# CÁC THUẬT TOÁN TÌM KIẾM

# BÀI TOÁN TÌM KIẾM

- Tìm kiếm SBD trong danh sách thi
- Tìm mặt hàng trên shoppee
- Tìm video trên youtube

***→ Thuật toán tìm kiếm hiệu quả đối với dữ liệu lớn***

# NỘI DUNG CHÍNH

BÀI TOÁN TÌM KIẾM

TÌM KIẾM TUẦN TỰ

TÌM KIẾM NHỊ PHÂN

# BÀI TOÁN TÌM KIẾM

- Bài toán: Có tồn tại số nguyên ngẫu nhiên x trong tập S các số nguyên đã cho không?
- Input: S là danh sách các số nguyên, x là số nguyên ngẫu nhiên cần tìm.
- Output: Vị trí của x trong S nếu x tồn tại trong danh sách S và -1 nếu không tồn tại.

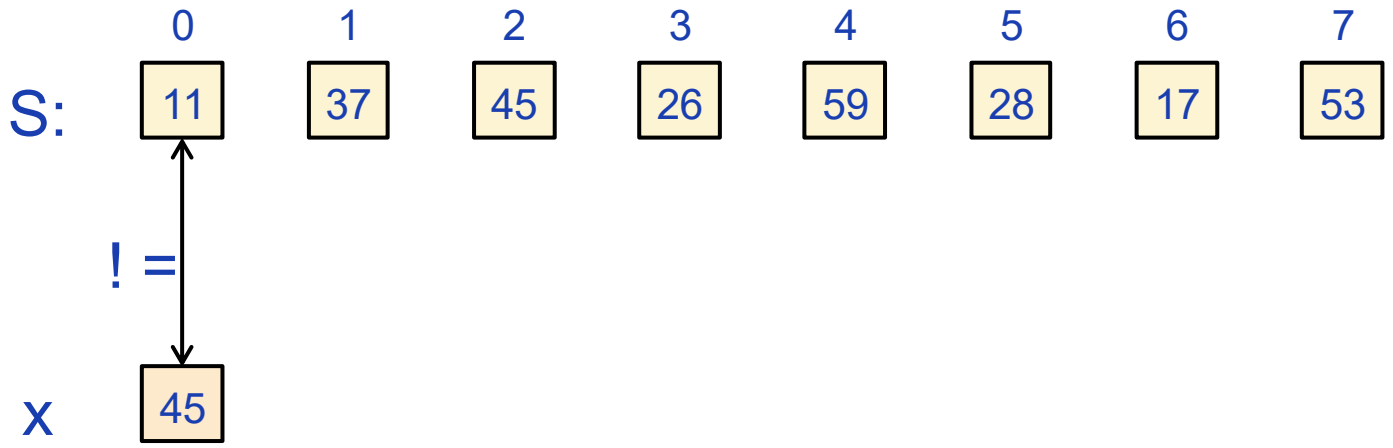
```
1 S = [11, 37, 45, 26, 59, 28, 17, 53]
2 x = 53
3 pos = seq_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
In S, 53 is at position 7.
```

# TÌM KIẾM NHỊ TUẦN TỰ

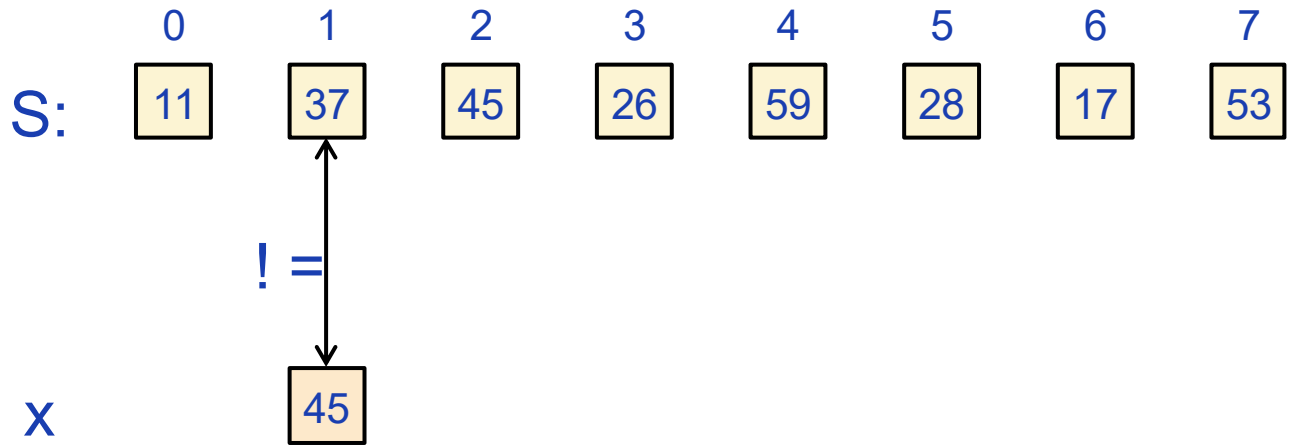
# THUẬT TOÁN TÌM KIẾM TUẦN TỰ

- Thuật toán tìm kiếm tuần tự so sánh giá trị x cần tìm với từng phần tử trong danh sách S đã cho.
- Khi bắt đầu tìm kiếm, nó trả về chỉ số 0 nếu x bằng với phần tử đầu tiên và nếu không nó sẽ thực hiện một so sánh với phần tử tiếp theo.



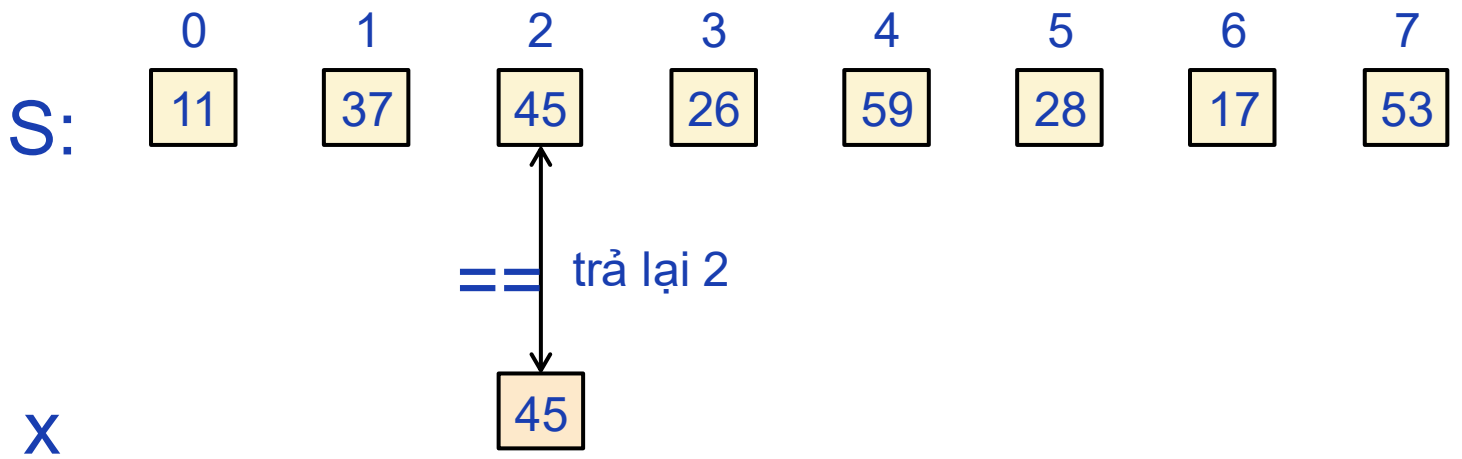
# THUẬT TOÁN TÌM KIẾM TUẦN TỰ

- Nếu x không bằng với phần tử tiếp theo S [1], chuyển sang phần tử tiếp theo.



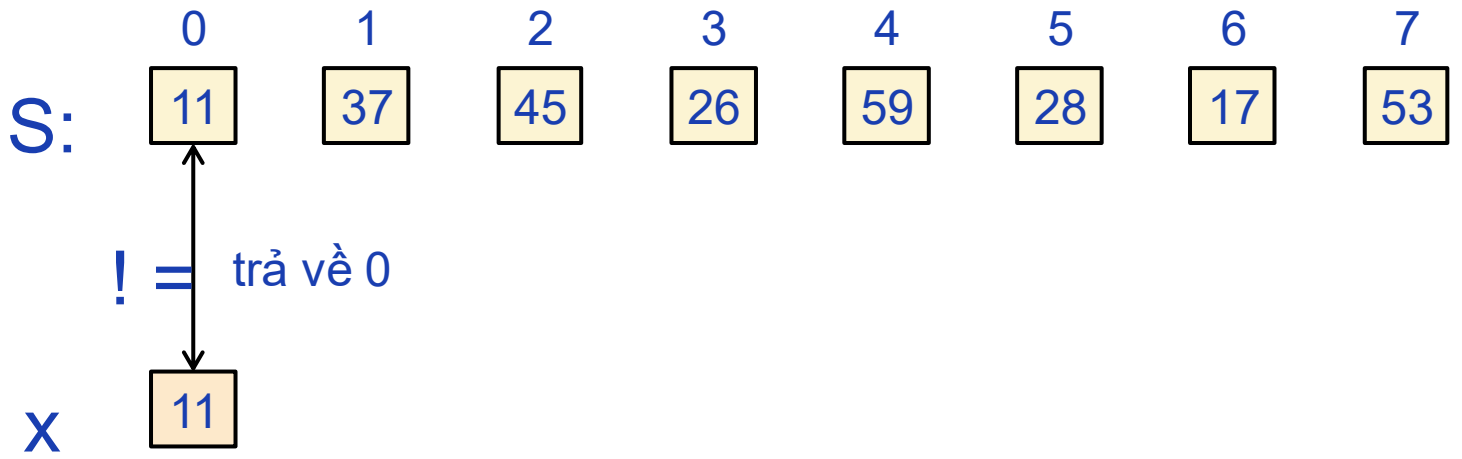
# THUẬT TOÁN TÌM KIẾM TUẦN TỰ

- Giá trị x bằng với phần tử S[2], vì vậy nó trả về 2 và kết thúc quá trình tìm kiếm,



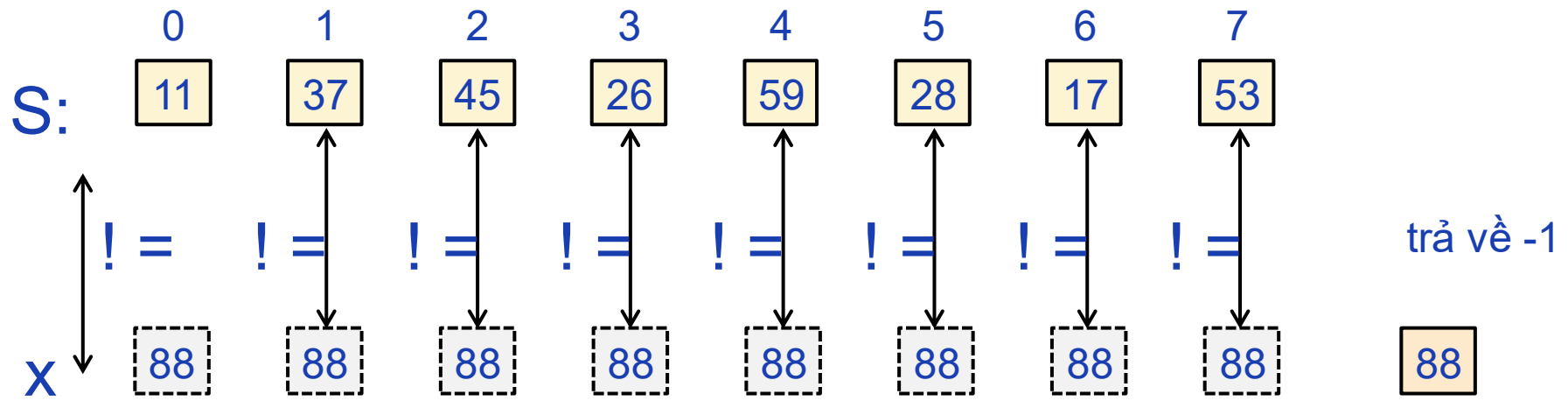
# THUẬT TOÁN TÌM KIẾM TUẦN TỰ

- Trong tìm kiếm tuần tự, nếu phần tử cần tìm là phần tử đầu tiên, thì chỉ cần một phép toán so sánh, và đây được gọi là trường hợp tốt nhất.



# THUẬT TOÁN TÌM KIẾM TUẦN TỰ

- Trong tìm kiếm tuần tự, nếu phần tử cần tìm không tồn tại trong S thì phải so sánh với tất cả các phần tử của S.
- Nếu có n phần tử trong S, thì cần thực hiện tổng cộng n thao tác so sánh. Nó được gọi là 'trường hợp tồi nhất.'



# CÀI ĐẶT THUẬT TOÁN TÌM KIẾM TUẦN TỰ

Hàm seq_num nhận nums và x làm đầu vào và thực hiện tìm kiếm tuần tự  phần tử x trong nums .

```
1 def seq_search(nums, x):
2     for i in range(len(nums)):
3         if x == nums[i]:
4             return i
5     return -1
```

Dòng 2-5

- Dòng 2 là vòng lặp for để tìm kiếm tuần tự các phần tử trong nums .
- Dòng 3 và 4 trả về ' i ' nếu nums [i] và x có giá trị bằng nhau.
- Dòng 5 trả về -1 bởi vì x không nằm trong nums.

# CÀI ĐẶT THUẬT TOÁN TÌM KIẾM TUẦN TỰ

## Sử dụng hàm seq_search ()

```
1 S = [11, 37, 45, 26, 59, 28, 17, 53]
2 x = 11
3 pos = seq_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
In S, 11 is at position 0.
```

```
1 S = [11, 37, 45, 26, 59, 28, 17, 53]
2 x = 77
3 pos = seq_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
In S, 77 is at position -1.
```

# ĐỘ PHỨC TẠP VỀ THỜI GIAN

- Độ phức tạp về thời gian của thuật toán tìm kiếm tuần tự là gì? Phân tích với ký hiệu Big-O.
- Đầu tiên, trong trường hợp tốt nhất, việc tìm kiếm được kết thúc sau một phép so sánh duy nhất, vì vậy độ phức tạp về mặt thời gian của thuật toán là O (1). Với trường hợp tồi nhất, cần phải so sánh n lần, vì vậy độ phức tạp là O(n).
- **Độ phức tạp thời gian trung bình của tìm kiếm tuần tự được phân tích như thế nào?**
- Giả sử rằng S có N phần tử. Khi tìm kiếm lần lượt từng phần tử của S, hãy tính xem cần thực hiện bao nhiêu phép toán so sánh (==).
- Giá trị x có thể nằm ở vị trí 1, 2, 3, … Nếu x nằm ở vị trí đầu tiên thì mất 1 phép so sánh, x nằm ở vị trí thứ 2 mất 2 phép so sánh, … x nằm ở vị trí N, cần thực hiện N lần phép so sánh. Do đó, tổng số lượng thao tác so sánh cần thiết cho N trường hợp là :

$$1 + 2 + \cdots + N = \frac{N(N+1)}{2}$$

- Vì vậy, thời gian tìm kiếm trung bình T (n) là tuyến tính.

$$T(N) = \frac{N(N+1)}{2} \div N = \frac{N+1}{2} \in O(n)$$

# Quizz. # 1

Hàm sau trả lại kết quả là gì. Phân tích mã và mô tả hành vi của thuật toán.

```python
def find_two(nums):
    x = y = 0
    for i in range(1, len(nums)):
        if nums[x] < nums[i]:
            x = i
        elif nums[y] > nums[i]:
            y = i
    return x, y
```

```python
nums = [11, 37, 45, 26, 59, 28, 17, 53]
i, j = find_two(nums)
print(nums[i], nums[j])
```

# Quizz. # 1

Sau đây là mã cho trò chơi 'đoán số'. Nếu tối đa = 100 và number = 51, có bao nhiêu số đếm sẽ được in ra khi chạy chương trình sau đây?

```
1  from random import randint
2
3  maximum = int(input("Enter the number of maximum: "))
4  number = int(input("Enter your guessing number: "))
5  count = 0
6  low, high = 1, maximum
7  while low < high:
8      mid = (low + high) // 2
9      count += 1
10     if mid == number:
11         print(f"Your number is {number}.")
12         break
13     elif mid > number:
14         high = mid - 1
15     else:
16         low = mid + 1
17 print(f"Total {count} times are searched.")
```

# VÍ DỤ - **TÌM SỐ LỚN NHẤT**

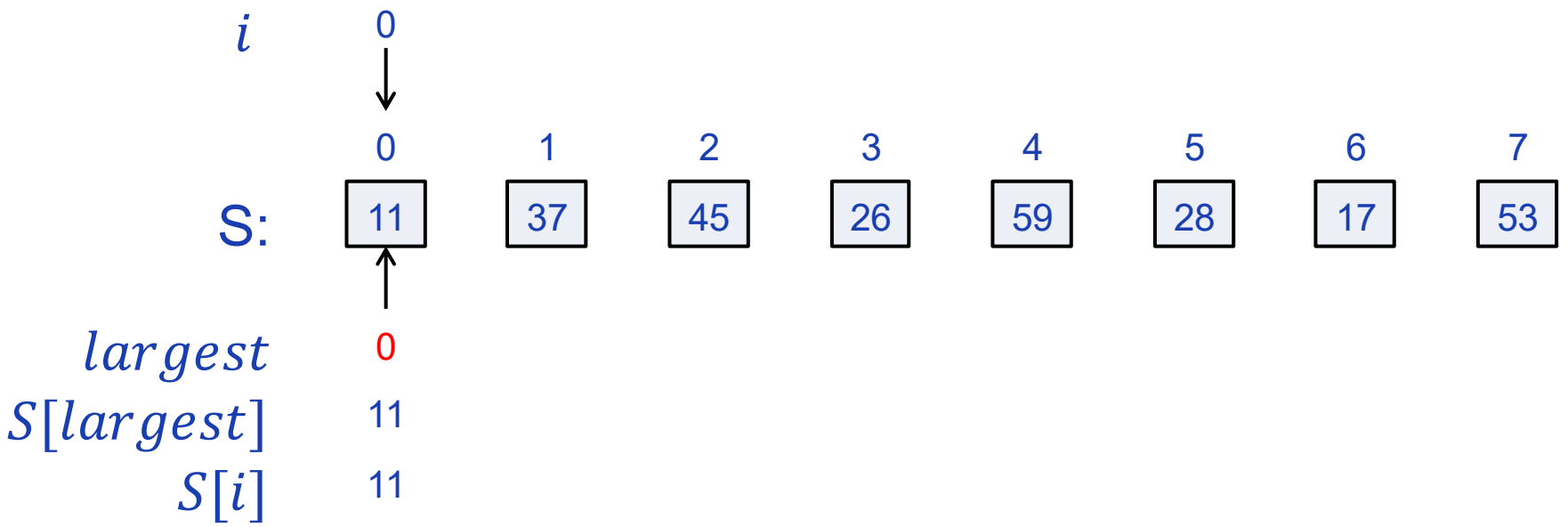
- **Bài toán:** Đưa ra số lớn nhất trong một tập hợp S các số nguyên đã cho?
- **Input:** S là danh sách các số nguyên.
- **Output:** Chỉ số của phần tử lớn nhất trong S.

```
1 nums = [11, 37, 45, 26, 59, 28, 17, 53]
2 pos = find_largest(nums)
3 print(f"The largest is {nums[pos]} at {pos}")
```

```
[11, 37, 45, 26, 59, 28, 17, 53]
The largest is 59 at 4
```

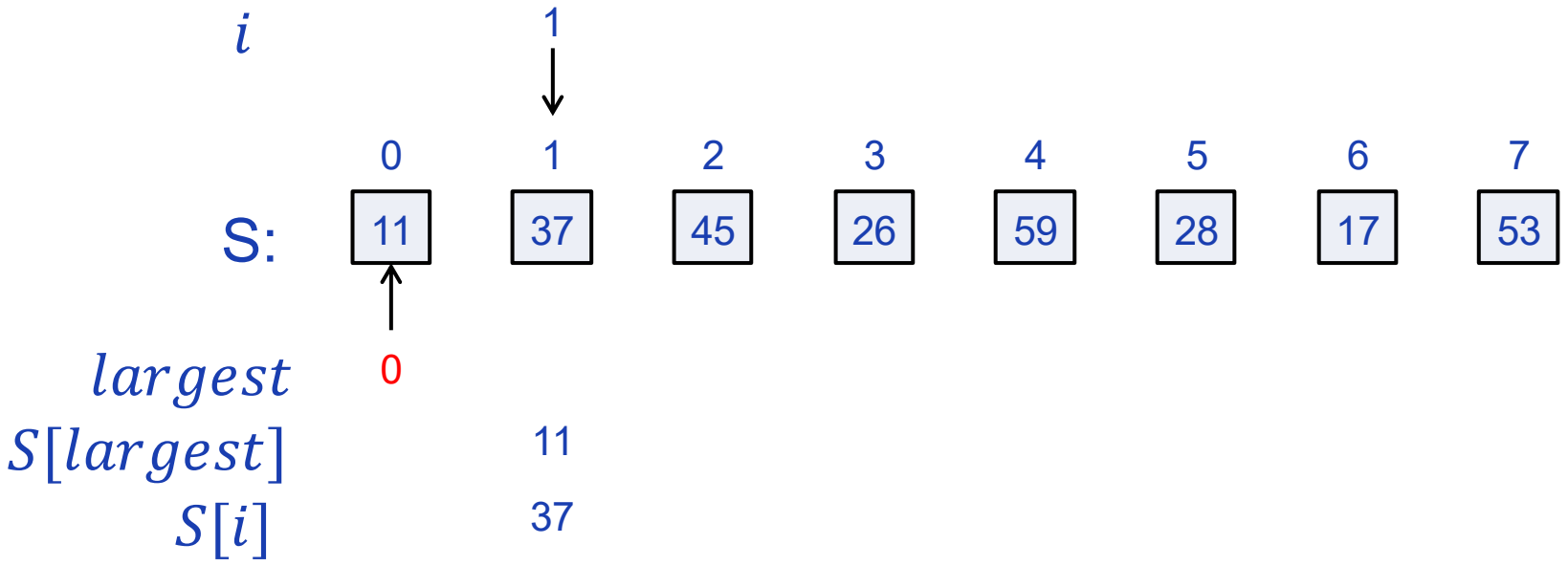
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Thuật toán tìm số lớn nhất có thể được thực hiện theo cách tương tự như thuật toán tìm kiếm tuần tự,
- Đầu tiên, giả sử rằng vị trí của số lớn nhất là phần tử đầu tiên của S.



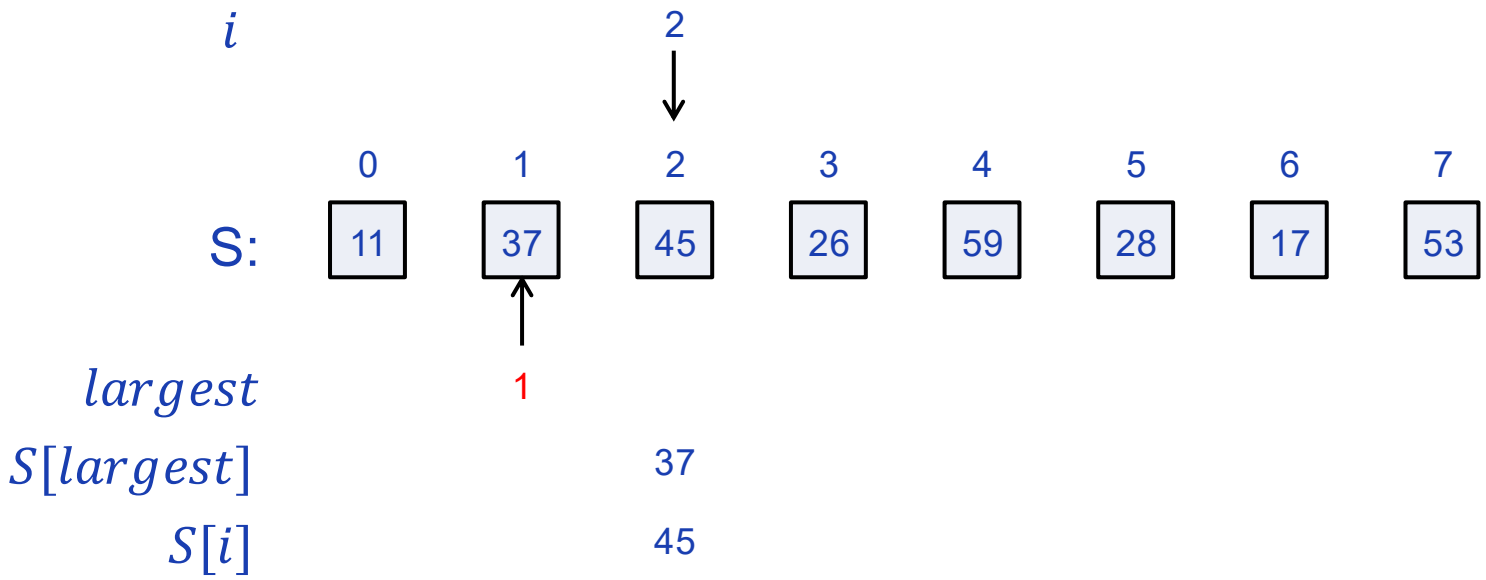
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Khi so sánh S [1] với S [largest] là phần tử lớn nhất hiện tại.
- Vì S [1] lớn hơn S [0], vị trí của số lớn nhất tạm thời (largest) được gán bằng 1.



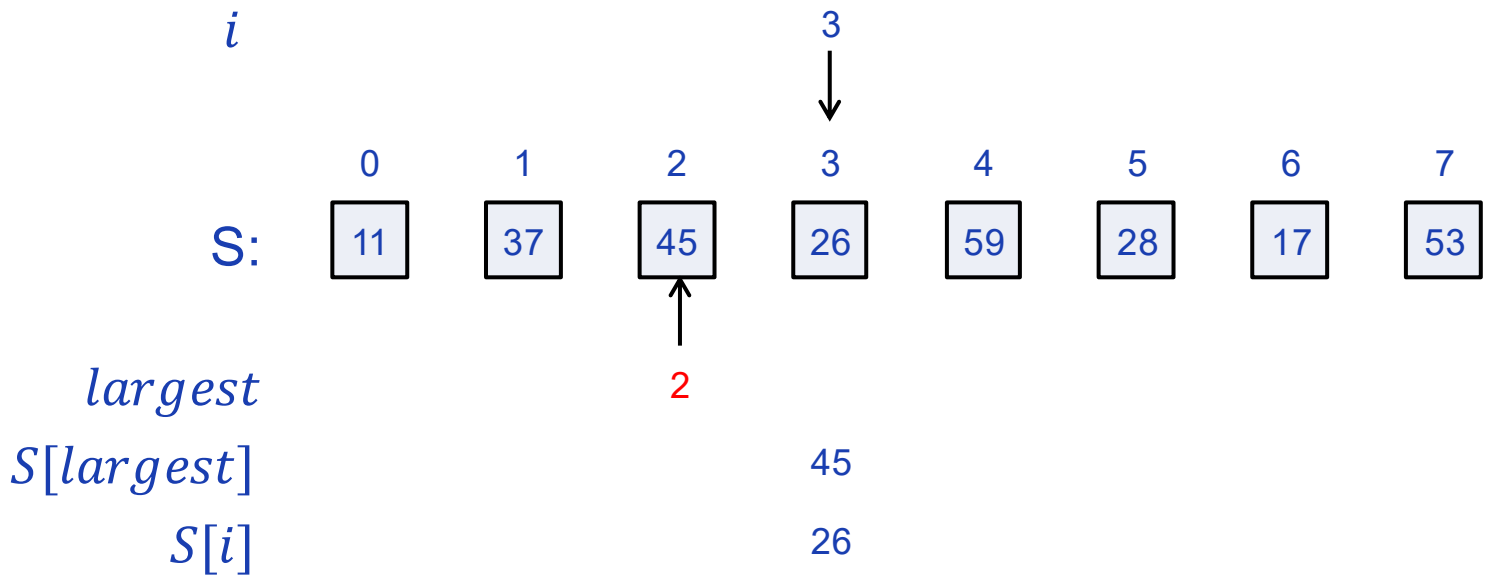
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Khi so sánh S [2] với S [largest], vì S [2] lớn hơn S [1], vị trí của số lớn nhất (largest) được gán bằng 2.



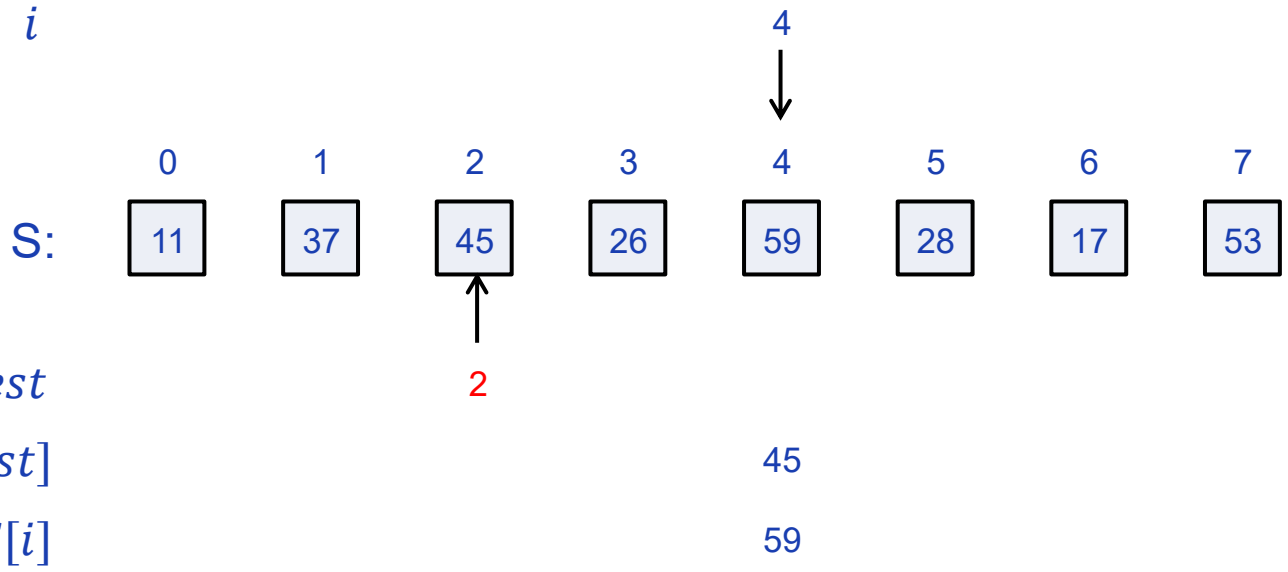
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Khi so sánh S [3] với S [largest] lớn nhất hiện tại, vì S [3] nhỏ hơn S [2] nên vị trí của số lớn nhất (largest) không thay đổi.



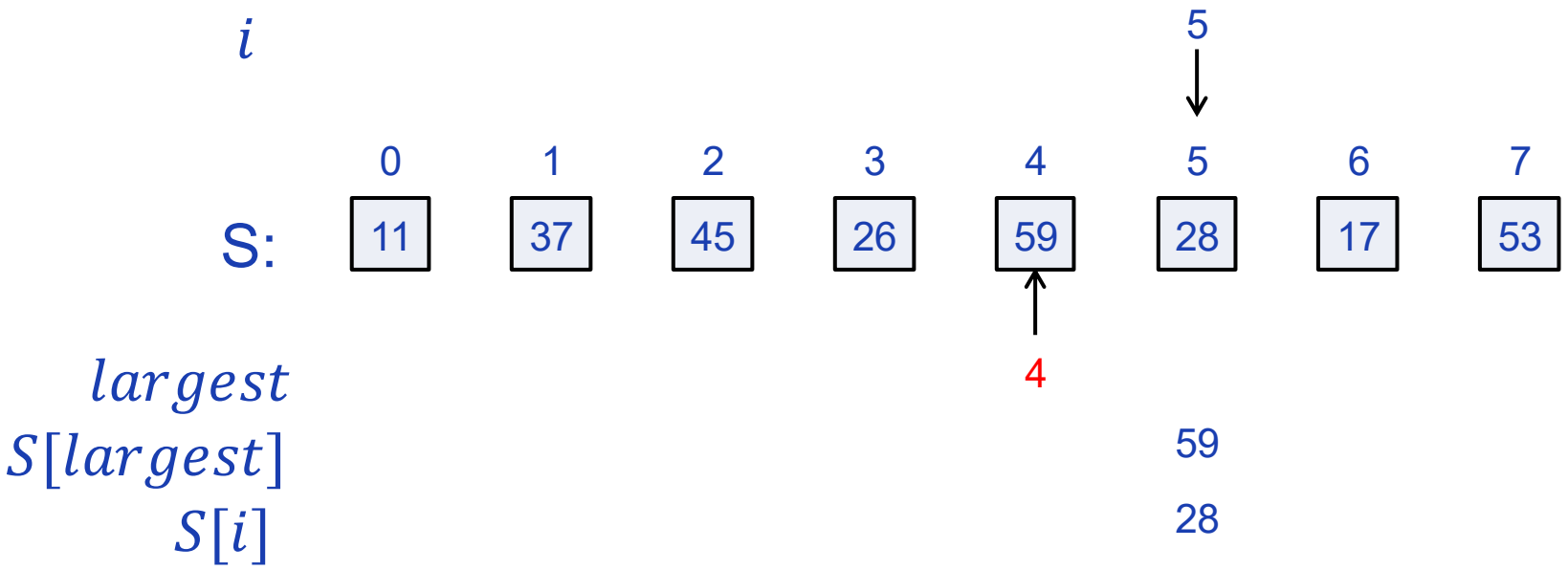
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Khi so sánh S [4] với S [largest] lớn nhất hiện tại, vì S [4] lớn hơn S [2], vị trí của số lớn nhất (largest) được gán bằng 4.



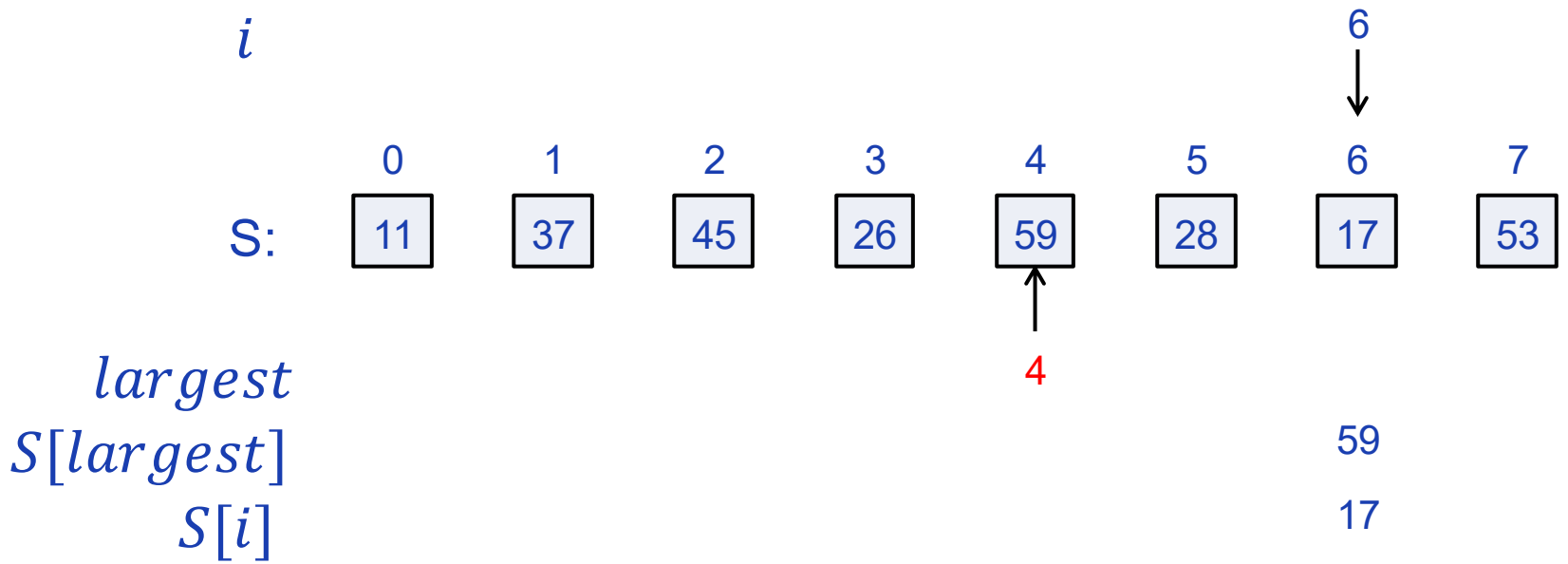
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Nếu không có giá trị nào lớn hơn S [largest] thì vị trí của số lớn nhất được giữ nguyên.



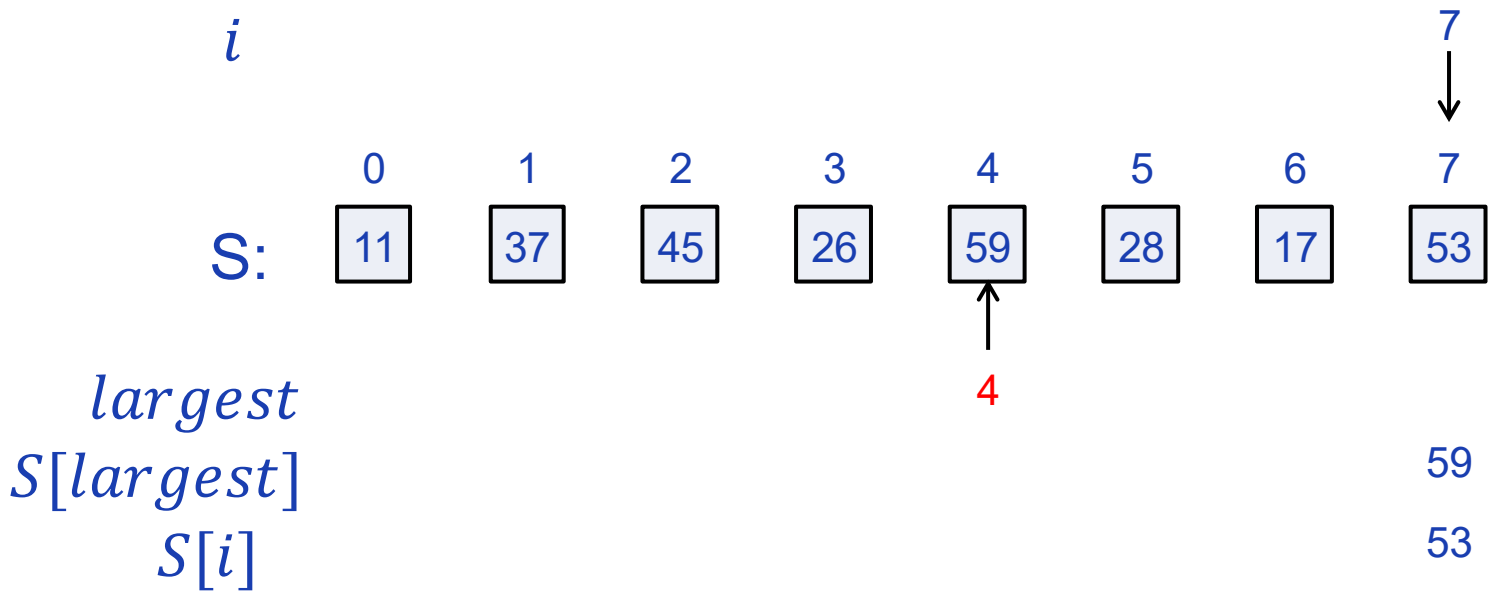
# VÍ DỤ - **TÌM SỐ LỚN NHẤT**

- Nếu không có giá trị nào lớn hơn S [largest] thì vị trí của số lớn nhất được giữ nguyên.



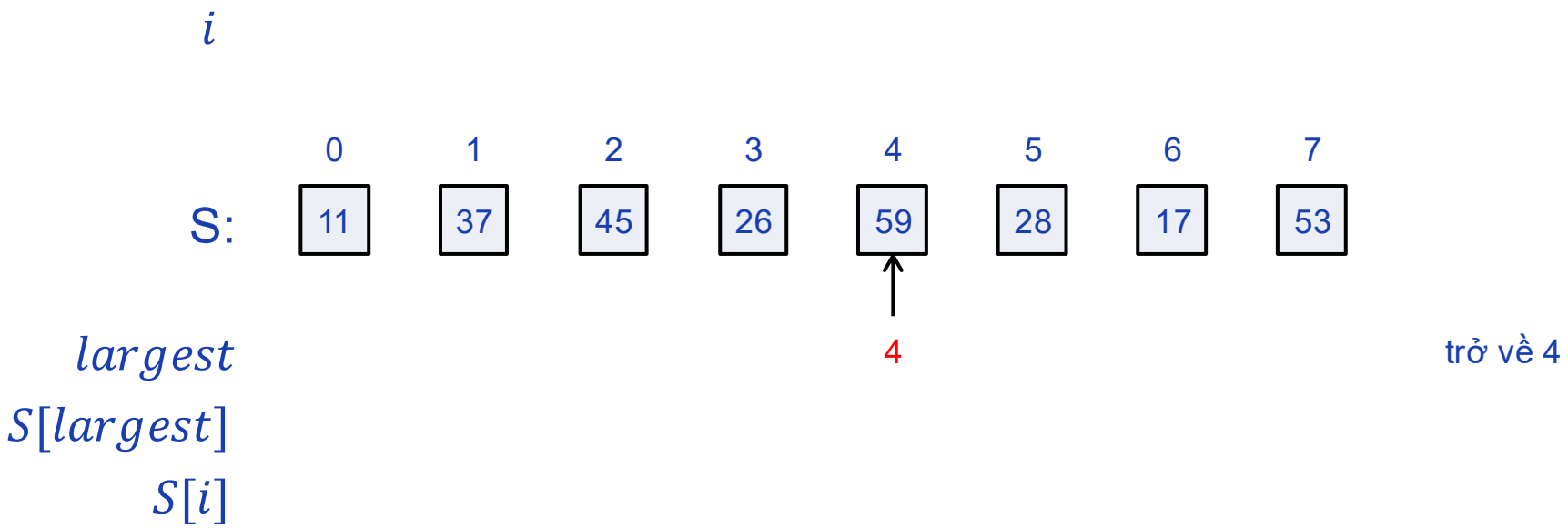
# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Nếu không có giá trị nào lớn hơn S [largest] thì vị trí của số lớn nhất được giữ nguyên.



# VÍ DỤ - TÌM SỐ LỚN NHẤT

- Sau khi duyệt qua tất cả các phần tử, largest là chỉ số của phần tử lớn nhất trong dãy



# VÍ DỤ - TÌM SỐ LỚN NHẤT

Áp dụng chiến lược đã sử dụng trước đó để cài đặt hàm find_largest () như sau.

```
1  def find_largest(nums):
2      largest = 0
3      for i in range(1, len(nums)):
4          if nums[largest] < nums[i]:
5              largest = i
6      return largest
```

Dòng 2-5

- Khởi tạo chỉ mục của phần tử đầu tiên trong danh sách nums thành giá trị lớn nhất.
- So sánh giá trị nums [largest] hiện tại và giá trị nums [ i ] từ phần tử thứ 2 tới phần tử cuối cùng của danh sách nums .
- Nếu giá trị phần tử thứ ' i ' lớn hơn giá trị phần tử lớn nhất tạm thời, thì cập nhật chỉ số của phần tử lớn nhất (largest) thành ' i '.
- Trả lại chỉ mục của phần tử lớn nhất trong danh sách nums.

**Bài tập *Thực hành 9.1. Thực hành tìm kiếm tuần tự***

Thời gian: 15 phút

Từ : 20h30 đến: 20h45

# TÌM KIẾM NHỊ PHÂN

# VÍ DỤ – TÌM KIẾM NHỊ PHÂN

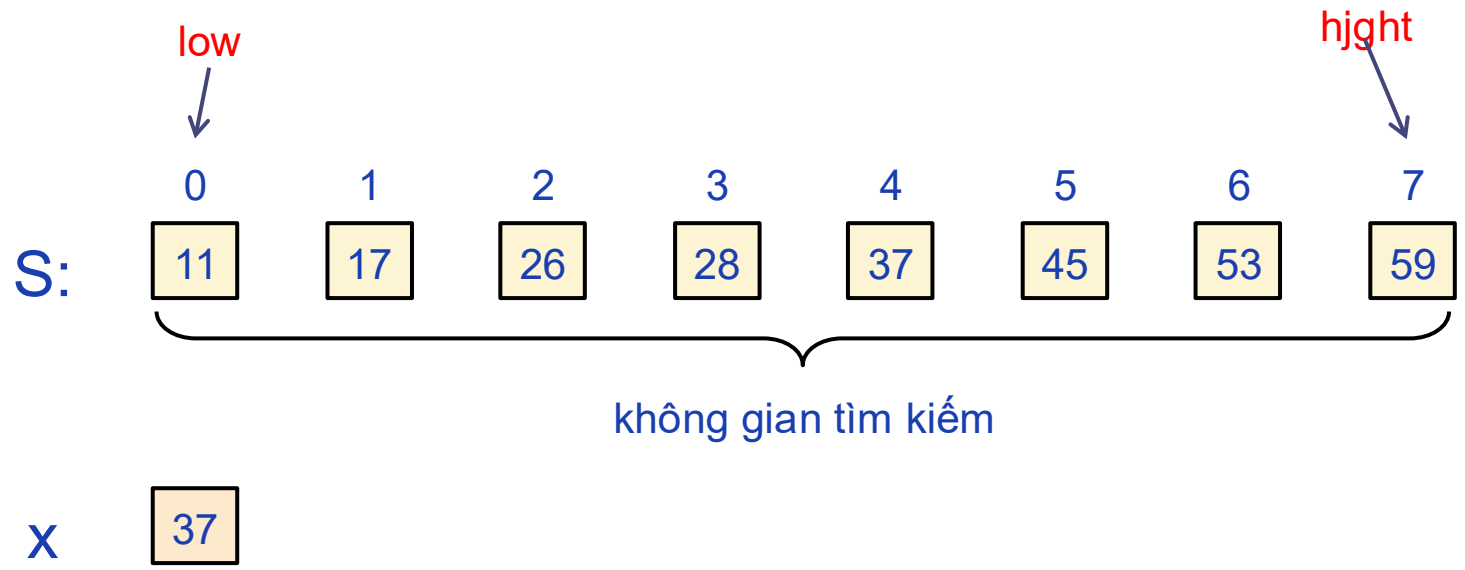
- Bài toán: Có tồn tại một số nguyên ngẫu nhiên x trong S, một tập hợp các số nguyên có thứ tự không?
- Dữ liệu vào: S là danh sách các số nguyên được sắp xếp theo thứ tự tăng dần, x là số nguyên ngẫu nhiên cần tìm
- Dữ liệu ra: Vị trí chỉ mục nếu x tồn tại trong danh sách S và -1 nếu không tồn tại.

```
1 S = [11, 17, 26, 28, 37, 45, 53, 59]
2 x = int(input("Input the number to search: "))
3 pos = bin_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
Input the number to search: 53
In S, 53 is at position 6.
```

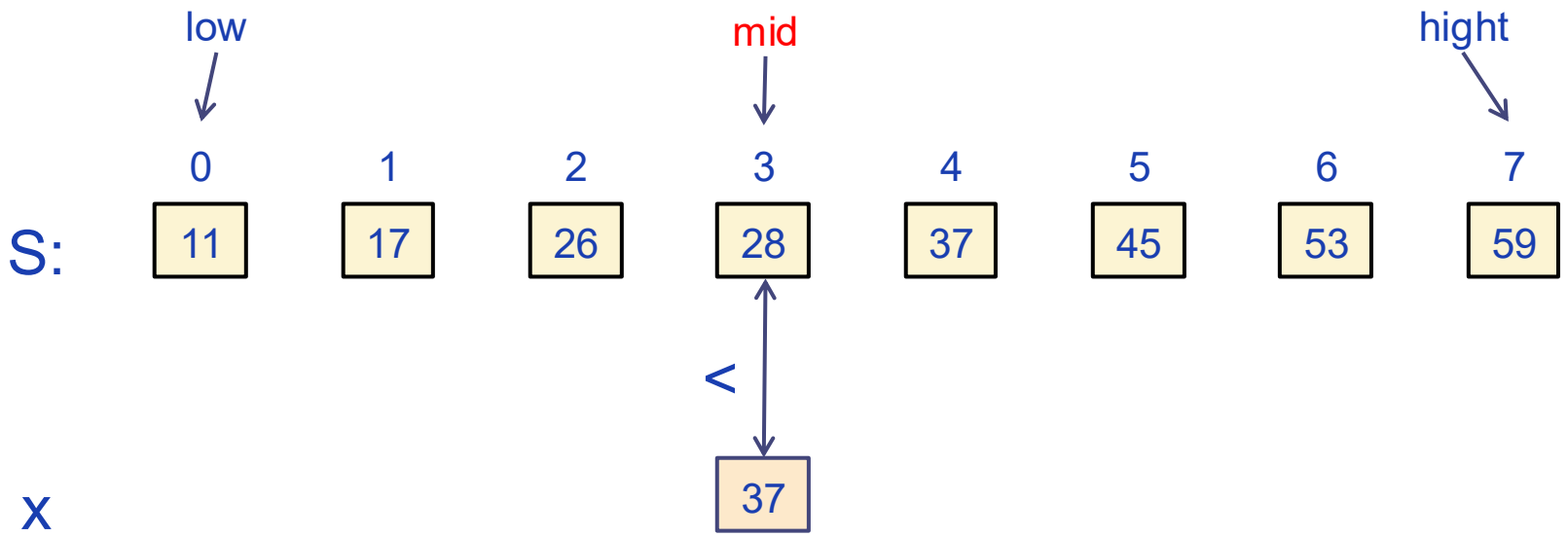
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Thuật toán tìm kiếm nhị phân tìm kiếm giá trị x trong danh sách S đã cho.
- Không gian tìm kiếm ban đầu nằm giữa low và high (thấp là chỉ mục phần tử đầu tiên và cao là chỉ mục phần tử cuối cùng) trong S.



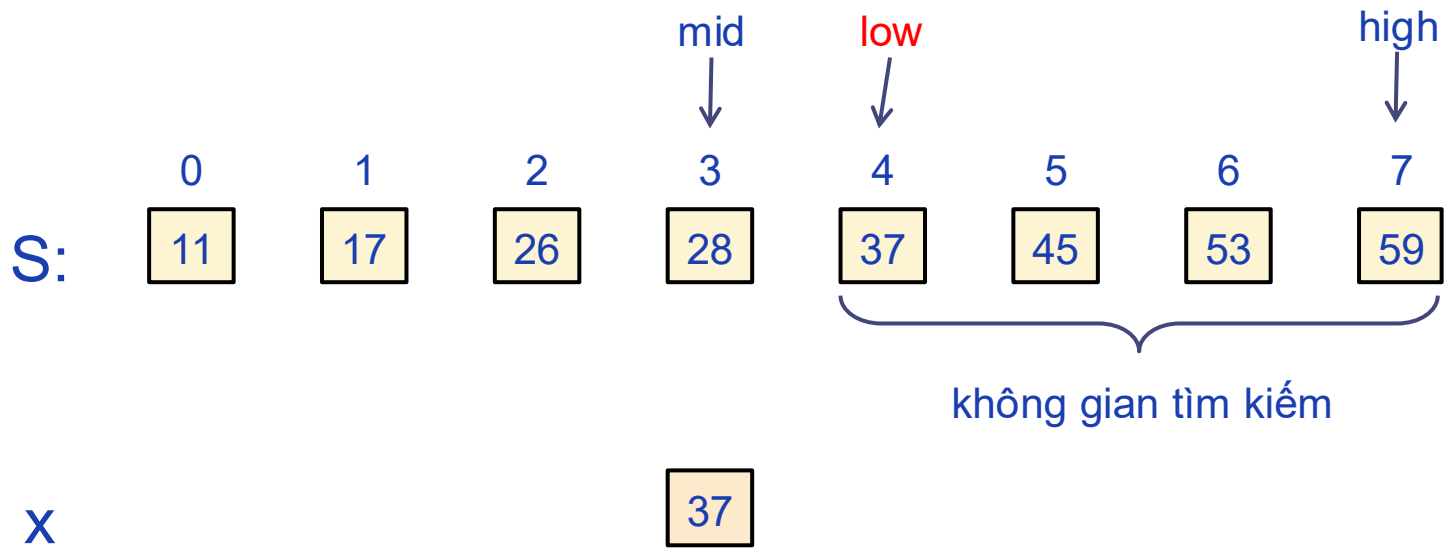
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Trong thuật toán tìm kiếm nhị phân, danh sách đầu vào được sắp xếp theo thứ tự tăng dần.
- So sánh giá trị x với giá trị trung bình ở giữa low và high.



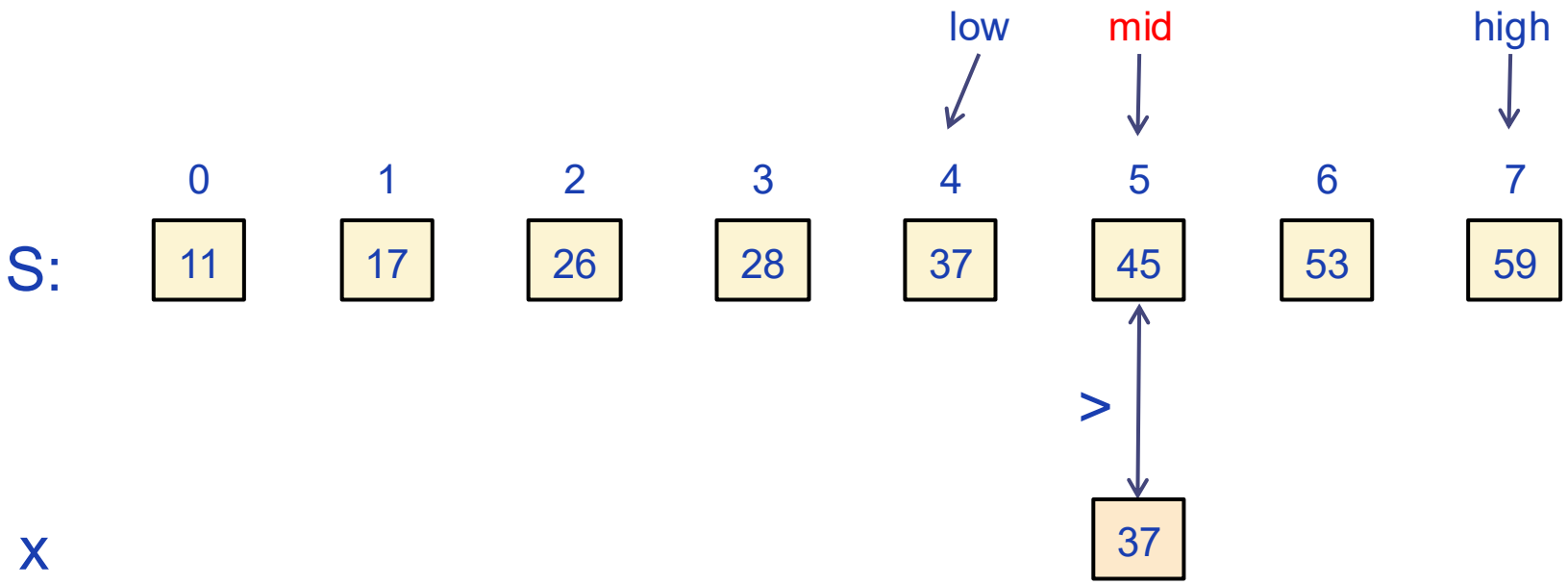
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Nếu S [giữa] nhỏ hơn x, x cần tìm sẽ nằm ở mid+ 1 và high.
- Do đó, low được thay đổi thành mid + 1, làm giảm một nửa không gian tìm kiếm từ [0, 7] thành [4, 7].



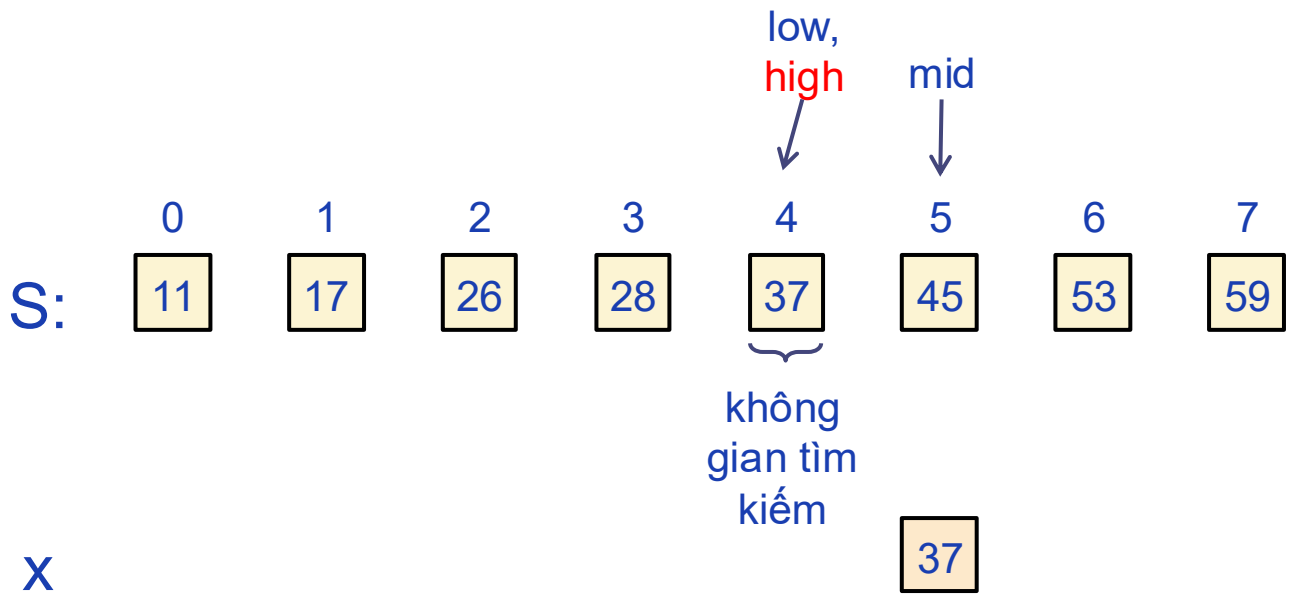
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- S [mid] lớn hơn x, vì vậy high được đổi thành mid-1.



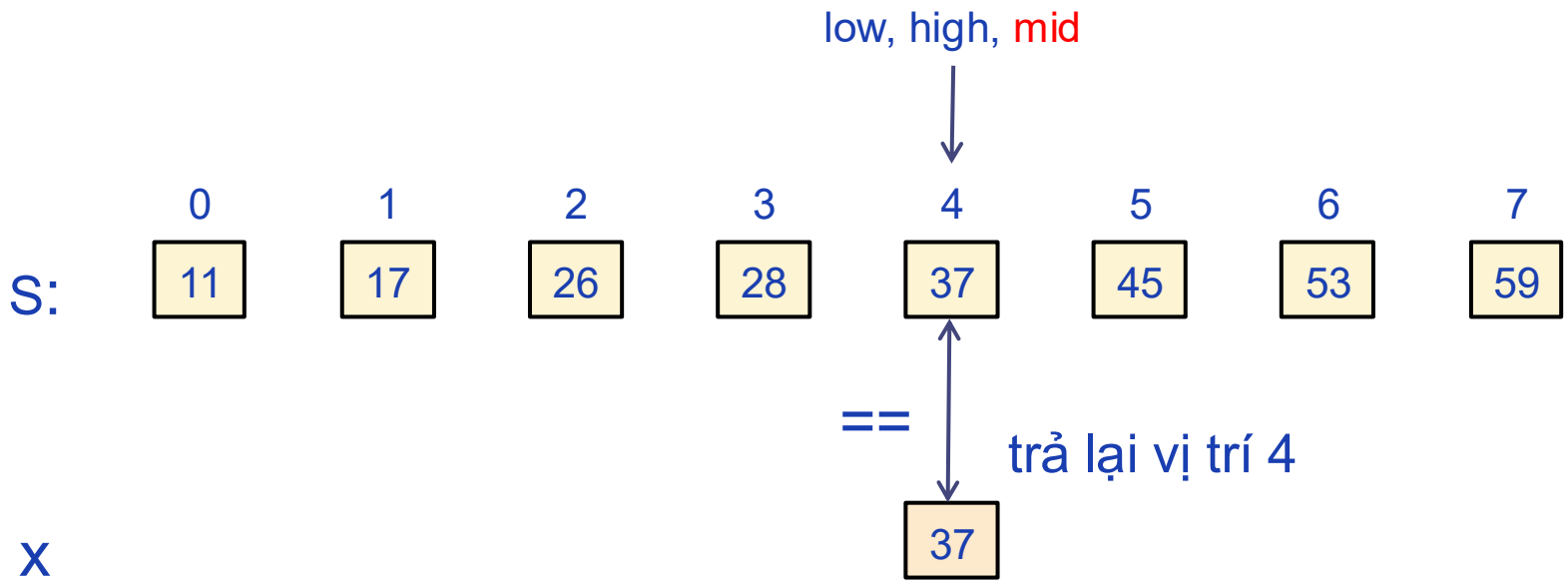
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Không gian tìm kiếm được giảm từ [4, 7] xuống [4, 4].



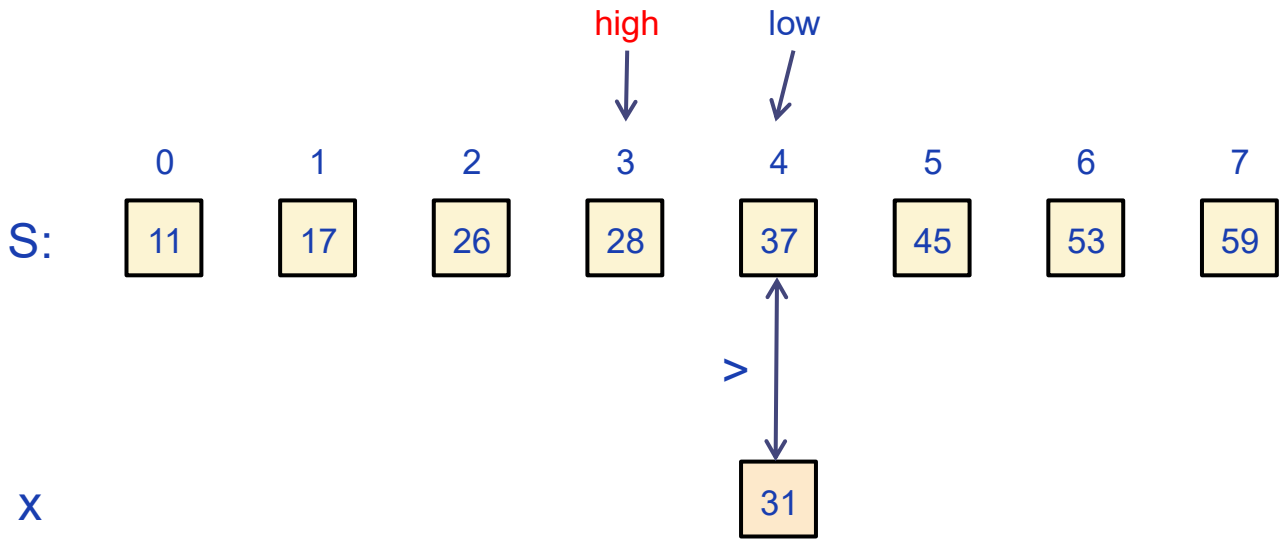
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- S [mid] == x, trả về giá trị giữa tại vị trí hiện tại.



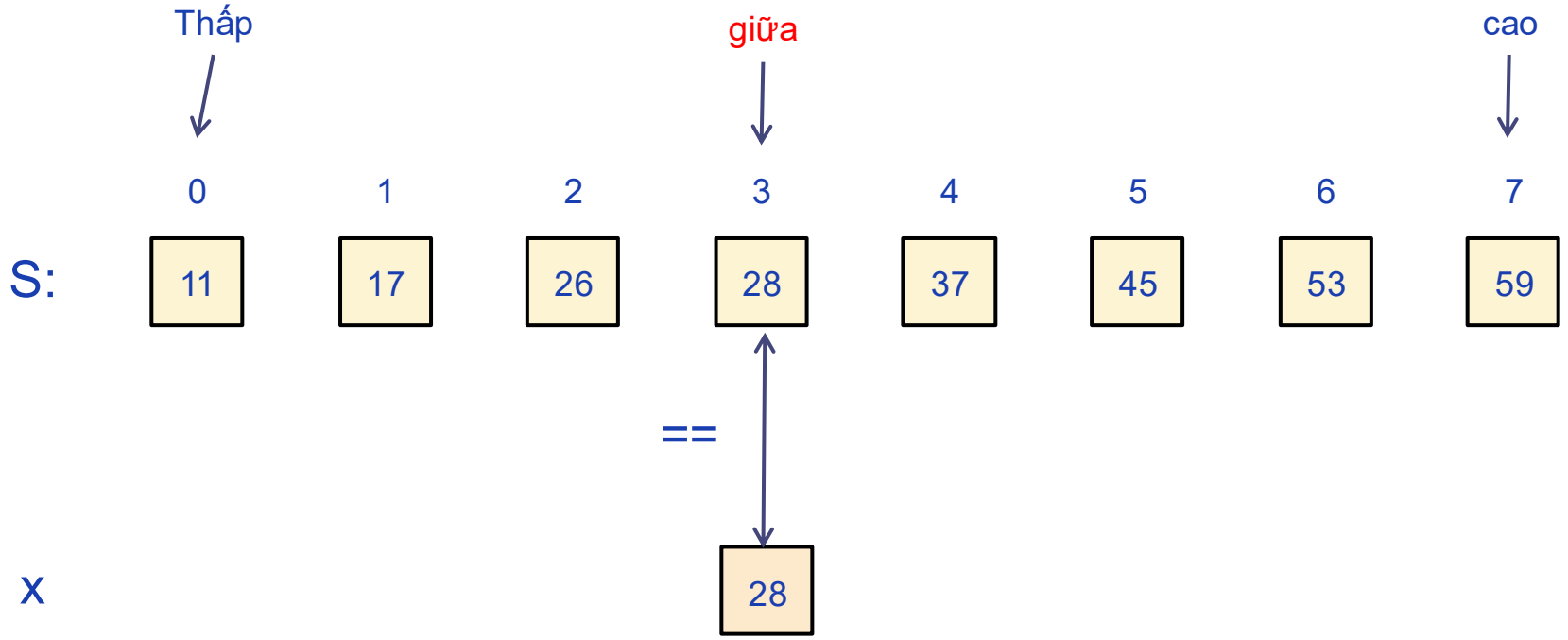
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Nếu giá trị cần tìm không tồn tại, chẳng hạn 31, vì x nhỏ hơn S [mid], giá trị high sẽ thay đổi thành mid - 1.
- Điều kiện low < high không thỏa mãn, vì vậy dừng vòng while và trả về -1.



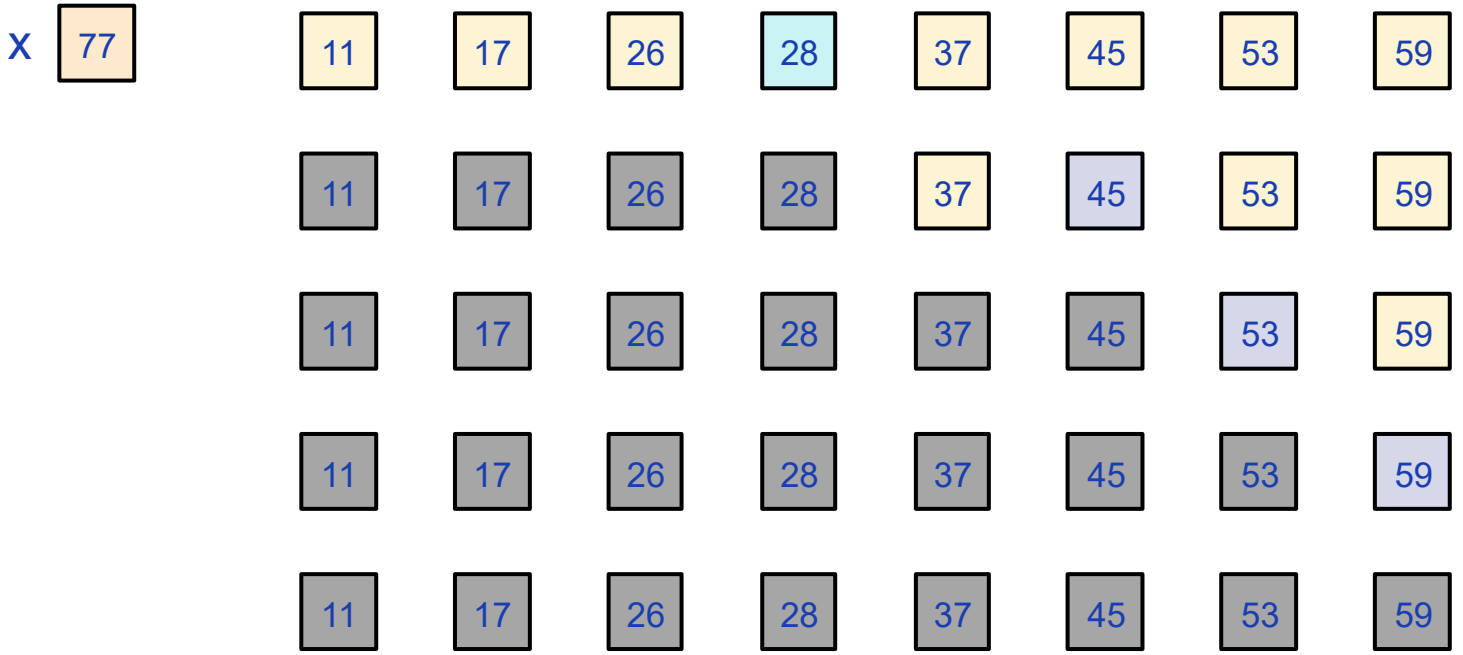
# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Trường hợp tốt nhất trong tìm kiếm nhị phân là gì? Khi bạn may mắn tìm kiếm giá trị giữa trong chữ S, bạn chỉ có thể thực hiện một phép so sánh để tìm ra phần tử.



# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

- Trường hợp xấu nhất trong tìm kiếm nhị phân là khi tìm kiếm một phần tử không tồn tại trong S.
- Trong ví dụ sau, cần thực hiện tổng cộng 4 phép toán so sánh để tìm kiếm 77.



# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

Hàm bin_search(nums, x) thực hiện thuật toán tìm kiếm nhị phân tìm kiếm phần tử x trong dãy nums.

```
1  def bin_search(nums, x):
2      low, high = 0, len(nums) - 1
3      while low <= high:
4          mid = (low + high) // 2
5          if nums[mid] == x:
6              return mid
7          elif nums[mid] > x:
8              high = mid - 1
9          else:
10             low = mid + 1
11     return -1
```

Dòng 2

- 'Low' và 'high' được khởi tạo lần lượt là phần tử đầu tiên và cuối cùng của nums .
- Trong phần bắt đầu của tìm kiếm nhị phân, toàn bộ số là không gian tìm kiếm. Do đó, không gian tìm kiếm là [0, len ( nums ) -1].

# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

Tìm kiếm nhị phân so sánh giá trị nằm giữa low và high.

```
1  def bin_search(nums, x):
2      low, high = 0, len(nums) - 1
3      while low <= high:
4          mid = (low + high) // 2
5          if nums[mid] == x:
6              return mid
7          elif nums[mid] > x:
8              high = mid - 1
9          else:
10             low = mid + 1
11     return -1
```

Dòng 3-10

- Giá trị giữa được tính là (low + high) // 2. Ở đây, toán tử // trả về thương số của phép chia.
- Nếu giá trị nums [mid] giống như x, tìm kiếm thành công nên trả về giá trị giữa.
- Nếu giá trị nums [mid] lớn hơn x, hãy thay đổi 'cao' thành giữa 1 và nếu nó nhỏ hơn x, hãy thay đổi 'thấp' thành giữa + 1.
- Nếu thấp hoặc giá trị cao thay đổi thành giữa + 1 hoặc giữa - 1, thì không gian tìm kiếm giảm đi một nửa.

# THUẬT TOÁN TÌM KIẾM NHỊ PHÂN

Trả về -1 nếu không tìm thấy x trong quá trình tìm kiếm nhị phân.

```
1  def bin_search(nums, x):
2      low, high = 0, len(nums) - 1
3      while low <= high:
4          mid = (low + high) // 2
5          if nums[mid] == x:
6              return mid
7          elif nums[mid] > x:
8              high = mid - 1
9          else:
10             low = mid + 1
11     return -1
```

Dòng 11

- Nếu giá trị thấp lớn hơn giá trị cao thì ngắt câu lệnh while.
- Nếu vậy, trả về -1 vì giá trị x không tồn tại trong S.

# ĐỘ PHỨC TẠP THỜI GIAN

- Độ phức tạp về thời gian của tìm kiếm nhị phân là gì? Đầu tiên, trường hợp tốt nhất có thể kết thúc tìm kiếm chỉ một lần, vì vậy nó là O (1). Sau đó, bạn sẽ phân tích độ phức tạp thời gian trong trường hợp xấu nhất của tìm kiếm nhị phân như thế nào?
- Giả sử rằng S có N phần tử. Kích thước của không gian tìm kiếm đầu tiên là N. Không gian thứ hai để tìm kiếm bằng một nửa của không gian ban đầu, do đó nó là N / 2. Không gian tìm kiếm tiếp theo lại bị giảm một nửa, là N / 4. Sự giải thích này có thể được khái quát như sau.

$$N, \frac{N}{2}, \frac{N}{2^2}, \cdots, \frac{N}{2^k}$$

- Tìm kiếm nhị phân kết thúc nếu chỉ có một không gian tìm kiếm, đó là N / $2^k$ = 1.

$$N = 2^k$$

$$\log_2 N = k$$

- Tối đa. số hoạt động tìm kiếm nhị phân là $\log_2 N$ , do đó độ phức tạp thời gian trong trường hợp xấu nhất của tìm kiếm nhị phân là O ($\log_2 N$ ).

# ĐỘ PHỨC TẠP THỜI GIAN

Thêm hàm *print()* vào *bin_search()* để kiểm tra độ phức tạp về thời gian của tìm kiếm nhị phân.

```
1  def bin_search(nums, x):
2      low, high = 0, len(nums) - 1
3      while low <= high:
4          mid = (low + high) // 2
5          print(low, high, mid)
6          if nums[mid] == x:
7              return mid
8          elif nums[mid] > x:
9              high = mid - 1
10         else:
11             low = mid + 1
12     return -1
```

Dòng 5

- In giá trị thấp, cao và trung bình trong câu lệnh vòng lặp để đếm số lần lặp lại tìm kiếm nhị phân.

# ĐỘ PHỨC TẠP THỜI GIAN

## Trường hợp tốt nhất: tìm kiếm phần tử 28, 1 phép so sánh

```
1 S = [11, 17, 26, 28, 37, 45, 53, 59]
2 x = int(input("Input the number to search: "))
3 pos = bin_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
Input the number to search: 28
0 7 3
In S, 28 is at position 3.
```

## Trường hợp xấu nhất: tìm kiếm phần tử 77, 4 phép so sánh

```
1 S = [11, 17, 26, 28, 37, 45, 53, 59]
2 x = int(input("Input the number to search: "))
3 pos = bin_search(S, x)
4 print(f"In S, {x} is at position {pos}.")
```

```
Input the number to search: 77
0 7 3
4 7 5
6 7 6
7 7 7
In S, 77 is at position -1.
```

# Quizz. # 1

Sau đây là mã cho trò chơi 'đoán số'. Nếu tối đa = 100 và number = 51, có bao nhiêu số đếm sẽ được in ra?

```
from random import randint

maximum = int(input("Enter the number of maximum: "))
number = int(input("Enter your guessing number: "))
count = 0
low, high = 1, maximum
while low < high:
    mid = (low + high) // 2
    count += 1
    if mid == number:
        print(f"Your number is {number}.")
        break
    elif mid > number:
        high = mid - 1
    else:
        low = mid + 1
print(f"Total {count} times are searched.")
```

**Bài tập *Thực hành 9.2. Thực hành tìm kiếm nhị phân***

Thời gian: 15 phút

Từ : 21h15 đến: 21h45

# CÂU HỎI