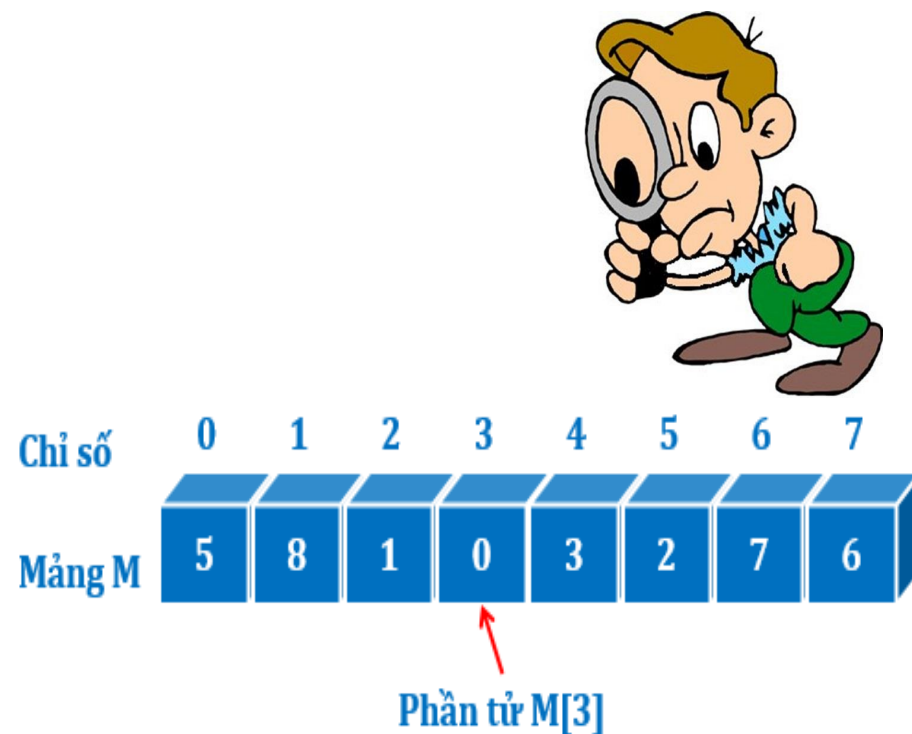
3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính (Linear search):

❖ Thuận toán:

Đầu vào: Danh sách **A** có **n** phần tử, giá trị khóa **X** cần tìm.

Đầu ra: Chỉ số **i** của phần tử **A[i]** trong **A** có giá trị khóa là **X**. Trong trường hợp không tìm thấy **i=-1**



3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính(Linear search):

❖ Các bước thực hiện:

Bước 1: Khởi gán **$i=0$** ;

Bước 2: So sánh $a[i]$ với giá trị x cần tìm, có 2 khả năng

+ **$a[i] == x$** tìm thấy x . **Dừng**;

+ **$a[i] != x$** sang bước 3;

Bước 3: $i=i+1$ // Xét tiếp phần tử kế tiếp trong mảng

Nếu $i==N$: Hết mảng. **Dừng**

Ngược lại: Lặp lại bước 2



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

↑
Phần tử $M[3]$

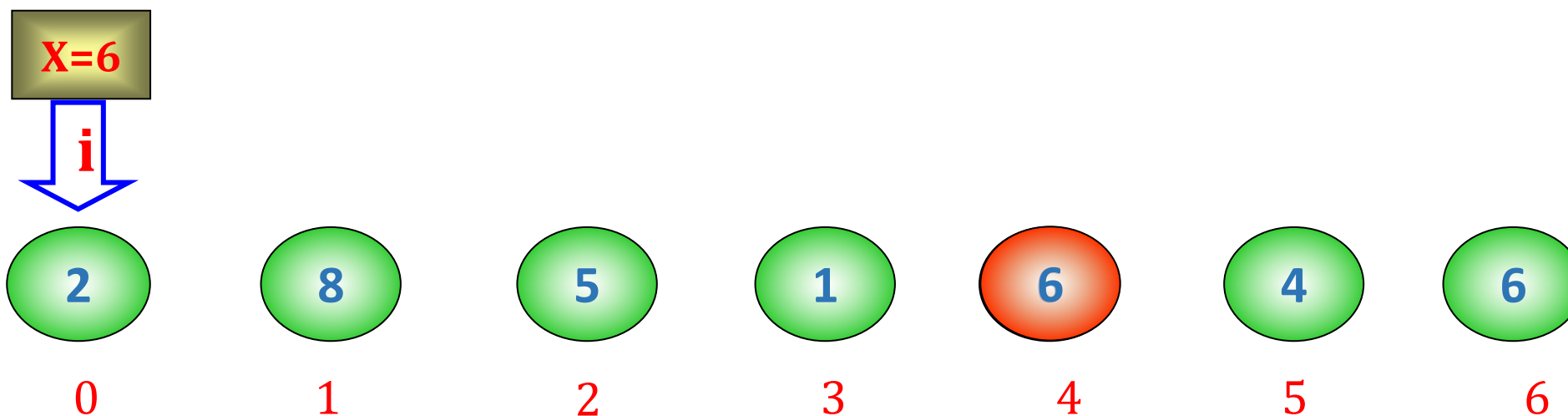
3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính (Linear search):

❖ Minh họa thuật toán:

Tìm thấy 6 tại vị trí 4



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính (Linear search):

❖ Minh họa thuật toán:

X=10



2

0

8

1

5

2

1

3

6

4

4

5

6

6

i=6, không tìm thấy



3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính(Linear search):

Cài đặt trên mảng

```
int LinearSearch(int a[],int n, int x)
{
    int i=0;
    while((i<n)&&(a[i]!=x))
        i++;
    if(i==n)
        return 0; //Tìm không thấy x
    else
        return 1; //Tìm thấy
}
```

Cài đặt trên danh sách đơn

```
Node* linearSearch(List A, int x)
{
    Node *p = A.pHead;
    while (!p)
    {
        if (p->info == x) return p;
        p = p->pNext;
    }
    return NULL;
}
```



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

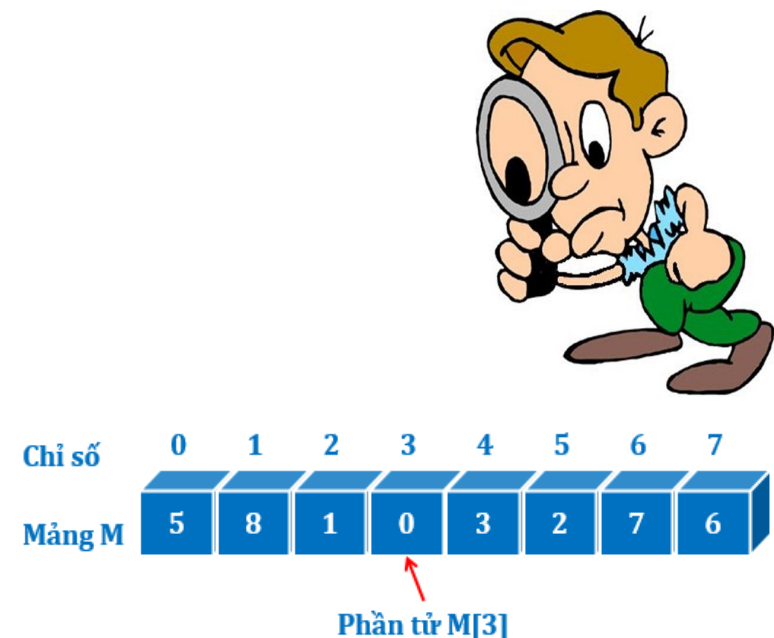
Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính (Linear search):

Đánh giá:

- ❖ Trường hợp tốt nhất (best case): A[0] chứa khóa X số lần lặp là 1, độ phức tạp hằng số **$O(1)$** .
- ❖ Trường hợp xấu nhất (worst case): A[] không có phần tử có khóa X, số lần lặp là n độ phức tạp tuyến tính **$O(n)$** .
- ❖ Trường hợp trung bình (average case): độ phức tạp tuyến tính **$O(n)$** .



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính(Linear search) cải tiến:

- ❖ **Phân tích:** Theo thuật toán tìm tuyến tính:
 - ✓ Cần phải kiểm tra điều kiện dừng khi xét hết danh sách ($i < n$).
 - ✓ Cần phải kiểm tra điều kiện dừng khi tìm thấy phần tử ai trong vòng lặp
- ❖ **Rút gọn điều kiện dừng.**



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính (Linear search) cải tiến:

❖ Ý tưởng:

- ✓ Thêm phần tử a_n có khóa **X** vào **A**, khi này **A** có **$n+1$** phần tử.

Phần tử thêm vào được gọi là phần tử cầm canh.

- ✓ Chỉ cần điều kiện dừng là tìm thấy phần tử a_i có khóa **X**



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính (Linear search) cải tiến:

❖ Thuật toán:

Đầu vào: Danh sách **A** có n phần tử, giá trị khóa x cần tìm.

Đầu ra: Chỉ số **i** của phần tử a_i trong **A** có giá trị khóa là **X**.

Trong trường hợp không tìm thấy **$i = -1$** .



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm tuyến tính(Linear search) cải tiến:

❖ Thuật toán:

Bước 1: Khởi gán **$i=0$** ; **$A[n]=X$** .

Bước 2: Trong khi ($a[i] \neq X$)

$i=i+1$

Bước 3: Nếu $i < N$. **Return i**

Ngược lại hết mảng. **Return -1**



Chỉ số	0	1	2	3	4	5	6	7
Mảng M	5	8	1	0	3	2	7	6

Phần tử M[3]

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm tuyến tính(Linear search) cải tiến:

Cài đặt trên mảng

```
int linearSearchA(int A[],int n,int x)
{
    int i = 0, A[n] = x;
    while (A[i] != x)
        i++;
    if (i < n) return i;
    else return -1;
}
```

Cài đặt trên danh sách đơn

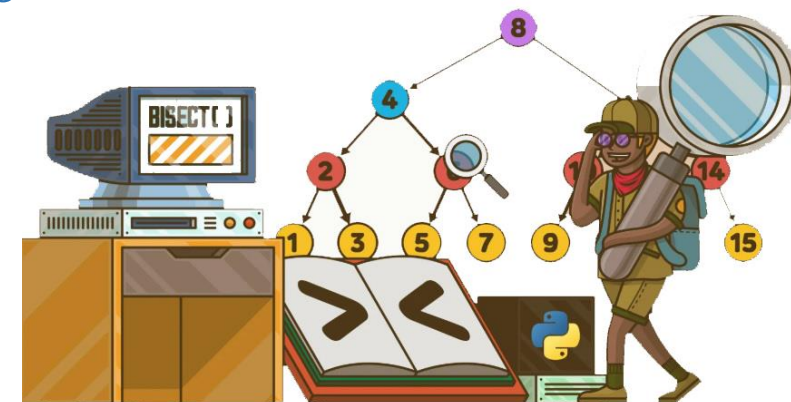
```
Node* linearSearchA(List A, int x)
{
    Node *p = A.pHead, *t = new Node(x);
    if (!t) throw "out of memory";
    addTail(A, t);
    while (p->info != x)
        p = p->pNext;
    if (p == A.pTail) return p;
    else return NULL;
}
```



3. CÁC GIẢI THUẬT TÌM KIẾM:

Tìm kiếm nhị phân(Binary search):

- ❖ Từ khóa: **Binary search**
- ❖ Điều kiện: Danh sách $A = \{a_0, a_1, \dots, a_{n-1}\}$ đã có thứ tự \mathcal{R} .
- ❖ Phân tích: Khi so sánh $A[i]$ với khóa X , dựa vào quan hệ thứ tự, có thể quyết định nên xét phần tử kế tiếp ở phần trước (hoặc phần sau) của $A[i]$ hay không.



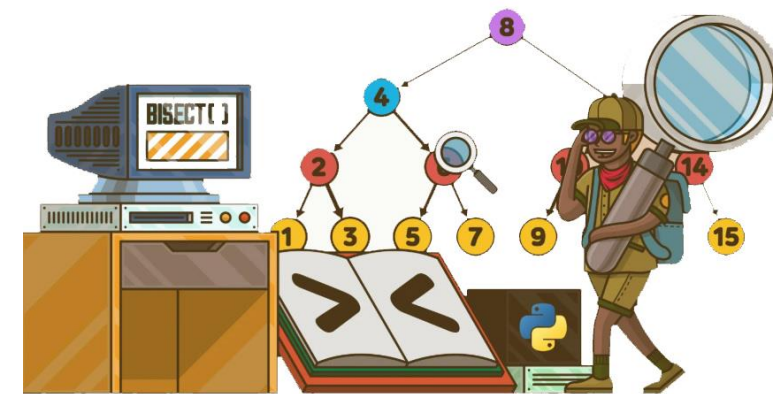
3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

❖ Ý tưởng:

- ✓ Giả sử ta xét **mảng có thứ tự tăng**, khi ấy ta có $a_{i-1} < a_i < a_{i+1}$
- ✓ Nếu $X > a_i$ thì X chỉ có thể xuất hiện trong đoạn $[a_{i+1}, a_{n-1}]$
- ✓ Nếu $X < a_i$ thì X chỉ có thể xuất hiện trong đoạn $[a_0, a_{i-1}]$
- ✓ Ý tưởng của giải thuật là tại mỗi bước ta so sánh X với phần tử **đứng giữa** trong dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này mà ta quyết định giới hạn dãy tìm kiếm ở nửa dưới hay nửa trên của dãy tìm kiếm hiện hành.



3. CÁC GIẢI THUẬT TÌM KIẾM:



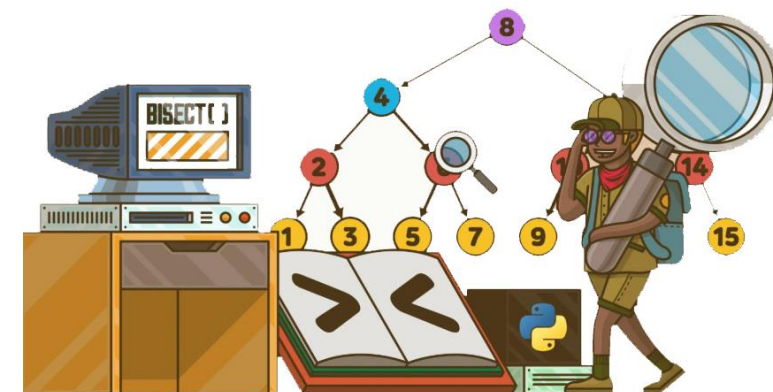
Tìm kiếm nhị phân(Binary search):

❖ Thuật toán:

Đầu vào: Danh sách **A** có **n** phần tử đã có thứ tự **ℳ**, giá trị khóa **X** cần tìm.

Đầu ra: Chỉ số **i** của phần tử **a_i** trong **A** có giá trị khóa là **X**.

Trong trường hợp không tìm thấy **i=-1**



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

Thuật toán: Giả sử dãy tìm kiếm hiện hành bao gồm các phần tử nằm trong a_{left} , a_{right} , các bước của giải thuật như sau:

❖ **Bước 1:** $\text{left}=0$; $\text{right}=\text{N}-1$;

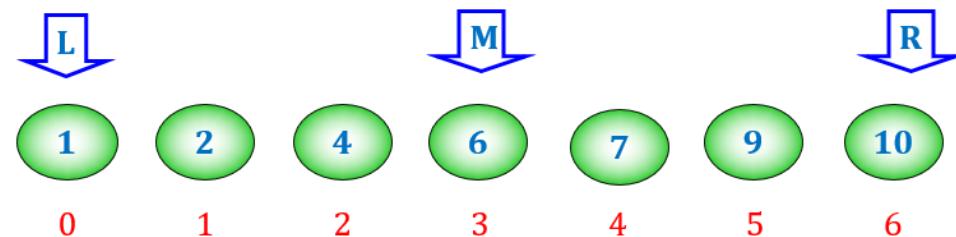
❖ **Bước 2:** $\text{mid}=(\text{left}+\text{right})/2$; //chỉ số phần tử giữa dãy hiện hành

➤ So sánh $a[\text{mid}]$ với x . Có 3 khả năng

- ✓ $a[\text{mid}]=X$: tìm thấy. **Dừng**
- ✓ $a[\text{mid}]>X$: $\text{Right}=\text{mid}-1$;
- ✓ $a[\text{mid}]<X$: $\text{Left}=\text{mid}+1$;

❖ **Bước 3:** Nếu $\text{Left} \leq \text{Right}$; // còn phần tử trong dãy hiện hành

Lặp lại **bước 2**. Ngược lại : **Dừng**

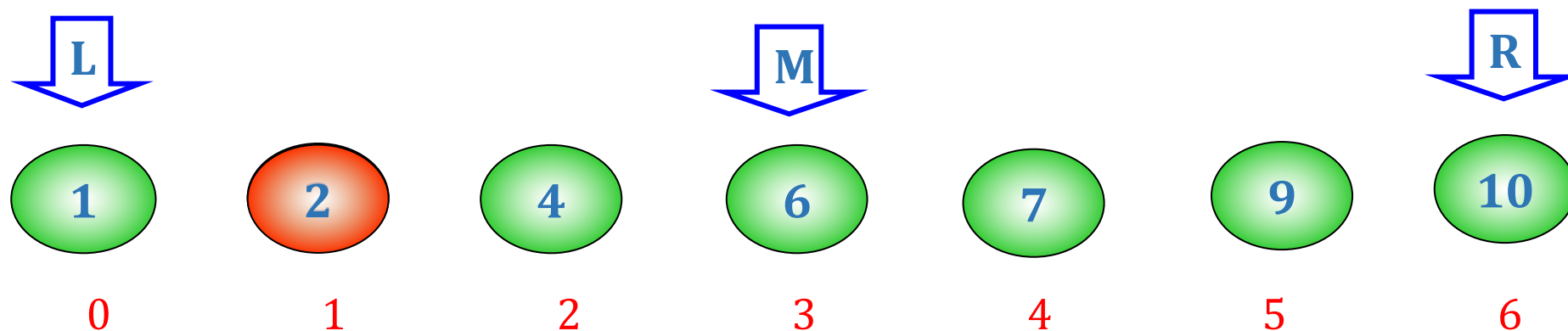


3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

❖ Minh họa thuật toán tìm **X=2**



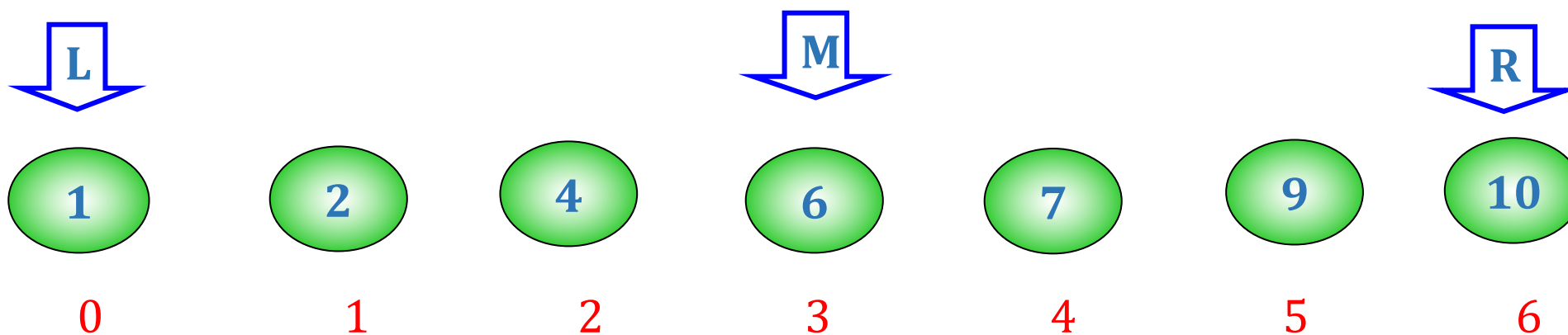
Tìm thấy 2 tại vị trí 1

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

❖ Minh họa thuật toán tìm **X=-1**



L=0

R=-1 => không tìm thấy X=-1

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

Cài đặt trên mảng

```
int BinarySearch(int a[],int n,int x)
{
    int left, right, mid, left=0, right=n-1;
    do
    {
        mid=(left+right)/2;
        if(a[mid]==x) return 1;
        else
        {
            if(a[mid]<x) left=mid+1;
            else right=mid-1;
        }
    }while(left<=right);
    return 0;
}
```

Cài đặt trên danh sách liên kết

Cài đặt trên cấu trúc liên kết khác là
Cây nhị phân tìm kiếm



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân(Binary search):

❖ Thực hiện từng bước tìm kiếm nhị phân trên mảng $A[]$ với $X=37$ như sau:

$A[] = 1 \ 3 \ 5 \ 7 \ 11 \ 13 \ 17 \ 19 \ 23 \ 29 \ 31 \ 37 \ 41 \ 43 \ 47 \ 53 \ 59$

Bước 1: $left = 0, right = 16$

Bước 2: $mid = (left + right) / 2 = 8 \Rightarrow A[mid] = 23 < X; \Rightarrow left = mid + 1 = 9.$

Bước 3: $left = 9, right = 16$. Quay lên bước 2 để tìm mid mới

.....

❖ Thực hiện từng bước tìm $X=3; X=13; X=53.$



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân:
(Binary search)

Binary search

steps: 0



Tìm kiếm tuyến tính:
(Linear search)

Sequential search

steps: 0



X=37

www.mathwarehouse.com

3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nhị phân (Binary search):

Đánh giá:

- ❖ Trường hợp tốt nhất: phần tử cần tìm ở đúng vị trí $(l+r) \div 2$ số lần lặp là **1** độ phức tạp hằng số **$O(1)$** .
- ❖ Trường hợp xấu nhất: số lần tìm là số lần chia đôi dãy đến khi dãy tìm kiếm còn 1 phần tử, số lần lặp khoảng **$\log_2(n)+1$** độ phức tạp logarith **$O(\log(n))$** .
- ❖ Trường hợp trung bình: độ phức tạp **$O(\log(n))$** .



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

- ❖ Từ khóa: **Interpolation Search**
- ❖ Điều kiện: Danh sách $A = \{a_0, a_1, \dots, a_{n-1}\}$ đã có thứ tự \mathcal{R} và giá trị khóa được rải đều trên danh sách.
- ❖ Phân tích: Giá trị khóa rải đều trên danh sách vị trí a_m chia danh sách tìm kiếm tương ứng với tỉ lệ giá trị X trong miền giá trị khóa của danh sách tìm kiếm.



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

❖ Ý tưởng:

- ✓ Thay vì xác định **$mid = (left + right) / 2$** như trong tìm kiếm nhị phân, xác định nội suy như sau:

$$mid = left + \frac{(right - left) * (X - A[left])}{(A[right] - A[left])}$$

- ✓ Các bước còn lại tương tự như tìm kiếm nhị phân.



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

❖ Thuật toán:

Đầu vào: Danh sách **A** có **n** phần tử **đã có thứ tự**, giá trị khóa **X** cần tìm.

Đầu ra: Chỉ số **i** của phần tử **a_i** trong **A** có giá trị khóa là **X**.

Trong trường hợp không tìm thấy **i=-1**



3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

Thuật toán:

- ❖ Bước 1: $\text{left}=0$; $\text{right}=\text{N}-1$;
- ❖ Bước 2: $\text{mid}=\text{left}+((\text{right}-\text{left})*(X-A[\text{left}]))/(A[\text{right}]-A[\text{left}])$;
 - So sánh $a[\text{mid}]$ với x . Có 3 khả năng
 - ✓ $a[\text{mid}]=X$: tìm thấy. **Dừng**
 - ✓ $a[\text{mid}]>X$: $\text{Right}=\text{mid}-1$;
 - ✓ $a[\text{mid}]<X$: $\text{Left}=\text{mid}+1$;
- ❖ Bước 3: Nếu $\text{Left} \leq \text{Right}$; // còn phần tử trong dãy hiện hành
Lặp lại bước 2. Ngược lại : **Dừng**

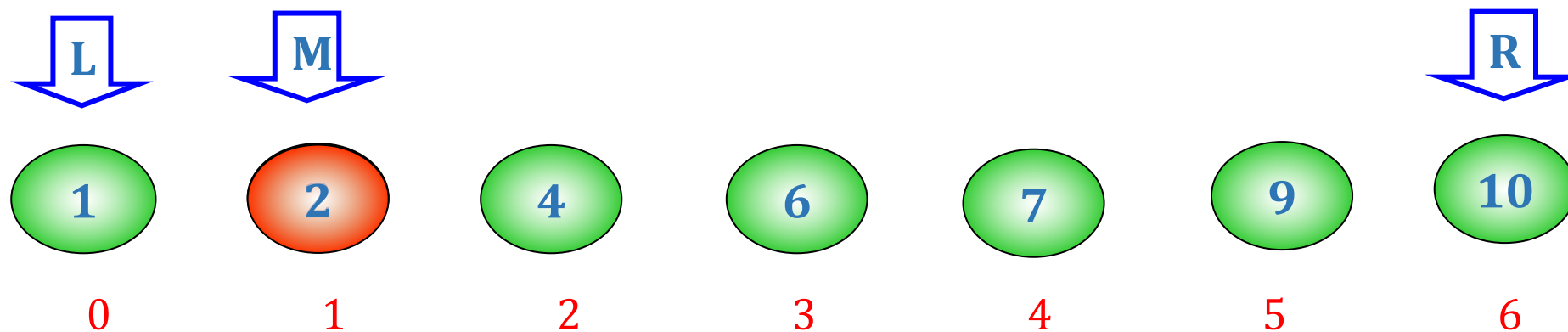


3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

❖ Minh họa thuật toán tìm **X=2**:



Tìm thấy 2 tại vị trí 1

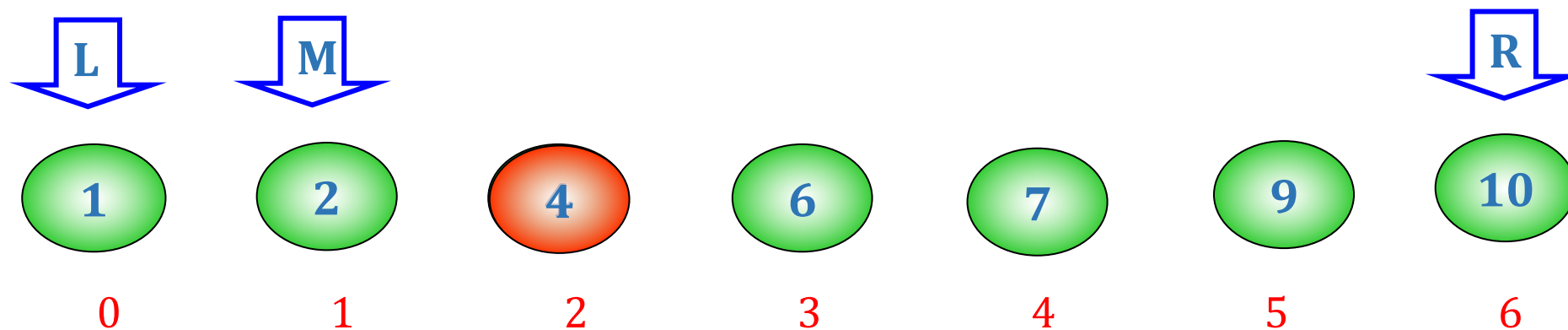


3. CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

❖ Minh họa thuật toán tìm $X=4$:



Tìm thấy 4 tại vị trí 2



CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

Cài đặt trên mảng:

```
int BinarySearch(int a[],int n,int x)
{
    int left, right, mid, left=0, right=n-1;
    while(left<=right
    {
        mid= left+(right-left)*(x-A[left])/(A[right]-A[lefy]);
        if(a[mid]==x) return mid;
        else
            if(a[mid]<x) left=mid+1;
            else right=mid-1;
    }
    return 0;
}
```



CÁC GIẢI THUẬT TÌM KIẾM:



Tìm kiếm nội suy (Interpolation Search):

❖ Thực hiện từng bước tìm kiếm nhị phân trên mảng $A[]$ với $X=37$ như sau:

$A[] = 1 \ 3 \ 5 \ 7 \ 11 \ 13 \ 17 \ 19 \ 23 \ 29 \ 31 \ 37 \ 41 \ 43 \ 47 \ 53 \ 59$

Bước 1: $left = 0, right = 16$

Bước 2: $mid = left + ((right - left) * (X - A[left])) / (A[right] - A[left]) = 10 \Rightarrow A[mid] = 31 < X;$
 $\Rightarrow left = mid + 1 = 11.$

Bước 3: $left = 11, right = 16$. Quay lên bước 2 để tìm mid mới

.....

❖ Thực hiện từng bước tìm $X=11; X=37; X=17; X=53$



BÀI NGHIÊN CỨU CÁC THUẬT TOÁN SẮP XẾP

- 1) Định nghĩa bài toán sắp xếp ?
- 2) Selection sort
- 3) Insertion sort
- 4) Counting sort
- 5) Radix sort

