建議對演算法與數據結構已經有足夠熟悉的學員跳過此章節

主要原因是我們課程的設計核心為實際去模擬當你真實遇到該問題時會有什麼樣的想法，但講義內容其實已經有部分暗示解法，但實際遇到問題時並不會知道接下來的題目會用到解法的方向。

Binary Search

很多題目可能第一眼會覺得跟 binary search 無關，但只要將題目換另一種形式呈現，就可以用 binary search 解決。
簡單來說能夠使用 binary search 的題目基本上都能簡化（參考 Reduction 章節）成能夠以讓範圍內任一個數字x，配上一個條件f(x) `True/False` 來思考的題目，當f(a)為True時，任何一個i>=a，f(i)都會是True，而當f(a)為False時，任何一個i<=a，f(i)都會是false，而我們的目的就是要知道所有的f(i)，並以此推算出所需要的答案。而大約 80% 的題目都可用切成兩半和符合條件的最小(大)值來簡化。

切成兩半範例

用一個簡單的題目說明：
給定一個被 rotated 的 sorted array，請問這個 array 被 rotated 幾次？（rotate 一次代表最左邊的數字會跑到最右邊）

這個題目其實只要找到最小的數字（原本最左邊）跑到哪個位置就可以知道 array 被 rotated 幾次。也就是可以把題目改成“尋找 array 中的最小值” 或是 “尋找 array 中第一個比目前最左邊小的數字”

轉換題目後我們以目前最左邊的數字為基準，大於等於他的數字為 True，小於他的為 False。在搜尋的過程中如果搜到的數字是 True 那代表左邊的數字也都會是 True（比基準大），因此可以忽視左半段只搜右半段 array。相反的，如果搜尋到的數字是 False，那代表右邊的數字也都會是 False（比基準小），因此可以忽視右半段只搜左半段。

以下使用 test case 講解：

```
Input: [3,4,5,1,2]
Output: 2

以下 l 代表左邊界，r 代表右邊界，T/F 分別代表 True/False

l=0 r=5       l=2 r=5       l=2 r=3
3,4,5,1,2 -> 3,4,5,1,2 -> 3,4,5,1,2
T ? ? ? ?     T T T ? ?     T T T F F

填完所有後 F 第一個出現的地方就是最小值
```

符合條件的符合條件的最小(大)值範例

這種類型題目通常會給一些條件你要在符合條件的情況下讓最後的值最小或最大，但需注意該答案有單調性(monotonic)，舉最小值為例，假設有一條公路長度為 n 公里，我們想知道每小時至少要跑幾公里，才能在 t 小時內時間走完他，那我們可以知道需要 n/t 公里/小時的速度才能滿足我們的需求，而我們發現若速度 > n/t 公里/小時也可以滿足我們的需求，這就是所謂的單調性(monotonic)

下面給一道題目做為範例：
給定若干跟長度不同的竿子，你總共可以切 n 刀，要怎麼在切完之後讓最長的竿子的值最小？
通常看到這個題目會覺得這跟 binary search 完全沒關，畢竟直觀來看完全沒一串數字讓我們來"搜尋"。但其實以上題目可以轉換成這種形式：給你 n 刀的機會，有辦法讓最長的竿子的長度為 x 公分嗎？
假設竿子最長是 10 公分，那答案可能會是 1 到 10 公分。如果 $x = 5$ 可以做到，就代表比 x 大的數值都可以（也就是右半部都是 True），相反的如果 $x = 5$ 不能做到，就代表比 x 小的數值都不能（也就是左半部都是 False）

Pseudo Implementation

以下說明 *r,l* 兩個指標移動的位置，配合 True/False 去判斷就能夠知道自己的指標目前指到哪裡。（以下以左邊是 true 右邊是 false 舉例）

Scenario one

```
while((r-l)>1) {
    mid = (l+r) / 2
    if(match condition){
        l = mid //把 l 移到最後一個 true
    } else {
        r = mid //把 r 移到第一個 false
    }
}
```

Scenario two

```
while(l<=r) {
    mid = (l+r) / 2
    if(match condition){
        l = mid + 1 //把 l 移到最後一個 true 的下一格
    } else {
        r = mid - 1 //把 r 移到第一個 false 的前一格
    }
}
```

Binary Search implementation tips

在實作 Binary Search 時，許多人最大的困惑之一，是 迴圈的結束條件 以及 `left`、`right` 指標的定位。如果這部分想不清楚，就很容易陷入錯誤。以下為各個條件下的思考方式。

1. 把 Binary Search 想像成 True/False 陣列搜尋

假設我們的搜尋條件可以被抽象成一個True/False陣列，整個過程就像在一個形狀為

`TTTT???FFFF`

的序列中尋找所有`?`的答案（`?`的位置是未知的 True 或 False）。
Binary Search 的核心，就是不斷縮小範圍，直到知道所有位子的答案是T還是F。

在這個模型下，你可以明確定義：

- **`left`** 指標：代表第一個問號的位置，或是最後一個 True 的位置。
- **`right`** 指標：代表最後一個問號的位置，或是第一個 False 的位置。

2. 思考 `mid` 判斷後該怎麼移動

每次計算 `mid` 並檢查條件時：

- 如果 `mid` 的結果是 **True**，代表分界點在右側，因此你應該移動 `left`，並維持`left` 是最後一個 True或是第一個?的位置的性質。
- 如果 `mid` 的結果是 **False**，代表分界點在左側，因此你應該移動 `right`，並維持 `right` 是第一個 False或是最後一個?的位置的性質。

關鍵就是：無論怎麼更新，都要確保 **`left`** 與 **`right`** 的語意一致，不要中途改變定義。

下面舉幾個例子

若搜尋途中整個True/False 陣列的形狀為這個樣子，而紅色的字為當前 `left` 跟 `right`的位置

`TTTT???FFFF`

當我們搜尋 middle 為T時，整個陣列會變成

`TTTTTT?FFFF`

而 middle則指向藍色的位子，為了維持`left` 與 `right` 的語意一致，所以這時候我們就要將 `left = middle`

另一個例子，若搜尋途中整個True/False 陣列的形狀為這個樣子，而紅色的字為當前 `left` 跟 `right`的位置

TTTT???FFFF

當我們搜尋 middle 為T時，整個陣列會變成

TTTTTT?FFFF

而 middle則指向藍色的位子，為了維持 `left` 與 `right` 的語意一致，所以這時候我們就要將 `left = middle + 1`

3. 迴圈結束條件的判斷

當搜尋結束時，True/False 陣列會收斂成類似：

TTTTFFFF

這時可以去想 `left` 與 `right` 照我們剛剛的語意，他的位子會在哪裡，譬如說若 `left` 在最後一個T(紅色位子)而 `right` 在第一個F(藍色位子)，

TTTTFFFF
因此就可以知道結束條件為 `left` 和 `right` 相差為1，因此條件就會是 `while(right - left > 1)` 也就是當他們相差>1時應該繼續搜尋

而另一個例子

這時可以去想 `left` 與 `right` 照我們剛剛的語意，他的位子會在哪裡，譬如說若 `left` 在第一個問號，這時會發現第一個問號其實是最後一個T後面那個位子(紅色位子)，而 `right` 在最後一個問號，這時會發現最後一個問號其實是第一個F前面那個位子(藍色位子)。

TTTTFFFF

因此就可以知道結束條件為 `left` > `right` 時，因此條件就會是 `while(left <= right)`也就是當 `left` 還沒大於 `right` 時就要繼續搜尋。