Big-O Blue

LECTURE 12

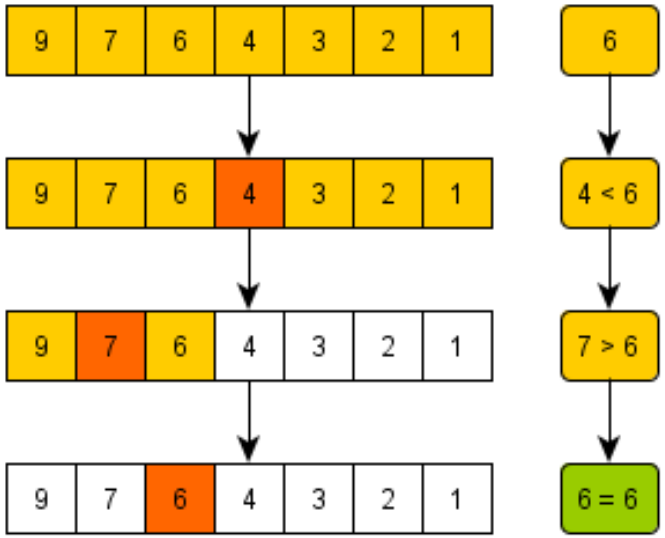
BINARY SEARCH

Big-O Coding

Website: www.bigocoding.com

Binary Search

Binary Search (Tìm kiếm nhị phân hay chặt nhị phân) là thuật toán tìm kiếm một phần tử trong mảng đã được sắp xếp.



Độ phức tạp: **O(log(N))**

Ý tưởng của thuật toán

Mảng ban đầu đã được sắp xếp sẵn.

0	1	2	3	4	5	6	7	8	9
V1	V2	V2	V4	V5	V6	V7	V8	V9	V10

So sánh phần tử cần tìm và phần tử chính giữa mảng để thu hẹp phạm vi tìm kiếm.

0	1	2	3	4	5	6	7	8	9
V1	V2	V2	V4	V5	V6	V7	V8	V9	V10

Tùy vào giá trị tìm kiếm sẽ loại số lượng phần tử **bên phải** hay **bên trái** ra khỏi phạm vi tìm kiếm.

0	1	2	3	4
V1	V2	V2	V4	V5

Quay trở lại bước so sánh trên đến khi mảng chỉ còn 1 phần tử cuối cùng.

Bài toán minh họa

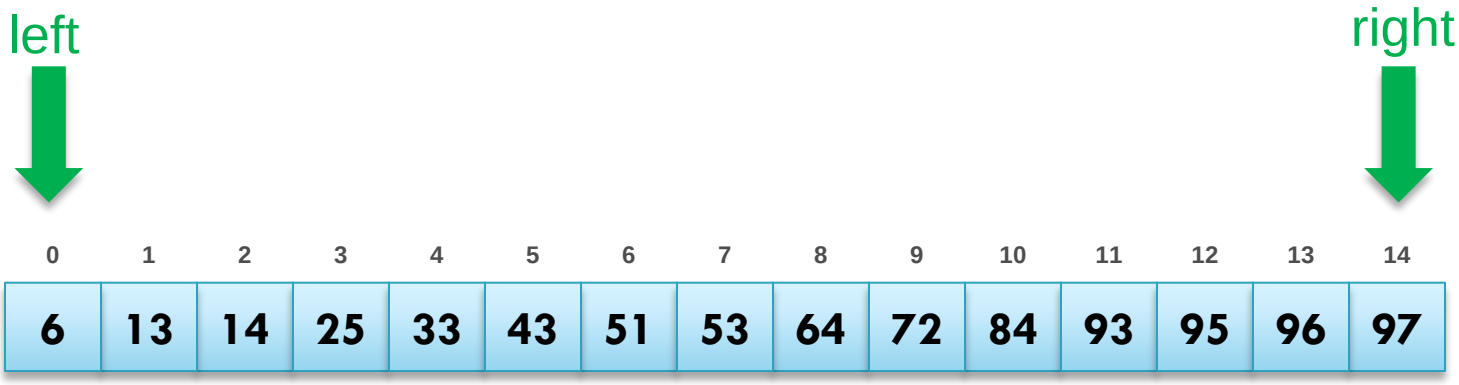
Cho mảng đã được sắp xếp **tăng dần.** Hãy tìm vị trí của phần tử có giá trị **x = 33** trong mảng.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	14	25	33	43	51	53	64	72	84	93	95	96	97

Bước 0: Chuẩn bị dữ liệu

Khởi tạo các giá trị ban đầu cho các biến.

- n = 15
- left = 0
- right = n – 1 = 14

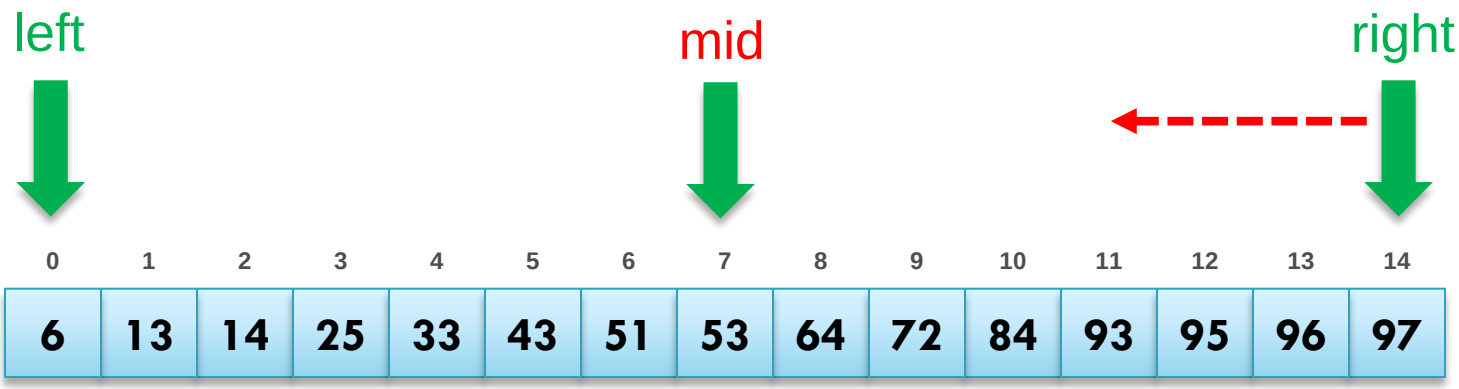


Bước 1: Chạy thuật toán lần 1

0 14

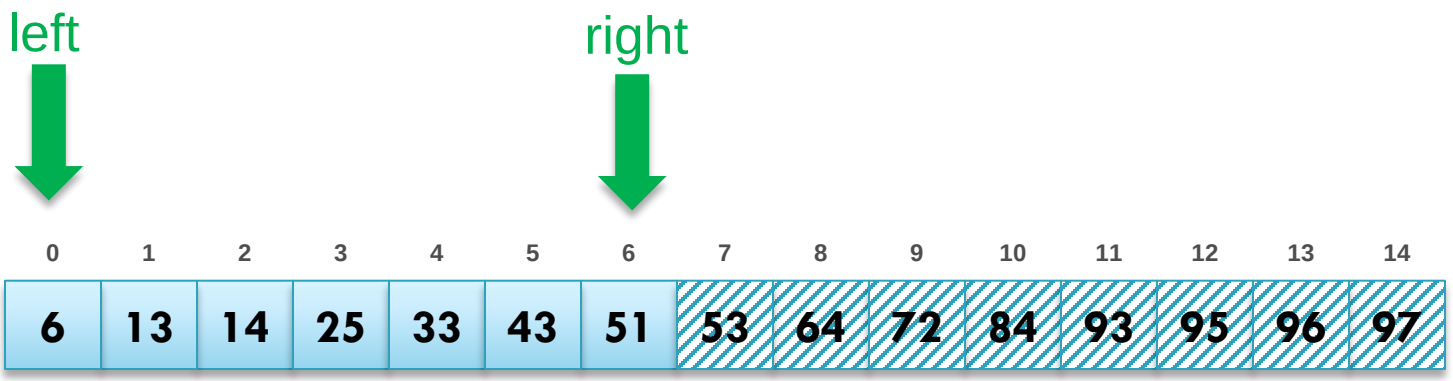
- mid = (left + right)/2 = 7
- x (33) < a[mid] (53)
- left = 0

➔ Tính lại: right = mid – 1 = 6



Bước 1: Chạy thuật toán lần 1

- Giá trị của left và right sau khi được tính lại:
  - left = 0
  - right = 6



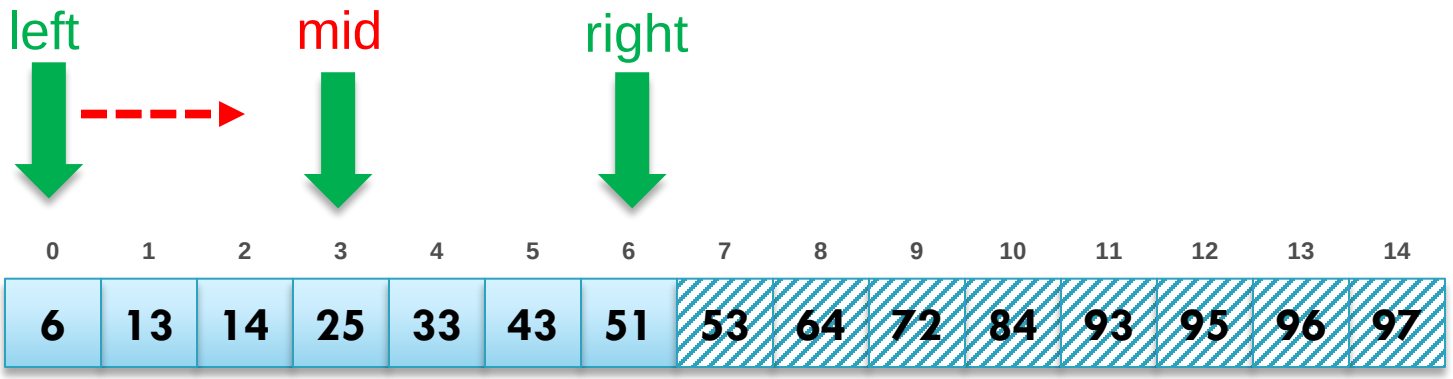
Bước 2: Chạy thuật toán lần 2

0 6

- mid = (left + right)/2 = 3
- x (33) > a[mid] (25)

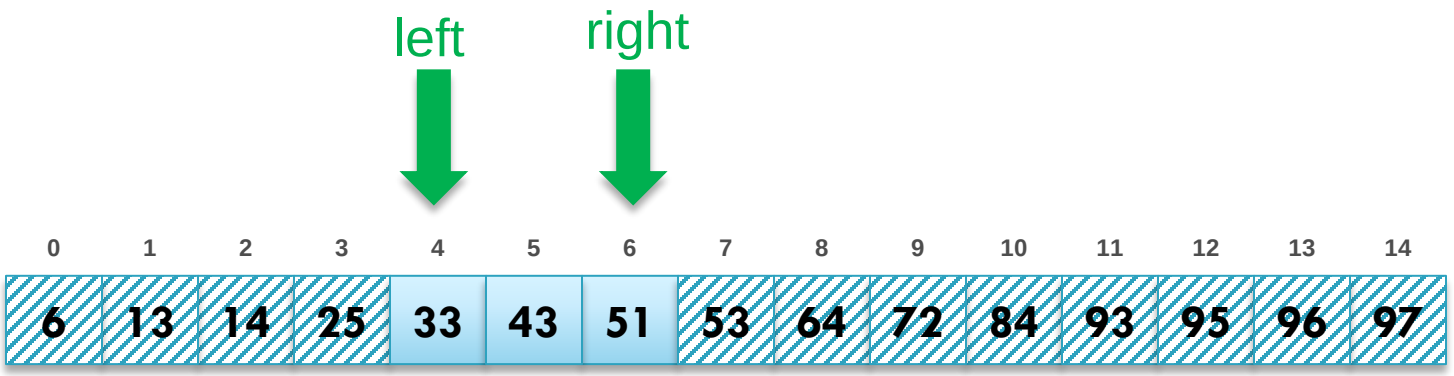
➔ Tính lại: left = mid + 1 = 4

- right = 6



Bước 2: Chạy thuật toán lần 2

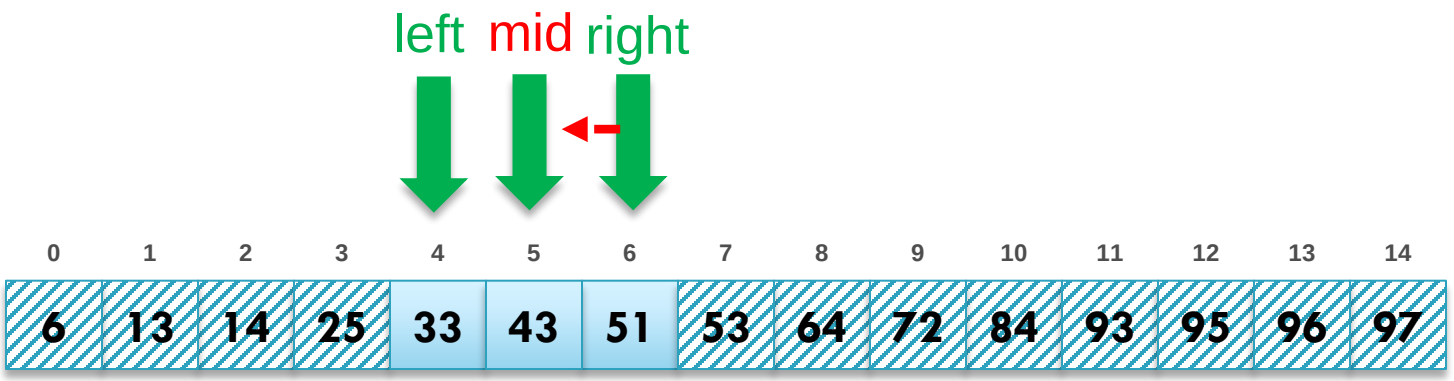
- Giá trị của left và right sau khi được tính lại:
  - left = 4
  - right = 6



Bước 3: Chạy thuật toán lần 3

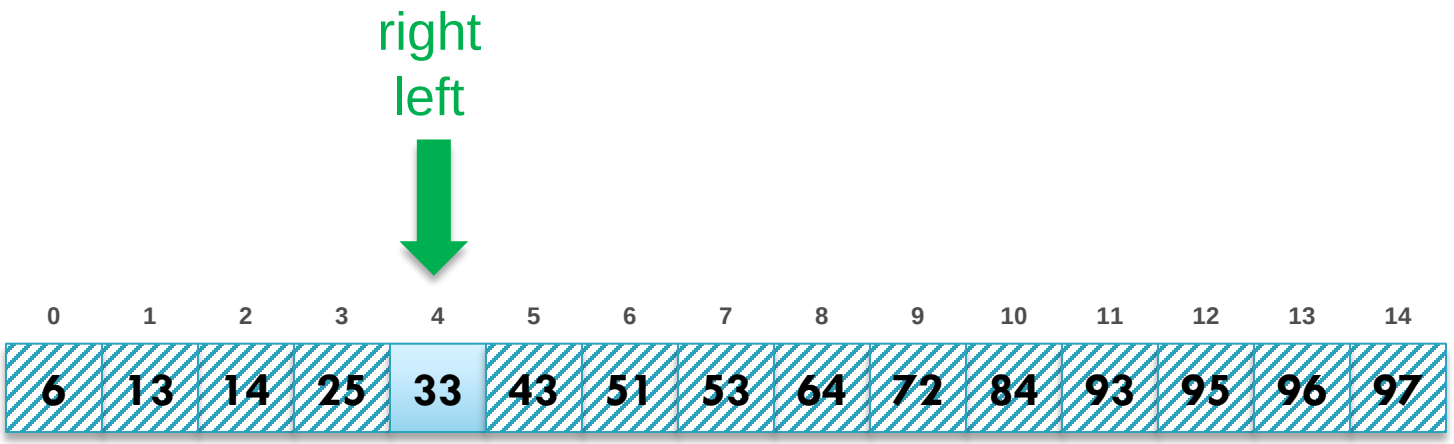
- mid = (left + right)/2 = 5
  (left = 4, right = 6)
- x (33) < a[mid] (43)
- left = 4

➔ Tính lại: right = mid – 1 = 4



Bước 3: Chạy thuật toán lần 3

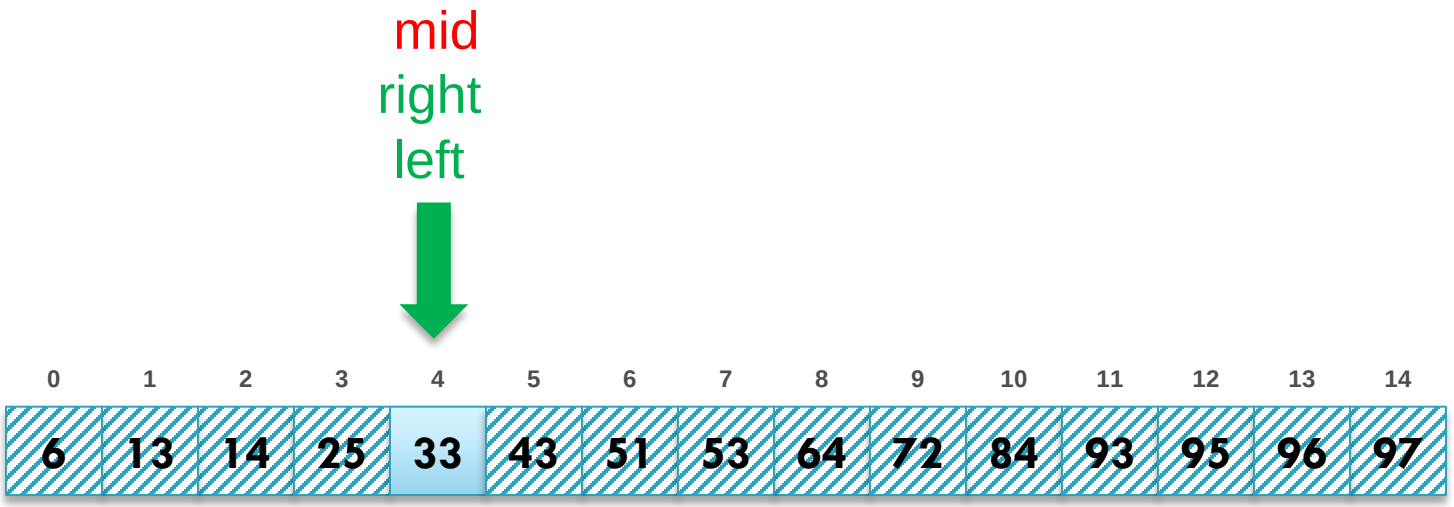
- Giá trị của left và right sau khi được tính lại:
  - left = 4
  - right = 4



Bước 4: Chạy thuật toán lần 4

- mid = (left + right)/2 = 4
  (left = 4, right = 4)
- x (33) == a[mid] (33)

Kết quả Tìm được **vị trí của x là 4**. Dừng thuật toán.

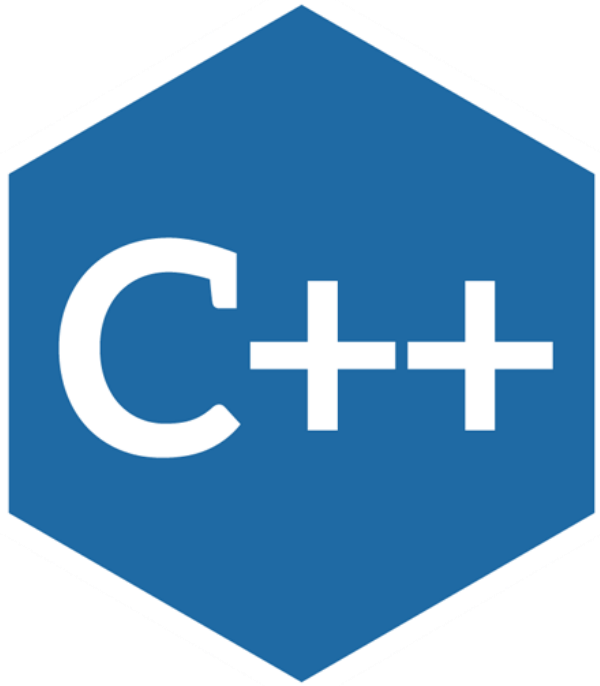


Nhận xét

Tìm kiếm nhị phân dựa vào **quan hệ giá trị** của các phần tử trong mảng để định hướng trong quá trình tìm kiếm, do vậy chỉ áp dụng được cho các dãy **đã có thứ tự**.

Khi mảng có biến động cần phải tiến hành sắp xếp lại. Tìm kiếm nhị phân cần phải xét đến thời gian sắp xếp lại mảng, thời gian sắp xếp này không nhỏ, phải cân nhắc khi thực hiện.

MÃ NGUỒN MINH HỌA BẰNG C++



Source Code Binary Search

```cpp
#include <iostream>
#include <vector>
using namespace std;
int binarySearch(const vector<int> &a, int left, int right, int x)
{
    while (left <= right)
    {
        int mid = (left + right) / 2;
        if (x == a[mid])
            return mid;
        else if (x < a[mid])
            right = mid - 1;
        else if (x > a[mid])
            left = mid + 1;
    }
    return -1;
}
```

Source Code Binary Search

```
18. int main()
19. {
20.     vector<int> a;
21.     int n, x, value;
22.     cin >> n >> x;
23.     for (int i = 0; i < n; i++)
24.     {
25.         cin >> value;
26.         a.push_back(value);
27.     }
28.     int result = binarySearch(a, 0, n - 1, x);
29.     cout << result;
30.     return 0;
31. }
```

Source Code Binary Search (Đệ quy)

```cpp
int binarySearch(const vector<int> &a, int left, int right, int x)
{
    if (left <= right)
    {
        int mid = left + (right - left) / 2;
        if (a[mid] == x)
            return mid;
        if (a[mid] > x)
            return binarySearch(a, left, mid - 1, x);
        return binarySearch(a, mid + 1, right, x);
    }
    return -1;
}
```

MÃ NGUỒN MINH HỌA BẰNG PYTHON

Source Code Binary Search

```
1.  def binarySearch(a, left, right, x):
2.      while left <= right:
3.          mid = (left + right) // 2
4.          if x == a[mid]:
5.              return mid
6.          elif x < a[mid]:
7.              right = mid - 1
8.          else:
9.              left = mid + 1
10.     return -1
```


```
11. if __name__ == '__main__':
12.     n, x = map(int, input().split())
13.     a = list(map(int, input().split()))
14.     result = binarySearch(a, 0, n-1, x)
15.     print(result)
```

Source Code Binary Search (Đệ quy)

```
1. def binarySearch(a, left, right, x):
2.     if left <= right:
3.         mid = (left + right) // 2
4.         if a[mid] == x:
5.             return mid;
6.         if a[mid] > x:
7.             return binarySearch(a, left, mid - 1, x)
8.         return binarySearch(a, mid + 1, right, x)
9.     return -1
```

MÃ NGUỒN MINH HỌA BẰNG JAVA

Source Code Binary Search

```
1.  public static int binarySearch(int a[], int left, int right, int x) {
2.      while (left <= right) {
3.          int mid = left + (right - left) / 2;
4.          if (x == a[mid]) {
5.              return mid;
6.          }
7.        else if (x < a[mid]) {
8.              right = mid - 1;
9.          }
10.       else {
11.             left = mid + 1;
12.         }
13.     }
14.     return -1;
15. }
```

Source Code Binary Search

```
16. public static void main (String[] args) {
17.     Scanner sc = new Scanner(System.in);
18.     int n = sc.nextInt();
19.     int x = sc.nextInt();
20.     int a[] = new int[n];
21.     for (int i = 0; i < n; i++) {
22.         a[i] = sc.nextInt();
23.     }
24.     System.out.print(binarySearch(a, 0, n - 1, x));
25. }
```

Source Code Binary Search (Đệ quy)

```java
public static int binarySearch(int a[], int left, int right, int x) {
    if (left <= right) {
        int mid = left + (right - left) / 2;
        if (x == a[mid]) {
            return mid;
        }
      else if (x < a[mid]) {
            return binarySearch(a, left, mid - 1, x);
        }
      else {
            return binarySearch(a, mid + 1, right, x);
        }
    }
    return -1;
}
```

MỘT SỐ HÀM BINARY SEARCH KHÁC CẦN LƯU Ý

Binary Search First (tìm phần tử đầu tiên)

Cho mảng đã được sắp xếp **tăng dần.** Tồn tại nhiều phần tử có giá trị giống nhau. Hãy tìm vị trí của phần tử **đầu tiên** có giá trị **x = 33** trong mảng.

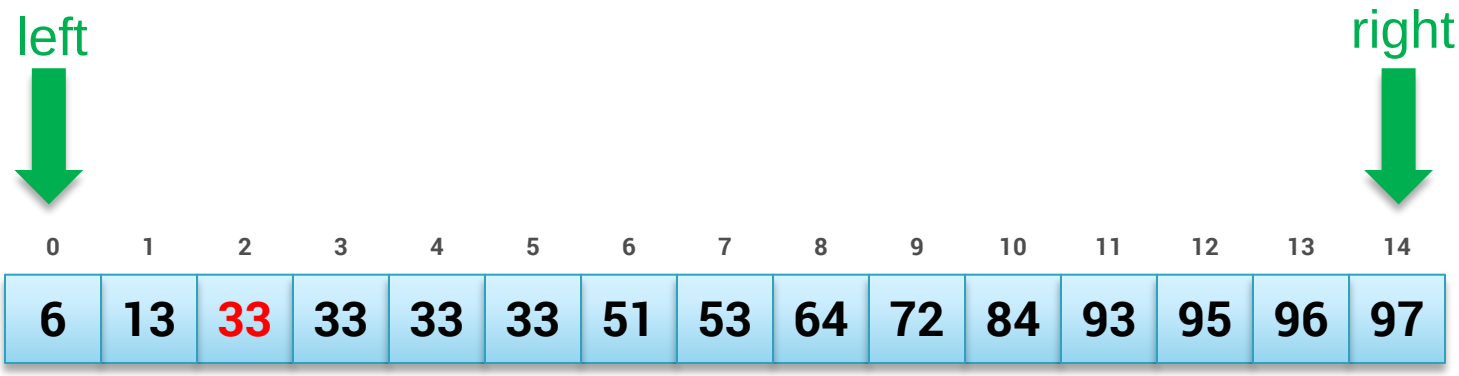
Trong mảng nếu tồn tại nhiều số 33 thì trả ra vị trí đầu tiên, nếu không có tồn tại số 33 thì trả về -1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
6	13	33	33	33	33	51	53	64	72	84	93	95	96	97

Binary Search First (tìm phần tử đầu tiên)

Bước 0: Khởi tạo các giá trị ban đầu cho các biến.

- n = 15
- left = 0
- right = n – 1 = 14

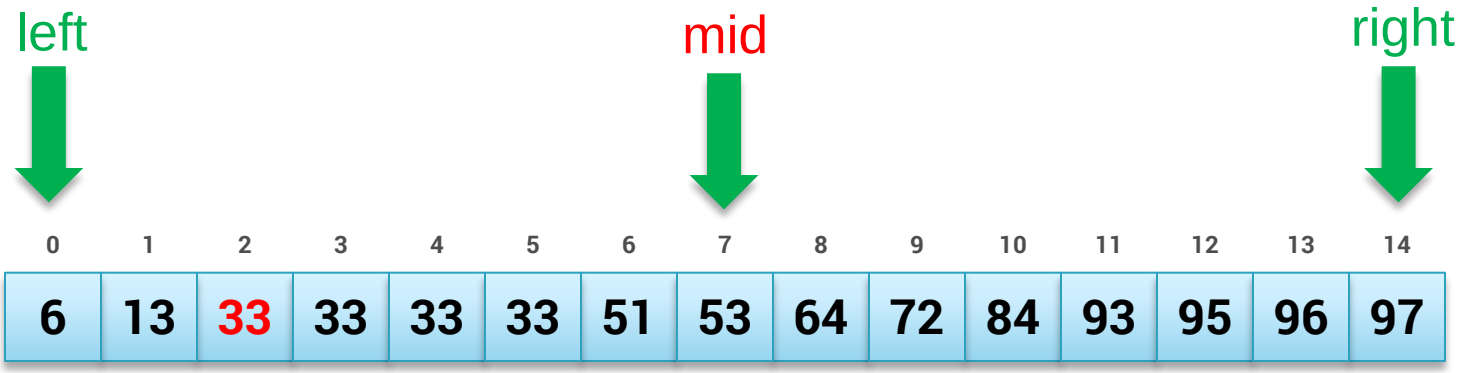


Bước 1: Chạy thuật toán lần 1

0 14

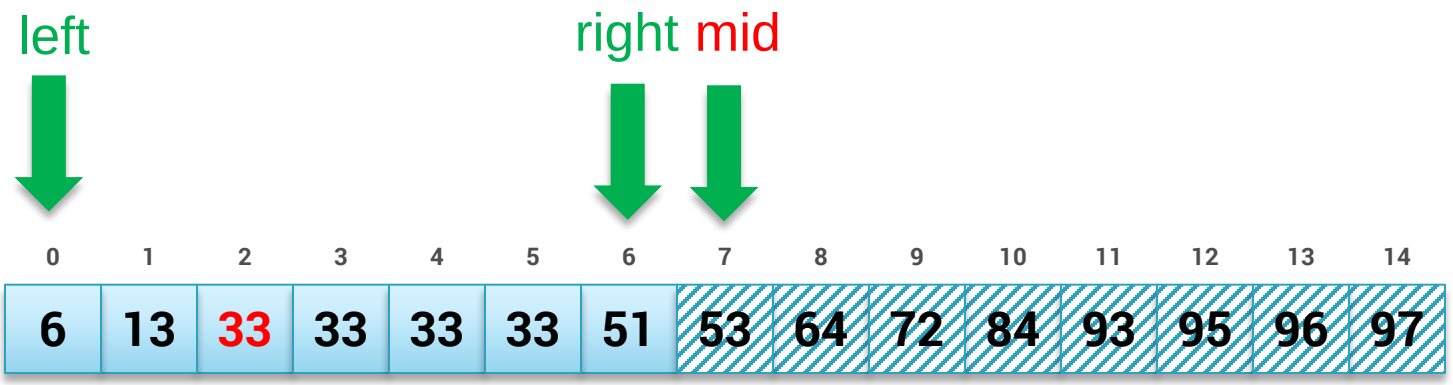
- mid = (left + right)/2 = 7
- x (33) < a[mid] (53)
- left = 0

➔ Tính lại: right = mid – 1 = 6



Bước 1: Chạy thuật toán lần 1

- Giá trị của left và right sau khi được tính lại:
  - left = 0
  - right = 6



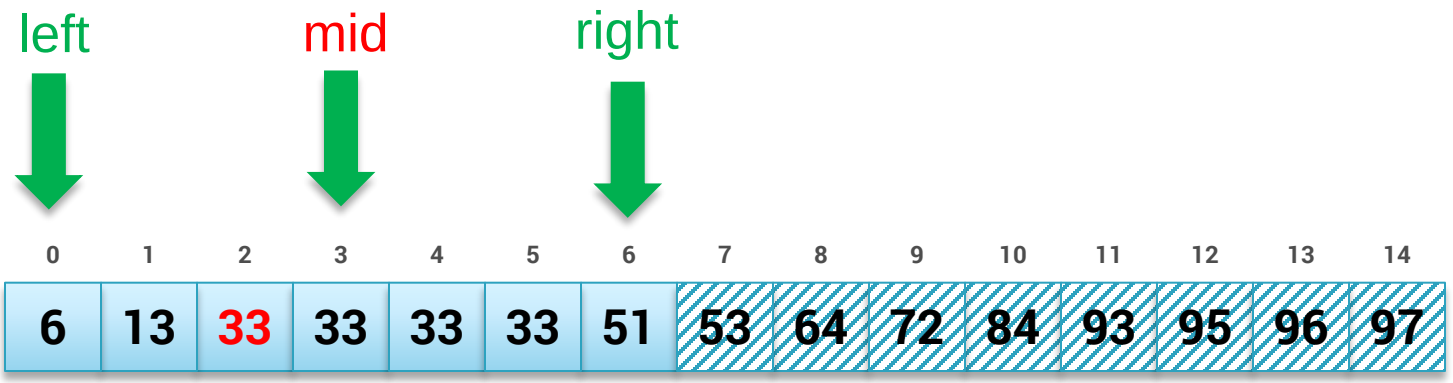
Bước 2: Chạy thuật toán lần 2

0 6

- mid = (left + right)/2 = 3
- x (33) == a[mid] (33)
- left = 0

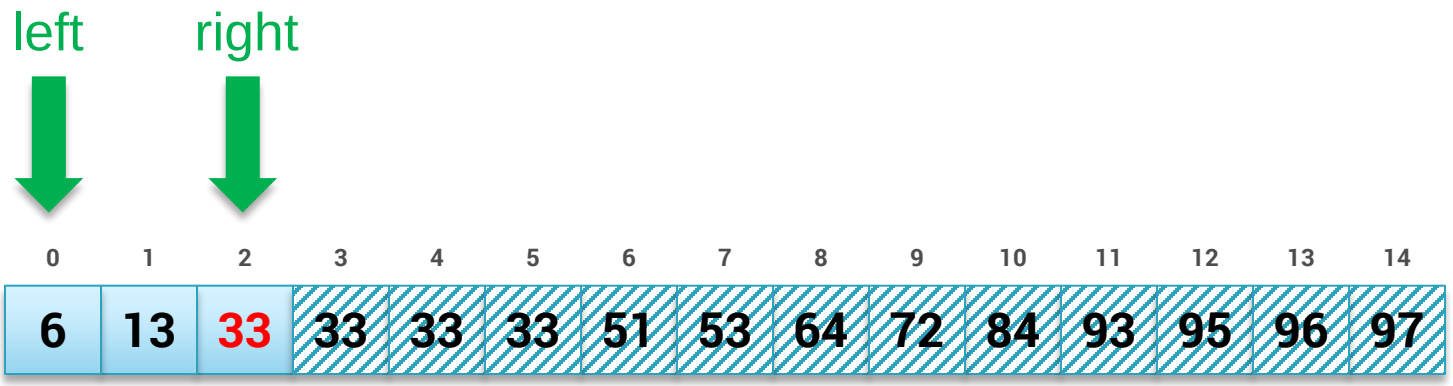
mid == left || x > a[mid – 1]

➔ Tính lại: right = mid – 1 = 2



Bước 2: Chạy thuật toán lần 2

- Giá trị của left và right sau khi được tính lại:
  - left = 0
  - right = 2



Bước 3: Chạy thuật toán lần 3

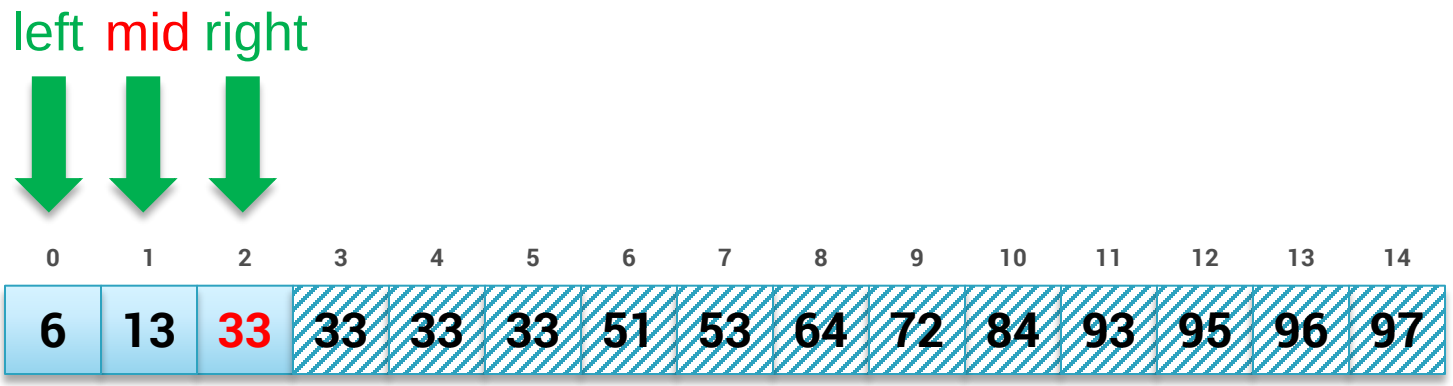
0 2

- mid = (left + right)/2 = 1
- x (33) > a[mid] (13)

➔ Tính lại: left = mid + 1 = 2

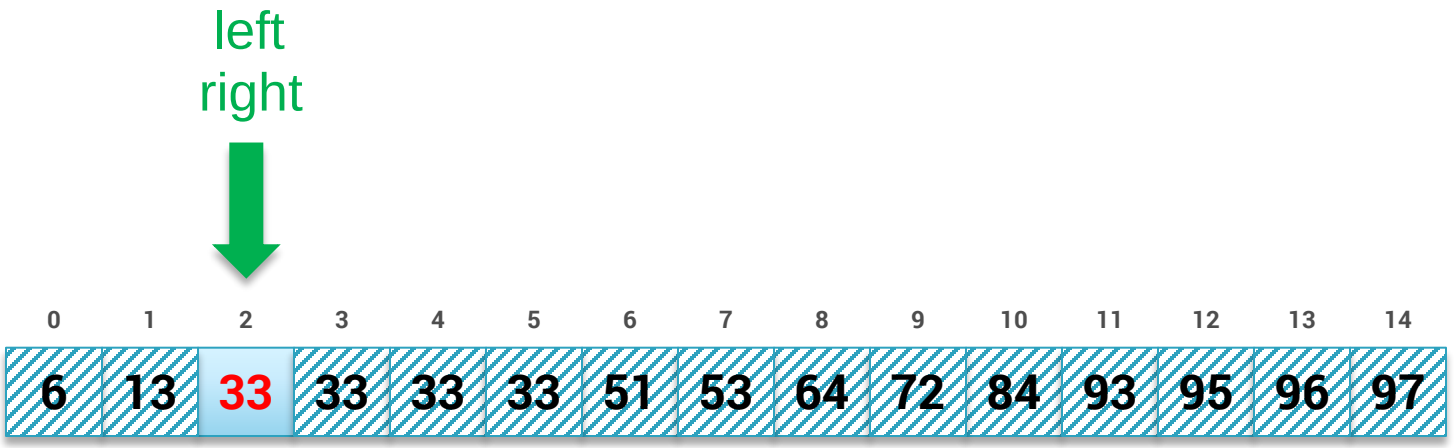
- right = 2

mid == left || x > a[mid – 1]



Bước 3: Chạy thuật toán lần 3

- Giá trị của left và right sau khi được tính lại:
  - left = 2
  - right = 2



Bước 4: Chạy thuật toán lần 4

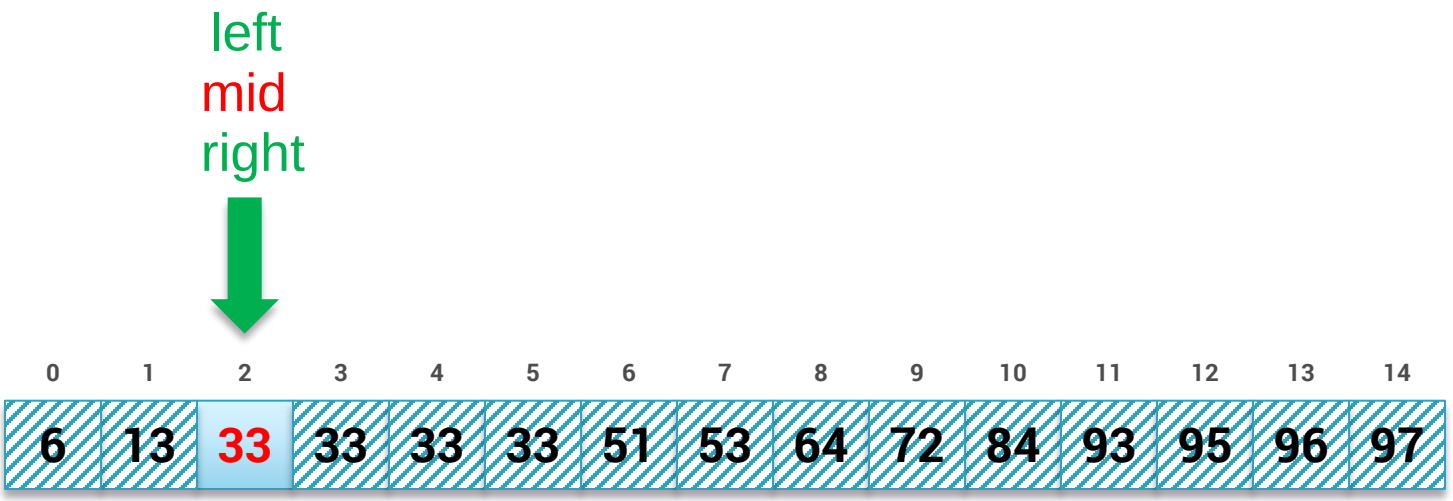
2 2

- mid = (left + right)/2 = 2
- x (33) == a[mid] (33)

mid == left || x > a[mid – 1]

Kết quả

Tìm được **vị trí của x là 2**. Dừng thuật toán.



Binary Search First (tìm phần tử đầu tiên)

```cpp
int bsFirst(const vector<int> &a, int left, int right, int x)
{
    if (left <= right)
    {
        int mid = left + (right - left) / 2;
        if ((mid == left || x > a[mid - 1]) && a[mid] == x)
            return mid;
        else if (x > a[mid])
            return bsFirst(a, (mid + 1), right, x);
        else
            return bsFirst(a, left, (mid - 1), x);
    }
    return -1;
}
```

Binary Search Last (tìm phần tử cuối cùng)

```cpp
int bsLast(const vector<int> &a, int left, int right, int x)
{
    if (left <= right)
    {
        int mid = left + (right - left) / 2;
        if ((mid == right || x < a[mid + 1]) && a[mid] == x)
            return mid;
        else if (x < a[mid])
            return bsLast(a, left, (mid - 1), x);
        else
            return bsLast(a, (mid + 1), right, x);
    }
    return -1;
}
```

Binary Search First (tìm phần tử đầu tiên)

```
1.  def bsFirst(a, left, right, x):
2.      if left <= right:
3.          mid = (left + right) // 2
4.          if (mid == left or x > a[mid - 1]) and a[mid] == x:
5.              return mid
6.          elif x > a[mid]:
7.              return bsFirst(a, mid + 1, right, x)
8.          else:
9.              return bsFirst(a, left, mid - 1, x)
10.     return -1
```

Binary Search Last (tìm phần tử cuối cùng)

```
1.  def bsLast(a, left, right, x):
2.      if left <= right:
3.          mid = (left + right) // 2
4.          if (mid == right or x < a[mid + 1]) and a[mid] == x:
5.              return mid
6.          elif x < a[mid]:
7.              return bsLast(a, left, mid - 1, x)
8.          else:
9.              return bsLast(a, mid + 1, right, x)
10.     return -1
```

Binary Search First (tìm phần tử đầu tiên)

```
1.  public static int bsFirst(int[] a, int left, int right, int x) {
2.      if (left <= right) {
3.          int mid = left + (right - left) / 2;
4.          if ((mid == left || x > a[mid - 1]) && a[mid] == x)
5.              return mid;
6.          else if (x > a[mid])
7.              return bsFirst(a, (mid + 1), right, x);
8.          else
9.              return bsFirst(a, left, (mid - 1), x);
10.     }
11.     return -1;
12. }
```

Binary Search Last (tìm phần tử cuối cùng)

```
1.  public static int bsLast(int[] a, int left, int right, int x) {
2.      if (left <= right) {
3.          int mid = left + (right - left) / 2;
4.          if ((mid == right || x < a[mid + 1]) && a[mid] == x)
5.              return mid;
6.          else if (x < a[mid])
7.              return bsLast(a, left, mid - 1, x);
8.          else
9.              return bsLast(a, mid + 1, right, x);
10.     }
11.     return -1;
12. }
```

DÙNG BINARY SEARCH TRONG THƯ VIỆN CỦA C++/PYTHON/JAVA

Hàm tìm kiếm nhị phân

binary_search: Trả về giá trị **true/false** khi tìm kiếm phần tử.

```
int a[] = {1, 1, 2, 2, 2, 3, 4, 5, 7};
int n = 9;
vector<int> v(a, a + n);
int x = 3;
bool result = binary_search(v.begin(),
v.end(), x);
```

true

~~**binary_search:**~~ Python không có hàm binary search.

Hàm tìm cận dưới ≥

0	1	2	3	4	5	6	7	8
1	1	2	2	2	3	4	5	7

lower_bound: Trả về iterator phần tử **đầu tiên lớn hơn hoặc bằng** giá trị tìm kiếm trong đoạn **[first**, last).

Cú pháp: lower_bound(FwdIt_First, FwdIt_last, x)

```cpp
int a[] = {1, 1, 2, 2, 2, 3, 4, 5, 7};
int n = 9;
vector<int> v(a, a + n);
int x = 3;
vector<int>::iterator low_value;
low_value = lower_bound(v.begin(), v.end(), x);
int index = low_value - v.begin();
cout << index;
```

bisect_left: Trả về ***vị trí*** **đầu tiên lớn hơn hoặc bằng** giá trị tìm kiếm trong đoạn **[first**, last).

Cú pháp: bisect_left(a, x, lo = 0, hi = len(a))

```python
if __name__ == '__main__':
    a = [1, 1, 2, 2, 2, 3, 4, 5, 7]
    n, x = 9, 3
    pos = bisect.bisect_left(a, x, 0, n)
#hoặc
  # pos = bisect.bisect_left(a, x)
    print(pos)
```

*(**NOTE**: C++ trả về iterator nhưng Python trả về index).*

index: 5

Hàm tìm cận trên >

0	1	2	3	4	5	6	7	8
1	1	2	2	2	3	4	5	7

upper_bound: Trả về iterator phần tử **đầu tiên lớn hơn** giá trị tìm kiếm trong đoạn [first, **last).**

Cú pháp: upper_bound(FwdIt_First, FwdIt_last, x)

```
int a[] = {1, 1, 2, 2, 2, 3, 4, 5, 7};
int n = 9;
vector<int> v(a, a + n);
int x = 3;
vector<int>::iterator upp_value;
upp_value = upper_bound(v.begin(), v.end(), x);
int index = upp_value - v.begin();
cout << index;
```

bisect_right: Trả về vị trí **đầu tiên lớn hơn** giá trị tìm kiếm trong đoạn [first, **last).**

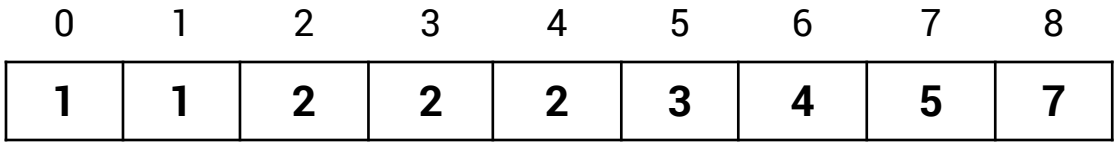
```
if __name__ == '__main__':
    a = [1, 1, 2, 2, 2, 3, 4, 5, 7]
    n, x = 9, 3
    pos = bisect.bisect_right(a, x, 0, n)
#hoặc
   #pos = bisect.bisect_right(a, x)
    print(pos)
```

*(**NOTE**: C++ trả về iterator nhưng Python trả về index).*

index: 6

C++: equal_range: *Trả về đoạn **[first, last)** thuộc kết quả tìm kiếm.*

Một số ví dụ khác về LB và UB



lower_bound: -1 → index: 0

upper_bound : 6 → index: 8

lower_bound: 10 → index: 9

THƯ VIỆN BINARY SEARCH JAVA

Hàm tìm kiếm nhị phân

Collections.binarySearch / Arrays.binarySearch: trả về vị trí phần tử có giá trị bằng giá trị tìm kiếm (nếu có nhiều giá trị thì trả về vị trí bất kì). Nếu không tìm thấy sẽ trả về một con số âm (–*insertion_point* – 1)

Trong đó: *insertion_point* là vị trí phần tử đầu tiên lớn hơn giá trị tìm kiếm.

```
int[] a = new int[]{1, 1, 2, 2, 2, 3, 4, 5, 7};
int result = Arrays.binarySearch(a, 3);
System.out.print(result);
```

5

Hàm tìm kiếm nhị phân

(–insertion_point – 1) insertion_point là vị trí phần tử đầu tiên lớn hơn giá trị tìm kiếm.

Nếu không tìm thấy sẽ ra kết quả như thế nào?

Tìm -1 ➔ kết quả là -1 suy ra insertion_point = 0.

Tìm 6 ➔ kết quả -9 suy ra insertion_point = 8.

Tìm 10 ➔ kết quả -10 suy ra insertion_point = 9.

```
int[] a = new int[]{1, 1, 2, 2, 2, 3, 4, 5, 7};
System.out.println(Arrays.binarySearch(a, -1));
System.out.println(Arrays.binarySearch(a, 6));
System.out.println(Arrays.binarySearch(a, 10));
```

```
-1
-9
-10
```

Hàm tìm kiếm nhị phân

Java không có các hàm tương ứng với lower_bound, upper_bound như của C++/Python. Phải tự cài đặt bằng tay.

Bên dưới là 2 đoạn code tham khảo lowerBound và upperBound trong Java. Dùng để tìm vị trí phần tử đầu tiên trong nửa đoạn **[left, right)** của mảng a có giá trị **lớn hơn hoặc bằng (lowerBound)** hoặc **lớn hơn (upperBound)**.

Nếu không có giá trị thỏa mãn thì trả về giá trị của biến **right**.

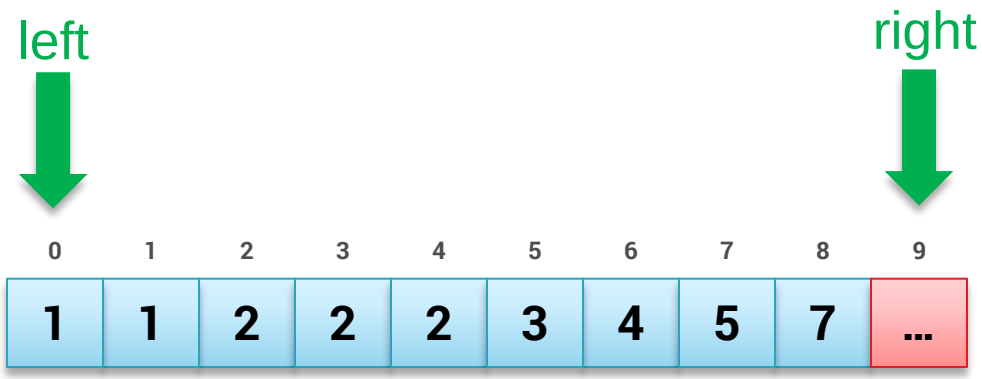
Binary Search - lowerBound

Tự cài đặt hàm lowerBound để tìm giá trị đầu tiên lớn hơn hoặc bằng giá trị tìm kiếm.

Tìm **x = 3**, sẽ trả về **vị trí 5**.

Bước 0: Khởi tạo các giá trị ban đầu cho các biến.

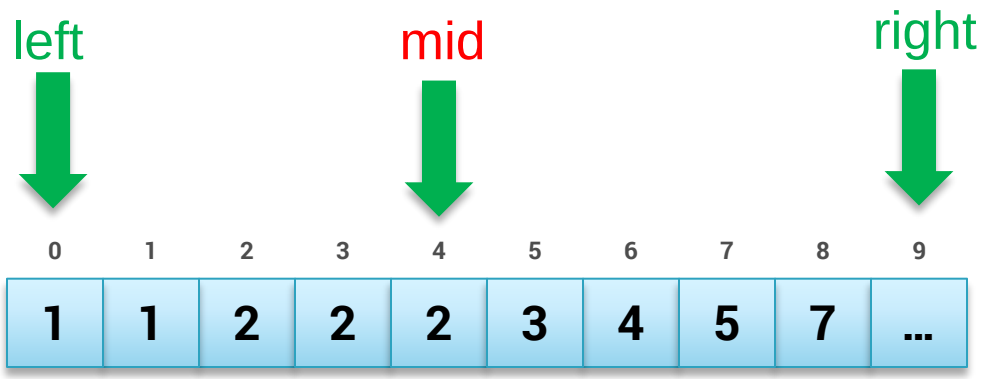
- n = 9
- left = 0
- right = n = 9
- pos = right = 9 (pos là giá trị trả về khi tìm thấy vị trí phù hợp)



Bước 1: Chạy thuật toán lần 1

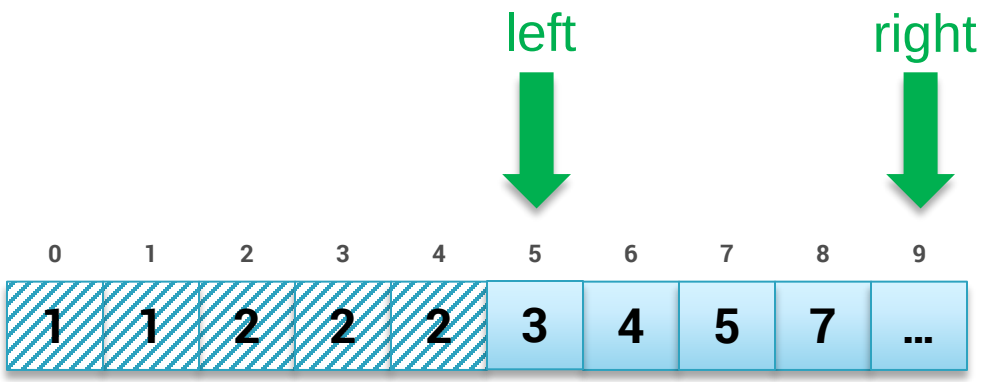
- pos = right = 9
  0 9
- mid = (left + right)/2 = 4
- x (3) > a[mid] (2)

➔ Tính lại: left = mid + 1 = 5



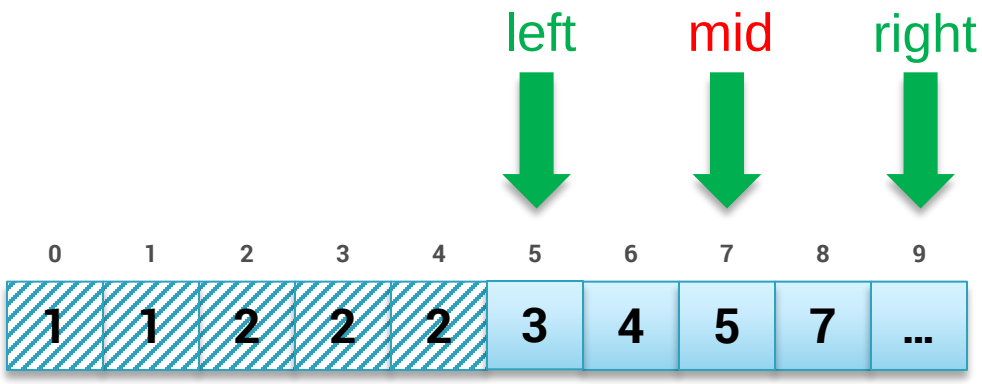
Bước 1: Chạy thuật toán lần 1

- Giá trị của left và right sau khi được tính lại:
  - left = 5
  - pos = right = 9



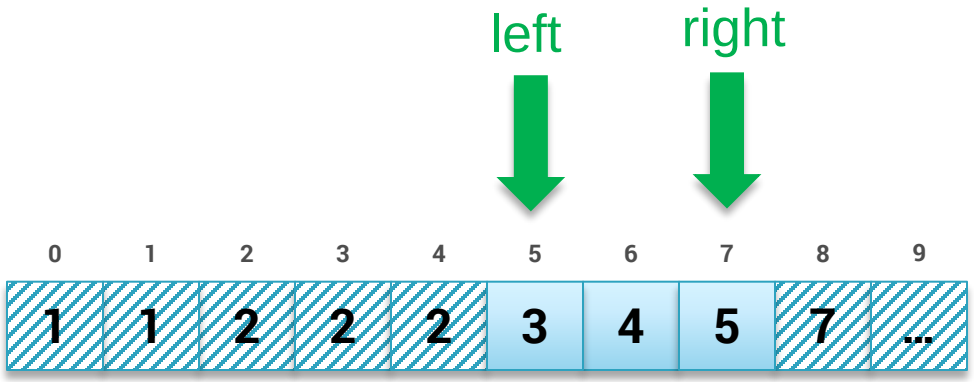
Bước 2: Chạy thuật toán lần 2

- mid = (left + right)/2 = 7
  (left = 5, right = 9)
- x (3) < a[mid] (5)
- Tính lại: pos = right = mid = 7.



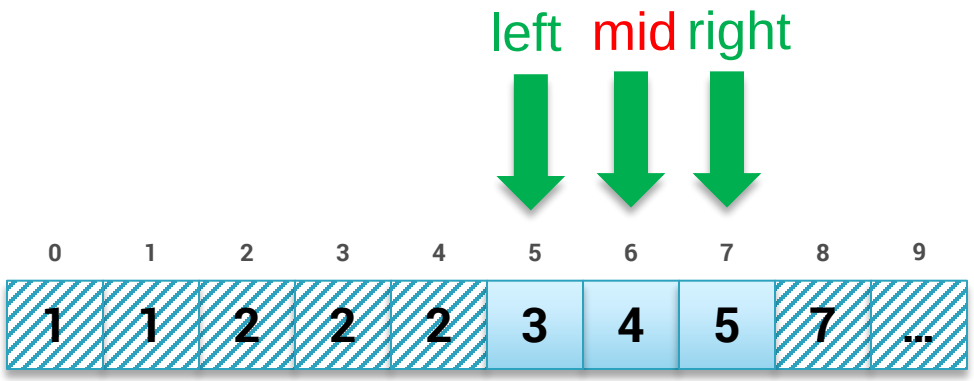
Bước 2: Chạy thuật toán lần 2

- Giá trị của left và right sau khi được tính lại:
  - left = 5
  - pos = right = 7



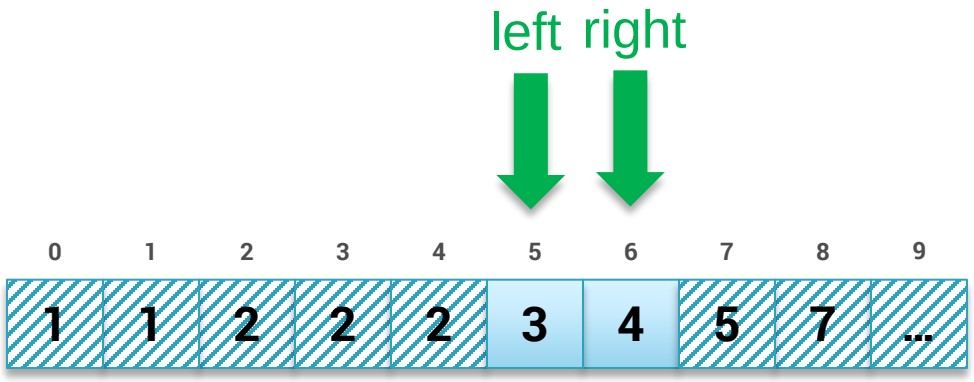
Bước 3: Chạy thuật toán lần 3

- mid = (left + right)/2 = 6 (left = 5, right = 7)
- x (3) < a[mid] (4)
- Tính lại: pos = right = mid = 6.



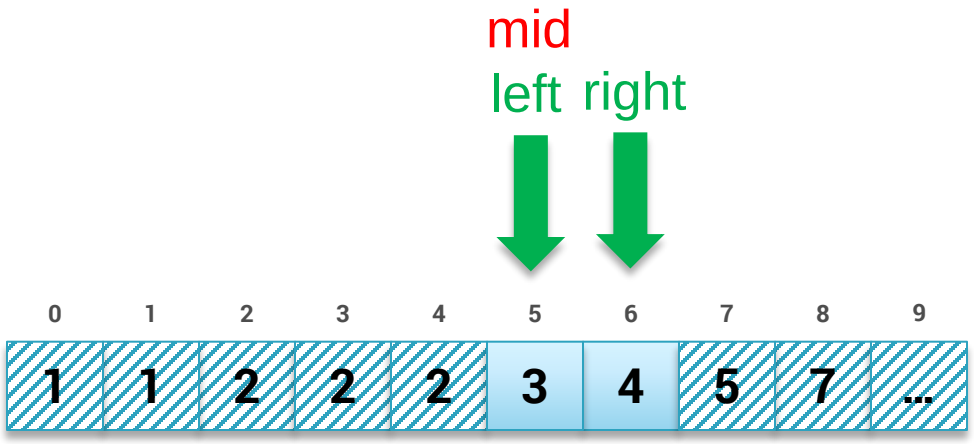
Bước 3: Chạy thuật toán lần 3

- Giá trị của left và right sau khi được tính lại:
  - left = 5
  - pos = right = 6



Bước 4: Chạy thuật toán lần 4

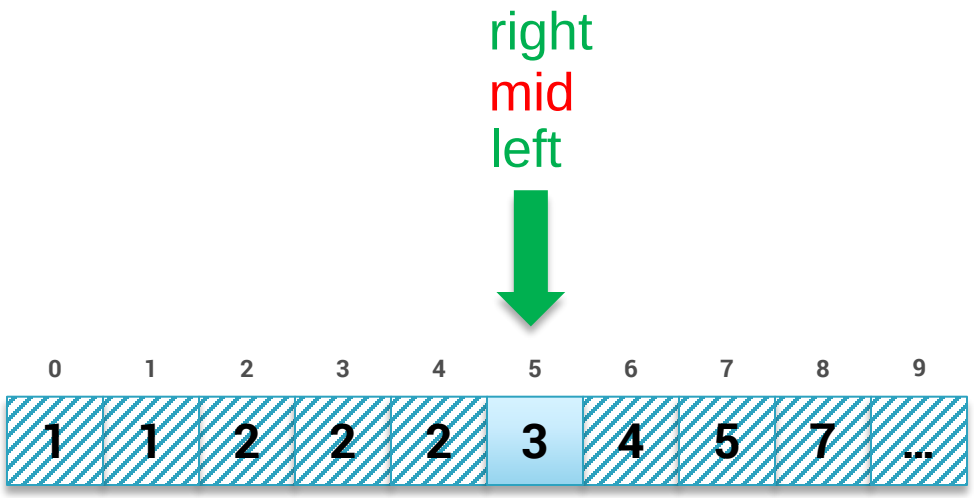
- mid = (left + right)/2 = 5
  (5 = left, 6 = right)
- x (3) == a[mid] (3)
- Tính lại: pos = right = mid = 5.



Bước 4: Chạy thuật toán lần 4

- pos = 5
- left (5) == right (5)

Kết quả Tìm được **vị trí của pos là 5**. Dừng thuật toán.



Code tham khảo lowerBound

```java
public static int lower_bound(int[] a, int left, int right, int x) {
    int pos = right;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (a[mid] >= x) {
            pos = mid;
            right = mid;
        }
        else {
            left = mid + 1;
        }
    }
    return pos;
}
```

Code tham khảo upperBound

```java
public static int upper_bound(int[] a, int left, int right, int x) {
    int pos = right;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (a[mid] > x) {
            pos = mid;
            right = mid;
        }
        else {
            left = mid + 1;
        }
    }
    return pos;
}
```

Hỏi đáp