

# 下載範例檔案

- [https://github.com/ck1001099/cpp\\_course\\_2019summer](https://github.com/ck1001099/cpp_course_2019summer)
- lesson8 -> Lecture -> sort\_template.cpp

# C++程式設計基礎

## lesson 8

陳毅

# 上週回顧

- 資料結構（ STL container ）
  - vector
  - map

# vector 簡介

- **vector**可視為會自動擴展容量的陣列。
  - 支援隨機存取
  - 在集合尾端增刪元素很快，但是在集合中間增刪元素比較費時
  - 使用動態陣列方式實作
- 容器類型：陣列

# map 簡介

- map具有一對一mapping 功能。
  - 第一個稱為關鍵字 (key)，每個關鍵字只能在map中出現一次。
  - 第二個稱為該關鍵字的值 (value)。
- 容器類型：字典
  - 在一本字典裡，一個單字 (key)只會出現一次。
  - 根據單字 (key)，我們可以查到該單字所對應的解釋 (value)。

# 本週概要

- 基礎演算法
  - 排序
    - 選擇排序法 (selection sort)
    - 泡泡排序法 (bubble sort)
  - 搜尋
    - 循序搜尋法 (sequential search)
    - 二元搜尋法 (binary search)
- 樹
- 進階演算法
  - DFS（深度優先搜尋）
  - BFS（廣度優先搜尋）

# 選擇排序法

# 簡介

- **選擇排序法(Selection Sort)**是排序演算法的一種，也是一種簡單容易理解的演算法。
- 其概念是反覆從未排序的數列中取出最小的元素，加入到另一個的數列，結果即為已排序的數列。

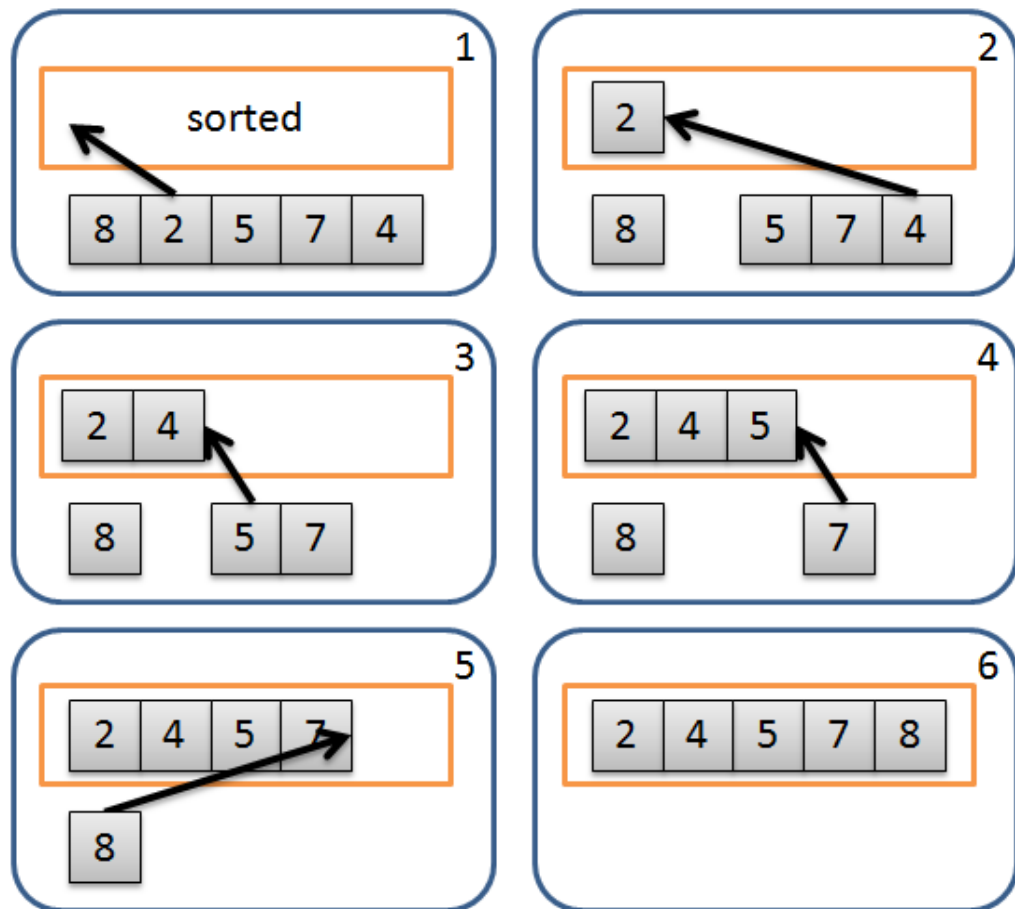


# 運算流程

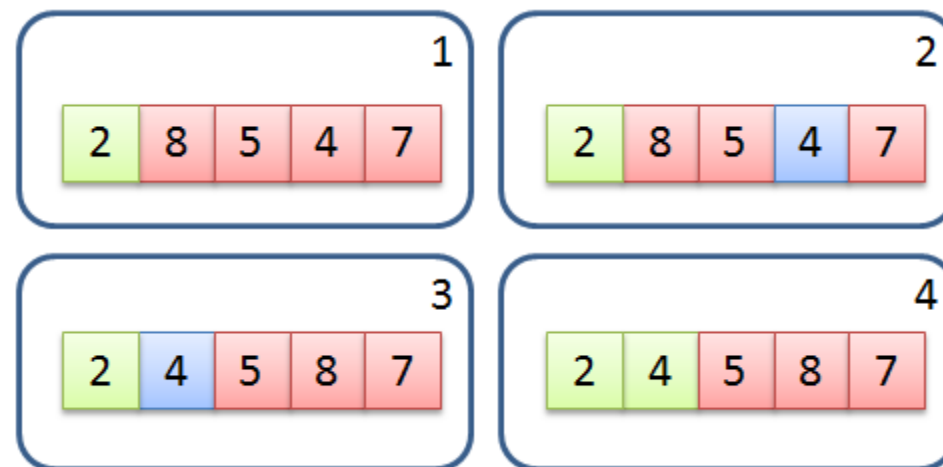
1. 從未排序的數列中找到最小的元素。
2. 將此元素取出並加入到已排序數列最後。
3. 重複以上動作直到未排序數列全部處理完成。

# 流程圖

## 額外空間

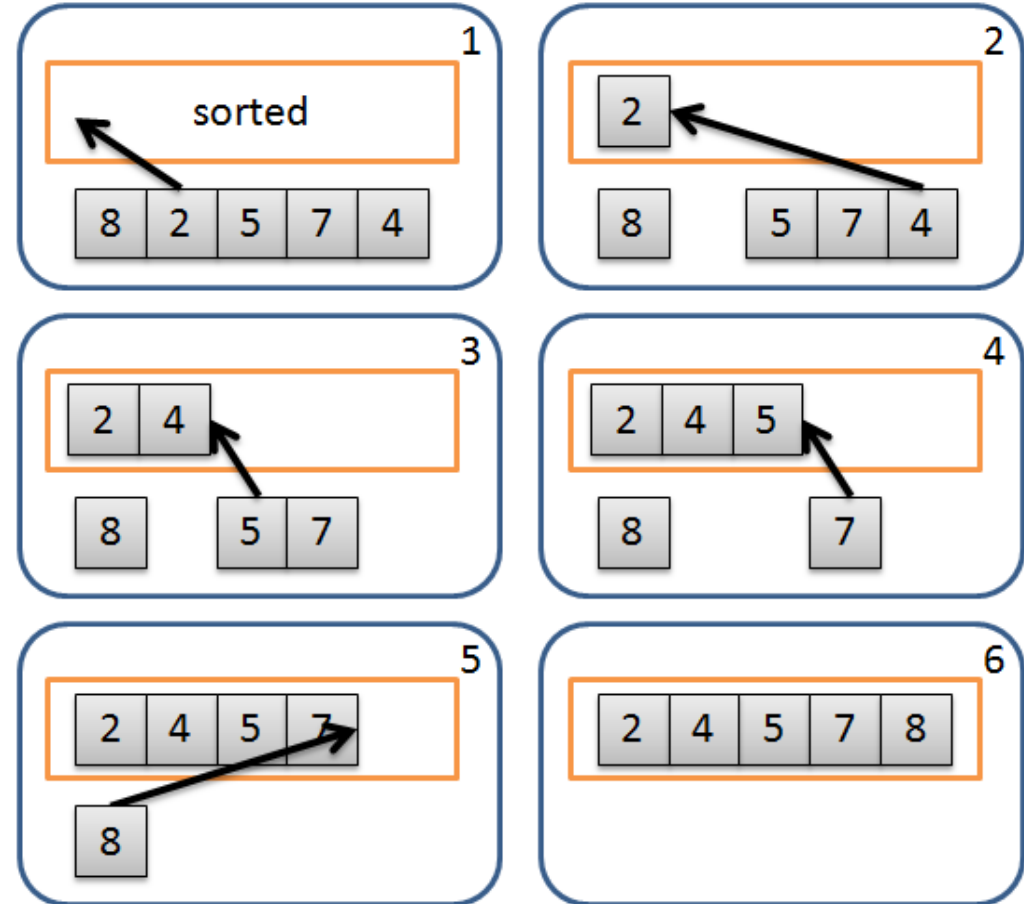


## In-place



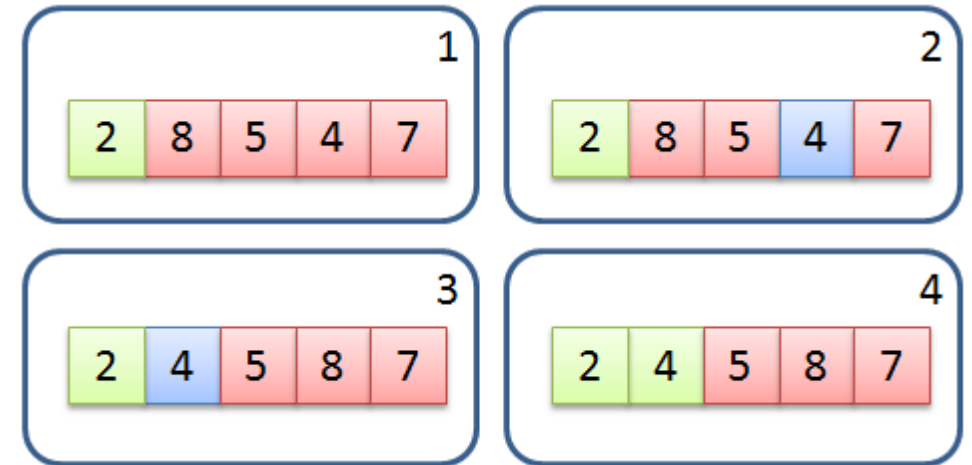
# 實作（額外空間）

```
function sort(list)
  var sorted = []
  while list.length > 0
    min = 取出 list 中最小元素
    sorted.add(min)
  end while
  list = sorted
end function
```



# 實作 (In-place)

```
function sort(list)
  for i = 0; i < list.length - 1; ++i
    minIndex = i;
    for j = i + 1; j < list.length; ++j
      if list[j] < list[minIndex]
        minIndex = j;
      end if
    end for
    if minIndex != i
      swap(list[minIndex], list[i]);
    end if
  end for
end function
```



# 複雜度分析

- 最佳時間複雜度： $O(n^2)$
- 平均時間複雜度： $O(n^2)$
- 最差時間複雜度： $O(n^2)$
- 空間複雜度： $O(1)$

# 泡泡排序法

# 簡介

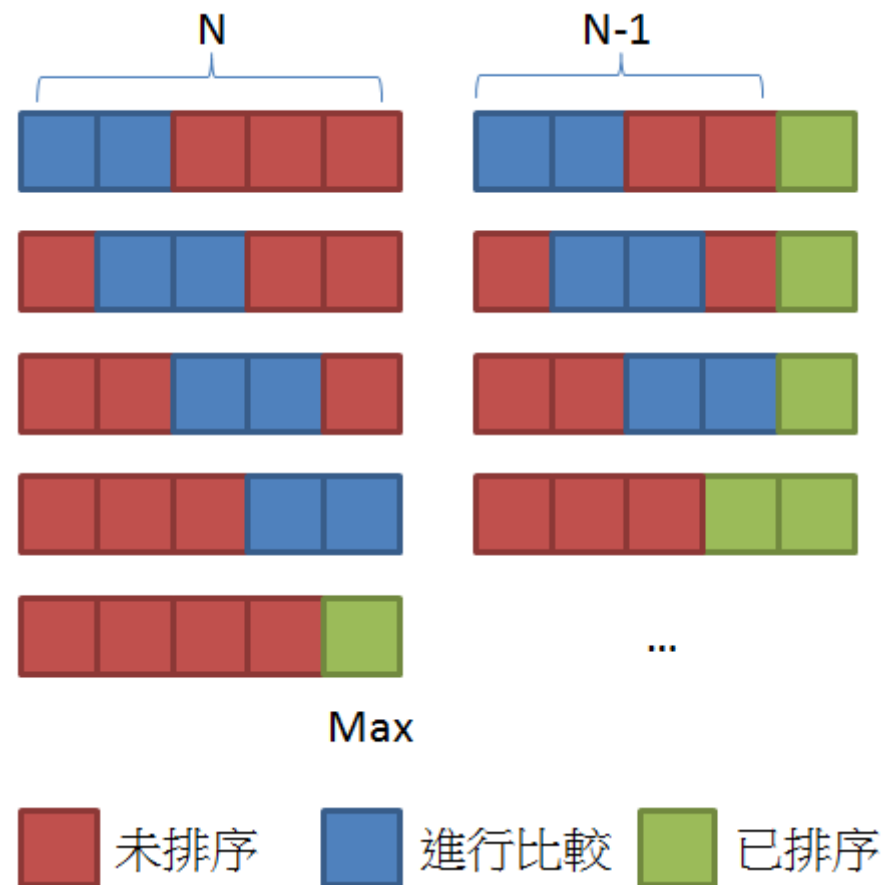
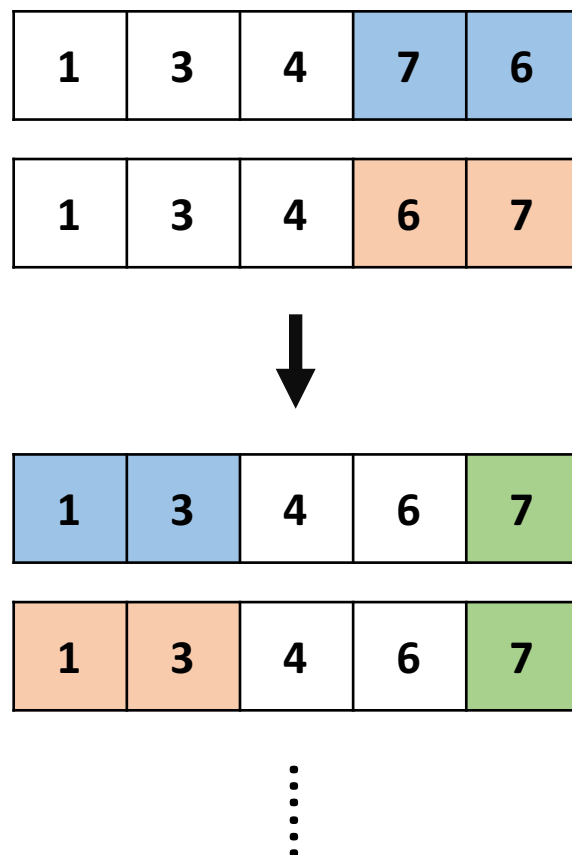
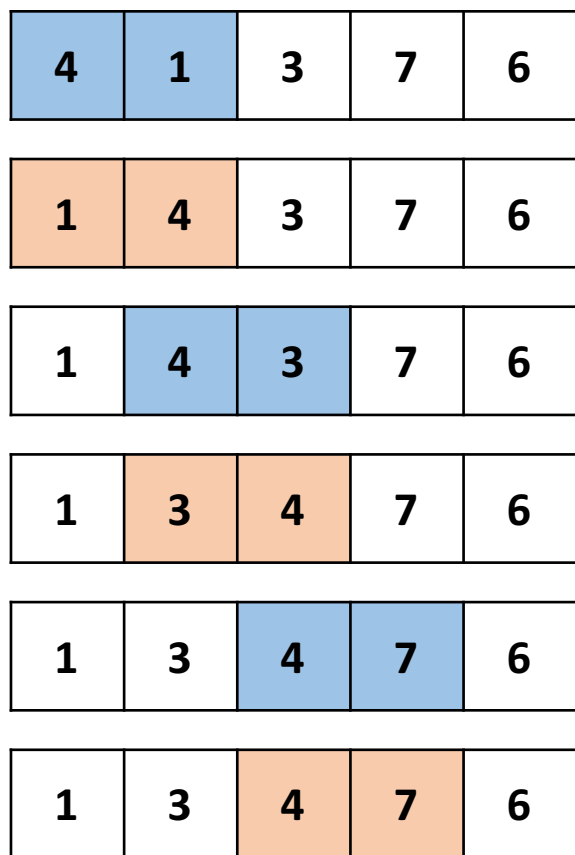
- **氣泡排序法(Bubble Sort)**是最容易理解和實作的一種**排序演算法**，也翻譯作冒泡排序法。
- 由於它很容易學習，所以也是許多演算法課程中第一個學習的排序演算法。
- 演算法過程會將**最大的數值移動到陣列最後面**，而較小的數值則**逐漸的往陣列前端移動**，就像有許多氣泡慢慢從底部浮出，因此成為氣泡排序法。

# 運算流程

1. 比較相鄰的兩個元素，若前面的元素較大就進行交換。
2. 重複進行1的動作直到最後面，最後一個元素將會是最大值。
3. 重複進行1,2的動作，每次比較到上一輪的最後一個元素。
4. 重複進行以上動作直到沒有元素需要比較。

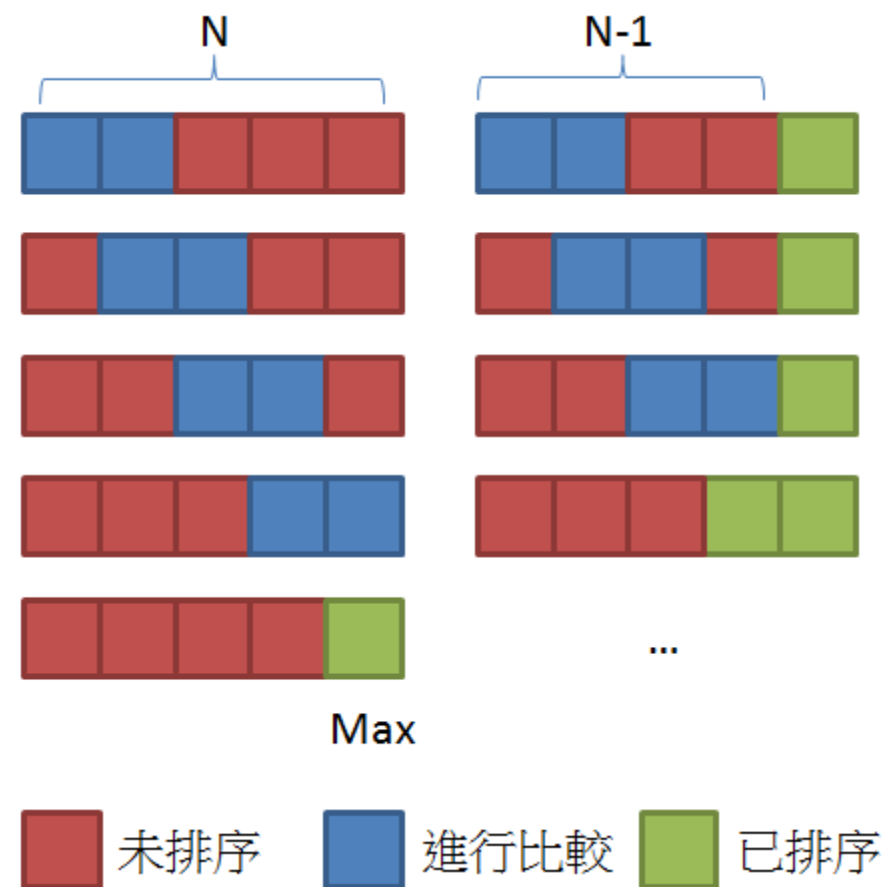


# 流程圖



# 實作

```
function sort(list)
    // 重複N次
    for i = 0 to list.length - 1
        // 比較到上一輪最後一個
        for j = 0 to list.length - i - 1
            if list[j] > list[j+1]
                swap (list[j], list[j+1])
            end if
        end for
    end for
end function
```



# 複雜度分析

- 最佳時間複雜度： $O(n \log n)$
- 平均時間複雜度： $O(n \log n)$
- 最差時間複雜度： $O(n^2)$
- 空間複雜度： $O(n)$

# 循序搜尋法

# 簡介

- **循序搜尋法 (Sequential search)**是相當直觀的**搜尋演算法**，又稱作**線性搜尋法**。
- 利用迴圈逐一比對要搜尋的資料，若相等則表示找到，並中斷迴圈，若全部都不等，則表示找不到。

# 實作

```
function search(list, target)
    // 重複N次
    for i = 0 to list.length - 1
        if list[i] == target
            return i
        end if
    end for
    return -1
end function
```

# 複雜度分析

- 最佳時間複雜度： $O(1)$
- 平均時間複雜度： $O(n)$
- 最差時間複雜度： $O(n)$

# 二元搜尋法



# 簡介

- **二元搜尋法 (Binary search)**是常用的**搜尋演算法**，又稱作二分搜尋法。二元搜尋法的原理與「終極密碼」的流程十分類似。
- 使用二元搜尋法前，**必須將資料排序**。
- 假設資料已從小到大排序。
- 演算法過程是**尋找編號上限與編號下限的中間項**，然後**比較中間項與搜索資料**。
  - 若中間項小於搜尋資料，則「 $\text{編號下限} = \text{中間項編號} + 1$ 」
  - 若中間項大於搜尋資料，則「 $\text{編號上限} = \text{中間項編號} - 1$ 」
  - 若相等，則尋找到資料。

# 流程

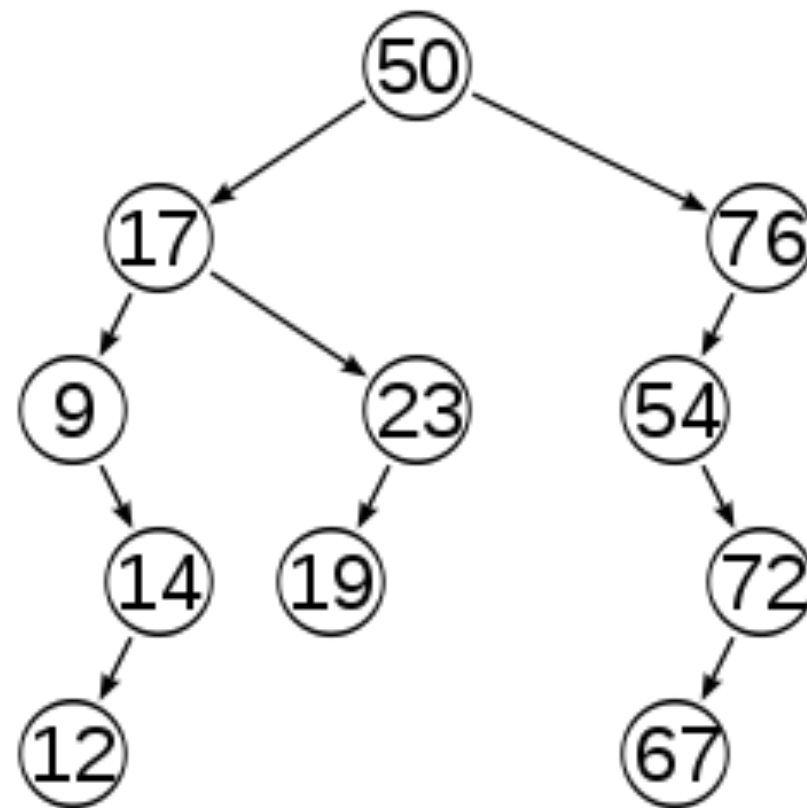
- Algorithm Visualizations
  - <https://www.cs.usfca.edu/~galles/visualization/Search.html>

# 實作

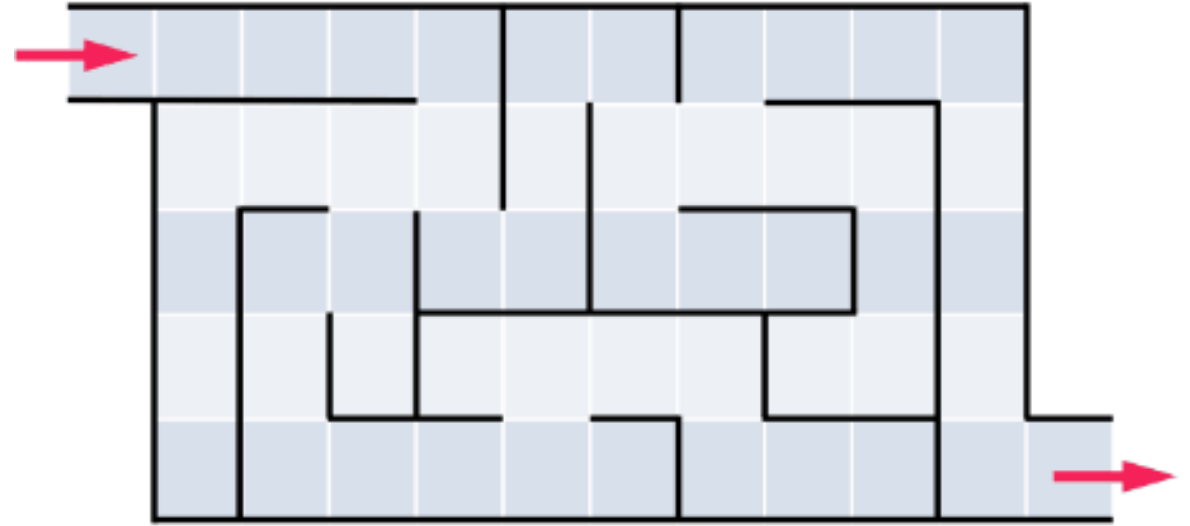
```
function binary_search(array, target) {  
  var L = 0, R = array.length - 1;  
  while(L<=R) {  
    var M = Math.floor((L+R)/2);  
    if(array[M]==target){  
      return M;  
    } else if(array[M]>target) {  
      R = M - 1;  
    } else {  
      L = M + 1;  
    }  
  }  
  return -1;  
}
```

# 樹 (tree)

一種資料結構，節點 (node) 與節點之間，以邊 (edge) 相連。

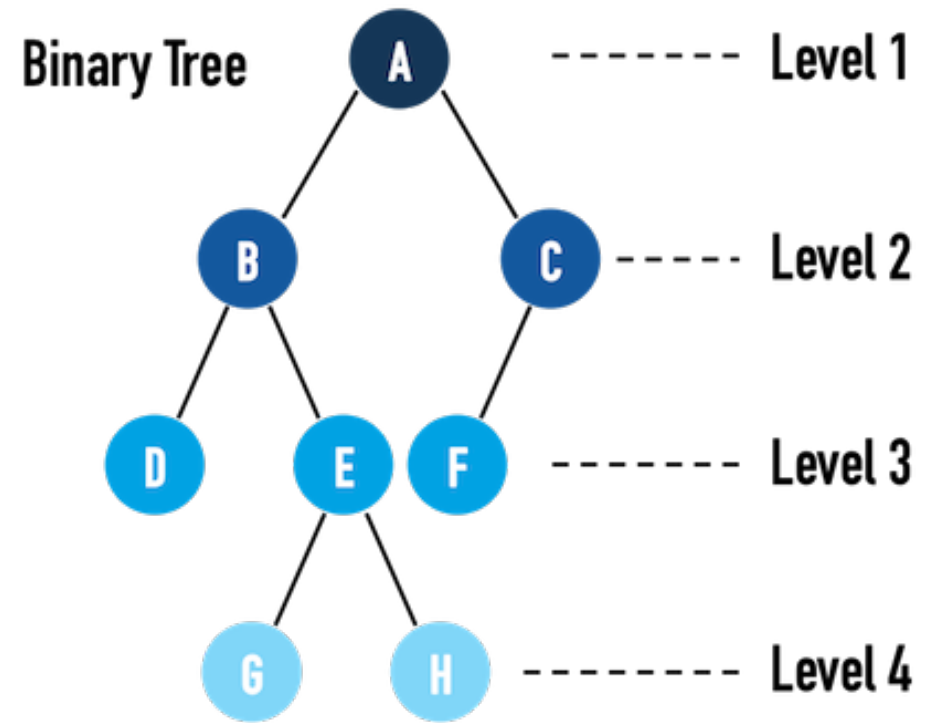


## Maze Problem



# DFS (Depth-First-Search)

沿著樹的深度遍歷樹的節點，儘可能深的搜尋樹的分支。當節點 $v$ 的所在邊都已被探尋過，搜尋將回溯到發現節點 $v$ 的那條邊的起始節點。



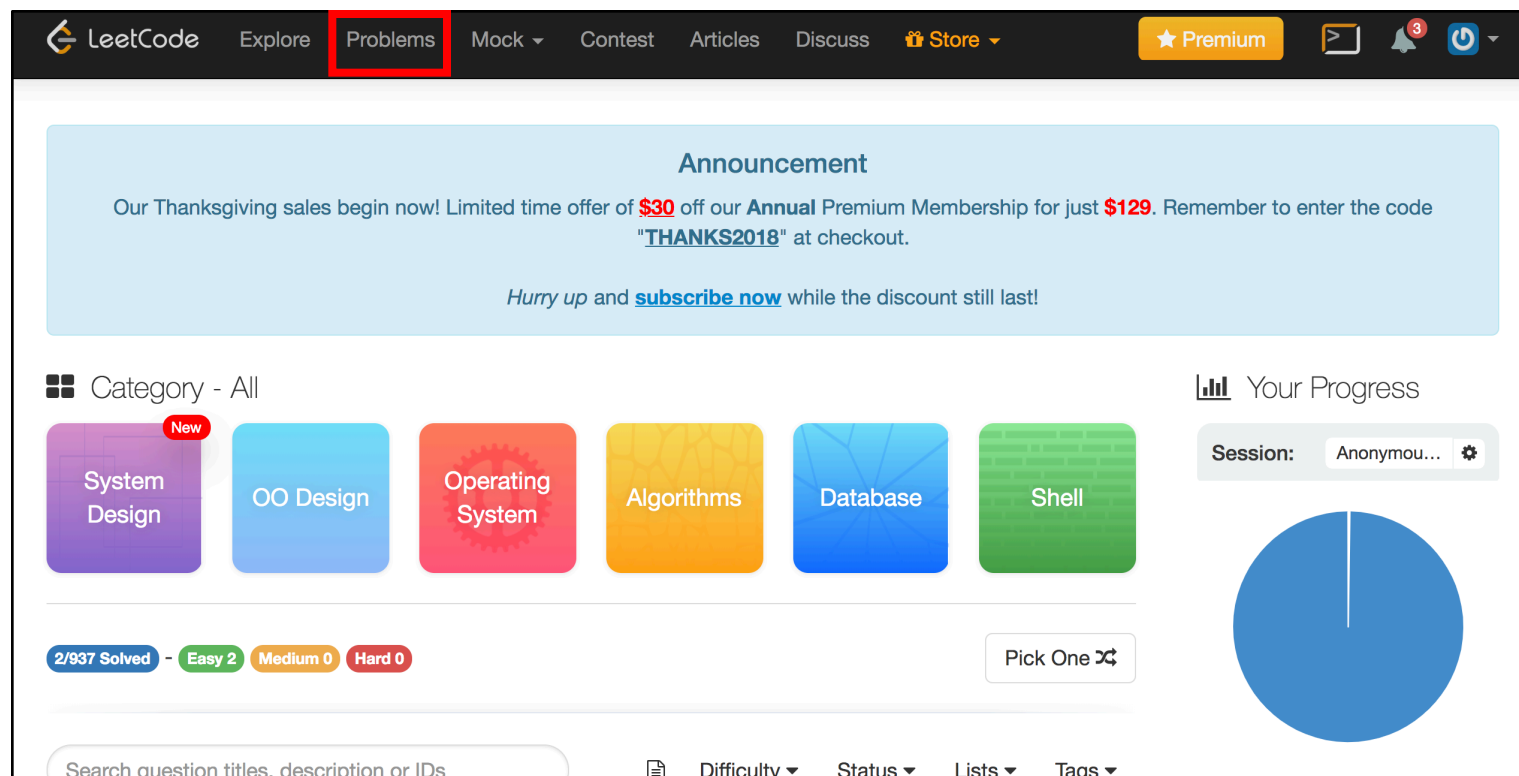
# BFS (Breadth-First Search)

從根節點開始，沿著樹的寬度遍歷樹的節點。如果所有節點均被存取，則演算法中止。

LeetCode











# LeetCode

- 可以實作更多的演算法的網站。
- ***LeetCode -> Problems***





# LeetCode

					Frequency
#	Title	Solution	Acceptance	Difficulty	🔒
✓ 1	Two Sum		39.2%	Easy	
2	Add Two Numbers		29.5%	Medium	
3	Longest Substring Without Repeating Characters		25.3%	Medium	
4	Median of Two Sorted Arrays		24.4%	Hard	
5	Longest Palindromic Substring		25.7%	Medium	
6	ZigZag Conversion		29.2%	Medium	
7	Reverse Integer		24.6%	Easy	
8	String to Integer (atoi)		14.2%	Medium	
9	Palindrome Number		39.6%	Easy	
10	Regular Expression Matching		24.5%	Hard	
11	Container With Most Water		40.2%	Medium	
12	Integer to Roman		48.4%	Medium	

# LeetCode

Description

Solution

Submissions

Discuss (999+)

i C++

## 1. Two Sum

Easy 8978 265

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have **exactly** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,  
return [0, 1].

```
1 class Solution {
2 public:
3     vector<int> twoSum(vector<int>& nums, int target) {
4         for (int i = 0 ; i < nums.size() ; i++){
5             for (int j = i+1 ; j < nums.size() ; j++){
6                 if (nums[i] + nums[j] == target){
7                     vector<int> ans {i,j};
8                     return ans;
9                 }
10            }
11        }
12    }
13};
```

Your previous code was restored from your local storage. [Undo](#)

1/937

Console

Contribute i

Run Code

Submit

# LeetCode

Description

Solution

Submissions

Discuss (999+)

Quick Navigation

View in Article

★★★★★ Average Rating: 4.83 (1620 votes)

## Solution

### Approach 1: Brute Force

The brute force approach is simple. Loop through each element  $x$  and find if there is another value that equals to  $target - x$ .

Java

```
1 public int[] twoSum(int[] nums, int target) {  
2     for (int i = 0; i < nums.length; i++) {  
3         for (int j = i + 1; j < nums.length; j++)  
4             ;  
5     }  
6 }
```

Copy

i C++

```
1 class Solution {  
2 public:  
3     vector<int> twoSum(vector<int>& nums, int target) {  
4         for (int i = 0 ; i < nums.size() ; i++){  
5             for (int j = i+1 ; j < nums.size() ; j++){  
6                 if (nums[i] + nums[j] == target){  
7                     vector<int> ans {i,j};  
8                     return ans;  
9                 }  
10            }  
11        }  
12    }  
13 };
```

Your previous code was restored from your local storage.

[Undo](#)

×

⋮

⌕

<

1/937

>

Console ▾

Contribute i

▶ Run Code

Submit

# LeetCode

Description

Solution


Submissions

Discuss (999+)

1. Two Sum

Recent Activity

New +




How much faster can this get ?

▲ 0

👁 19

jagzviruz created at: 3 hours ago | No replies yet.




$O(n\log(n))$  solution in python

▲ 0

👁 18

utferris created at: 4 hours ago | No replies yet.




Accepted C++  $O(n)$  Solution

▲ 301

👁 131.3K

naveed.zafar created at: September 13, 2014 12:57 AM | Last Reply: DemonSalute 5 hours ago



Runtime: 20 ms, faster than 100.00% of Python online submissions for Two Sum.

▲ 0

👁 125

lei1989 created at: 16 hours ago | Last Reply: Richard\_Hsu 5 hours ago

<https://leetcode.com/problems/two-sum/>

# 補充資料

- 演算法筆記：<http://www.csie.ntnu.edu.tw/~u91029/>
- 演算法與資料結構：<http://alrightchiu.github.io/SecondRound/mu-lu-yan-suan-fa-yu-zi-liao-jie-gou.html>

獨數解

# 程式設計思路

- 對於每一個空格，分別確認

- 同一個九宮格

- 同一橫列

- 同一直行

● = 1, 2, 3, 4, 5, 6, 7, 8, 9

5	3	●		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

# 程式設計思路

- 對於每一個空格，分別確認
  - 同一個九宮格
  - 同一橫列
  - 同一直行

● = 1, 2, 3, 4, 5, 6, 7, 8, 9


5	3	●		7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



# 程式設計思路

- 對於每一個空格，分別確認
  - 同一個九宮格
  - 同一橫列
  - 同一直行

 = 1, 2, 3, 4, 5, 6, 7, 8, 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

# 程式設計思路


- 對於每一個空格，分別確認

- 同一個九宮格

- 同一橫列

- 同一直行

 = 1, 2, 3, 4, 5, 6, 7, 8, 9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

# 測試資料

- [https://github.com/ck1001099/cpp\\_course\\_2019summer](https://github.com/ck1001099/cpp_course_2019summer)
- lesson8 -> Challenge -> testcase1.in
- lesson8 -> Challenge -> testcase2.in

# 作業

- 從以下題目任選兩題完成，下次上課時檢查。
  - a059：完全平方和
  - b680：百米賽道編排
  - d550：物件排序
  - a225：明明愛排列
  - d555：平面上的極大點
- Reading: 課本7.5
- 若遇到作業問題，歡迎隨時寄信至：[ck1001099@gmail.com](mailto:ck1001099@gmail.com)