19. 二分搜尋法

2025資訊研究社 算法班
Made with ❤️ by jheanlee

什麼是二分搜尋法

現在給你一本有未知頁數的字典，要你翻到第345頁。你會怎麼做?

作法一: 從第一頁開始翻，翻344次
作法二: 從最中間開始翻，頁碼較大就往後，較小就往前

作法二是平均下來效率較高的作法

什麼是二分搜尋法

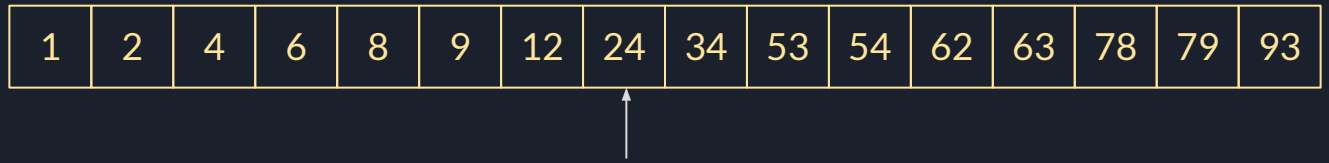
二分搜尋法用於**已排序**的資料

將中間值與目標比大小，向較接近的方向移動

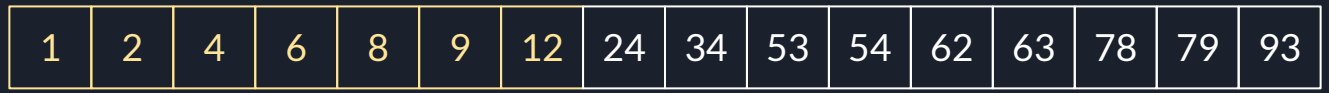
目標: 12

1	2	4	6	8	9	12	24	34	53	54	62	63	78	79	93

目標: 12

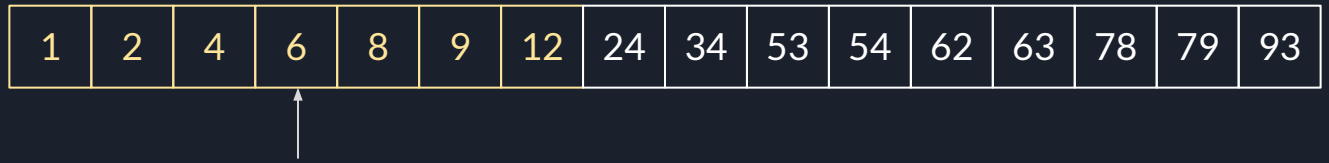


24 > 12
12不可能在右邊

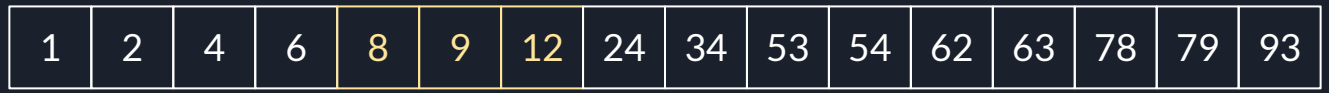


黃色為目前目標可能範圍

目標: 12

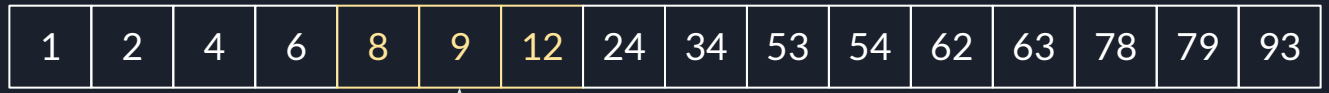


6 < 12
6不可能在左邊

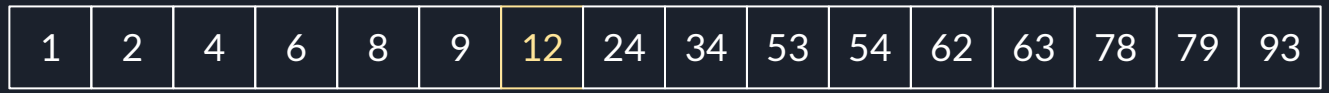


黃色為目前目標可能範圍

目標: 12



9 < 12
12不可能在左邊

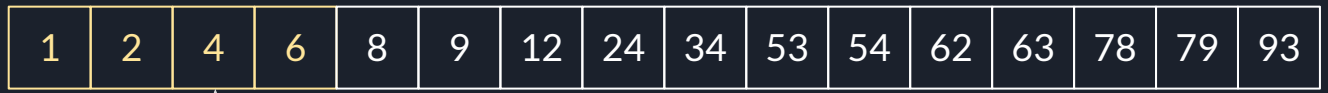


黃色為目前目標可能範圍

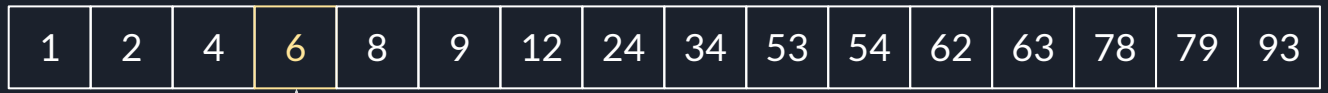

1
2
4
6
8
9
12
24
34
53
54
62
63
78
79
93
12 = 12
得出位置

Q: 如果目標不存在呢

目標: 5



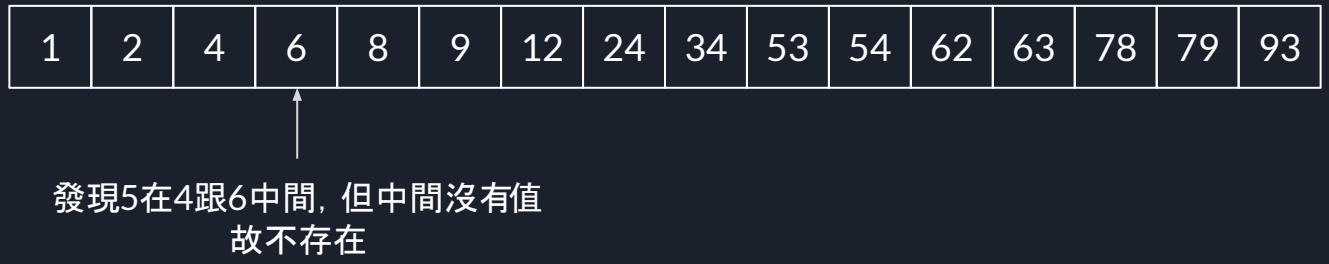
4 < 5
5不可能在左邊



6 > 5
5不可能在右邊

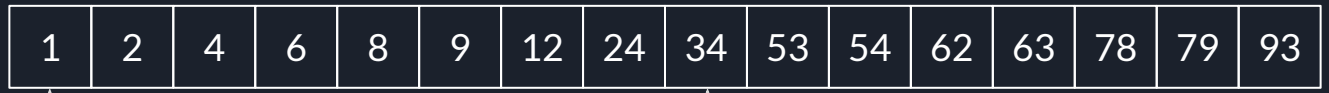
黃色為目前目標可能範圍

Q: 如果目標不存在呢



二分搜尋實作

左閉右開寫法
[l, r)



可能範圍左邊邊界(含)
left

可能範圍中間值
middle
middle = (left + right) / 2

可能範圍右邊邊界(不含)
right

二分搜尋實作

核心邏輯:

```
if (values[mid] == target) {  // 找到目標
  // 將邊界向內提或回傳
} else if (values[mid] < target) {  // 目標在中間值右邊
  // 將左側邊界縮減
} else if (values[mid] > target) {  // 目標在中間值左邊
  // 將右側邊界縮減
}
```

```cpp
//  iterative binary search, [l, r), returns the smallest index where value[index] >= target
int lower_bound(vector<int> &values, int target) {
  int left = 0, right = values.size();

  while (left < right) {
    int mid = left + (right - left) / 2;

    if (values[mid] == target) {
      right = mid;  //  close up to lower boundary
    } else if (values[mid] < target) {
      left = mid + 1;
    } else if (values[mid] > target) {
      right = mid;
    }
  }

  return left;
}
```

左閉右開寫法
lower_bound
尋找**大於等於**目標的最小值

```
index:  0 1 2 3 4 5 6 7
values: 1 2 4 4 5 5 5 6
```

```
lower_bound(values, 5) binary_search (lower_bound) 4
lower_bound(values, 3) binary_search (lower_bound) 2
```

```cpp
//  iterative binary search, [l, r), returns the smallest index where values[index] > target
int upper_bound(vector<int> &values, int target) {
  int left = 0, right = values.size();

  while (left < right) {
    int mid = left + (right - left) / 2;

    if (values[mid] == target) {
      left = mid + 1;
    } else if (values[mid] < target) {
      left = mid + 1;
    } else if (values[mid] > target) {
      right = mid;
    }
  }

  return left;
}
```

左閉右開寫法
upper_bound
尋找**大於**目標的最小值

```
index:  0 1 2 3 4 5 6 7
values: 1 2 4 4 5 5 5 6
```

upper_bound(values, 5) binary_search (upper_bound) 7

upper_bound(values, 3) binary_search (upper_bound) 2

C++內建的std::lower_bound和std::upper_bound

最常用的用法(範例請見程式碼)

```
std::lower_bound(iterator_first, iterator_last, target);

std::upper_bound(iterator_first, iterator_last, target);
```

iterator_first為欲搜尋範圍之起始位置迭代器;
iterator_last為欲搜尋範圍之終止位置迭代器
搜尋範圍左閉右開(不包含iterator_last)

回傳值為迭代器

Leetcode 35. Search Insert Position

給予一個**遞增排列**好的 array 以及目標值。若目標值在 array 中，回傳他的位置；不在的話，回傳目標值應插入的位置。

*array中沒有重複值

Example 1:

> **Input:** nums = [1,3,5,6], target = 5
> **Output:** 2

Example 2:

> **Input:** nums = [1,3,5,6], target = 2
> **Output:** 1

Example 3:

> **Input:** nums = [1,3,5,6], target = 7
> **Output:** 4

Leetcode 35. Search Insert Position

```cpp
class Solution {
private:
  int lower_bound(vector<int> &values, int target) {
    int left = 0, right = values.size();

    while (left < right) {
      int mid = left + (right - left) / 2;

      if (values[mid] == target) {
        right = mid;  //  close up to lower boundary
      } else if (values[mid] < target) {
        left = mid + 1;
      } else if (values[mid] > target) {
        right = mid;
      }
    }

    return left;
  }

public:
  int searchInsert(vector<int>& nums, int target) {
    int index = lower_bound(nums, target);
    return index;
  }
};
```

Leetcode 1539. Kth Missing Possive Number

給予一個嚴格遞增的array及正整數k，回傳array中第k個缺少的正數。

Example 1:

> **Input:** arr = [2,3,4,7,11], k = 5
> **Output:** 9
> **Explanation:** The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5^{th} missing positive integer is 9.

Example 2:

> **Input:** arr = [1,2,3,4], k = 2
> **Output:** 6
> **Explanation:** The missing positive integers are [5,6,7,...]. The 2^{nd} missing positive integer is 6.

Leetcode 1539. Kth Missing Possive Number

```cpp
class Solution {
public:
  int findKthPositive(vector<int>& arr, int k) {
    int left = 0, right = arr.size();

    while (left < right) {
      int mid = left + (right - left) / 2;
      int missing_count = arr[mid] - (mid + 1);

      if (missing_count < k) {
        left = mid + 1;
      } else {
        right = mid;
      }
    }

    return left + k;
  }
};
```

Leetcode 2540. Minimum Common Value

給予兩個遞增的array，回傳兩者共同元素中的最小值，如果沒有共同元素則回傳1。

Example 1:

> **Input:** nums1 = [1,2,3], nums2 = [2,4]
> **Output:** 2
> **Explanation:** The smallest element common to both arrays is 2, so we return 2.

Example 2:

> **Input:** nums1 = [1,2,3,6], nums2 = [2,3,4,5]
> **Output:** 2
> **Explanation:** There are two common elements in the array 2 and 3 out of which 2 is the smallest, so 2 is returned.

Leetcode 2540. Minimum Common Value

```cpp
class Solution {
private:
  bool binary_search(vector<int> &values, int target) {
    int left = 0, right = values.size();

    while (left < right) {
      int mid = left + (right - left) / 2;

      if (values[mid] == target) {
        right = mid;  //  close up to lower boundary
      } else if (values[mid] < target) {
        left = mid + 1;
      } else if (values[mid] > target) {
        right = mid;
      }
    }

    return values[left] == target;
  }
```

將lower_bound的邏輯改一下,
若有找到回傳true;沒有則回傳false

Leetcode 2540. Minimum Common Value

```
public:
  int getCommon(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) {
      return getCommon(nums2, nums1);
    }

    for (int num: nums1) {
      if (binary_search(nums2, num)) {
        return num;
      }
    }

    return -1;
  }
};
```