排序與搜尋
Sorting & Searching

Eason

February 5, 2024

Outline

排序的意義

排序的意義

- 讓你的序列以一種特定的方式來排列
- 特定的方式可以是：數字大小關係、字典序等等
- 排序的目的是為了增加搜尋的效率
- 目前世界上有好幾十種的排序演算法，接下來我會挑其中三個比較常用的來介紹
- 為了方便介紹，以下會用由小到大排序來講解

氣泡排序法 Bubble Sort

氣泡排序法

核心想法

每次都把當前最大的元素推到陣列的最後面

- 先來看個影片 `Bubble_Sort`

氣泡排序法

核心想法

每次都把當前最大的元素推到陣列的最後面

- 先來看個影片 `Bubble_Sort`
- 顧名思義就像氣泡一樣，從水底慢慢浮上去

氣泡排序法

核心想法

每次都把當前最大的元素推到陣列的最後面

- 先來看個影片 `Bubble_Sort`
- 顧名思義就像氣泡一樣，從水底慢慢浮上去
- 想像今天有一個長度為 `n` 的陣列要進行氣泡排序

氣泡排序法

核心想法

每次都把當前最大的元素推到陣列的最後面

- 先來看個影片 `Bubble_Sort`
- 顧名思義就像氣泡一樣，從水底慢慢浮上去
- 想像今天有一個長度為 `n` 的陣列要進行氣泡排序
- 第 `k` 輪就把第 `k` 大的元素推到右邊第 `k` 個 $(\mathtt{k} = 1 \sim \mathtt{n})$

氣泡排序法

核心想法

每次都把當前最大的元素推到陣列的最後面

- 先來看個影片 Bubble_Sort
- 顧名思義就像氣泡一樣，從水底慢慢浮上去
- 想像今天有一個長度為 n 的陣列要進行氣泡排序
- 第 k 輪就把第 k 大的元素推到右邊第 k 個 $(k = 1 \sim n)$
- 那經過 n 輪之後，整個陣列就排序好了
- 時間複雜度：$O(n^2)$
- 參考程式碼：bubble_sort.cpp

合併排序法 Merge Sort

合併排序法

核心想法

將陣列拆解成多個小元素，並在合併時將元素進行排序

- 每次都將陣列一分為二，分到不能分為止

合併排序法

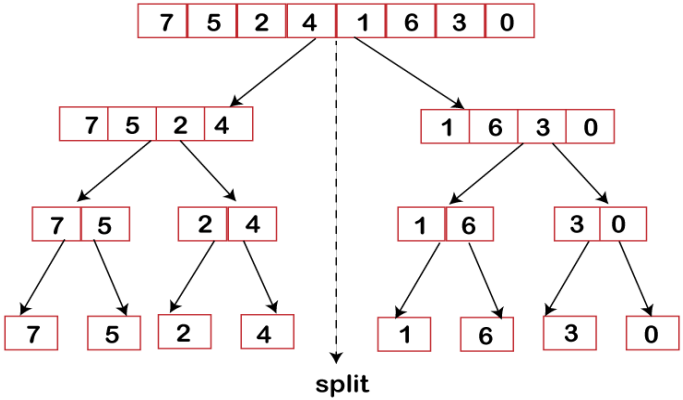


Figure 1: Merge Sort Divide Phase

合併排序法

核心想法

將陣列拆解成多個小元素，並在合併時將元素進行排序

- 最後再兩兩合併，並且讓合併後的陣列維持已排序的狀態

合併排序法

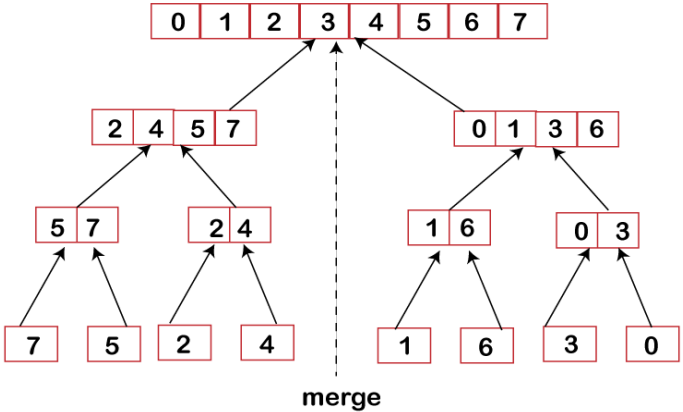


Figure 2: Merge Sort Combine Phase

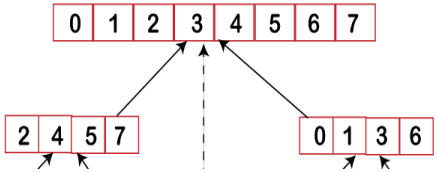
合併排序法

核心想法

將陣列拆解成多個小元素，並在合併時將元素進行排序

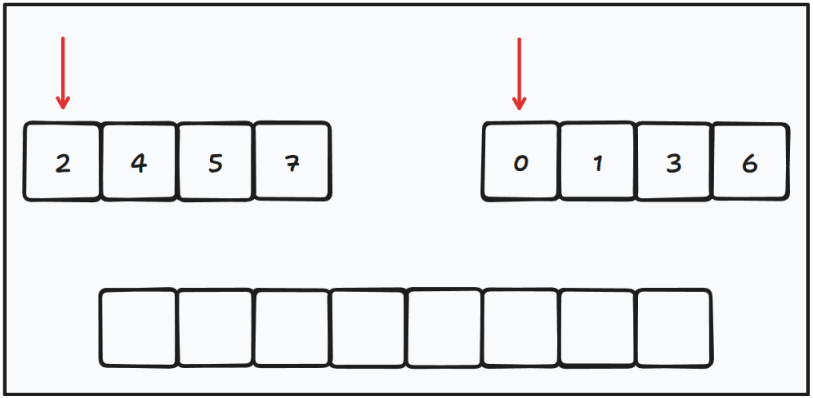
- 到這邊應該可以想到直接用遞迴來實作
- 拆解不難，合併也不難
- 重點在於如何在合併的同時將陣列排序

合併排序法

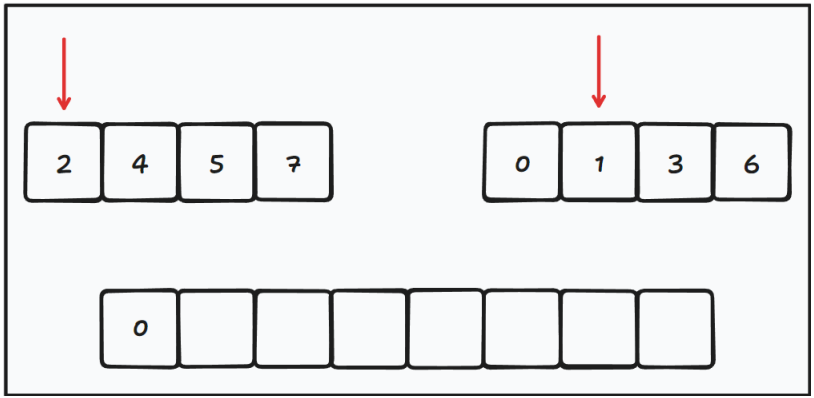


- 以這塊來舉例
- 把問題簡化成：如何將兩個已經排序好的陣列，合併成一個排序好的陣列
- 可以發現只需要把兩個陣列中最小的抓出來，再一個一個放到最終的陣列

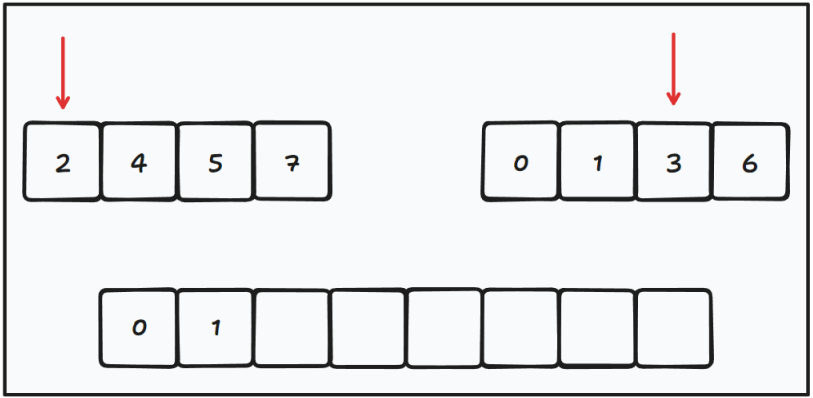
合併排序法



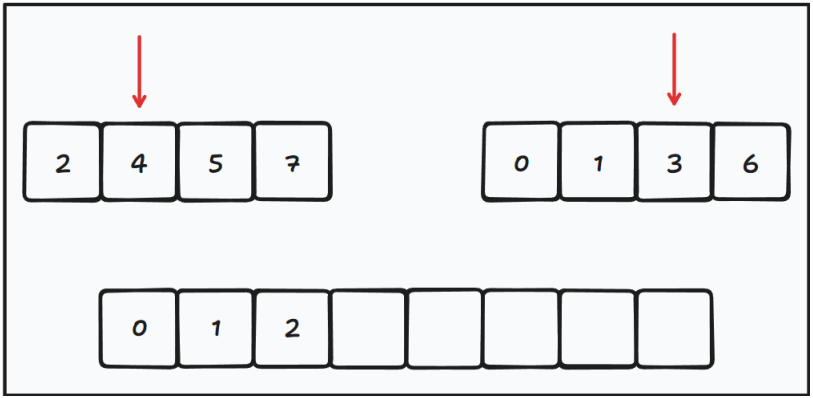
合併排序法



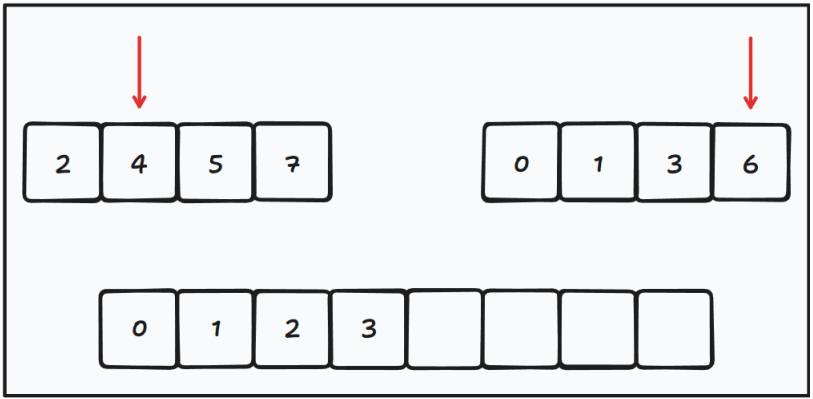
合併排序法



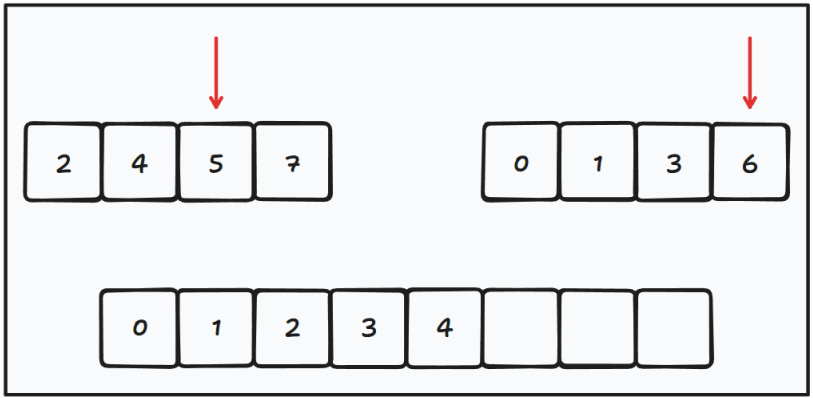
合併排序法



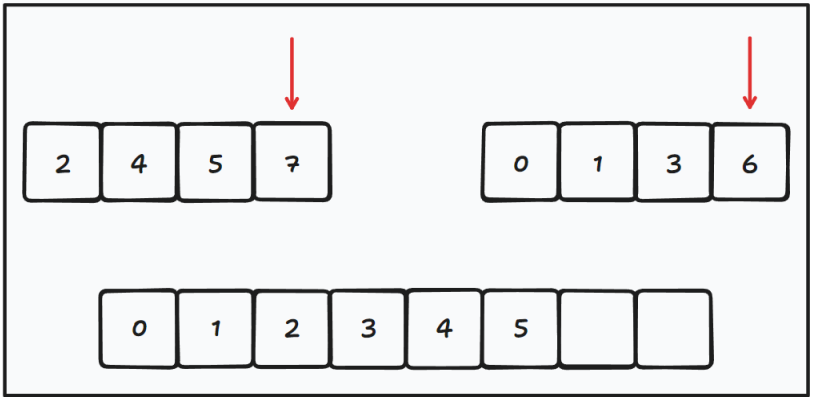
合併排序法



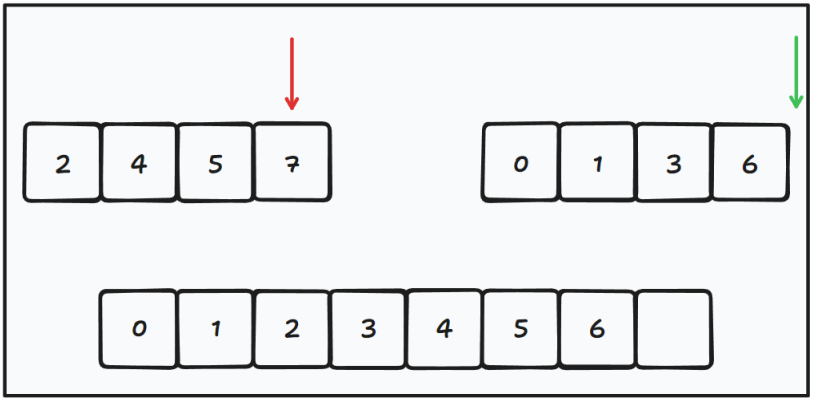
合併排序法



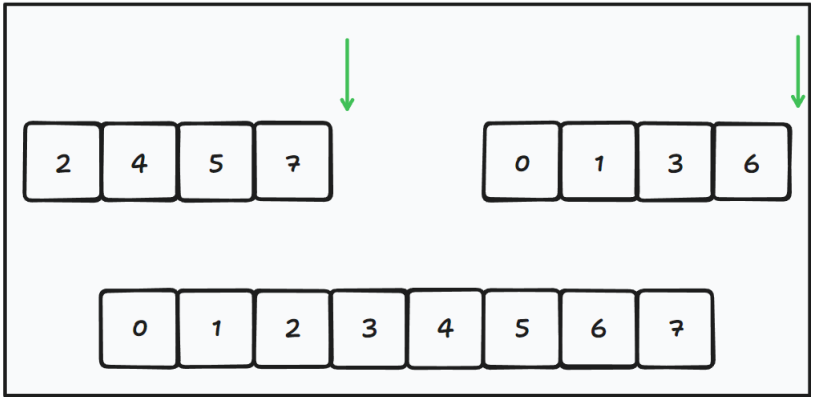
合併排序法



合併排序法



合併排序法



合併排序法

- 在實作上其實只需遍歷兩個陣列比大小即可
- 為了確保完整遍歷兩個陣列，所以在比完大小之後要多寫兩個迴圈，讓指標指到底
- 參考程式碼：`merge_sort.cpp`

合併排序法

- 每層需要 $O(n)$ 的時間來合併
- 總共有 $\log_2 n$ 層，所以整體的時間複雜度是 $O(n \log n)$
- 算是目前數一數二快的排序演算法

合併排序法

- 每層需要 $O(n)$ 的時間來合併
- 總共有 $\log_2 n$ 層，所以整體的時間複雜度是 $O(n \log n)$
- 算是目前數一數二快的排序演算法
- 合併排序算是分治演算法的其中一種
- 所以把合併排序的實作搞懂，分治其實也算學會 8 成了

練習題

TIOJ 1287

排序裸題

TIOJ 1080 分治經典題：逆序數對

在數列 S 中，如果 S 的第 i 項及第 j 項滿足 $S_i > S_j$ 且 $i < j$，則 (i, j) 為一組逆序數對，找出 S 中共有幾組逆序數對

CSES 1095 Exponentiation 分治經典題：快速冪

給你 n 組 (a, b)，求每一組 a^b 除以 $10^9 + 7$ 的餘數

CSES 1643 分治經典題：最大連續和

給你 n 個數字，找出這 n 個數字的最大連續和

快速排序法 Quick Sort

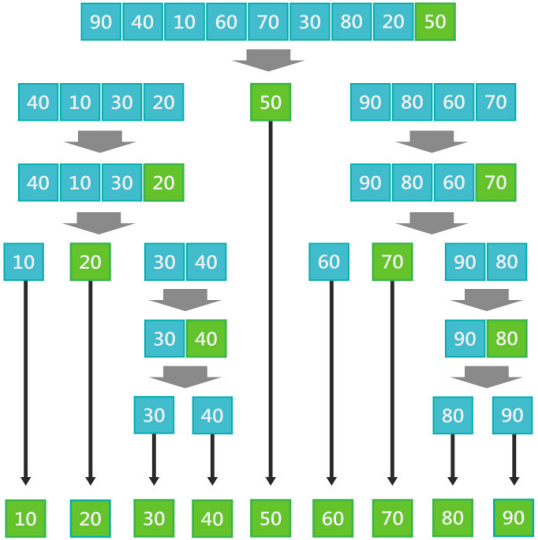
快速排序法

- 快速排序的實現方式也是利用分治演算法
- 但是它比 Merge Sort 少了一個遞迴的常數
- 所以它在絕大多數的情況會比 Merge Sort 還快

快速排序法：實作流程

1. 隨機在陣列中挑一個基準值
2. 把所有比基準值小的元素放在它的左邊
3. 再把所有比基準值大的元素放在它的右邊
   - 對左邊的所有元素重複上述步驟
   - 對右邊的所有元素重複上述步驟

快速排序法：圖解


90 40 10 60 70 30 80 20 50
40 10 30 20
50
90 80 60 70
40 10 30 20
90 80 60 70
10
20
30 40
60
70
90 80
30 40
90 80
30
40
80
90
10
20
30
40
50
60
70
80
90

快速排序法：複雜度分析

- 平均複雜度：$O(n \log n)$
- 最差複雜度：$O(n^2)$
- 如果選基準值的順序恰好是排序好的順序
- 那複雜度就會直接飆到 $O(n^2)$
- 但是發生的機率極低，所以可以放心使用

內建的排序法

內建的排序法

- 手刻排序好麻煩，有沒有更快的方法
- 那就使用 `std::sort()`
- 在使用之前要先 `#include<algorithm>`
- 我相信大家都會用 ~~`using namespace std;`~~
- 所以以下就簡稱 ~~`sort()`~~

sort()

使用方法

```
sort (first, last, cmp);
```

- `first` 起始位置，以記憶體位置來表示

sort()

使用方法

```
sort (first, last, cmp);
```

- first 起始位置，以記憶體位置來表示
- last 終止位置，也是以記憶體位置來表示

sort()

使用方法

```
sort (first, last, cmp);
```

- `first` 起始位置，以記憶體位置來表示
- `last` 終止位置，也是以記憶體位置來表示
- `cmp` 比較函式，決定要用甚麼方式排序，預設由小到大

sort()

使用方法

```
sort (first, last, cmp);
```

- first 起始位置，以記憶體位置來表示
- last 終止位置，也是以記憶體位置來表示
- cmp 比較函式，決定要用甚麼方式排序，預設由小到大
- 簡單來說 sort() 會把 [first, last) 之間的元素以 cmp 進行排序

sort()

使用方法

```
sort (first, last, cmp);
```

- first 起始位置，以記憶體位置來表示
- last 終止位置，也是以記憶體位置來表示
- cmp 比較函式，決定要用甚麼方式排序，預設由小到大
- 簡單來說 sort() 會把 [first, last) 之間的元素以 cmp 進行排序
- 而 sort() 的排序演算法正是快速排序
- 直接來看參考程式碼：std_sort.cpp

cmp

- cmp 函式須為 bool 型態
- cmp 需要傳入兩個參數，參數的型態要跟數列的型態相同
- cmp 回傳值表示該順序是否正確。true 表示正確、false 表示錯誤
- 若是錯誤，則會把 cmp 參數的兩個值對調
- 透過 cmp，我們可以將較複雜的結構，如 pair、tuple 或甚至是自己寫的 struct 自訂排序方式

stable_sort()

- 用法跟 sort() 相同
- 差別在於 stable_sort() 是以 Merge Sort 實作
- 如果真的很怕 sort() 跑到 $O(n^2)$ 再來用這個

二分搜尋法 Binary Search

二分搜

終極密碼

從 $1 \sim 100$ 中猜一個數字，假設你猜到 k，我會告訴你答案在 $1 \sim k$ 還是 $k \sim 100$

- 相信大家小時候都玩過這個猜數字的遊戲

二分搜

終極密碼

從 $1 \sim 100$ 中猜一個數字，假設你猜到 k，我會告訴你答案在 $1 \sim k$ 還是 $k \sim 100$

- 相信大家小時候都玩過這個猜數字的遊戲
- 你有想過這個遊戲的最佳策略是甚麼嗎？

二分搜

終極密碼

從 $1 \sim 100$ 中猜一個數字，假設你猜到 k，我會告訴你答案在 $1 \sim k$ 還是 $k \sim 100$

- 相信大家小時候都玩過這個猜數字的遊戲
- 你有想過這個遊戲的最佳策略是甚麼嗎？
- 其實就是每次都選擇中位數

二分搜

終極密碼

從 $1 \sim 100$ 中猜一個數字，假設你猜到 k，我會告訴你答案在 $1 \sim k$ 還是 $k \sim 100$

- 相信大家小時候都玩過這個猜數字的遊戲
- 你有想過這個遊戲的最佳策略是甚麼嗎？
- 其實就是每次都選擇中位數
- 這樣能確保每次都可以把搜尋範圍砍半

二分搜

終極密碼

從 $1 \sim 100$ 中猜一個數字，假設你猜到 k，我會告訴你答案在 $1 \sim k$ 還是 $k \sim 100$

- 相信大家小時候都玩過這個猜數字的遊戲
- 你有想過這個遊戲的最佳策略是甚麼嗎？
- 其實就是每次都選擇中位數
- 這樣能確保每次都可以把搜尋範圍砍半
- 而這就是二分搜在做的事

二分搜的作用

- 可以找出某個元素是否在陣列中
- 可以找出元素在陣列排序後的位置
- 而且複雜度很優

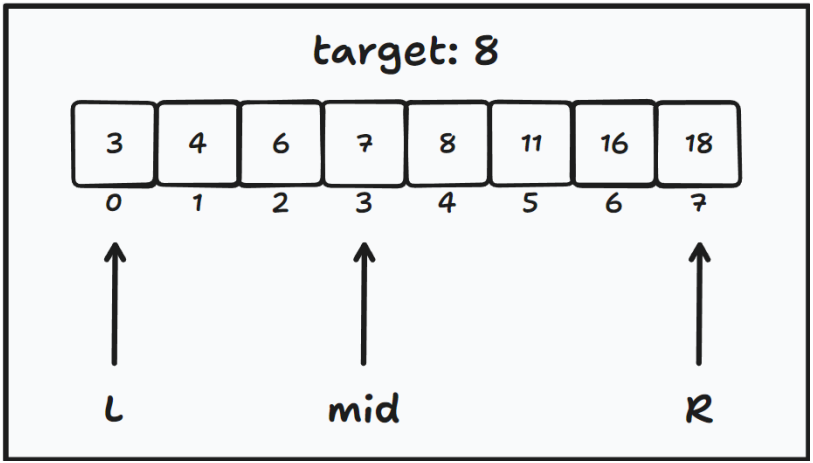
實作二分搜

- 在實作之前要先確認陣列具有單調性
- 換句話說，陣列要保持非嚴格遞增或非嚴格遞減
- 還是很文言文 XD
- 簡單來說就是要先把陣列排序好

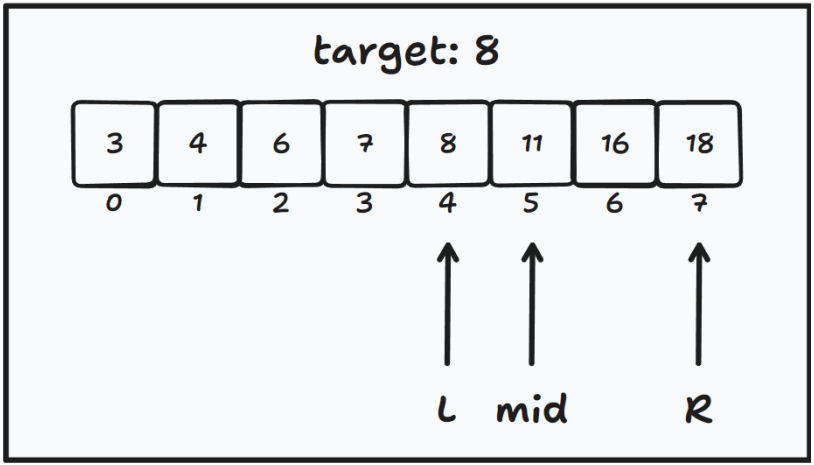
實作二分搜

- 先定義左界 `L` 為陣列的頭、右界 `R` 為陣列的尾
- 而中間值 `mid` 為左右界平均，即 $(L + R)/2$
- 比較 `mid` 與答案 `target` 的關係
  - `target < arr[mid]`：將 `R` 設為 `mid` -1
  - `target == arr[mid]`：`mid` 即為答案
  - `target > arr[mid]`：將 `L` 設為 `mid` $+1$
- 只要 $[L, R]$ 是合法區間，就可以做二分搜
- 換句話說，我們要搜尋到 $[L, R]$ 是非法的為止，即 $L > R$

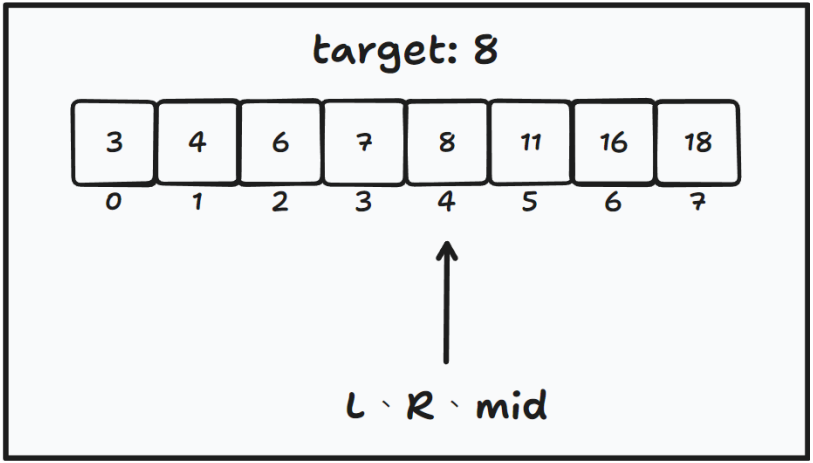
實作二分搜：圖解



實作二分搜：圖解



實作二分搜：圖解



實作二分搜：複雜度分析

- 由於每次都將搜尋範圍除以 2，最多除到範圍變成 1 為止
- 所以整體時間複雜度是 $O(\log n)$
- 參考程式碼：`binary_search.cpp`

練習題

Codeforces EDU Binary Search

二分搜裸題

lower_bound & upper_bound

lower_bound 和 upper_bound 的差別

- 兩者皆是用二分搜來找元素的位置
- 假設我要找的元素是 k
- lower_bound 會回傳第一個 $\geq$ k 的元素位置
- upper_bound 會回傳第一個 $>$ k 的元素位置

lower_bound 實作

- 因為要找第一個 $\geq$ k 的元素位置
- 所以可以把 $>$ k 和 $=$ k 的情況放在一起討論

lower_bound 實作

- 因為要找第一個 $\geq$ k 的元素位置
- 所以可以把 $>$ k 和 $=$ k 的情況放在一起討論
- 也因為 $\geq$ k 的元素都可能是答案，所以在這個情況，把右界 R 設為 mid
- 而 < k 的元素一定不是答案，所以在這個情況下，直接把左界 L 設為 mid + 1
- 參考程式碼：lower_bound.cpp

upper_bound 實作

- 原理跟 lower_bound 相同
- 但 upper_bound 要的是 $>$ k 的元素
- 所以把 lower_bound.cpp 的第 18 行改成 $>$ 即可

手刻好麻煩 QQ

std::binary_search()

```
binary_search(first, last, k);
```

std::lower_bound()

```
lower_bound(first, last, k);
```

std::upper_bound()

```
upper_bound(first, last, k);
```

- 當然 C++ 還是有內建搜尋法的
- first：搜尋起始位置，以指標表示
- last：搜尋終止位置，以指標表示
- k：搜尋的值

內建搜尋法：回傳值

- `binary_search`：布林值，表示 `k` 是否在陣列中
- `lower_bound`：記憶體位置，第一個 $\geq$ `k` 的元素的記憶體位置
- `upper_bound`：記憶體位置，第一個 $>$ `k` 的元素的記憶體位置
- 參考程式碼：`std_search.cpp`

搜尋法的應用

TOJ 55

給你 N 個數字及 M 筆查詢，每筆查詢給一個數字 k，求 k 在數列中出現幾次

- 運用 lower_bound 取得第一個 k 出現位置
- 再用 upper_bound 取得第一個大於 k 的數字的出現位置
- 將這兩個值相減即為 k 的出現次數
- 參考程式碼：toj55.cpp

對答案二分搜

對答案二分搜

- 相信大家都會實作二分搜了
- 來講講二分搜的進階應用
- 直接來看題目

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性
- 所以我可以對服務直徑進行二分搜

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性
- 所以我可以對服務直徑進行二分搜
- 而在判斷的時候只要看這個半徑是不是合法的

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性
- 所以我可以對服務直徑進行二分搜
- 而在判斷的時候只要看這個半徑是不是合法的
- 如果是，那就表示比他大的半徑一定合法，所以將右界設為 mid

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性
- 所以我可以對服務直徑進行二分搜
- 而在判斷的時候只要看這個半徑是不是合法的
- 如果是，那就表示比他大的半徑一定合法，所以將右界設為 `mid`
- 如果不是合法的，那就往比他大的半徑去搜尋，將左界設為 `mid + 1`

對答案二分搜

ZJ c575 基地台

給你 N 個服務點的座標，你可以在上面放 K 個基地台，求基地台的服務直徑最少要多少，才可以包含所有的服務點

- 如果我把服務直徑列出來，可以發現它具有單調性
- 所以我可以對服務直徑進行二分搜
- 而在判斷的時候只要看這個半徑是不是合法的
- 如果是，那就表示比他大的半徑一定合法，所以將右界設為 `mid`
- 如果不是合法的，那就往比他大的半徑去搜尋，將左界設為 `mid + 1`
- 參考程式碼：`zj_c575.cpp`

練習題

CSES 1620

一間工廠有 n 台機器，並且預計需要生產 m 個產品，但是每台機器產生一個商品的時間不同，題目會給所有機器生產一個產品的時間，請問如果要生產 m 個產品至少需要多久。

CSES 1085

給定一個長度為 n 的數列，並且要求把數列切成 m 個區間，並且請在這 m 個區間中，最大的區間和。

結語

實作的重要

- 雖然前面講的幾乎都是模板題
- 有些甚至用一些內建的函式就可以做了
- 但希望大家能夠自己實作一次
- 才可以確保你真的有學會，並且會活用這個演算法
- 跳脫演算法初始的框架

關鍵字

- 排序
  - Radix Sort
  - Counting Sort
- 搜尋
  - 建表
  - 前綴和
  - 資料結構
  - 雙指針
  - 滑動窗口 Sliding Window
  - 單調隊列