

# Tìm kiếm và sắp xếp

Duc-Minh Vu @ Phenikaa - ORLab

# Nội dung

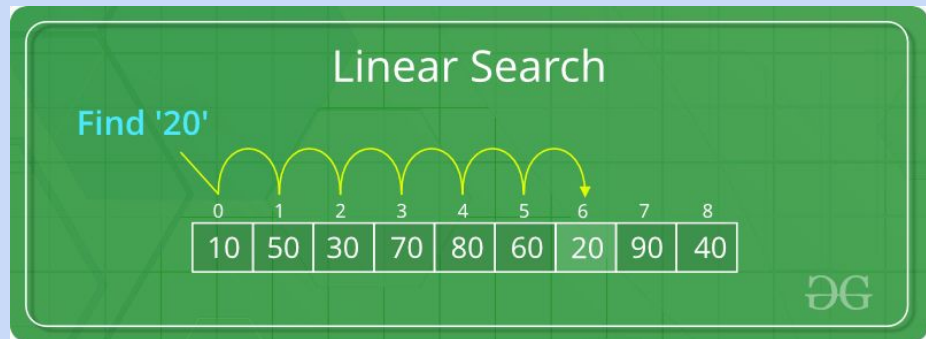
- Thuật toán tìm kiếm tuyến tính.
- Thuật toán tìm kiếm nhị phân.
- Một số hàm trong C/C++ hỗ trợ tìm kiếm tuyến tính/tìm kiếm nhị phân.
-

# Bài toán tìm kiếm

- Cho một danh sách  $n$  phần tử, tìm phần tử thỏa mãn điều kiện nào đó.
  - Có giá trị bằng giá trị  $X$  cho trước.
  - Có giá trị lớn hơn  $X$  cho trước
  - ...
  - $\Rightarrow$  thỏa mãn một điều kiện logic nào đó, gọi là  $f(X)$
- Tùy vào cách thức lưu trữ thông tin, có các phương thức tìm kiếm khác nhau
  - Không thứ tự: tìm kiếm tuyến tính
  - Có thứ tự: tìm kiếm nhị phân, tam phân, nội suy, etc.
- Nếu miền giá trị bé, có thể tìm kiếm trong  $O(1)$

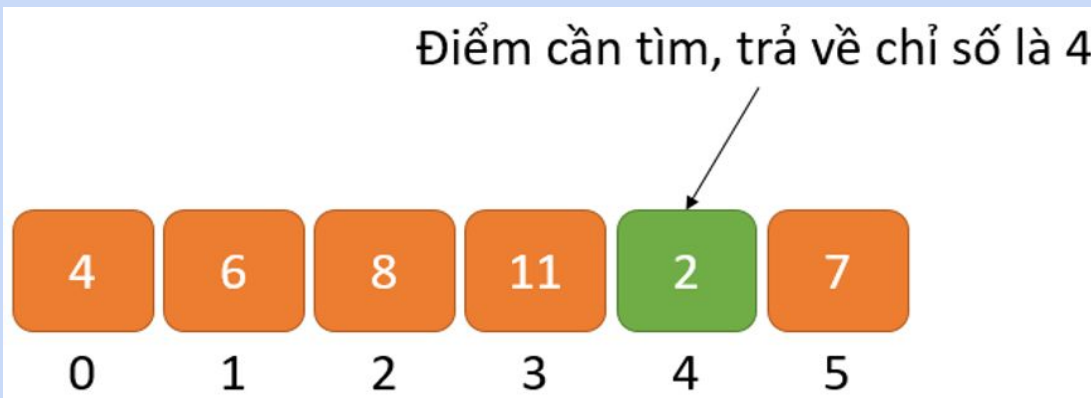
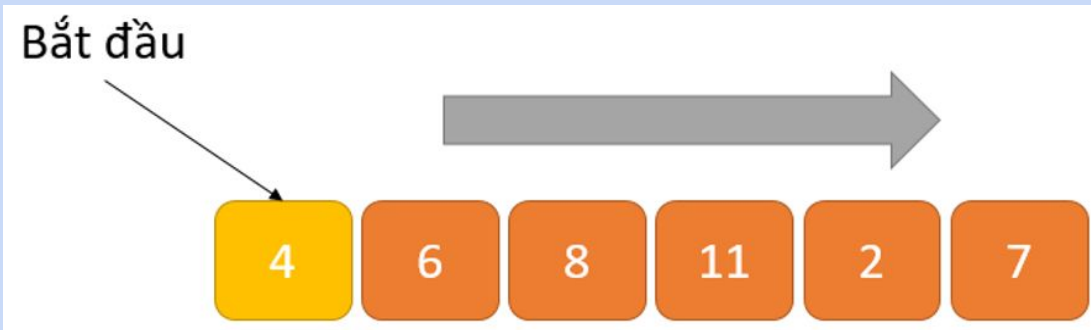
# Tìm kiếm với độ phức tạp tuyến tính

- [Linear Search - GeeksforGeeks](#)
- [Thuật toán tìm kiếm tuyến tính \(tek4.vn\)](#)
- Áp dụng với các danh sách tuyến tính: mảng, danh sách liên kết, etc.
- Duyệt từ đầu đến cuối danh sách và kiểm tra biểu thức logic.
- Độ phức tạp:  $O(n \cdot g(M))$  với  $n$  là kích thước danh sách và  $M$  là thời gian lớn nhất cho việc kiểm tra điều kiện.
  - Thông thường  $O(g(M))=1$



```
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

$O(n)$



Tìm kiếm vị trí số 2 trong danh sách sau

Điều kiện logic: bằng số 2.  
Trả về: vị trí của số 2 tìm thấy

## Functions in <algorithm>

### Non-modifying sequence operations:

<b>all_of</b> <small>C++11</small>	Test condition on all elements in range (function template )
<b>any_of</b> <small>C++11</small>	Test if any element in range fulfills condition (function template )
<b>none_of</b> <small>C++11</small>	Test if no elements fulfill condition (function template )
<b>for_each</b>	Apply function to range (function template )
<b>find</b>	Find value in range (function template )
<b>find_if</b>	Find element in range (function template )
<b>find_if_not</b> <small>C++11</small>	Find element in range (negative condition) (function template )
<b>find_end</b>	Find last subsequence in range (function template )
<b>find_first_of</b>	Find element from set in range (function template )
<b>adjacent_find</b>	Find equal adjacent elements in range (function template )
<b>count</b>	Count appearances of value in range (function template )
<b>count_if</b>	Return number of elements in range satisfying condition (function template )
<b>mismatch</b>	Return first position where two ranges differ (function template )
<b>equal</b>	Test whether the elements in two ranges are equal (function template )
<b>is_permutation</b> <small>C++11</small>	Test whether range is permutation of another (function template )
<b>search</b>	Search range for subsequence (function template )
<b>search_n</b>	Search range for elements (function template )

[<algorithm>  
- C++  
Reference  
\(cplusplus.com\)](http://cplusplus.com)

Các hàm này đều có thể tự triển khai “dễ dàng”.

Một số hàm sử dụng tìm kiếm/duyệt tuần tự trong C++

# Các phiên bản của tìm kiếm trên STL C++

- [find - C++ Reference \(cplusplus.com\)](#) : tìm kiếm phần tử đầu tiên trong một danh sách cho trước (với điều kiện bằng)
- [find\\_if - C++ Reference \(cplusplus.com\)](#) : trả về vị trí phần tử đầu tiên thỏa mãn một điều kiện cho trước tự định nghĩa.
- [find\\_if\\_not - C++ Reference \(cplusplus.com\)](#) : tìm kiếm vị trí phần tử đầu tiên không thỏa mãn một điều kiện cho trước tự định nghĩa/
- [find\\_first\\_of - C++ Reference \(cplusplus.com\)](#): tìm kiếm vị trí đầu tiên của một phần tử trong tập A mà nó xuất hiện trong tập B.
- ...

## ✓ Tìm các số chia hết cho 3

Cho một số tự nhiên  $N$ . Hãy đếm xem từ 1 đến  $N$  có bao nhiêu số chia hết cho 3.

### Dữ liệu vào:

- Một số tự nhiên  $N$

### Kết quả:

- Một số là kết quả của bài toán

### Ví dụ:

#### Input

10

#### Output

3

- a) Viết thuật toán có độ phức tạp  $O(n)$
- b) Viết thuật toán có độ phức tạp  $O(1)$

[Tìm các số chia hết cho 3 - LQDOJ: Le Quy Don Online Judge](#)



## ✓ Tìm số hạng thứ n

Cho dãy số 2, 5, 8, ...

Em hãy tìm số hạng thứ n với n được nhập từ bàn phím

### Dữ liệu vào:

- Một dòng gồm một số nguyên dương n ( $0 < n \leq 10^9$ )

### Kết quả:

- Một dòng số hạng thứ n

### Ví dụ:

**Input**

4

**Output**

11

- a) Viết thuật toán có độ phức tạp  $O(n)$
- b) Viết thuật toán có độ phức tạp  $O(1)$

[Tìm số hạng thứ n - LQDOJ: Le Quy Don Online Judge](#)

## Chia dãy (THT'15)

Có một dãy các số nguyên  $a_1, a_2, \dots, a_n$ . Ta chia dãy số này thành 2 dãy con như sau:

- Dãy con thứ nhất gồm  $k$  số đầu tiên trong dãy đã cho và tổng các phần tử của dãy con này là  $T_1$ .
- Dãy con thứ hai gồm các số còn lại của dãy số đã cho và tổng các phần tử của dãy con này là  $T_2$ .

**Yêu cầu:** Tìm số nguyên dương  $k$  là độ dài của dãy con thứ nhất sao cho  $|T_1 - T_2|$  nhỏ nhất.

**Chú ý:** Nếu có hơn một số  $k$  thỏa mãn thì ghi ra số  $k$  nhỏ nhất.

### Dữ liệu

- Dòng đầu tiên ghi một số nguyên dương  $n$  ( $n \leq 1000000$ )
- Dòng thứ hai ghi  $n$  số nguyên  $a_1, a_2, \dots, a_n$  với  $|a_i| \leq 10$  ( $1 \leq i \leq n$ ), mỗi số cách nhau một dấu cách.

### Kết quả

- Ghi ra một số nguyên dương  $k$  thỏa mãn yêu cầu của đề bài.

### Sample Input

```
6
4 7 1 1 4 6
```

### Sample Output

```
2
```

- a) Viết thuật toán có độ phức tạp  $O(n^2)$
- b) Viết thuật toán có độ phức tạp  $O(n)$

[Chia dãy \(THT'15\) - LQDOJ: Le Quy Don Online Judge](#)

# Làm sao để có thể tìm kiếm tốt hơn $O(n)$ ?

1. Khai thác thông tin về dữ liệu.
2. Tổ chức lại cách lưu trữ dữ liệu

# Tìm kiếm với độ phức tạp hằng số

- Cho một mảng A có N phần tử và các phần tử có giá trị nằm trong khoảng  $0..10^6$ .
  - Cho một số X, kiểm tra xem X có tồn tại trong mảng A hay không.
  - Thực hiện thao tác kiểm tra M lần (e.g.  $M \sim 10^5$  lần).
- Tìm kiếm tuyến tính chỉ khả dĩ khi tìm, e.g. 10-100 lần, trừ khi bạn có thể ngồi đợi hàng tiếng - Độ phức tạp  $O(M*N)$
- Khai thác miền giá trị để “lưu trữ”.
  - Sử dụng một mảng phụ b[ $0..10^6$ ] phần tử.
  - $a[i] = x$  thì đánh dấu  $b[x] = \text{true}$  (hoặc bằng 1) // nghĩa là x tồn tại trong mảng A.
  - Kiểm tra số X có tồn tại trong mảng A hay không thì tương đương với kiểm tra  $b[X]$  có bằng 1 hay không.
  - Độ phức tạp  $O(N+M)$  với  $O(N)$  độ phức tạp do tiền xử lý tạo mảng B; và  $O(M)$  là độ phức tạp của M truy vấn.

## Điểm danh vắng mặt

Một lớp học nọ của Boss Small có  $N$  học sinh. Một ngày đẹp trời, Boss Small nhận thấy số học sinh đi học chỉ có  $M$  người, ít hơn  $N$  nên Boss quyết định nhờ bạn điểm danh các học sinh trong lớp. Hãy viết một chương trình cho biết số thứ tự của các học sinh vắng mặt **theo thứ tự tăng dần**.

Biết rằng, lớp học đánh số thứ tự cho học sinh từ 1 cho đến  $N$ .

### Input

- Dòng đầu tiên chứa hai số nguyên dương lần lượt là  $N$  và  $M$  ( $1 \leq M < N \leq 10^5$ )
- Dòng thứ hai chứa  $M$  số nguyên khác nhau từng đôi một, có giá trị trong đoạn  $[1, N]$ .

### Output

In ra một danh sách các số nguyên, là số thứ tự của những học sinh vắng mặt, theo thứ tự tăng dần.

### Ví dụ

#### Input 1

```
5 3
5 2 3
```

[Copy](#)

#### Output 1

```
1 4
```

[Copy](#)

[Điểm danh vắng mặt - LQDOJ: Le Quy Don Online Judge](#)

# Tìm kiếm với độ phức tạp logarit - tìm kiếm nhị phân

- Làm sao để tăng tốc độ tìm kiếm với dữ liệu đã được sắp xếp (e.g. có thứ tự)
  - Thực hiện được khi dữ liệu được lưu trữ có tổ chức
- [Thuật toán tìm kiếm nhị phân \(tek4.vn\)](http://tek4.vn)
- Thuật toán tìm kiếm nhị phân áp dụng với các dãy có thứ tự được lưu trên mảng/vector
  - Khai thác tính chất có thứ tự
- Áp dụng quy tắc chia để trị để làm hẹp không gian tìm kiếm
  - So sánh giá trị cần tìm kiếm với giá trị “nằm giữa” để thu hẹp không gian tìm kiếm.
  - Mỗi lần không gian tìm kiếm giảm  $\frac{1}{2}$  so với lần trước đó.

# Tìm kiếm nhị phân

- Áp dụng quy tắc chia để trị
  - Giả sử tìm kiếm  $x$  trên đoạn  $A[l..r]$  có thứ tự không giảm
  - Nếu  $l == r$ , so sánh  $x$  với  $A[l]$  và dừng.
  - Ngược lại, so sánh phần tử cần tìm với phần tử nằm giữa của danh sách,  $A[m]$  với  $m = (l+r)/2$ 
    - Nếu  $A[m] == x$ : dừng
    - Nếu  $A[m] > x$ : tìm kiếm trong khoảng  $A[l..m-1]$  //các số trong khoảng  $A[m..r]$  đều  $> x$
    - Nếu  $A[m] < x$ : tìm kiếm trong khoảng  $A[m+1..r]$  //các số trong khoảng  $A[m..r]$  đều  $< x$
- Triển khai thuận tiện bằng đệ quy/lặp
- Độ phức tạp:  $O(\log n)$  với  $n$  là kích thước dữ liệu.

$$T(n) = T(n/2) + O(1)$$

# Binary Search

Search 23

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

23 > 16  
take 2<sup>nd</sup> half

L=0	1	2	3	M=4	5	6	7	8	H=9
2	5	8	12	16	23	38	56	72	91

23 < 56  
take 1<sup>st</sup> half

0	1	2	3	4	L=5	6	M=7	8	H=9
2	5	8	12	16	23	38	56	72	91

Found 23,  
Return 5

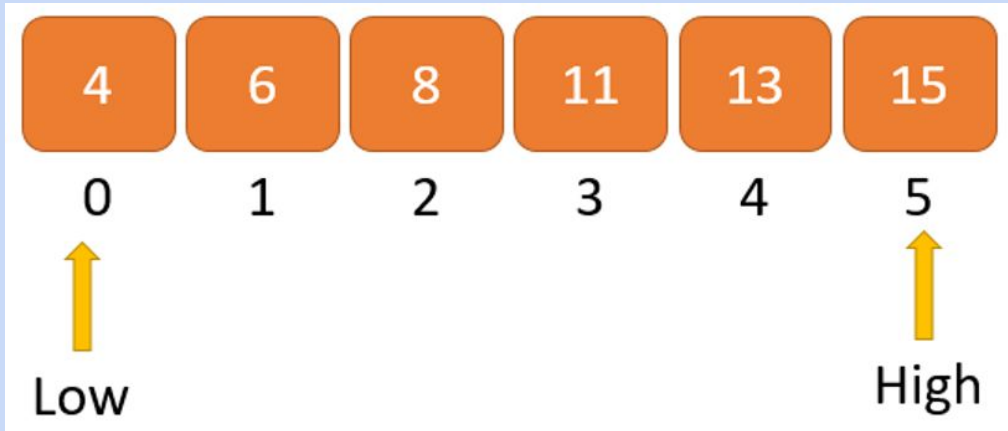
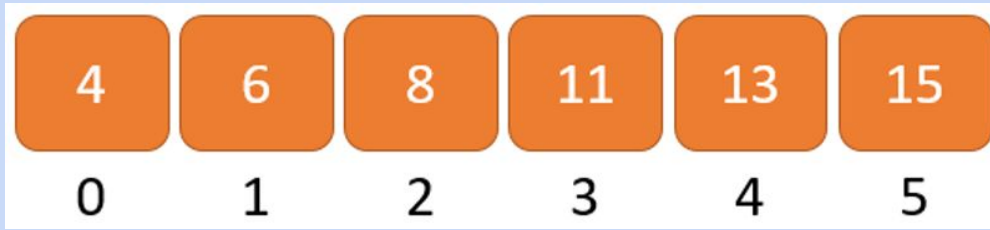
0	1	2	3	4	L=5, M=5	H=6	7	8	9
2	5	8	12	16	23	38	56	72	91

1. L = 0, H = 9, M = 4
2. L = 5, H = 9, M = 7
3. L = 5, H = 6, M = 5

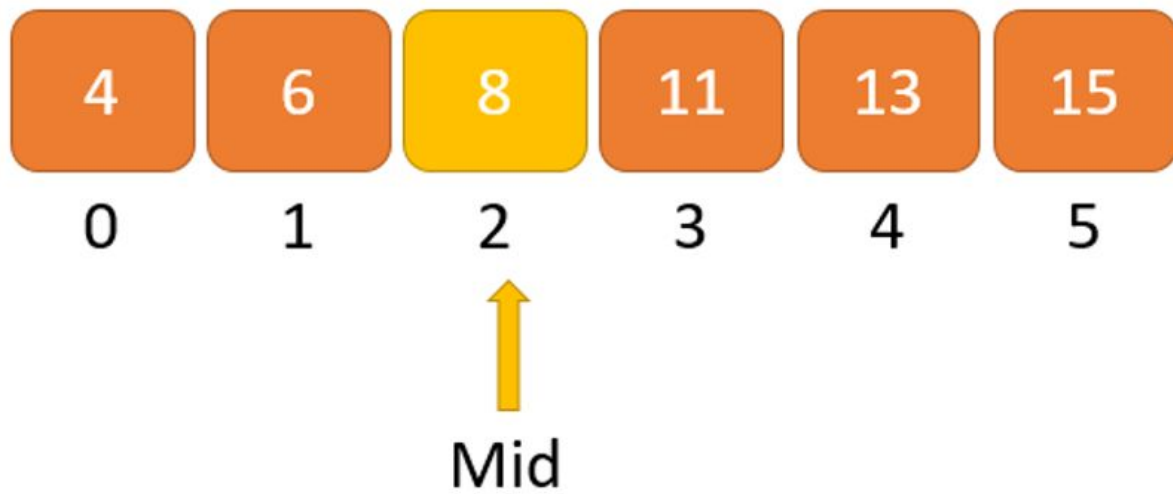


Tìm kiếm phần tử 23 trong mảng tăng dần gồm 9 phần tử

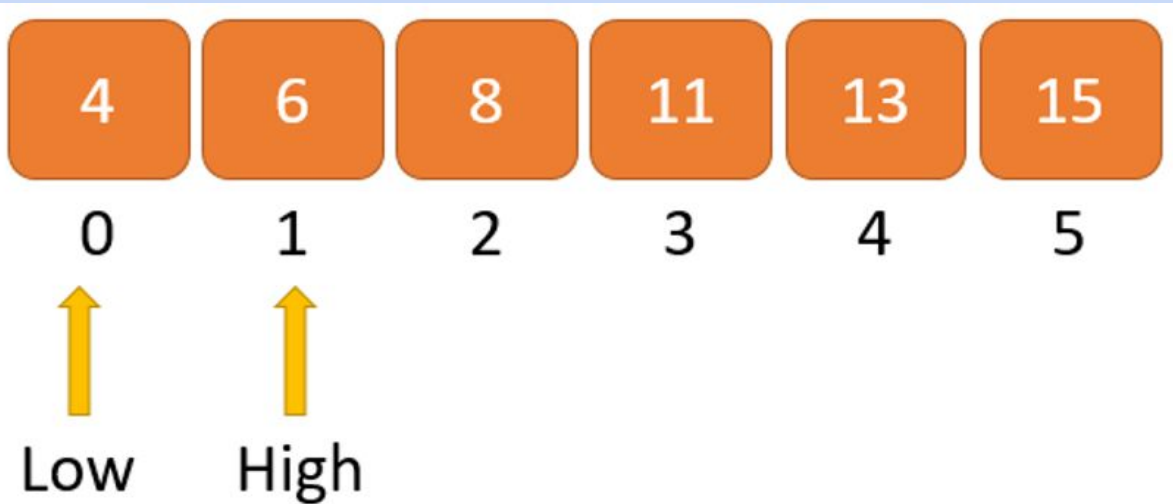




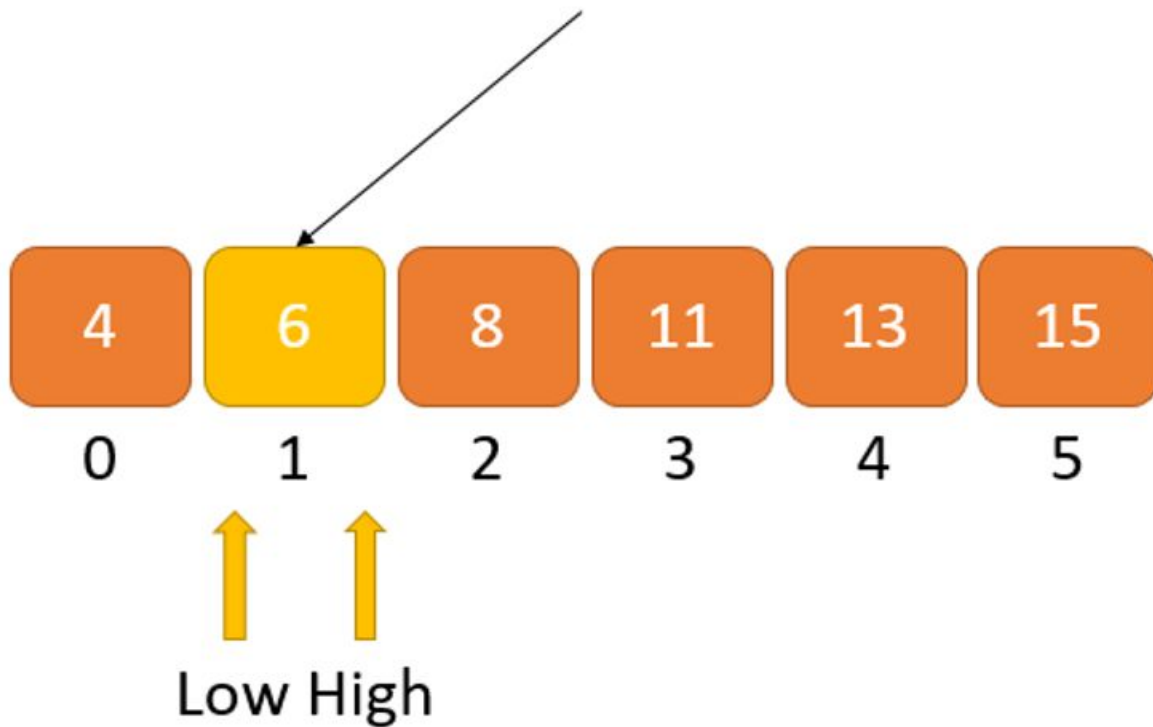
Tìm kiếm phần tử có giá trị 6 trong mảng tăng dần gồm 6



Tìm kiếm phần tử có giá trị 6 trong mảng tăng dần gồm 6



Phần tử cần tìm



Tìm kiếm phần tử có giá trị 6 trong mảng tăng dần gồm 6

```

1. // A recursive binary search function.It returns
2. // location of x in given array arr[l..r] is present,
3. // otherwise -1
4. int binarySearch(int arr[], int l, int r, int x)
5. {
6.     if (r == l )
7.         return arr[r] == x ? r : -1;
8.
9.     int mid = l + (r - l) / 2;
10.
11.     // If the element is present at the middle itself
12.     if (arr[mid] == x)
13.         return mid;
14.
15.     // If element is smaller than mid, then it can only be present in left subarray
16.     if (arr[mid] > x)
17.         return binarySearch(arr, l, mid - 1, x);
18.
19.     // Else the element can only be present in right subarray
20.     return binarySearch(arr, mid + 1, r, x);
21. }

```

Ref: [Binary Search -  
GeeksforGeeks](https://www.geeksforgeeks.org/binary-search/)

Code mẫu:

[5s8XVL - Online C++  
Compiler & Debugging Tool  
- Ideone.com](https://www.5s8xvl.com/)

Tìm kiếm nhị phân bằng đệ quy

```
1. // An iterative binary search function.It returns
2. // location of x in given array arr[l..r] if present,
3. // otherwise -1
4. int binarySearch(int arr[], int l, int r, int x)
5. {
6.     while (l <= r)
7.     {
8.         int m = l + (r - l) / 2;
9.
10.        // Check if x is present at mid
11.        if (arr[m] == x) return m;
12.
13.        // If x greater, ignore left half
14.        if (arr[m] < x)
15.            l = m + 1;
16.        // If x is smaller, ignore right half
17.        else
18.            r = m - 1;
19.    }
20.    // if we reach here, then element was not present
21.    return -1;
22. }
```

Ref: [Binary Search - GeeksforGeeks](#)

Code mẫu: [TBkcWX - Online C++ Compiler & Debugging Tool - Ideone.com](#)

Tìm kiếm nhị phân bằng code tuần tự

# Tìm kiếm nhị phân trong STL C++

- Áp dụng trên mảng/vector có thứ tự (e.g. giả sử tăng dần)
- [Tổng quan về tìm kiếm nhị phân | How Kteam](#)

**Binary search** (operating on partitioned/sorted ranges):

<b>lower_bound</b>	Return iterator to lower bound (function template )
<b>upper_bound</b>	Return iterator to upper bound (function template )
<b>equal_range</b>	Get subrange of equal elements (function template )
<b>binary_search</b>	Test if value exists in sorted sequence (function template )

# Tìm kiếm nhị phân trong STL C++

- Áp dụng trên mảng/vector có thứ tự (e.g. giả sử tăng dần)
  - [binary\\_search - C++ Reference \(cplusplus.com\)](#): trả về true nếu phần tử x trong danh sách tìm kiếm.
  - [equal\\_range - C++ Reference \(cplusplus.com\)](#): trả về vị trí l và r trong danh sách tiếp kiểm các các phần tử trong đoạn [l, r - 1] đều bằng giá trị x.
  - [lower\\_bound - C++ Reference \(cplusplus.com\)](#): trả về vị trí đầu tiên trong danh sách có giá trị  $\geq x$ .
  - [upper\\_bound - C++ Reference \(cplusplus.com\)](#): trả về vị trí đầu tiên trong danh sách có giá trị  $> x$ .

# Các thuật toán tìm kiếm khác có độ phức tạp logarithm

- Cũng khai thác ý tưởng chia đoạn thành nhiều đoạn con
- Thuật toán tìm kiếm nội suy (interpolation search)
  - [Thuật toán tìm kiếm nội suy \(tek4.vn\)](http://tek4.vn)
  - [Interpolation Search - GeeksforGeeks](http://GeeksforGeeks)
- Tìm kiếm tam phân
  - [Ternary Search - GeeksforGeeks](http://GeeksforGeeks)
  - Chia ra làm ba đoạn và tìm kiếm trên đoạn tương ứng
- etc.



# Thực hành

- Triển khai thuật toán tìm kiếm tuyến tính/tìm kiếm nhị phân
- Thực hiện thao tác tìm kiếm  $M$  lần trên cùng 1 dãy số (có  $10^5$  phần tử được sinh ngẫu nhiên).
- Báo cáo thời gian thực hiện cho  $M = 100$ ,  $M = 10^5$ ,  $M = 10^6$ .