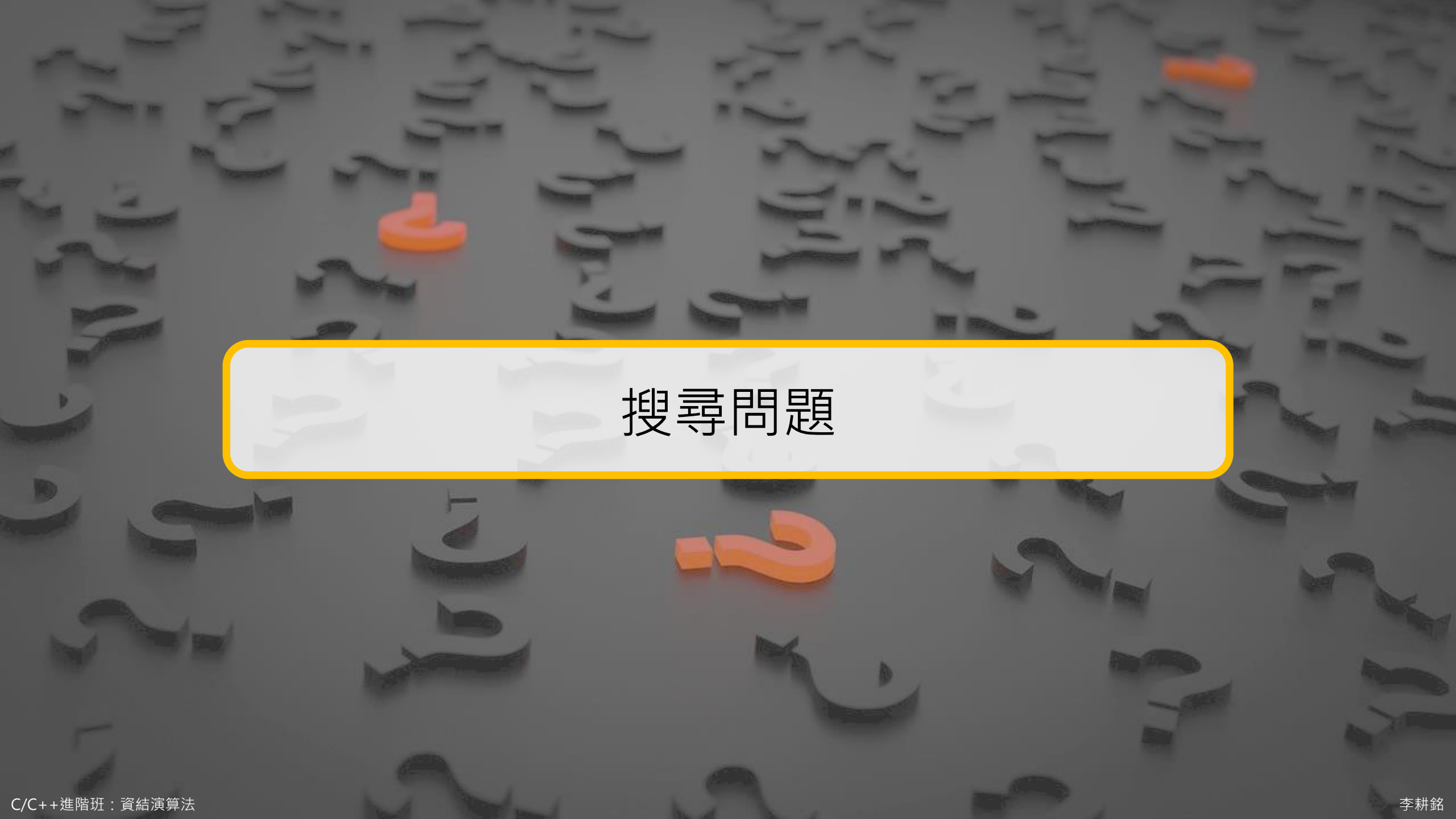
C/C++ 進階班
演算法

搜尋
(Search)

李耕銘

課程大綱

- 搜尋問題
- 搜尋方式
  1. 循序搜尋（Sequential Search）
  2. 二分搜尋（Binary Search）
  3. 二元搜尋樹搜尋 (Binary Tree Search)
  4. 插補搜尋 (Interpolation Search)
  5. 黃金切割搜尋 (Golden Section Search)
  6. 雜湊搜尋 (Hasing Search)
  7. 費氏搜尋(Fibonacci Search)
- 搜尋總結
- 實戰練習



搜尋問題

搜尋問題

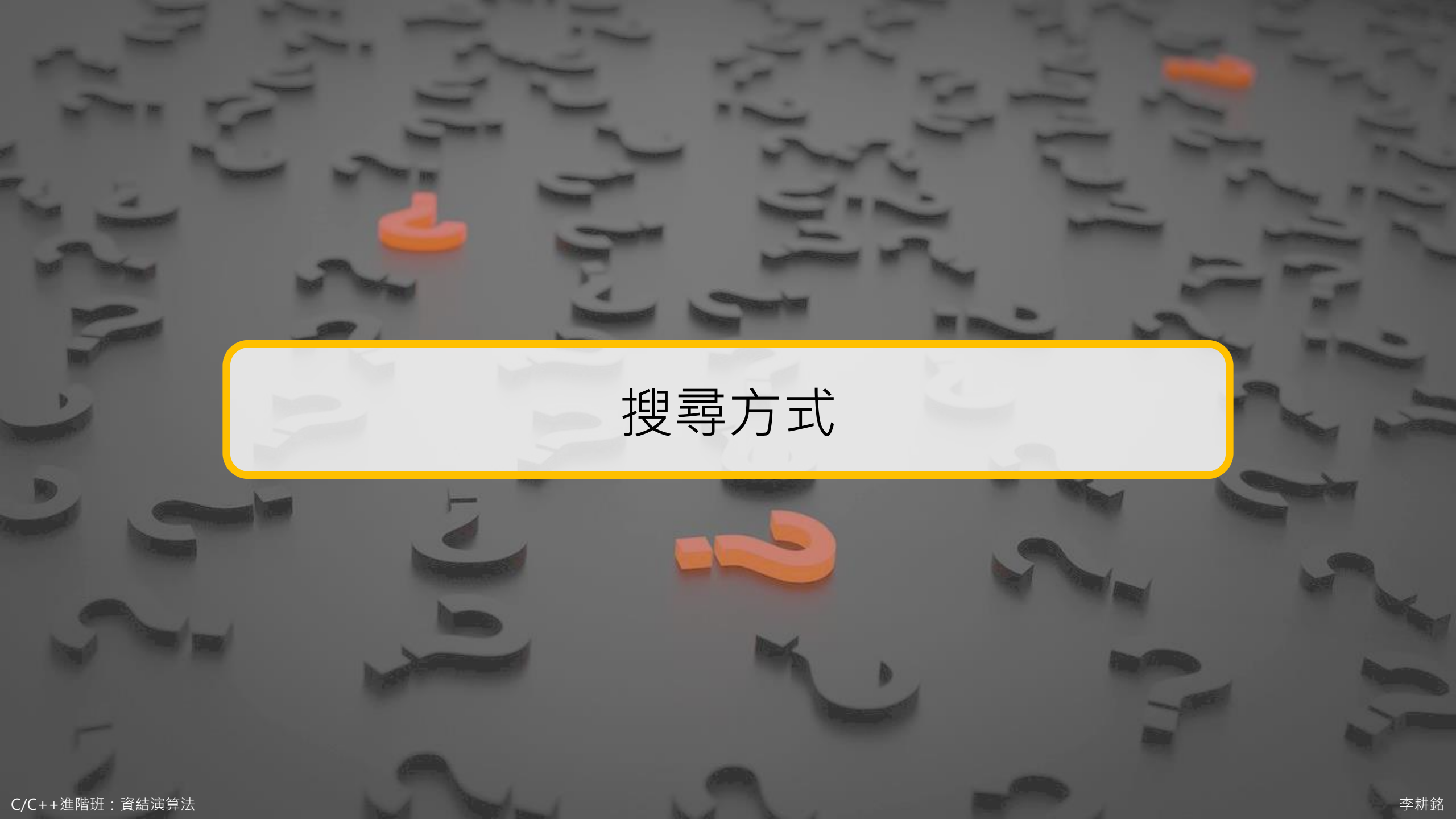
- 搜尋定義
  - ➢ 在集合中找出具特定鍵值(Key) 的元素
    - ✓ 該集合可以是未排序或已排序
  - ➢ 不同的資料結構會影響到搜尋的方式與效率
  - ➢ 同樣具有 Worst/Average/Best case
    - ✓ 通常 Best case 都是 O(1)
    - ✓ 運氣超好 der

搜尋問題

- 搜尋方式分類
  - ➢ 內部搜尋
    - ✓ 搜尋過程中的資料可以全部載入記憶體中
  - ➢ 外部搜尋
    - ✓ 搜尋過程中的資料不可以全部載入記憶體中
  - ➢ Successful Search
    - ✓ 成功找到該筆資料在資料結構中的位置
  - ➢ Unsuccessful Search
    - ✓ 確定該資料並不存在於該資料結構中

搜尋問題

- 本章講的搜尋，使用的資料結構皆為陣列
  - ➢ 其實 Sequential container 都可以
    - ✓ Vector
    - ✓ Linked list
  - ➢ 雜湊搜尋、二元搜尋樹本章不重提
    - ✓ 請至資料結構班的講義自行查看
  - ➢ 費氏搜尋法與費氏堆疊也不提
    - ✓ 費氏堆疊與紅黑樹是大學資料結構的兩大魔王
    - ✓ 以後可能會講，但不是現在這門課



搜尋方式

循序搜尋

- 從第一筆資料一路找到最後一個
  - 其實就是暴力解！
  - 資料不需要事先經過排序
  - 複雜度為：O(n)

35	52	68	12	47	52	36	52	74	27

循序搜尋

```
int sequential_search(int data[], int len, int target){
    for(int i=0;i<len;i++){
        if(data[i]==target)
            return i;
        else if(i==len-1)
            return -1;
    }
}
```

35	52	68	12	47	52	36	52	74	27

循序搜尋

- Worst case
  - 找到最後一個才找到：O(n)
- Best case
  - 第一個就找到了：O(1)
- Average case
  - 平均要找的次數：$\frac{1+2+3+\cdots n}{n} = \frac{n+1}{2}$
  - O(n/2)= O(n)

二分搜尋

- 每次搜尋時，用刪去法刪去一半的可能
  - 資料必須事先排序好，才知道要刪哪一半
  - 需要支援隨機存取(索引值)，否則效能低落
  - 每次搜尋會分成三種狀況
    1. 確定找或找不到資料！
    2. 沒有找到資料，但它會出現在陣列的前半部
    3. 沒有找到資料，但它會出現在陣列的後半部

二分搜尋

要找 27：

12	27	35	36	47	52	52	52	68	74

取出陣列的中位數：52 (索引值 10/2=5)

12	27	35	36	47	52	52	52	68	74

52 > 27，目標在左邊 0~4 索引值的區間

12	27	35	36	47	52	52	52	68	74

二分搜尋

12	27	35	36	47	52	52	52	68	74

取出陣列的中位數：35 (索引值 5/2=2)

12	27	35	36	47	52	52	52	68	74

35 > 27，目標在左邊 0~1 索引值的區間

12	27	35	36	47	52	52	52	68	74

二分搜尋

12	27	35	36	47	52	52	52	68	74

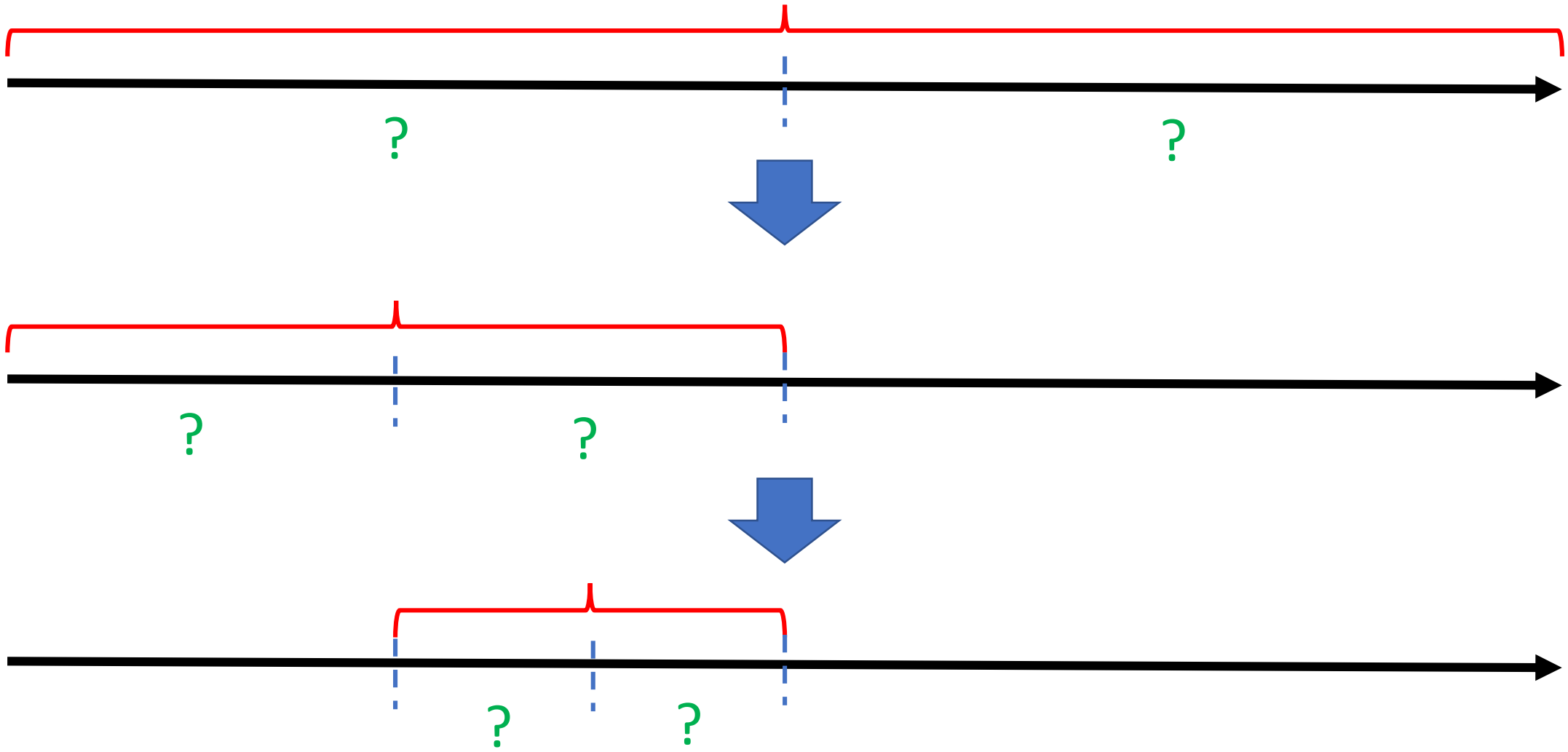
取出陣列的中位數：27 (索引值 2/2=1)

12	27	35	36	47	52	52	52	68	74

27 = 27，結束，回傳目前的索引值 1

12	27	35	36	47	52	52	52	68	74

二分搜尋

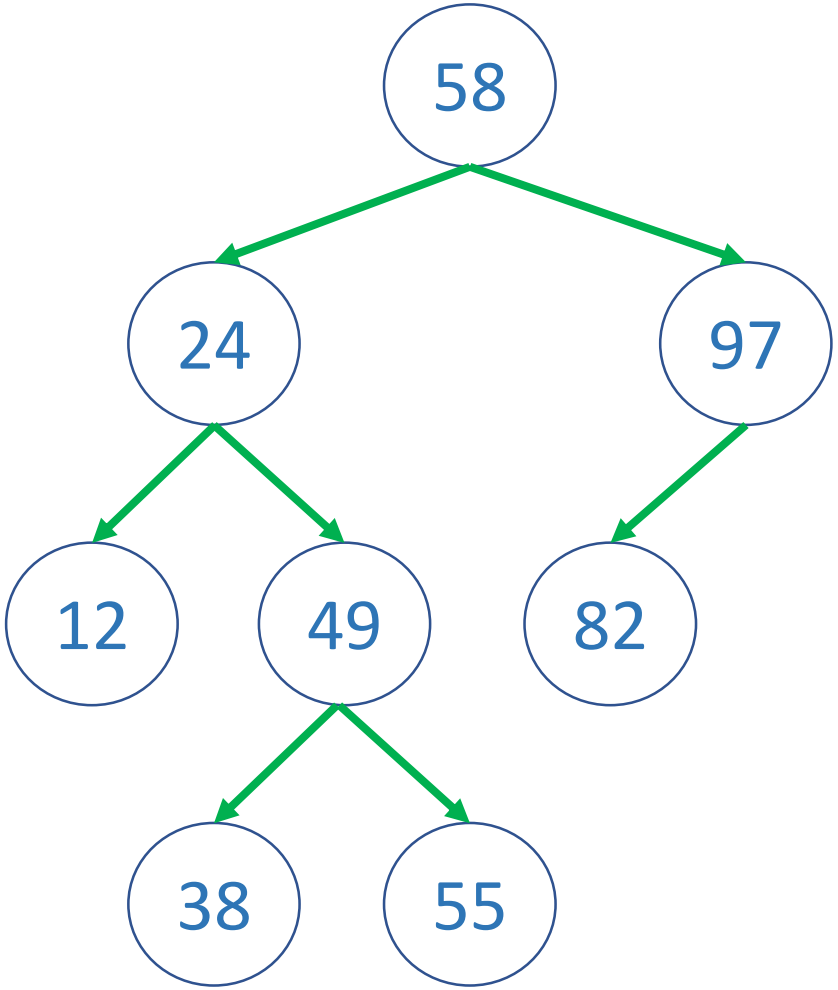


二分搜尋

- Worst case
  - 找到最後一個才找到：$O(log_2 n)$
- Best case
  - 第一個就找到了：O(1)
- Average case
  - 平均要找的次數：$\frac{1+2+3+\cdots\lfloor log_2 n \rfloor}{\lfloor log_2 n \rfloor} = \frac{\lfloor log_2 n \rfloor + 1}{2}$
  - $O(log_2 n)$

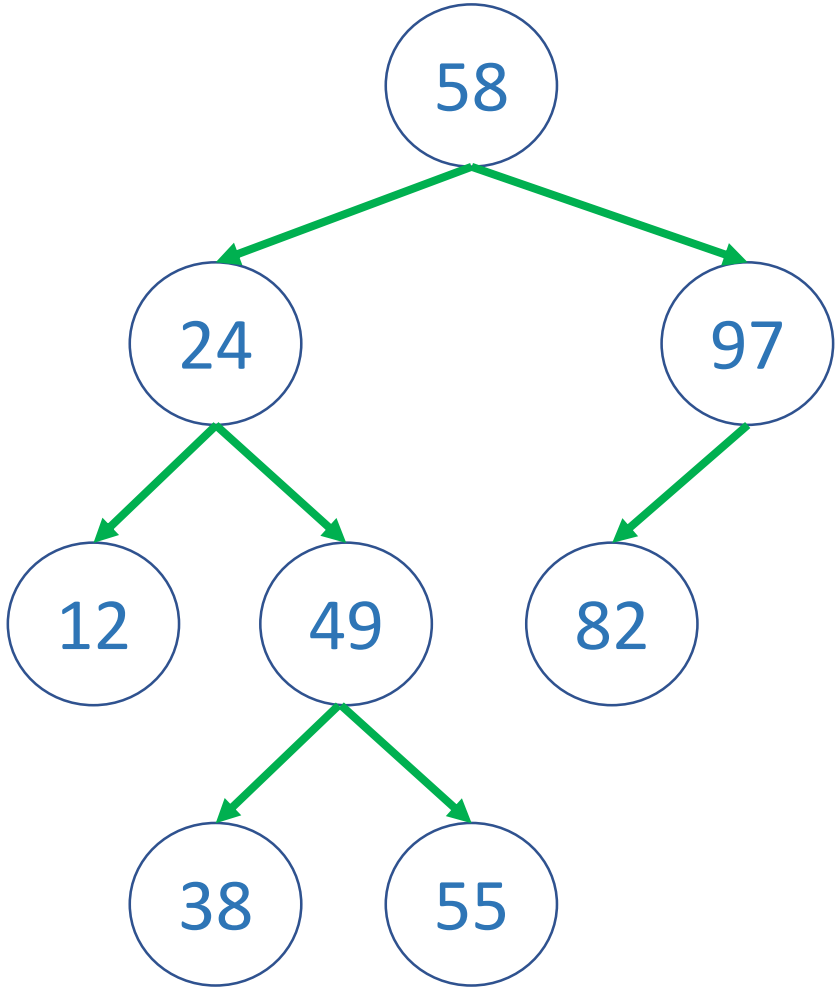
二元搜尋樹

- 當要找的編號跟現處節點一致
  - ➢ 結束
- 當要找的編號跟現處節點不一致
  - ➢ 尋找的編號比節點編號小
    - ✓ 往左節點移動
  - ➢ 尋找的編號比節點編號大
    - ✓ 往右節點移動
- 當現在的節點為 leaf node
  - ➢ 結束，回傳空指標



二元搜尋樹

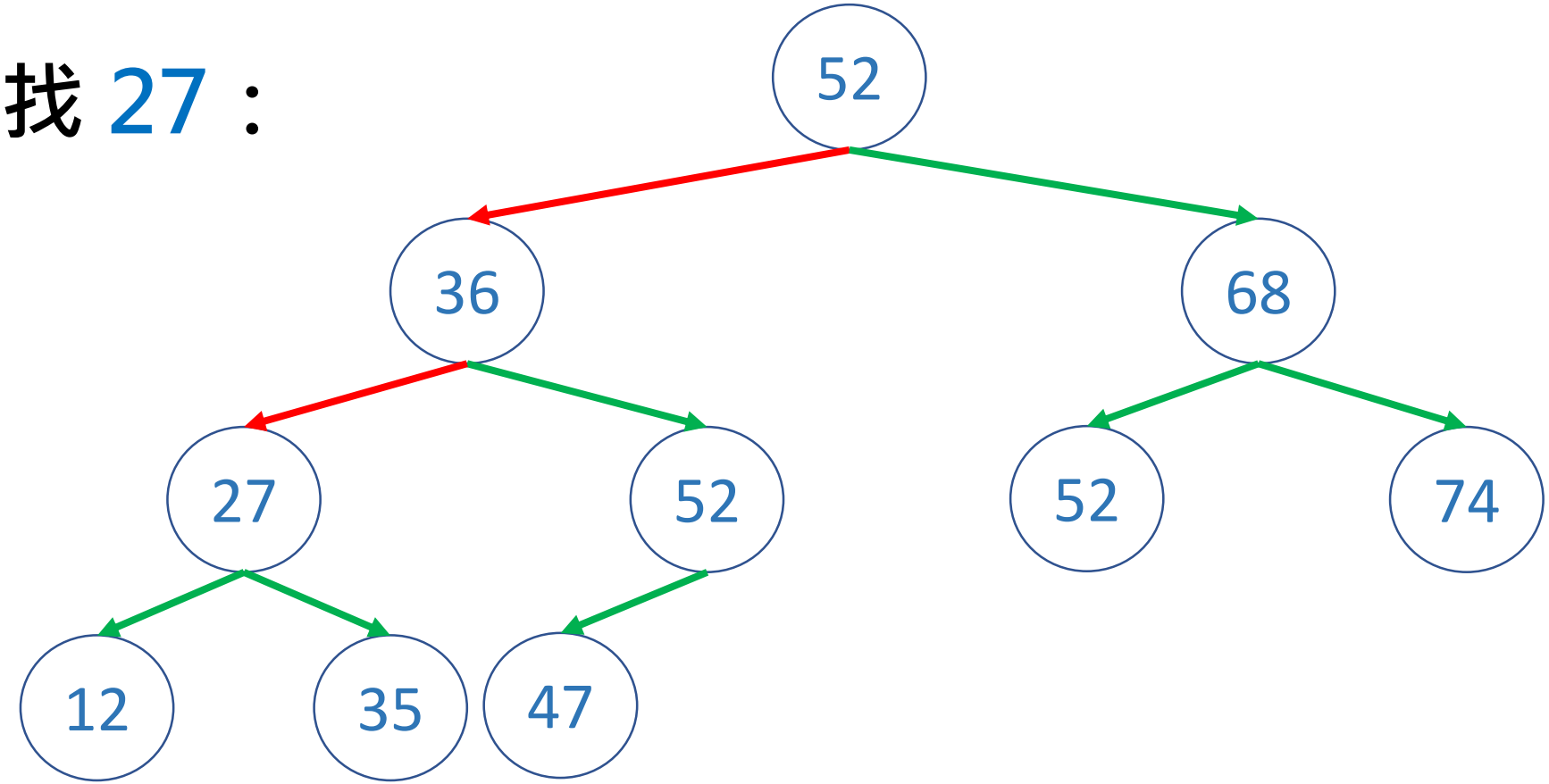
- 若是完滿二元樹
  - ✓ 每經過一次分岔就可以刪去 ½
  - ✓ n 次搜尋，可以找到 $2^n - 1$筆資料
  - ✓ 搜尋次數 ~ log_2(資料數目)
  - ✓ 搜尋：O(log_2N)
  - ✓ 新增：O(log_2N)
  - ✓ 刪除：O(log_2N)
  - ✓ 通常資料結構/演算法的 log 底數為2



二元搜尋樹

12	27	35	36	47	52	52	52	68	74

要找 27：



Example Code

Mission

對已排序好的數列執行二分搜尋法。

二分搜尋

```
int Binary_Search(int data[],int lower,int upper,int target){
    if(upper<lower)
        return -1;
    int middle = (lower+upper)/2;
    if(data[middle]==target)//中位數==target
        return middle;
    else if(data[middle]>target)
        return Binary_Search(data,lower,middle-1,target);
    else if(data[middle]<target)
        return Binary_Search(data,middle+1,upper,target);
}
```

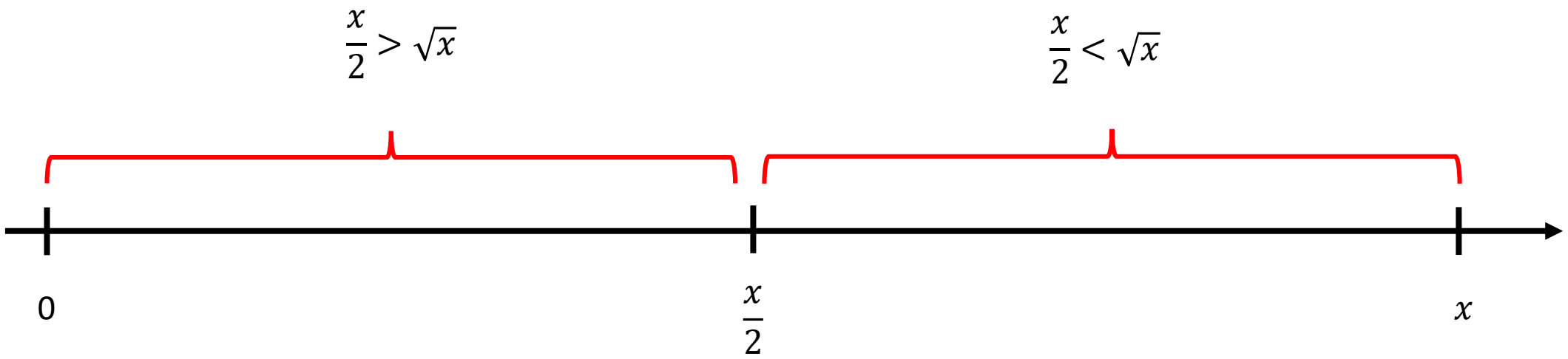
Practice 1

Mission

讓使用者輸入一個大於 1 的數 x 與可容許誤差 E，而後透過二分搜尋法求出 $\sqrt{x}$ 的值，誤差範圍需在 E 以內。

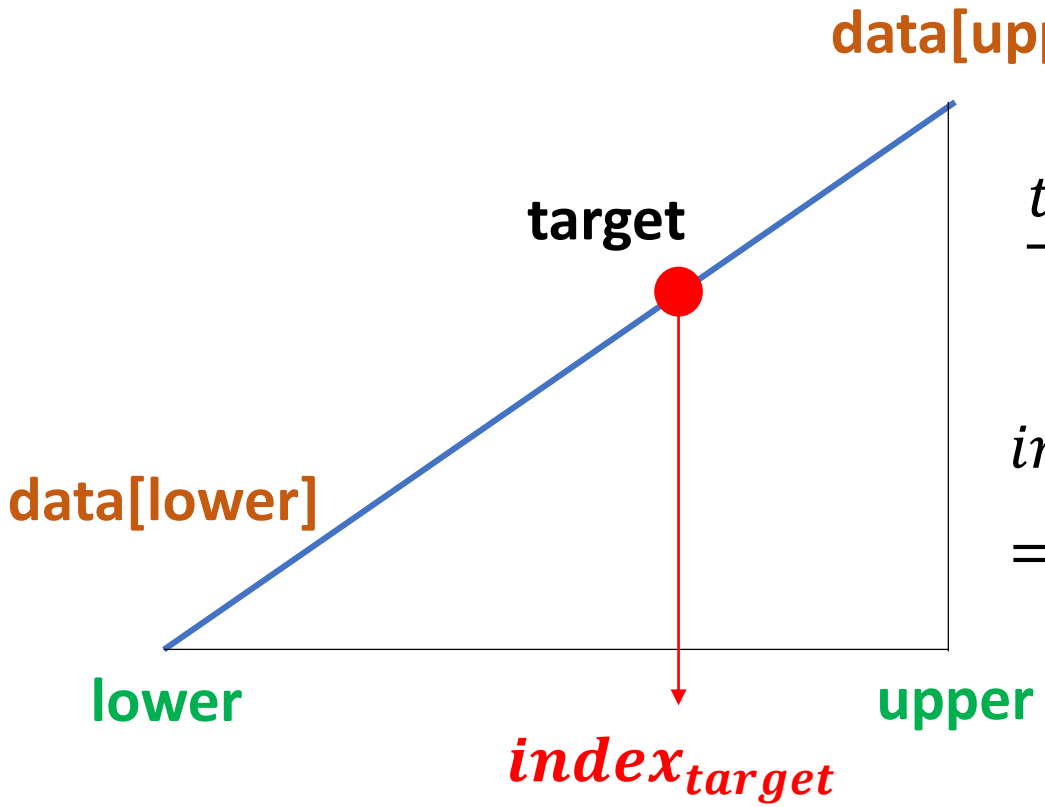
```
Please enter a number(>1) and error :
5 0.000001
The square root is 2.23607
```

Practice 1



插補搜尋

- 二分逼近法的改良版，用內插法找出資料
  - ➢ 假設資料是平均散佈的狀況



$$\frac{target - data[lower]}{index_{target} - lower} = \frac{data[upper] - data[lower]}{upper - lower}$$

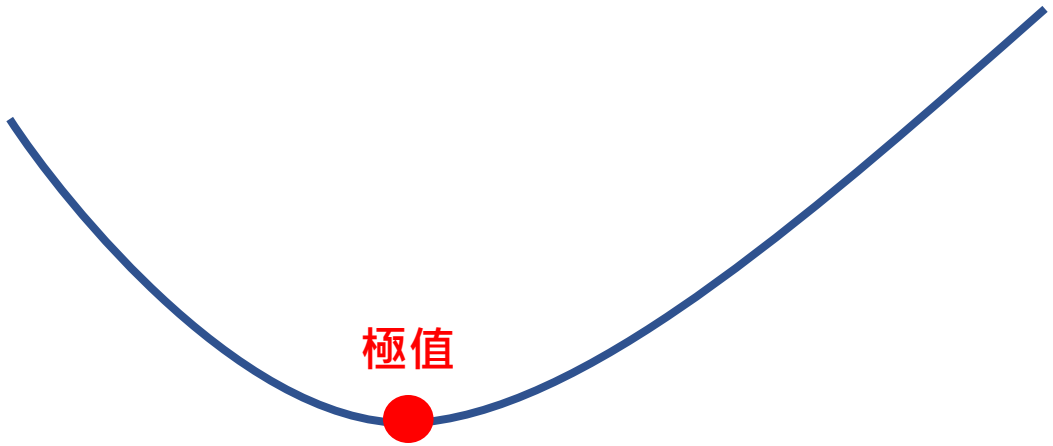
$$index_{target} = lower + \frac{(target - data[lower])(upper - lower)}{data[upper] - data[lower]}$$

插補搜尋

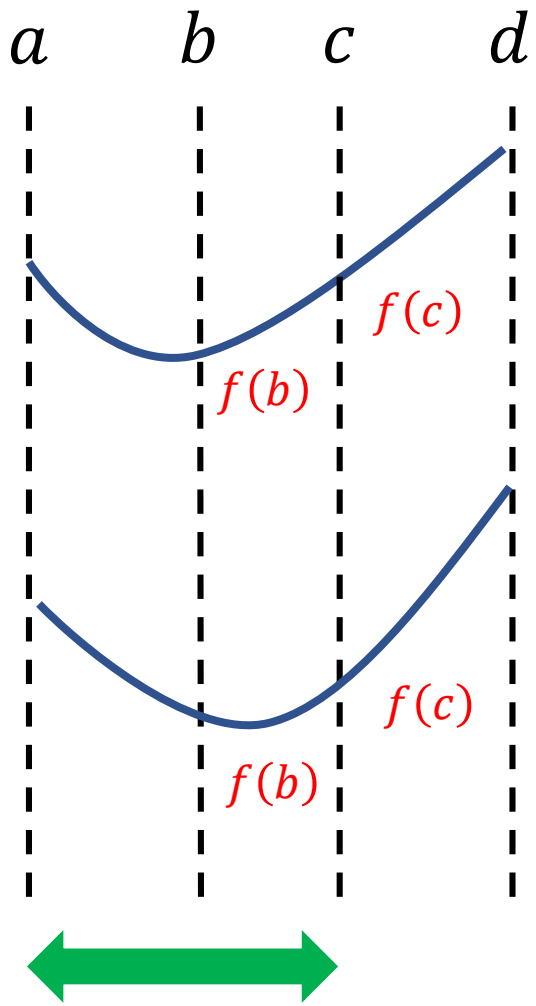
```
int Interpolation_Search(int data[],int lower,int upper,int target){
    if(upper<lower)
        return -1;
    int upper_data = data[upper];
    int lower_data = data[lower];
    int index = lower + (target-lower_data)*(upper-lower)/(upper_data-lower_data);
    if(data[index]==target)
        return index;
    else if(data[index]>target)
        return Interpolation_Search(data,lower,index-1,target);
    else if(data[index]<target)
        return Interpolation_Search(data,index+1,upper,target);
}
```

黃金切割搜尋
(Golden Section Search)

- 特殊狀況下的搜尋方式
  - 可找出函式在某區間的極值
  - 只適用於單峰函式 (unimodal function)



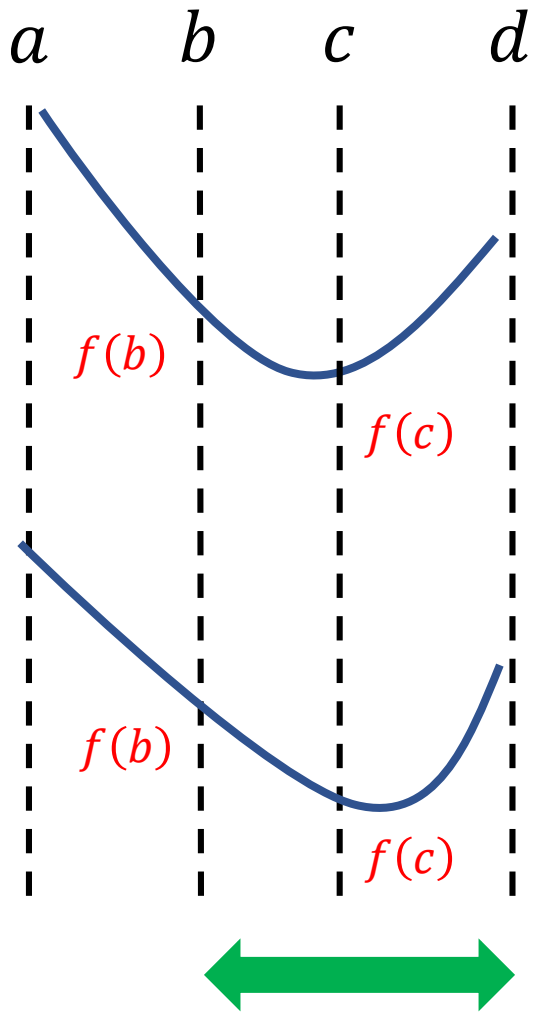
黃金切割搜尋
(Golden Section Search)



$if\ f(c) > f(b)$

→ 極值落在 [a,c]

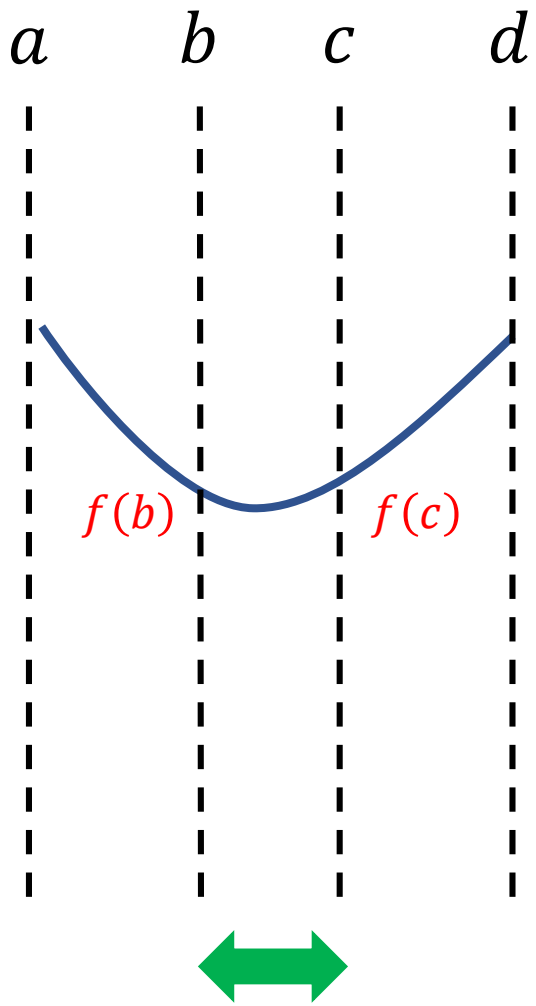
黃金切割搜尋
(Golden Section Search)



$if\ f(b) > f(c)$

→ 極值落在 [b,d]

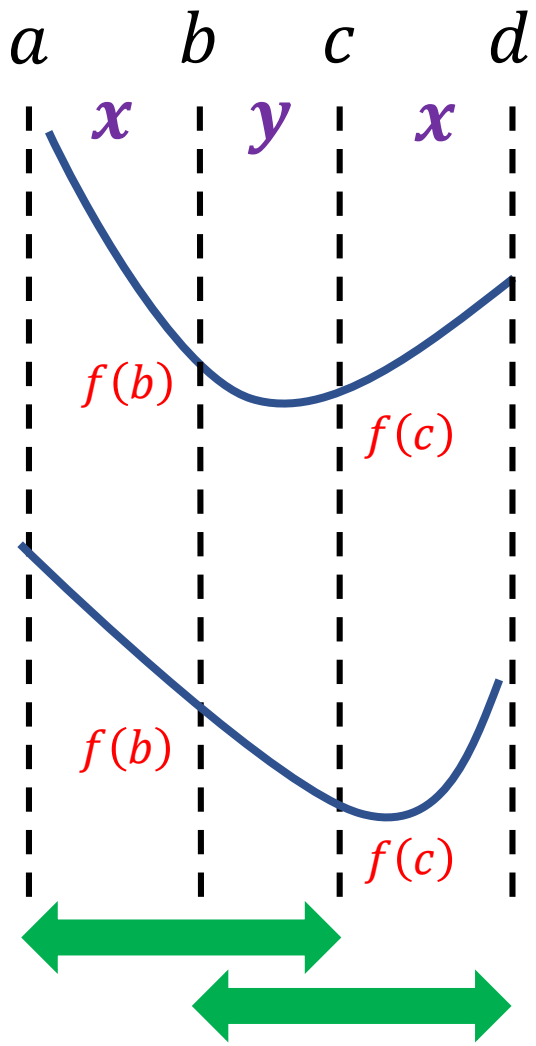
黃金切割搜尋
(Golden Section Search)



$if\ f(b) = f(c)$

→ 極值落在 [b,c]

黃金切割搜尋
(Golden Section Search)



$$Q：\overline{ab}、\overline{bc}、\overline{cd} = ?$$

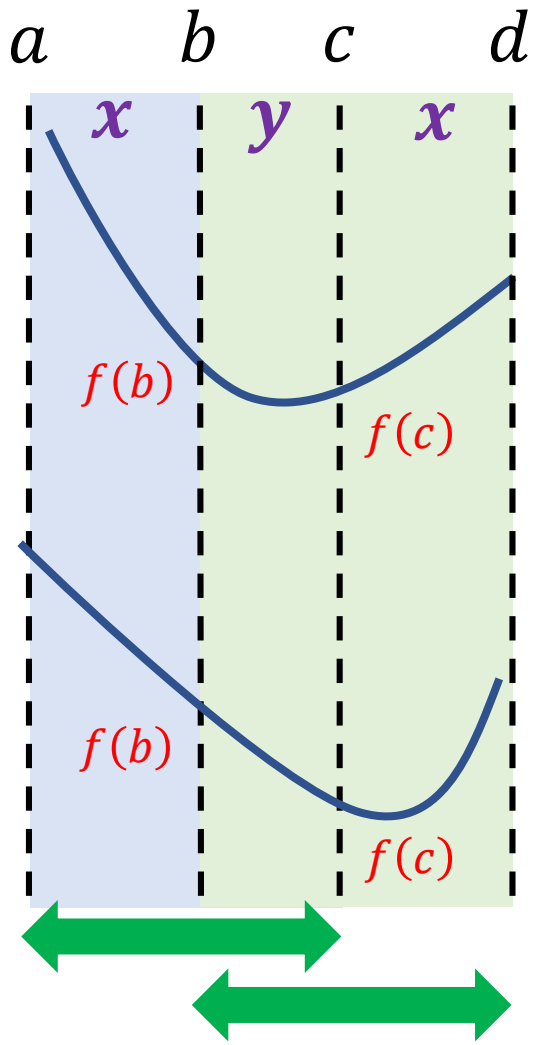
兩端需對稱，故 $\overline{ac} = \overline{bd}$

$$let\ \overline{ab} = \overline{cd} = x, \overline{bc} = y$$

為了讓 b、c 點可以繼續沿用：

$$\frac{\overline{ab}}{\overline{ad}} = \frac{\overline{bc}}{\overline{bd}} \rightarrow \frac{x}{2x+y} = \frac{y}{x+y}$$

黃金切割搜尋
(Golden Section Search)



$$Q：\overline{ab}、\overline{bc}、\overline{cd} = ?$$

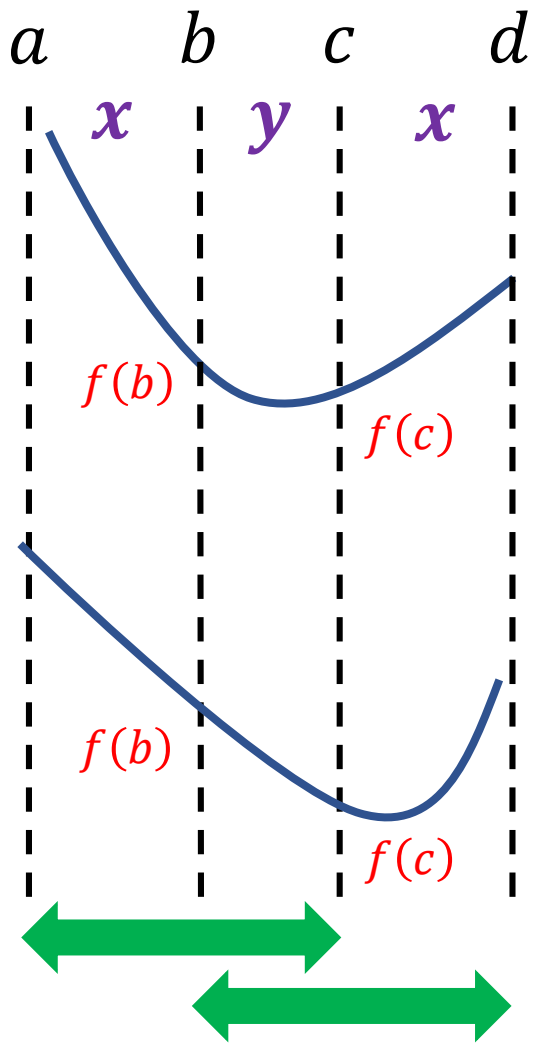
兩端需對稱，故 $\overline{ac} = \overline{bd}$

$$let\ \overline{ab} = \overline{cd} = x, \overline{bc} = y$$

為了讓 b、c 點可以繼續沿用：

$$\frac{\overline{ab}}{\overline{ad}} = \frac{\overline{bc}}{\overline{bd}} \rightarrow \frac{x}{2x+y} = \frac{y}{x+y}$$

黃金切割搜尋
(Golden Section Search)



$$\frac{\overline{ab}}{\overline{ad}} = \frac{\overline{bc}}{\overline{bd}} \rightarrow \frac{x}{2x+y} = \frac{y}{x+y}$$

$$x^2 + xy = 2xy + y^2$$

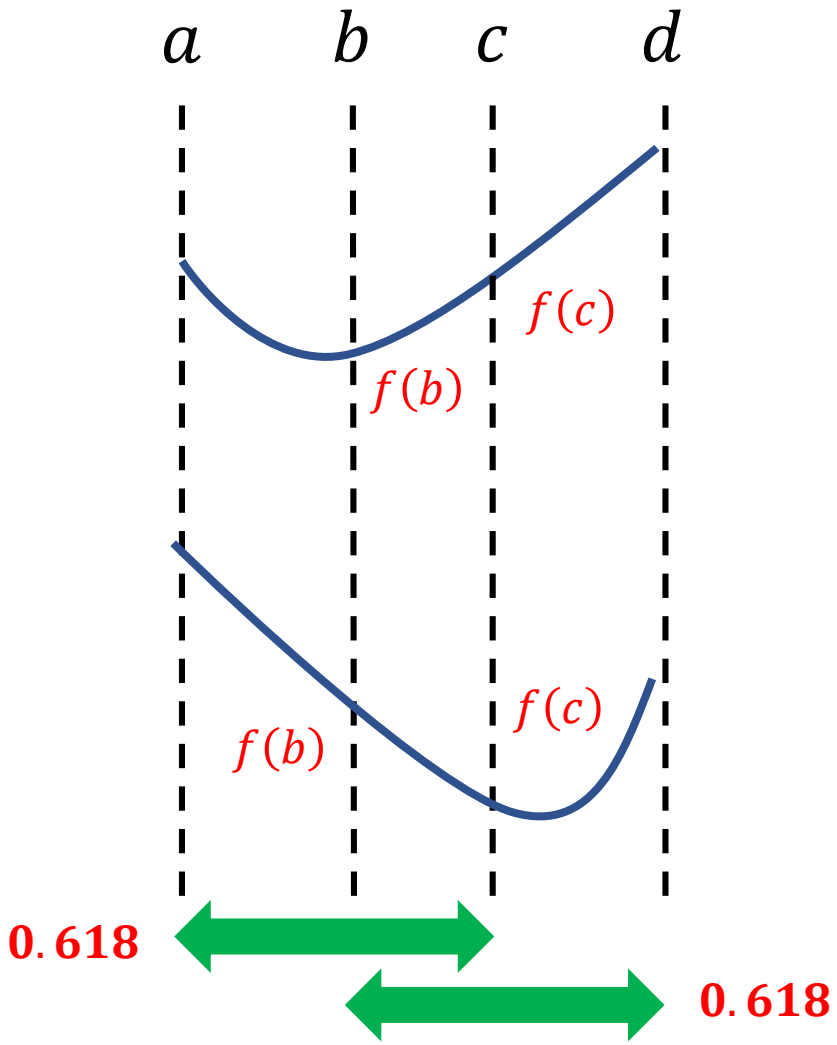
$$x^2 - xy - y^2 = 0$$

$$x = \frac{y \pm \sqrt{5y^2}}{2} = \frac{1 \pm \sqrt{5}}{2} y$$

$$\frac{\boldsymbol{x}}{\boldsymbol{y}} = \frac{\mathbf{1} + \sqrt{\mathbf{5}}}{\mathbf{2}}$$

$$\frac{\boldsymbol{x+y}}{\boldsymbol{2x+y}} = \frac{\mathbf{-1} + \sqrt{\mathbf{5}}}{\mathbf{2}} \approx \mathbf{0.618}$$

黃金切割搜尋
(Golden Section Search)



$$\frac{\overline{ac}}{\overline{ad}} = \frac{1+\sqrt{5}}{2} \approx \mathbf{0.618}$$ 下第一刀

$$\frac{\overline{ac}}{\overline{ad}} = \frac{\overline{bd}}{\overline{ad}} = \frac{\overline{cd}}{\overline{bd}} = \frac{\overline{ab}}{\overline{ac}} \sim 0.618$$

$if\ f(c) > f(b)$

→ 極值落在 [a,c]

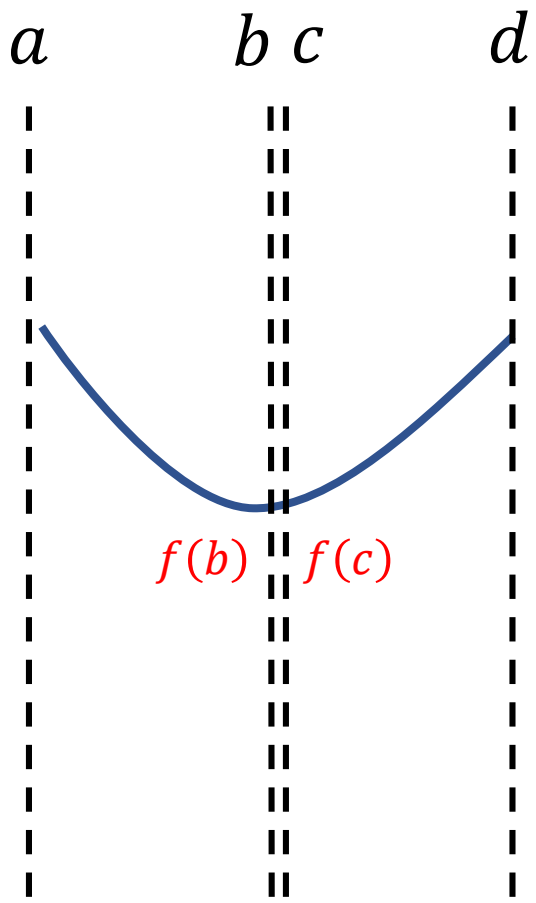
→切出 e，$\frac{\overline{ce}}{\overline{ac}} = \frac{1+\sqrt{5}}{2}$

$if\ f(c) < f(b)$

→ 極值落在 [b,d]

→切出 e，$\frac{\overline{be}}{\overline{bd}} = \frac{1+\sqrt{5}}{2}$

黃金切割搜尋
(Golden Section Search)

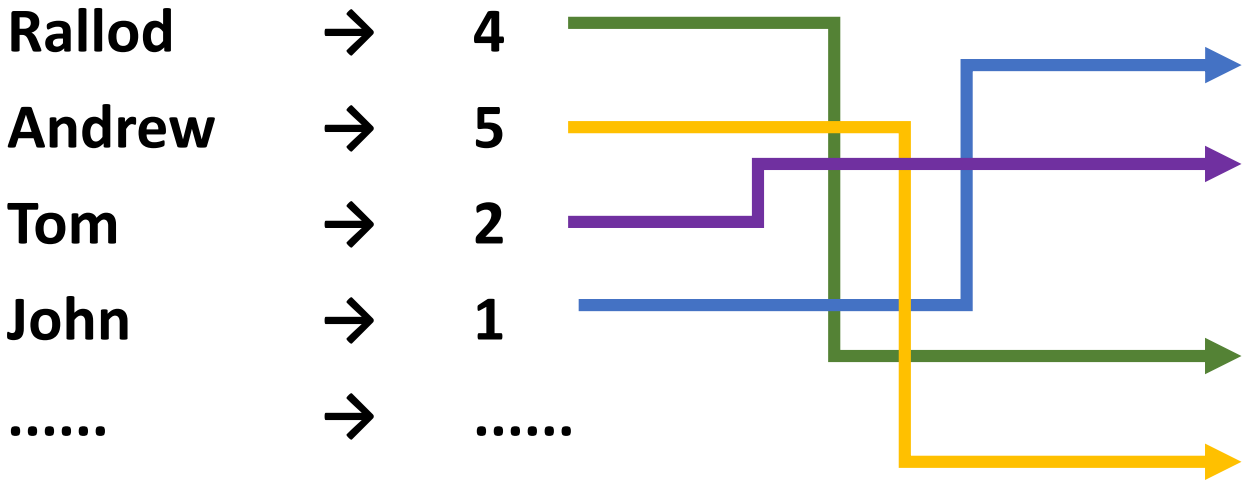


- 為何不直接在中點附近取兩個點？
- 如果在中點附近取兩個很接近的點
  - 每呼叫兩次函式，縮減一半的空間
  - 每呼叫一次函式，縮減 25% 的空間
- 黃金切割搜尋可以複用上一輪的運算結果
  - 每呼叫一次函式，縮減 38.2% 的空間
- 黃金切割搜尋在同樣呼叫次數下效率較高

雜湊搜尋

給定任意 input，output 必須介於 0~m-1

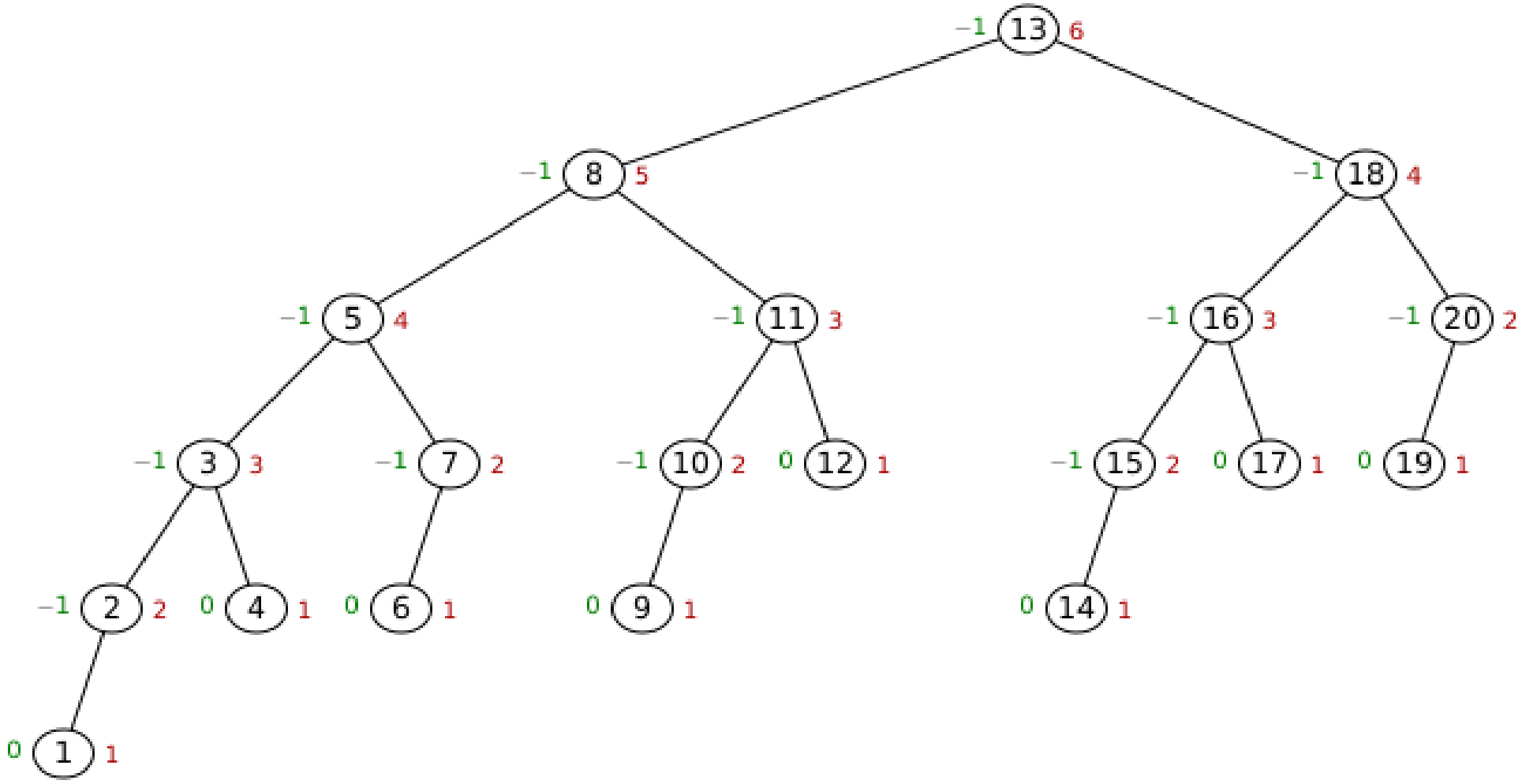
$$hash: U \rightarrow \{0, 1, \ldots, m-1\}$$



Index	Name	Score
0	David	83
1	John	95
2	Tom	82
3	Sherry	61
4	Rallod	78
5	Andrew	85
6	Helen	96
...	...	...

m

費氏搜尋


-1 13 6
-1 8 5
-1 18 4
-1 5 4
-1 11 3
-1 16 3
-1 20 2
-1 3 3
-1 7 2
-1 10 2
0 12 1
-1 15 2
0 17 1
0 19 1
-1 2 2
0 4 1
0 6 1
0 9 1
0 14 1
0 1 1

Practice 2

Mission

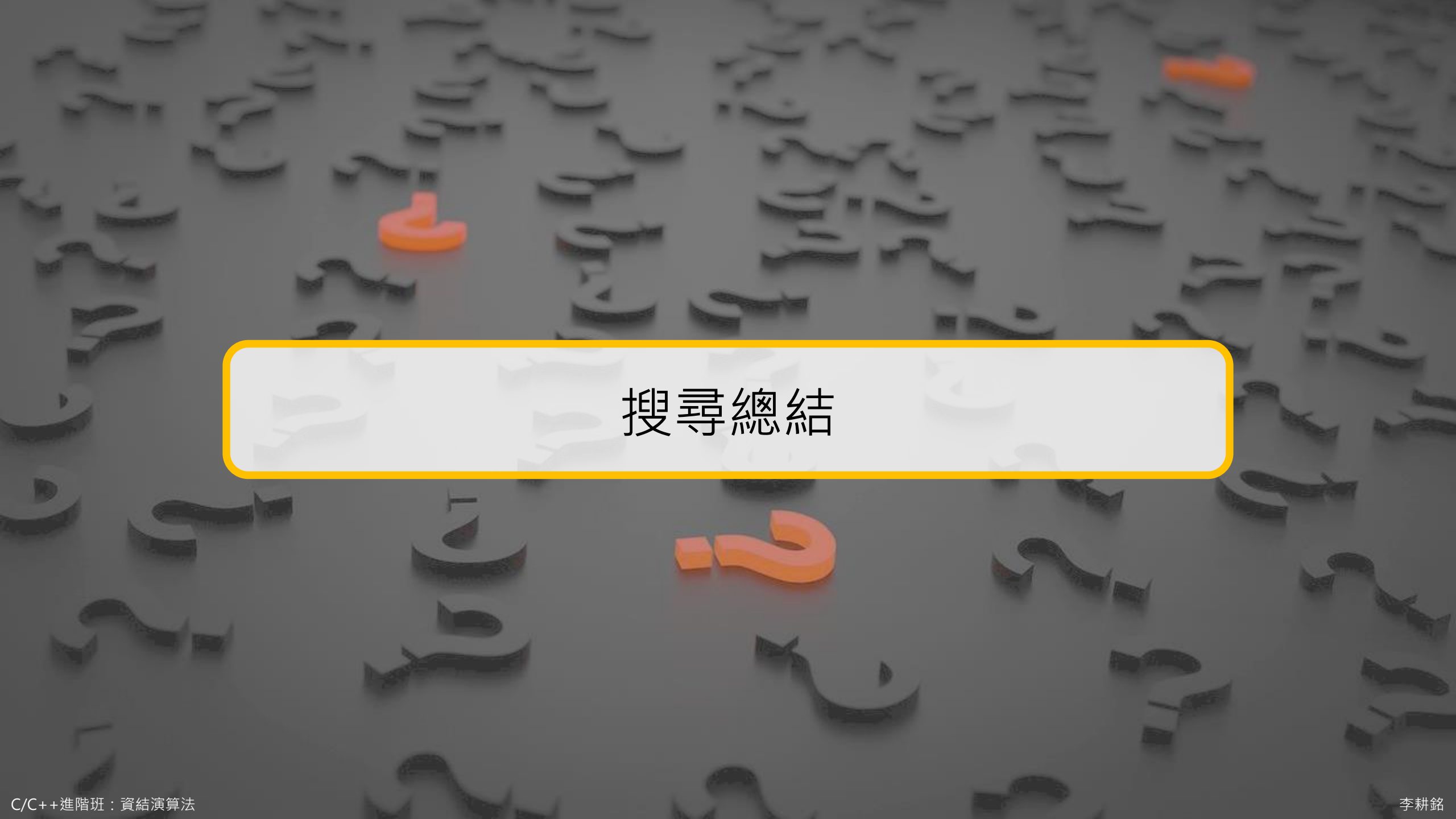
Try LeetCode #852. Peak Index in a Mountain Array

Let's call an array arr a mountain if the following properties hold:

- arr.length >= 3
- There exists some i with 0 < i < arr.length - 1 such that:
  - arr[0] < arr[1] < ... arr[i-1] < arr[i]
  - arr[i] > arr[i+1] > ... > arr[arr.length - 1]

Given an integer array arr that is guaranteed to be a mountain, return any i such that arr[0] < arr[1] < ... arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1].

Ref : https://leetcode.com/problems/peak-index-in-a-mountain-array/



搜尋總結

總結

- 資料如果沒有經過任何處理只能使用循序搜尋
- 事先排序好，就能夠使用二分搜尋、插補搜尋
- 事先建立好雜湊表，就能夠用雜湊搜尋
- 但是資料前處理也需要時間
  - 只要處理過一次，往後每次搜尋都能大幅加速
  - 前人種樹後人乘涼
- 二分搜尋最重要、最常用、最常考

總結

搜尋複雜度	最好複雜度	平均複雜度	最壞複雜度
線性搜尋	$O(1)$	$O(n)$	$O(n)$
二分搜尋	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$
二元樹搜尋	$O(1)$	$O(\log_2 n)$	$O(\log_2 n)$
插補搜尋	$O(1)$	$O(\log_2 n)$	$O(n)$
雜湊搜尋	$O(1)$	$O(1)$	$O(1)$

實戰練習

Practice 3

Mission

Try LeetCode #35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

Ref：https://leetcode.com/problems/search-insert-position/

Practice 4

Mission

Try LeetCode #278. First Bad Version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Ref：https://leetcode.com/problems/first-bad-version/

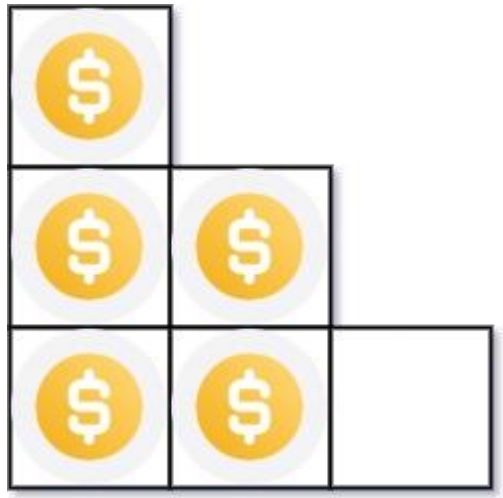
Practice 5

Mission

Try LeetCode #441. Arranging Coins

You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase may be incomplete.

Given the integer n, return the number of complete rows of the staircase you will build.



Ref：https://leetcode.com/problems/arranging-coins/