

中學進階程式設計與APCS檢測

吳邦一

2021 年 7 月 26 日



Outlines

- 中學進階程式設計與APCS實作題檢測
- 預備知識
- 遞迴
- 排序與搜尋
- 佇列與堆疊、滑動視窗
- 貪心與掃描線演算法
- 分治
- 動態規劃
- 基本圖論
- 樹

課綱中程式設計與演算法的內容

- 七年級：基本程式語法
- 八年級：陣列、搜尋、排序
- 九年級專題
- 高中部定必修：樹、圖、遞迴、分治
- 高中選修(加深加廣)：資料結構、搜尋、排序、分治、貪心、DFS、BFS

學了程式以後會變成如何

- (0) 編號從0開始而不是從1開始
- (1) 知道為何萬聖節=聖誕節(Oct 31=Dec 25)
- (2) 認為警告 (warning)是可以忽略的
- (3) 樹葉在下，樹根在上
- (4) $3/2=1$
- (5) 誤認一公斤是1024公克 ($1\text{Kg}=1024\text{g}$)
- (6) 學了一堆語言但是不大會說話
- (7) 學語言第一句是Hello world
- (8) 句子結尾是分號而不是句號
- (9) 認為 $a = a + 1$ 和 $2!=1$ 都是對的
- (10) 會從 1, 2, 4, 8, 16, ... 一直背到1024
- 以上純屬玩笑

程式設計與競技程式

- 競技程式(competitive programming)是根據程式競賽而逐漸發展出來的一種程式設計領域，其內涵以程式設計、資料結構與演算法為主。
- 程式競賽指的是
 - 如IOI與ICPC，以解題為主的競賽
 - 針對有明確輸入與答案的題目撰寫程式，裁判端將選手的程式進行編譯，餵入設計好的測試資料，以選手程式的輸出與執行時間來裁判其是否正確
 - APCS實作題也是競技程式的題型

競技程式

- 程式設計不等於資訊科技
 - 但是是很重要的一部分
- 競技程式不等於程式設計
 - 但是是很重要的一部分
 - 就像定點投籃與運球不等於打籃球，但有助於增進打籃球的能力。
- 為何特別受到重視？
 - 比起其他創意類型的比賽，競技程式可以被公平客觀而廣泛的檢測

中學生重要競程比賽與檢定

- 資訊學科能力競賽
 - 校內初賽
 - 地區複賽(10月)
 - 全國決賽(12月)
- TOI(資奧培訓營)
 - 一階段(1!)海選(3月)
 - 二階選拔、國手選拔、IOI國際大賽
- NPSC
 - 台大主辦(11~12月)
 - 國高中可參賽
- 少年圖靈賽
 - 精誠公司舉辦
- 東南亞青少年程式賽
 - 電腦學會主辦
- APCS檢定
 - 1、6、10月，一年3次
- CPE(大學生程式檢定)
 - 每年四次

競程特殊技巧

興趣啟
發

基礎程式

基本資料結構
與演算法進階資料結構
與演算法

APCS

二三級

四級

五級

CPE

1~3題

4~5題

6~7題

全國賽三等獎20名

TOI 1 !

全國賽一二等獎10名

TOI 2 ! (前12)

IOI 國手(TOI前4)

CPE 早期難度較高，現在難度已經下降了，6題以上的難度不穩定，目前主要作為某些大學基礎能力的檢定(畢業門檻)。

中學生視為社會人士要收報名費500(降為400)

競程學習資源---網路資源無限多

- 初階入門：
 - 教科書
 - 給新手的C++教學(Code風景區)
- 程式網站提供題目和Auto-Judge
 - ZeroJudge (高中生流行的解題網站)
 - TIOJ(建中)
 - ITSA (E-tutor) (教育部，成大)
 - Uva (英文，ICPC考古題，CPE考題來自於此)
 - CodeForces (可累積個人排名聲望)
- 教學網站
 - 演算法筆記(中文)
 - usaco.guide (English)
- 會google之後很容易搜尋到相關資訊，但英文居多。
- FB有些社團可發問，但未必友善

我的建議

- 學一些基本的程式絕對是好事，即使將來不走這條路
- 升學還是一件重要的事，可以有追夢的勇氣，但也要冷靜的反思與規劃
 - 競賽可以让你了解自己與他人
 - 好好與家人溝通，對中學生而言，沒有家人的支持無法走下去
 - 善設停損，喜歡程式等大學再做也不見得太晚
- 參加營隊活動認識老師與朋友對你非常有幫助，因為不管是程式設計或是其他，成功的重要因素都是----

如何達到神乎其技(神之一招)



少了佐為就沒有棋靈王，沒有塔矢亮也成就不了進藤光

對手跟隊友一樣重要

成功更重要的要素

三井壽

(Mitsui Hisashi)

綽號 永不放棄的男人

性別 男

生日 5月22日

身高 1.84公尺 (6英尺1/2英寸)

體重 70 公斤 (154 磅)

職業 高中生

所屬 湘北高中3年3班

位置 控球後衛(SG)

背號 14



■ 教練！我想打籃球



學習之路

- 初學階段（從0開始到APCS三級）
 - 以探索培養興趣為主要目的
 - 熟悉基本的程式觀念，跟著基本教科書或解題網站練習一些簡單的題目。
 - 應達成目標：看到簡單的題目會構思程式怎麼寫，遇到bug會知道如何找錯誤
 - 應熟悉技能：基本程式指令(運算分支迴圈)、陣列、遞迴。
 - 如果自學，建議以沒有物件導向的C++(或C)開始。如果學校有課程，跟著學校課程會比較容易(先學Python也無妨)

學習之路

- 中級階段（APCS 四五級）
 - 接觸基本資料結構與演算法觀念
 - AP325 或 競技程式的教材
 - 多練習一些題目。
 - 如果初學非C++，必須轉到C++
 - 應熟悉技能：APCS五級及以內的題目
 - C++ STL常用資料結構與函數
 - 嘗試參加一些檢定與比賽（線上與實體賽）
 - 參加好的營隊（台清交的學生有辦一些營隊，收費不多，有些甚至幾乎免費）
 - 有些營隊要有基本能力才能參加

學習之路

- 進階選手階段
 - 增加基本資料結構與演算法的技能
 - 不在STL中的進階資料結構
 - 特殊演算法
 - CodeForce
 - 進階競程教材與網路上資料
 - 以全國賽與TOI為目標

學習之路

- 不變的心法
 - 學而不思則殆，思而不學則惘
 - 要去思考，也要學習別人的程式寫法與經驗
- 學校的學科還是很重要的
 - 特別是英文、數學以及中文溝通能力(閱讀寫作)

什麼是 APCS ?

- 公家辦的「程式設計檢測」，2016年起開始，目前已辦理14次(2021.01)，目前為止都是免費，是具有**公信**力之檢測。
 - 一開始是為了AP(大學先修)課程所開辦的檢測，後來教育部納入升學
 - 適性揚才的政策下，找一個科系族群夠大又有公信力的檢測
- 一年舉辦 **3次** (約在2月、6月、10月)
 - 提供學生自我評量程式設計**能力**
 - 評量大學程式設計先修課程**學習成效**
 - 作為**大學選才**(個人申請)的參考依據
 - 檢驗**民眾個人**程式設計能力
 - 學界或**業界**的選才參考依據
- 目前每次參加名額約3000人
- 北部都會區試場往往開放報名後隨即滿額

APCS

成績應用面

升
學

特殊選才、APCS組、
個人申請

大
學

課程免修、抵免

資
格
競
賽

資訊能力競賽、科展等

其
他

第二專長教師程式設計能力
認定

APCS檢測內容

- 分觀念題與實作題，可分開報名
- 觀念題為選擇題，實作題為競程形式，但為賽後裁判而非online judge
- 實作題以實際撰寫完整程式解決問題為主
 - 可自行選擇以 C, C++, Java, Python 撰寫程式
 - 檢測時間：4 道試題，共計 150 分鐘
 - 計分方式：滿分100分/五級分，每題依照通過測資數給分(與競賽不同)，通常每題有20筆測資，每筆測資五分。

檢測成績級別

程式設計觀念題		程式設計實作題		
級別	原始總分範圍	級別	原始總分範圍	說明
五	90 ~ 100	五	350 ~ 400	具備常見資料結構與基礎演算程序運用能力
四	70 ~ 89	四	250 ~ 349	具備程式設計與基礎資料結構運用能力
三	50 ~ 69	三	150 ~ 249	具備基礎程式設計與基礎資料結構運用能力
二	30 ~ 49	二	50 ~ 149	具備基礎程式設計能力
一	0 ~ 29	一	0 ~ 49	尚未具備基礎程式設計能力

APCS實作題的內容

- 雖然每次的題目都不一樣(AP不會有考古題)，但內涵其實幾乎固定
 - 第一題：基本的輸入輸出、運算、判別式與迴圈。
 - 第二題：簡單的陣列運用，通常一維50分二維50分。
 - 第三題：遞迴、排序與搜尋、或簡單資料結構(stack and queue)
 - 第四題：貪心、分治或DP。
- 第三第四題或許混合，但內容不脫此範圍。
- APCS的範圍僅為常見資料結構，所以不使用進階資料結構(例如某些樹狀資料結構)應該都可以解，但有些題目用進階資料結構可能比較好寫。
- Python都可以通過，但是Python要達到四五級還是有其不利的因素(雖然可能有較寬鬆的時間限制)，例如少了編譯器優化
- APCS與程式競賽的題型是一樣的，但目的與檢測方式不同，所以略有差異：APCS的題目通常比較直接，重程式技巧而非思考，不會埋小陷阱。

APCS學習的書籍與資源

- 競程有很多書與網路資源，多半是英文或簡體中文，缺點是內容太多
- 一般基礎程式學習大概可以到達實作題 3~4級的程度，而且要去熟悉競程上機考試的形式
- 初學者推薦兩本免費的書
 - AP325 (APCS範圍)
 - 網址在臉書社團” APCS實作題檢測”
 - 目前剛好325頁，另外含題目與測資
 - Competitive Programmer' s Handbook (競程範圍)
 - <https://cses.fi/book/book.pdf>
 - Antti Laaksonen
- 網站資源：APCS官網與競程的網路資源
 - 一位學生整理的學習筆記 [YUI HUANG 演算法學習筆記](#)

AP325講義

- 325意為3-to-5，這是一份為了三級分的人所撰寫的免費講義，依照程式技巧區分為0~8章，除了文字講解外，內含例題與習題共121題，。
 - 包含多次APCS過去考題的第三第四題。
 - 每題附測資與答案，可以自己練習。
 - 目前有些online-judge站台有這些題目，例如台中一中的Zero-judge。
- 獲得方式：在臉書搜尋”APCS實作題檢測”社團。此社團是為了開放此講義所設，以便發布更新與勘誤資訊，也方便使用者提問與討論。
- 以下內容多採自該講義。



大學程式設計先修檢測
Advanced Placement Computer Science

AP₃₂₅

預備知識

- 講義中列出一些重要的預備知識，不清楚的人值得看看
 - 基本C++模板與輸入輸出
 - 程式測試與測試資料
 - 複雜度估算（程式執行的效率，輸入資料量的函數）
- 須留意的事
 - 整數overflow
 - 浮點數的rounding error
 - 判斷式的short-circuit evaluation
 - 編譯器優化

「遞迴是小事化小，要記得小事化無。」
「暴力或許可以解決一切，但是可能必須等到地球毀滅。」
「遞迴搜尋如同末日世界，心中有樹而程式(城市)中無樹。」

遞迴

遞迴

- 遞迴在數學上是函數直接或間接以自己定義自己，在程式上則是函數直接或間接呼叫自己。
- 遞迴是一個非常重要的程式結構，在演算法上扮演重要的角色，許多的演算法策略都是以遞迴為基礎出發的，例如分治與動態規劃。
- 遞迴通常使用的時機有兩類

再舉一個很有名的費式數列 (Fibonacci) 來作為例子。費式數列的定義：

$$F(1) = F(2) = 1;$$

$$F(n) = F(n-1) + F(n-2) \text{ for } n > 2.$$

- 根據定義來實作。
- 以遞迴來進行窮舉暴搜

以程式來實作可以寫成：

```
int f(int n) {  
    if (n <= 2) return 1;  
    return f(n-1) + f(n-2);  
}
```

P-1-1. 合成函數 (1)

令 $f(x) = 2x - 1$, $g(x, y) = x + 2y - 3$ 。本題要計算一個合成函數的值，例如
 $f(g(f(1), 3)) = f(g(1, 3)) = f(4) = 7$ 。

Time limit: 1 秒

輸入格式：輸入一行，長度不超過 1000，它是一個 f 與 g 的合成函數，但所有的括弧與逗號都換成空白。輸入的整數絕對值皆不超過 1000。

輸出：輸出函數值。最後答案與運算過程不會超過正負 10 億的區間。

範例輸入：

$f \ g \ f \ 1 \ 3$

範例輸出：

7

遞迴重要的是觀念，寫起來簡單

```
// p 1.1a
#include <bits/stdc++.h>
int eval(){
    int val, x, y, z;
    char token[7];
    scanf("%s", token);
    if (token[0] == 'f') {
        x = eval();
        return 2*x - 1;
    } else if (token[0] == 'g') {
        x = eval();
        y = eval();
        return x + 2*y - 3;
    } else {
        return atoi(token);
    }
}

int main() {
    printf("%d\n", eval());
    return 0;
}
```

依照定義寫就可以
遞迴去抓一個引數

Q-1-5. 二維黑白影像編碼 (APCS201810)

假設 n 是 2 的幕次，也就是存在某個非負整數 k 使得 $n = 2^k$ 。將一個 $n \times n$ 的黑白影像以下列遞迴方式編碼：

如果每一格像素都是白色，我們用 0 來表示；

如果每一格像素都是黑色，我們用 1 來表示；

否則，並非每一格像素都同色，先將影像均等劃分為四個邊長為 $n/2$ 的小正方形後，然後表示如下：先寫下 2，之後依續接上左上、右上、左下、右下四塊的編碼。

輸入編碼字串 S 以及影像尺寸 n ，請計算原始影像中有多少個像素是 1。

Time limit: 1 秒

輸入格式：第一行是影像的編碼 S ，字串長度小於 1,100,000。第二行為正整數 n ， $1 \leq n \leq 1024$ ，中 n 必為 2 的幕次。

輸出格式：輸出有多少個像素是 1。

範例輸入：

2020020100010

8

範例輸出：

17

```
1  #include <stdio>
2  char str[1200000]; // the input string
3  int idx; // the current position to be checked
4  // given the size n, return the area. global input string
5  int rec(int n) {
6      char code=str[idx]; // the current code
7      idx++; // move to next position
8      if (code=='1') return n*n;
9      if (code=='0') return 0;
10     // code='2'
11     n/=2;
12     int area=0; // loop 4 times for 4 sub-area
13     for (int i=0; i<4;i++) area+=rec(n);
14     return area;
15 }
16 int main() {
17     int n;
18     scanf("%s%d",str,&n);
19     idx=0;
20     printf("%d\n",rec(n));
21     return 0;
```

遞迴爆搜：P-1-7. 子集合乘積

遞迴的寫法時間複雜度是 $O(2^n)$ 而迴圈的寫法時間複雜度是 $O(n \cdot 2^n)$ 。

- 輸入 n 個正整數，請計算其中有多少組合的相乘積除以 P 的餘數為 1，每個數字可以選取或不選取但不可重複選，輸入的數字可能重複。
 $P=10009$ ， $0 < n < 26$ 。

```
// subset product = 1 mod P, using recursion
#include<bits/stdc++.h>
using namespace std;
int n, ans=0;
long long P=10009, A[26];
// for i-th element, current product=prod
void rec(int i, int prod) {
    if (i>=n) { // terminal condition
        if (prod==1) ans++;
        return;
    }
    rec(i+1, (prod*A[i])%P); // select A[i]
    rec(i+1, prod); // discard A[i]
    return;
}

int main() {
    scanf("%d", &n);
    for (int i=0; i<n; i++) scanf("%lld", &A[i]);
    ans=0;
    rec(0, 1);
    printf("%d\n", ans-1); // -1 for empty subset
    return 0;
}
```

「完全無序，沒有效率；排序完整，增刪折騰；完美和完整不是同一回事。」
(sorted array vs. balanced tree)

「上窮碧落下黃泉，姐在何處尋不見，人間測得上或下，不入地獄，就是上天；
天上地下千里遠，每次遞迴皆減半，歷經十世終不悔，除非無姐，終能相見。」
(binary search)

SORT AND SEARCH

排序

- 排序通常有幾個運用的時機：
 - 需要把相同資料排在一起
 - 便於快速搜尋
 - 做為其他演算法的執行順序，例如Sweep-line, Greedy, DP。
- 簡單的使用方式，單欄位資料由小到大
 - `sort(a, a+10); // int a[N];`
 - `sort(v.begin(), v.end()); // vector<int> v;`
- 由大到小反序
 - key 變號
 - 使用比較函數，可以用內建的 `sort(a, a+10, greater<int>())`

多欄位資料或其他比較大小的要求

- 如果初學者不想一開始學太多，還是用基本的
 - 定義結構。 `struct Point { int x,y; }`
 - 定義比較函數。

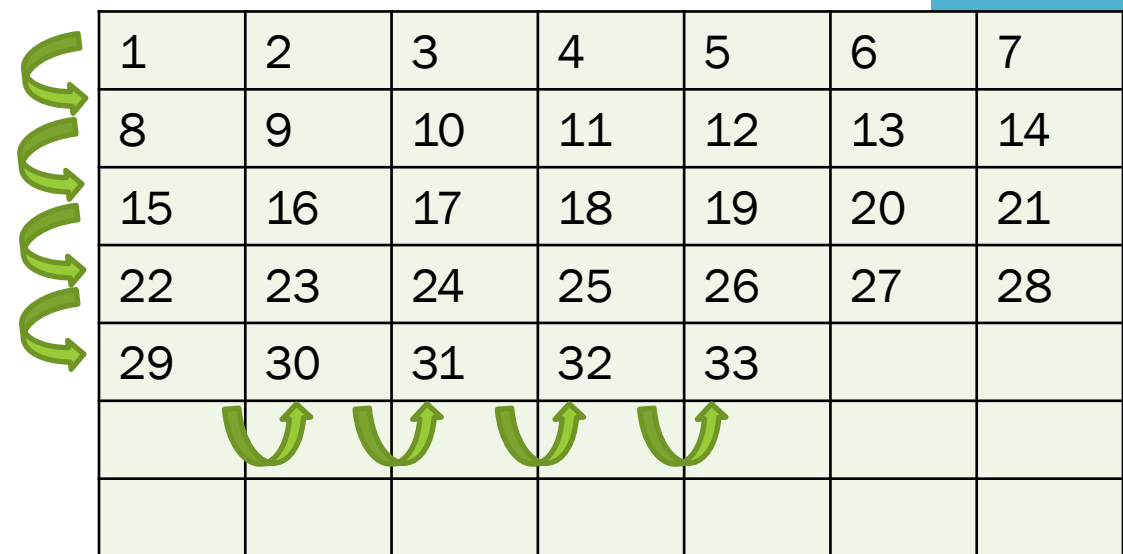
```
bool cmp(Point p, Point q) {  
    return p.x+p.y < q.x+q.y;  
}
```
 - C++ 要比C好寫
- 考試與比賽就是要會偷懶：善用C++內建的pair
 - pair是一個內建雙欄位的class（看成結構就好），用來處理兩個欄位的結構時，可以免除自己定義結構與比較函數。大小的比較是字典順序(lexicographic order)，也就是先比第一欄位再比第二欄位。
 - 詳情請參考文件或講義

搜尋

- 在一個序列中找到某條件的元素
 - 線性搜尋：一個一個往下找，在無序的序列中也只好這麼做，複雜度 $O(n)$
 - 有函數可呼叫，但不一定要學
 - 二分搜：用於已排序的序列，複雜度 $O(\log n)$
- 標準二分搜寫法：
 - 維護一區間，每次將中間點拿來比較，找到或者區間減半
 - 請參考講義
- 二分搜有很多應用時機，也因此很容易寫錯，注意兩件事：
 - 定義好搜尋的區間是 $[a, b]$ 還是 $[a, b)$ 或其他。（會影響裡面的寫法）
 - 檢查在剩下兩個元素時是否正確
- 程式設計的一個重要掌握邏輯正確的要領：在一個區塊(迴圈)中保持重要的不變特性(invariant)，例如二分搜的invariant就是要搜尋的目標始終在此區間中。

另一種二分搜：對初學者的教學啟發

- find the first $a[] \geq x$
while ($p + \text{jump} < n \ \&\& \ a[p + \text{jump}] < x$)
 $p += \text{jump};$
while ($p + 1 < n \ \&\& \ a[p + 1] < x$)
 $p++;$
return $p + 1;$
- 與線性搜尋比較，複雜度由 $O(n)$ 變成 $O(n/\text{jump} + \text{jump})$ ， $= O(\sqrt{n})$
when $\text{jump} = \sqrt{n}$



1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33		

把一維陣列折成二維陣列

另一種二分搜

```
// binary search the first >=x between a[0..n-1]
int jump_search(int a[], int n, int x) {
    if (a[0]>=x) return 0; // check the first
    int po=0; // the current position, always < x
    for (int jump=n/2; jump>0; jump/=2) { //jump distance
        while (po+jump<n && a[po+jump]<x)
            po += jump;
    }
    return po+1;
}
```

- 跳躍距離每次折半，只要還 $< x$ ，能跳就跳。
- 優點：在不同的運用的情形下比較不會寫錯。
- 有很多進階的二元樹資料結構都有退化的陣列作法，複雜度由Log變成根號。可參考我寫的小故事：小方的 Binary Indexed Tree

C++ STL

- C++的內建二分搜
 - `binary_search()` // find if exist
 - `lower_bound()` // find the first $\geq x$
 - `upper_bound()` // find the first $> x$
- 可以先只學一個 `lower_bound()`
- 可以自訂比較函數
- 如果資料並未排序，後果自己負責(並不會報錯，但很可能是錯的)
- 其他C++好用的內建容器(資料結構)
 - `set/multiset`，`map/multimap`
 - 平衡的Binary Search Tree
 - APCS不會出非用不可的題目，但有些題目用了可以有更簡單的寫法。
 - 行有餘力再學。

Q-2-7. 互補團隊 (APCS201906)

- 前 m 個英文大寫字母每個代表一個人物，以一個字串表示一個團隊，字串由前 m 個英文大寫字母組成，不計順序也不管是否重複出現，有出現的字母表示該人物出現在團隊中。兩個團隊沒有相同的成員而且聯集起來是所有 m 個人物，則這兩個團隊稱為「互補團隊」。輸入 m 以及 n 個團隊，請計算有幾對是互補團隊。我們假設沒有兩個相同的團隊。
- Time limit: 1秒
- 輸入格式：第一行是兩個整數 m 與 n ， $2 \leq m \leq 26$ ， $1 \leq n \leq 50000$ 。第二行開始有 n 行，每行一個字串代表一個團隊，每個字串的長度不超過100。
- 輸出格式：輸出有多少對互補團隊。

- 範例輸入：

```
10 5
AJBA
HCEFGGC
BIJDAIJ
EFCDHGI
HCEFGA
```

- 範例輸出：

```
2
```

解

- 可以用字串做
 - 每一個字串化成一個該集合的”標準型”，也就是排序好且字元不重複。
 - 將這些字串排序
 - 對每一個字串，算出互補字串，然後進行二分搜
- 也可以用整數做，一個整數表示一個集合
 - 利用位元運算把每一個字串集合轉換成一個整數。
 - bit 0表示A在不在，bit 1表示B，...
 - 其餘做法相同
 - 速度最快
- 用整數表示集合的方法是一種常用的資料表示法，使用時配合位元運算
- 當集合的大小超過64時，可以用陣列做，也可以用C++內建的bitset

字串+binary_search

```
8 int main() {
9     int m=0, n=0;
10    cin >> m >> n;
11    for (int i=0; i<n; i+=1) {
12        string s;
13        int alpha[26]={0};
14        cin >> s;
15        for (int i=0; i<s.size(); i+=1) // 1 for existing
16            alpha[s[i]-'A'] = 1;
17        // build set-string and complement-string
18        for (int k=0; k<m; k+=1)
19            if (alpha[k]) teams[i] += ('A'+k); // append a char
20            else comp[i] += ('A'+k);
21    }
22    // sort for binary search
23    sort(teams, teams+n);
24    int ans=0;
25    for (int i=0; i<n; i+=1) { // O(nlogn) check if complement is in teams
26        if (binary_search(teams, teams+n, comp[i]))
27            ans += 1;
28    }
29    cout << ans/2 << endl;
```

alpha[0]表示A在不在

每一對會算到兩次

轉數字 + set (也可以用sort)

```
5 int main() {
6     int m=0, n=0;
7     int teams[50000]={0};
8     char s[110];
9     set<int> D;
10    scanf("%d%d", &m, &n);
11    int ff = (1<<m)-1;
12    for (int i=0; i<n; i++) {
13        scanf("%s", s);
14        int len=strlen(s);
15        for (int j=0; j<len; j++) // 1 for existing
16            teams[i] |= 1<<(s[j] - 'A');
17        D.insert(ff-teams[i]); // insert the complement into set
18    }
19    int ans=0;
20    for (int i=0; i<n; i++) { // O(nlogn) check if complement is in teams
21        if (D.find(teams[i])!=D.end())
22            ans++;
23    }
24    printf("%d\n", ans/2);
```

m個1

P-2-15. 圓環出口 (APCS202007)

- 有 n 個房間排列成一個圓環，以順時針方向由 0 到 $n - 1$ 編號。玩家只能順時針方向依序通過這些房間。每當離開第 i 號房間進入下一個房間時，即可獲得 $p(i)$ 點。玩家必須依序取得 m 把鑰匙，鑰匙編號由 0 至 $m-1$ ，兌換編號 i 的鑰匙所需的點數為 $Q(i)$ 。一旦玩家手中的點數達到 $Q(i)$ 就會自動獲得編號 i 的鑰匙，而且手中所有的點數就會被「全數收回」，接著要再從當下所在的房間出發，重新收集點數兌換下一把鑰匙。遊戲開始時，玩家位於 0 號房。請計算玩家拿到最後一把鑰匙時所在的房間編號。
- 前綴和 + 二分搜
 - `prefix_sum`，前 i 個的總和 ($i = 1..n$)。
 - 正數的前綴和為一遞增序列，
- 請見講義

原陣列

4	1	3	3	6	2	1	2	5
---	---	---	---	---	---	---	---	---

prefix-sum 前綴和

4	5	8	11	17	19	20	22	27
---	---	---	----	----	----	----	----	----

- 在位置 i 找 x , $\text{sum}(a[i: x]) \geq Q$ ，相當於找 $p[x] \geq p[i-1] + Q$
- 超過尾端的話，就扣除剩餘後，從頭找
- 留意題目的定義：離開該房間才獲得點數

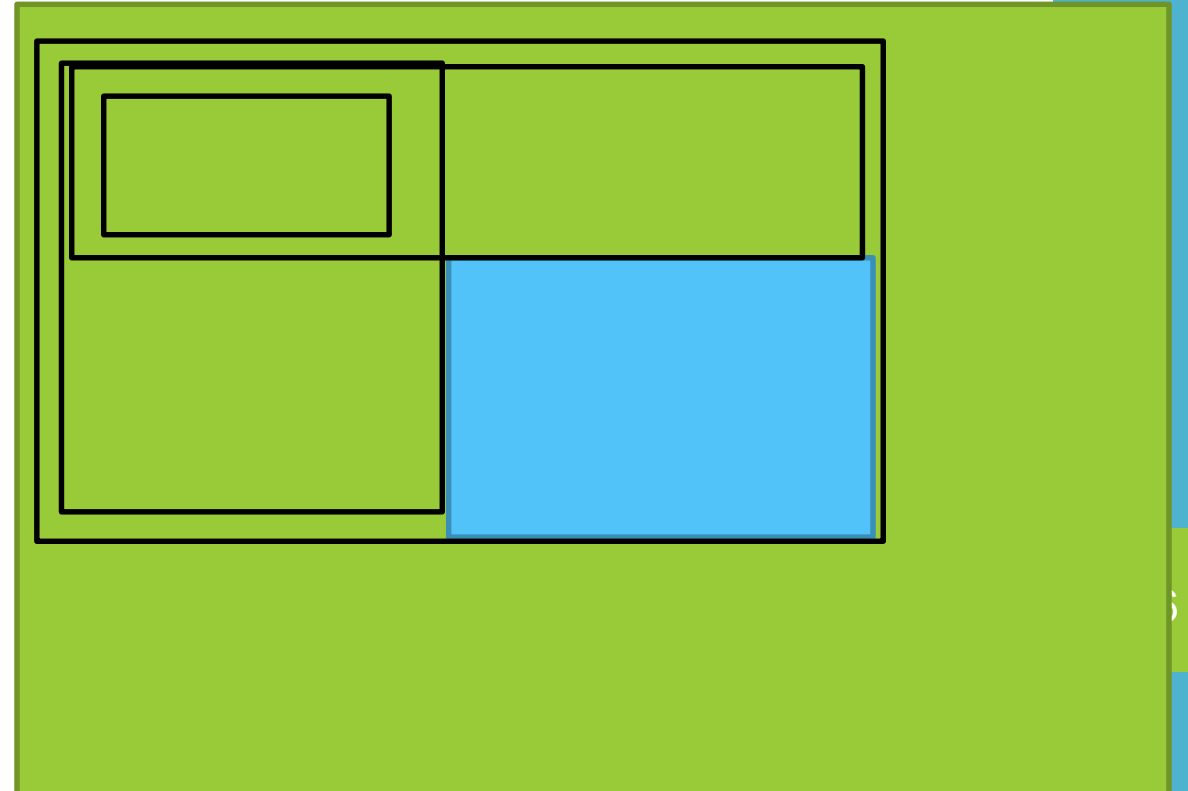
前綴和是一個常用的前處理

- 對一個數列 $a[1:n]$ ，前綴和 $psum[i]$ 是 a 的前 i 項總和。
- $O(n)$ 可以求出所有 $psum$
 - $psum[0] = 0$
for ($i=1$; $i \leq n$; $i++$) $psum[i] = psum[i-1] + a[i]$;
- 原序列最好從 1 開始
- 小心不要 overflow
- 可以在 $O(1)$ 查詢區間和
 $sum[i:j] = psum[j] - psum[i-1]$



2D prefix sum

- $\text{psum}[i, j] = \text{sum}(a[0:i][0:j])$
- $O(1)$ 算任意一個子矩陣總和
- $\text{sum}(a[r1:r2][c1:c2]) = \text{psum}[r2, c2] - \text{psum}[r1-1, c2] - \text{psum}[r2, c1-1] + \text{psum}[r1-1, c1-1]$



佇列與堆疊、滑動視窗

array、vector、queue and stack

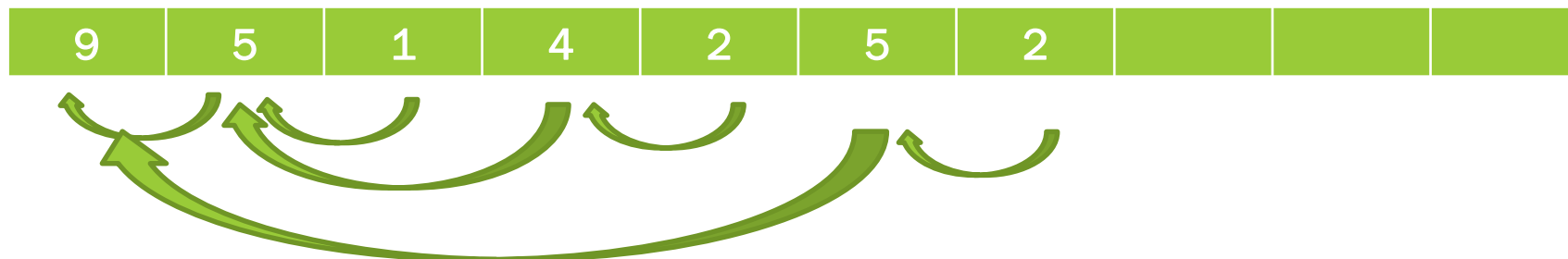
- vector可看作是可變長度的陣列，如果初學不適合一下子學太多，可以以陣列為主，因為大部分的題目並不會有空間的問題。
 - 非用不可的場合是sparse graph與tree。所以在陣列熟練之後，還是建議學起來。
- 佇列(queue)、堆疊(stack)與雙向佇列(deque, double-ended queue)都是STL中提供的容器，但也很容易自己製作。
 - 除了queue與deque在某些時候會有空間的問題之外，自製並沒有比使用內建來得麻煩，可能還更簡單。而空間的問題一般在題目中並不會遇到。
- 自製的方法一般的課本都有，這裡的重點不再自製而是如何運用與何時運用。

運用Queue的例子

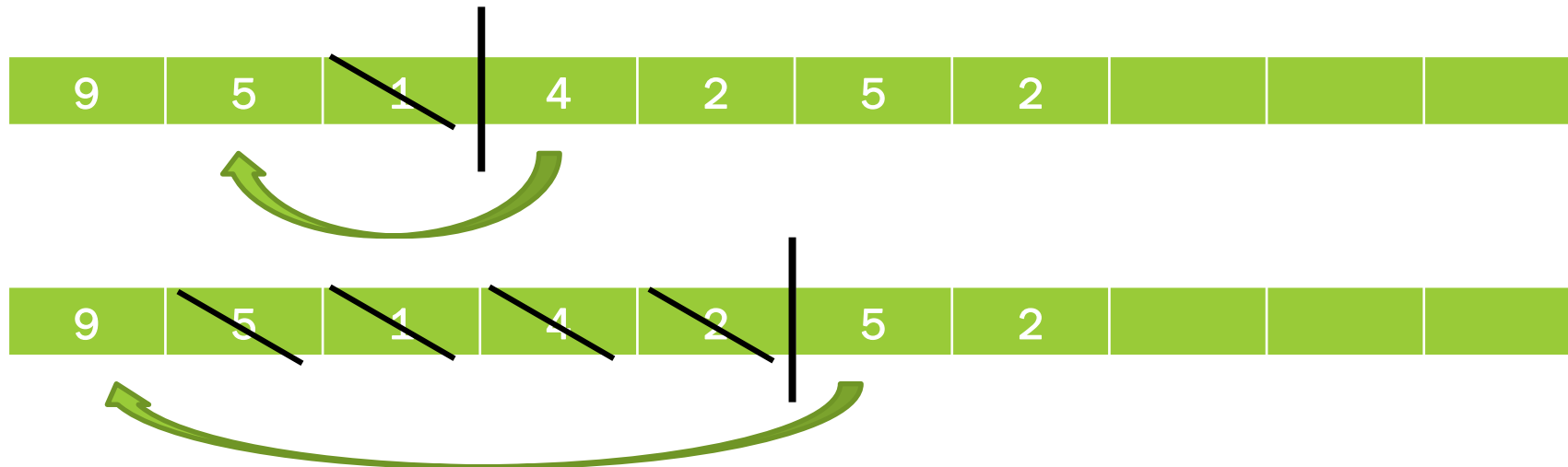
- 模擬題
- graph與tree的BFS (breadth-first search)
- DAG的topological sort
- 例題
 - P-3-1 樹的高度與根 (bottom-up) (APCS201710)
 - Q-7-5 闖關路線 (APCS201910)
 - 詳見講義

stack 的例子：P-3-4 (APCS201902)

- 給一個整數序列a，對每一個 $a[i]$ ，找到他前方離他最近比他大的(高人)



- 最天真無邪又直接的方法就是：對每一個 i ，從 $i-1$ 開始往前一一尋找
- 改善：假設由前往後掃過去的時候，那些沒有用？
 - $a[i-1] \leq a[i]$ ，那麼 $a[i-1]$ 不可能是 i 之後的人的高人，因為由後往前找的時候會先碰到 $a[i]$ ，如果 $a[i]$ 不夠高， $a[i-1]$ 也一定不夠高。
 - 如果我們丟掉那些沒有用的，會剩下甚麼呢？
 - 一個遞減序列。維護好這個遞減序列，就可以有效率的解此題



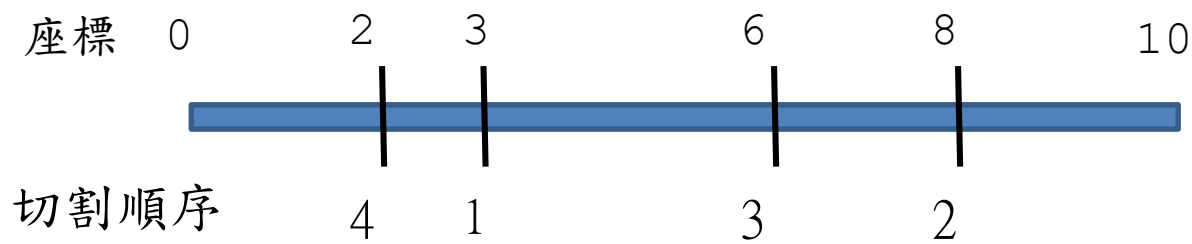
- 以一個stack存此遞減序列，在 $a[i]$ 時，從後往前看 $\leq a[i]$ 的都沒有用了；碰到第一個 $> a[i]$ 的就是他的高人，並且把 $a[i]$ 放入stack。
- 延伸：Q-3-5. 帶著板凳排雞排的高人（APCS201902）
 - 在此遞減序列上做二分搜（因此不要用stack）

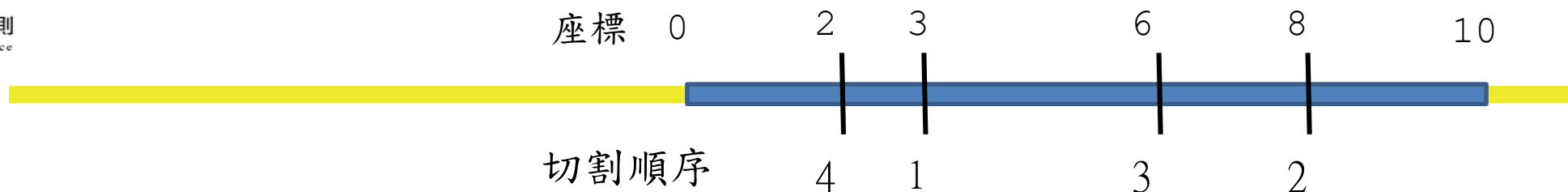
- for 迴圈內有個while，
複雜度會不會壞掉？
- 事實上不會，整個程式的複雜度是 $O(n)$
- 雖然while可能某次做很多次pop()，但總共只會做 $O(n)$ 次
 - amortized analysis
 - 均攤分析

```
// p_3_4a, stack for index
#include <bits/stdc++.h>
using namespace std;
#define N 300010
#define oo 10000001
int a[N];
stack<int> S; // for index
int main() {
    int i,n;
    long long total=0; // total distance
    scanf("%d",&n);
    S.push(0);
    a[0]=oo;
    for (i=1; i<=n; i++) {
        scanf("%d",&a[i]);
    }
    for (i=1; i<=n; i++) {
        while (a[S.top()] <= a[i])
            S.pop();
        total += i - S.top();
        S.push(i);
    }
    printf("%lld\n",total);
    return 0;
}
```

APCS202101

- 給長度為 L 的棍子上 n 個切割點以及它們的切割順序，計算每個點在切割時所在的棍子長度，也就是最接近該點的左右已切割點的距離， 0 與 L 看成最早的已切割點。輸出每個點切割時所在棍子長度的總和。
- 第1點切在3，成本 = 10
- 第2點切在8，成本 = $10 - 3 = 7$ （切割時的棍子長度）
- 第3點切在6，成本 = $8 - 3 = 5$
- 第4點切在2，成本 = $3 - 0 = 3$





- 解法：這一題有很多種解法，其中之一依照座標排序後，每一點切割時的左端就是在他左邊，切割順序小於它而離他最近的切割點。
- 堆疊內：尚未找到右端的點
 - 必為遞增
- 碰到下一點 p 時
 - 堆疊內晚於 p 的點，它們的右端就是 p ， pop
 - p 的左端是堆疊的 top
- 成本 = 所有的右端點座標減去所有的左端點座標

```

00 # using stack for those whose right endpoint are not found
01 # their orders must be monotonic increasing
02 n,len = map(int, input().split())
03 cut = []
04 for i in range(n):
05     cut.append([int(x) for x in input().split()]) # [pos,ord]
06 cut.sort() # sort by position
07 cut.append([len,0])
08 stack = [[0,0]] # [pos,ord]
09 cost = 0
10 for p in cut:
11     while stack[-1][1] > p[1]: # stack[-1] is top
12         cost += p[0] # p is right endpoint of top
13         stack.pop()
14     cost -= stack[-1][0] # top is left endpoint of p
15     stack.append(p)
16 # end for
17 print(cost)

```

Python的範例

P-3-6. 砍樹 (APCS202001)

- N棵樹種在一排，現階段砍樹必須符合以下的條件：「讓它向左或向右倒下，倒下時不會超過林場的左右範圍之外，也不會壓到其它尚未砍除的樹木。」。你的工作就是計算能砍除的樹木。若 $c[i]$ 代表第 i 棵樹的位置座標， $h[i]$ 代表高度。向左倒下壓到的範圍為 $[c[i]-h[i], c[i]]$ ，而向右倒下壓到的範圍為 $[c[i], c[i]+h[i]]$ 。如果倒下的範圍內有其它尚未砍除的樹就稱為壓到，剛好在端點不算壓到。
- 我們可以不斷找到滿足砍除條件的樹木，將它砍倒後移除，然後再去找下一棵可以砍除的樹木，直到沒有樹木可以砍為止。無論砍樹的順序為何，最後能砍除的樹木是相同的。

砍樹的解

- 假設由前往後一一檢視，能砍就砍了，不能砍的暫時放著。想一想，暫時不能砍的樹何時會變成可以砍？
 - 除非他後面的樹被砍，否則不可能。
 - 用一個stack把暫時不能砍的樹放起來，每次下一棵樹被砍時，往前檢查。

```
S.push(0);  
for (i=1;i<=n;i++) { // scan from left to right  
    // if i is removable  
    if (c[i]-h[i]>=c[S.top()] || c[i]+h[i]<=c[i+1]) {  
        total++;  
        high = max(high, h[i]);  
        // backward check remaining tree in stack  
        while (c[S.top()+h[S.top()]<=c[i+1]) {  
            total++; high = max(high, h[S.top()]);  
            S.pop();  
        }  
    } else { // i is not removable  
        S.push(i);  
    }  
}
```

滑動視窗

- 接下來要介紹滑動視窗的技巧，這個名字可能不是很統一，基本上是以兩個指標維護一段區間，然後逐步將這區間從前往後(從左往右)移動。
 - 雙指針
 - 爬行法
- Q-3-12. 完美彩帶 (APCS201906)
 - 有一條細長的彩帶，總共有 m 種不同的顏色，彩帶區分成 n 格，每一格的長度都是 1，每一格都有一個顏色，相鄰可能同色。長度為 m 的連續區段且各種顏色都各出現一次，則稱為「完美彩帶」。請找出總共有多少段可能的完美彩帶。請注意，兩段完美彩帶之間可能重疊。
 - 有很多寫法

格子編號	1	2	3	4	5	6	7	8	9	10
顏色編號	1	4	1	7	6	4	4	6	1	7

- 維護 $[j, i]$ 是一個無同色的區間

```

27 disc(a,n); // discretization
28 // start sliding window, window = [j,i]
29 for (i=0, j=0; i<n; i++) {
30     int c=a[i];
31     if (col[c]) { // color already in window
32         // clear before color c
33         while (a[j] != c) {
34             col[a[j]] = 0;
35             j++;
36         }
37         j++; // next starting position
38     }
39     else // new color
40         col[c] = 1;
41     if (i-j+1 == m) // a perfect colored window
42         total++;
43 }
44 printf("%d\n",total);

```

- 另解：對每一個位置 i ，找出最遠的左端 $\text{left}[i]$ ，滿足 $[\text{left}[i], i]$ 是無同色區間。若 $i - \text{left}[i] + 1 == m$ 則是完美色帶
- 找 left 的方法
 - 對每一個位置 i ，找出前一個同色的位置 $\text{prev}[i]$

格子編號	1	2	3	4	5	6	7	8	9	10
顏色編號	1	4	1	7	6	4	4	6	1	7
prev	0	0	1	0	0	2	6	5	3	4

- $\text{left}[i] = \max(\text{left}[i-1], \text{prev}[i]+1)$ ，不能小於這兩個位置

格子編號	1	2	3	4	5	6	7	8	9	10
顏色編號	1	4	1	7	6	4	4	6	1	7
prev	0	0	1	0	0	2	6	5	3	4
left	1	1	2	2	2	3	7	7	7	7

「貪心通常比較簡單，但多半不能解決問題，在演算法與人生中都是這樣。」
AP325，第4章，貪心演算法與掃描線演算法

GREEDY ALGORITHM 貪心演算法

Greedy algorithm

- 簡單的例子：令狐沖去少林寺觀光，少林寺有四種代幣，面額分別是[1, 5, 10, 50]，令狐沖拿了n元去換代幣，請問以這四種代幣湊成n元，最少要多少枚代幣？
- 最直覺的做法：由面額大的能換就換。
- 這就是貪心算法。
 - 算法由一連串決定組成(由大到小逐一考慮各種代幣的數量)，每次考慮當前最好決定(能換盡量換)，選擇後就不再更改。
 - 算法的正確性必須經過證明，內容就是要證明所下的決定是對的，也就是一定有一個最佳解包含了貪心法所做的決定，而證明的方法通常都是用換的：假設有一個最佳解跟貪心法做的決定不一樣，那麼，我們可以換成一個與貪心法一樣的決定，而解會更好，或至少不會變差。

例題P-4-2. 笑傲江湖之三戰

- 少林寺與五嶽劍派等正教派出 n 位高手，而你方也有 n 位高手，每個人都有一個戰力值。雙方要進行一對一的對戰，每個人不管輸或贏都只能比一場，假設兩人對戰時，戰力值高的獲勝。對於對方的每一位高手，你可以指定派哪一位與其對戰，為了解救盈盈，你希望贏越多場越好，平手則沒有幫助。請計算出最多能贏多少場。

- 假設我方目前最弱的戰力是a而對方最弱的是b，
 - 如果 $a \leq b$ ，則a對誰都不會贏，是沒用的，可以忽略。
 - 否則 $a > b$ ，我們宣稱「一定有一個最佳解是派a去出戰b」
 - 假設在一個最佳解中，a對戰x而y對戰b，我們將其交換改成「a對b而y對x」。根據我們的假設 $y \geq a$ ，交換後a可以贏b而y對x的戰績不會比a對x差，所以交換後勝場數不會變少。
 - 派a對戰b一定可以得到最佳解，並將a與b移除後，繼續依照上述原則挑選。

敵：[3, 5, 5, 7, 8]



2是沒用的

我：[2, 6, 6, 8, 8]

敵：[3, 5, 5, 7, 8]



我：[6, 6, 8, 8]

敵：[3, 5, 5, 7, 8]

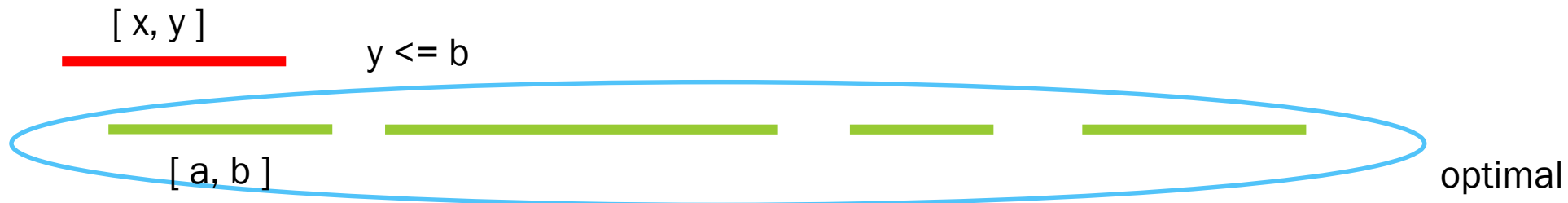


我：[6, 6, 8, 8]


```
// P_4_2, one-on-one  $O(n \log n)$ , sorting+greedy
#include <bits/stdc++.h>
using namespace std;
#define N 100010
int main() {
    int n, enemy[N], ours[N];
    scanf("%d", &n);
    for (int i=0; i<n; i++)
        scanf("%d", &enemy[i]);
    for (int i=0; i<n; i++)
        scanf("%d", &ours[i]);
    sort(enemy, enemy+n); // sort enemy power
    sort(ours, ours+n); // sort our power
    int win=0; // num of win
    for (int i=0, j=0; i<n && j<n; i++) { // for each of our power
        // j is currently weakest enemy
        if (ours[i] > enemy[j]) { // can win some enemy
            win++;
            j++; // next enemy
        }
        // otherwise, ours[i] is useless
    }
    printf("%d\n", win);
    return 0;
}
```

P-4-4. 幾場華山論劍(activity selection)

- 華山派決定每年都舉辦非常多場的華山論劍，每一場都有自己的開始時間與結束時間，參加者必須全程在場，所以不能在同一時間參加兩場。令狐沖拿到了今年所有場次的資料，希望能參加越多場越好，以便盡速累積經驗值，請你幫忙計算最多可以參加幾場。
 - 數線上有若干線段，要挑選出最多的不重疊的線段。
- 若 $[x, y]$ 是現存線段中右端點最小的，一定有個最佳解挑選 $[x, y]$ 。
 - 證明：假設最佳解沒有挑 $[x, y]$ ，令 $[a, b]$ 是最佳解中右端點最小的。根據我們的假設， $y \leq b$ ，因此將 $[x, y]$ 取代 $[a, b]$ 不會重疊到任何最佳解中的其他線段，我們可以得到另外一個最佳解包含 $[x, y]$ 。



```
↵  
// P_4_4, activity selection, sorting+greedy↵  
#include <bits/stdc++.h>↵  
using namespace std;↵  
#define N 100010↵  
struct ACT{↵  
    int s,f; // start and finish time↵  
};↵  
// compare finish time↵  
bool cmp(ACT p, ACT q) {↵  
    return p.f<q.f;↵  
}↵  
int main() {↵  
    int n;↵  
    ACT ac[N];↵  
    scanf("%d",&n);↵  
    for (int i=0;i<n;i++)↵
```

```
        scanf("%d%d",&ac[i].s, &ac[i].f);↵  
    sort(ac, ac+n, cmp); // sort from small to large↵  
    int endtime=-1, total=0;↵  
    for (int i=0; i<n; i++) {↵  
        if (ac[i].s>endtime) { // compatible↵  
            total++;↵  
            endtime=ac[i].f;↵  
        }↵  
    }↵  
    printf("%d\n",total);↵  
    return 0;↵  
}↵  
↵
```

排序的偷懶寫法：利用pair

```
// P_4_4b, activity selection, sorting+greedy
#include <bits/stdc++.h>
using namespace std;
#define N 100010

int main() {
    int n;
    vector<pair<int,int>> act;
    scanf("%d",&n);
    for (int i=0;i<n;i++) {
        int s,t;
        scanf("%d%d",&s,&t);
        act.push_back({t,s}); // key = finish time
    }
    sort(act.begin(), act.end()); // sort by finish
    int endtime=-1, total=0; // end of previous activity
    for (auto p: act) {
        if (p.second>endtime) { // compatible
            total++;
            endtime=p.first; // set end-time
        }
    }
    printf("%d\n",total);
    return 0;
}
```

Q-4-6. 少林寺的自動寄物櫃 (APCS201710)

- 少林寺的自動寄物櫃系統存放物品時，是將 N 個物品堆在一個垂直的貨架上，每個物品各佔一層。系統運作的方式如下：每次只會取用一個物品，取用時必須先將在其上方的物品貨架升高，取用後必須將該物品放回，然後將剛才升起的貨架降回原始位置，之後才會進行下一個物品的取用。
- 每一次升高貨架所需要消耗的能量是以這些物品的總重來計算。現在有 N 個物品，第 i 個物品的重量是 $w(i)$ 而需要取用的次數為 $f(i)$ ，我們需要決定如何擺放這些物品的順序來讓消耗的能量越小越好。
- 解： $w(i)/f(i)$ 由小排到大。
- 證明方法：若相鄰兩個反序時，交換可以得到更好的解

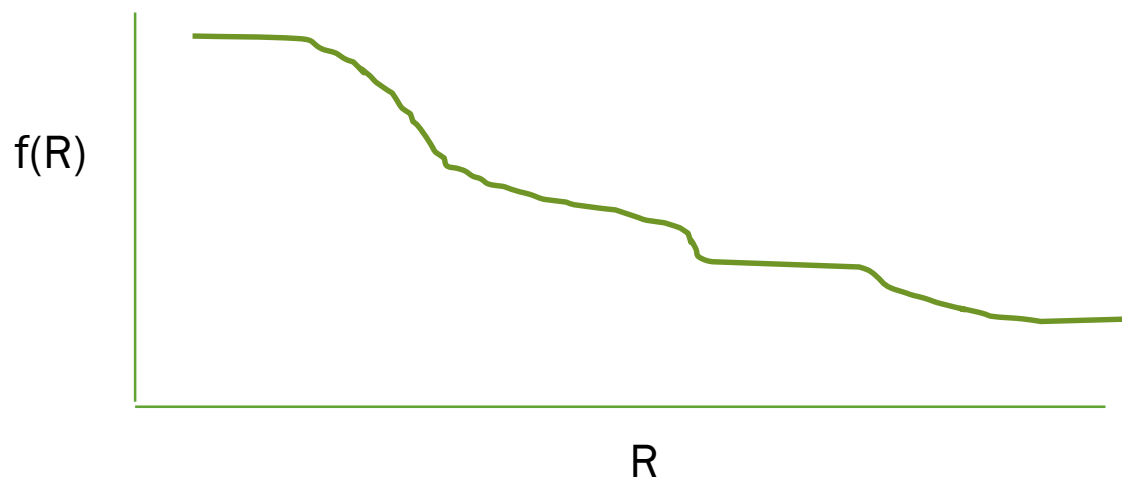
P-4-9. 基地台 (APCS201703)

- 直線上有 N 個要服務的點，每架設一座基地台可以涵蓋直徑 R 範圍以內的服務點。輸入服務點的座標位置以及一個正整數 K ，請問：在架設 K 座基地台以及每個基地台的直徑皆相同的條件下，基地台最小直徑 R 為多少？
- 服務點可以看成數線上的點，基地台可以看成數線上的線段
 - 以 K 根長度相同的線段蓋住所有的點，最小的線段長度。
- 這一題是給數量 K 要求長度 R 。要解決這個問題，先看以下問題：
 - 輸入給數線上 N 個點以及 R ，請問最少要用幾根長度 R 的線段才能蓋住所有輸入點。
 - 「一定有一個最佳解是將一根線段的左端放在最小座標點上。」



外掛二分搜

- 用長度 R 的 K 根線段蓋住所有點
- 假設 $f(R)$ 是給定長度 R 的最少線段數，那麼上述的演算法可以求得 $f(R)$ 。
- 當 R 增加時， $f(R)$ 必然只會相同或減少
 - 因為用更長的線段去蓋相同的點，不會需要更多線段。
- 所以我們可以二分搜來找出最小的 R ，滿足 $f(R) \leq K$ 。



```
// check if k segment of length r is enough↵
bool enough(int r) {↵
    int nseg=k, endlne = -1; // current covered range↵
    for (int i=0; i<n; i++) {↵
        if (p[i] <= endlne) continue;↵
        if (nseg == 0) return false;↵
        // use a segment to cover↵
        nseg--; // remaining segments↵
        endlne = p[i] + r;↵
    }↵
    return true;↵
}↵
↵
int main() {↵
    scanf("%d%d", &n, &k);↵
    for (int i=0; i<n; i++)↵
        scanf("%d", p+i);↵
    sort(p, p+n);↵
    // binary search, jump to max not-enough length↵
    int len = 0, L = p[n-1] - p[0];↵
    for (int jump=L/2; jump>0; jump>>=1) {↵
        while (len+jump<L && !enough(len+jump))↵
            len += jump;↵
    }↵
    printf("%d\n", len+1);↵
}
```


Sweep line algorithm

- P-4-11. 線段聯集 (APCS 201603)
 - 輸入數線上的 N 個線段，計算線段聯集的總長度。
- 想像有根掃描線，從左往右掃，維護好已經看到的線段的聯集 S
- 碰到某線段左端時，代表有一個線段要加入 S 。
 - 從左往右掃的好處是，當一個線段 $[x, y]$ 加入時，最多只會跟 S 中的一根線段(最後一根)有交集！因為他的左端 x 不會在 S 中任何一根線段的左方(記得吧，我們從左往右)。
 - 什麼時候沒交集呢？如果 x 大於 S 中的最大右端。
 - 而且，如果這個線段與 S 沒交集，後面的線段也都不會再跟 S 有交集，因為後面的線段的左端都大於等於 x 。

```
11 int main() {
12     int n, total=0; // total length
13     Seg s[N];
14     scanf("%d",&n);
15     for (int i=0; i<n; i++)
16         scanf("%d%d", &s[i].left, &s[i].right);
17     sort(s, s+n, cmp); // sort by left
18     Seg last = s[0]; // last segment in the union
19     for (int i=1; i<n; i++) { // insert each segment
20         if (s[i].left > last.right) { // disjoint
21             total += last.right - last.left;
22             last = s[i];
23             continue;
24         }
25         // else part, merge last and s[i]
26         last.right = max(last.right, s[i].right);
27     }
28     total += last.right - last.left; // don't forget
29     printf("%d\n",total);
```

分治：「一刀均分左右，兩邊各自遞迴，返回合併之日，解答浮現之時。」

「架勢起得好，螳螂鬥勝老母雞。」

在分治的架構下，簡單的資料結構與算法往往也有好的複雜度。」

分治演算法

分治

- 主要步驟
 - 切割。將問題切割成若干個子問題，通常是均勻地切成兩個。
 - 分別對子問題遞迴求解。
 - 合併。將子問題的解合併成原問題的解。
- 所謂的子問題切割是「相同問題而比較小的輸入資料」
 - 第二步驟是透過遞迴呼叫來解子問題
 - 除了終端條件外，第二步驟什麼都不必做。
- 分治算法的迷人之處：在思考一個問題的解的時候，你不需要去想解的步驟，只要去想「如何將子問題的解合併成整個問題的解」就可以了。

- 課本典型的分治例子：
 - 快速排序法、合併排序法、快速傅立葉轉換(FFT)、矩陣乘法、整數乘法
- 分治在思考問題設計演算法時是很重要策略，但一般題目中歸屬分治的題目比較少
 - 很多問題運用了資料結構後就有掃描線的解法
 - 也有些解法往往分類到其他類型，例如樹的分治與倍增法

分治的複雜度

- 分治算法的複雜度通常需要解遞迴式 $T(n) = a \times T(n/b) + f(n)$
 - Master theorem
- 常見的情形

遞迴式	時間複雜度	說明
$T(n) = T(n/b) + O(1)$	$O(\log(n))$	切兩塊其中一塊不需要，如二分搜
$T(n) = T(n/b) + O(n)$	$O(n)$	每次資料減少一定比例
$T(n) = 2T(n/2) + O(1)$	$O(n)$	例如找最大值
$T(n) = 2T(n/2) + O(n)$	$O(n \log(n))$	如merge sort
$T(n) = 2T(n/2) + O(n \log(n))$	$O(n \log^2(n))$	分割合併花 $O(n \log(n))$
$T(n) = 2T(n/2) + O(n^2)$	$O(n^2)$	$f(n)$ 大於 n 的一次方以上，結果皆為 $f(n)$

架勢起得好，螳螂鬥勝老母雞

- 分治或許不會讓我們一下子找到效率最好的解，但是如果使用分治，即使合併的方法很天真，往往也很容易找到突破天真算法複雜度的解法
- P-5-2. 最大連續子陣列(分治)(同P-4-13)
 - 有一個整數陣列 $A[0:n-1]$ ，請計算 A 的連續子陣列的最大可能總和，空陣列的和以0計算。
- 如果我們要計算陣列在 $[L, R-1]$ 區間的最大連續和
 - 一刀平均分兩段
 - 左遞迴，右遞迴
 - 跨兩邊的解？對於左邊，我們就笨笨的從中點往左一直累加，計算每一個可能左端的區間和(也就是左邊的suffix-sum)，然後取最大；同樣的，對於右邊計算所有prefix-sum的最大，然後兩者相加就是跨過中點的最大區間和。

```
if (le >= ri) return 0;↵
if (le+1 == ri) return max(a[le], (LL) 0);↵
int mid=(le+ri)/2;↵
// recursively solve left and right parts↵
LL largest=max(subarr(a, le, mid), subarr(a, mid, ri));↵
// find largest sum cross middle↵
LL lmax=0, rmax=0;↵
// max suffix sum of the left↵
for (LL sum=0, i=mid-1; i>=le; i--) {↵
    sum += a[i];↵
    lmax=max(lmax, sum);↵
}↵
// max prefix sum of the right↵
for (LL sum=0, i=mid; i<ri; i++) {↵
    sum += a[i];↵
    rmax=max(rmax, sum);↵
}↵
return max(largest, lmax+rmax);↵
}↵
```

- 這樣用迴圈笨笨的做，時間複雜度如何？
- $T(n)=2T(n/2)+O(n)$ ，答案是 $O(n\log(n))$ ！因為它有一個聰明的外殼架構。
- 雖然每一次合併都是笨笨的做，但是任一個資料參與合併的次數只有 $\log(n)$ 。

P-5-4. 反序數量 (APCS201806)

- 考慮一個數列 $A[1:n]$ 。如果 A 中兩個數字 $A[i]$ 和 $A[j]$ 滿足 $i < j$ 且 $A[i] > A[j]$ ，也就是在前面的比較大，則我們說 $(a[i], a[j])$ 是一個反序對 (inversion)。定義 $W(A)$ 為數列 A 中反序對數量。例如，在數列 $A=(3, 1, 9, 8, 9, 2)$ 中，一共有 $(3, 1)$ 、 $(3, 2)$ 、 $(9, 8)$ 、 $(9, 2)$ 、 $(8, 2)$ 、 $(9, 2)$ 一共6個反序對，所以 $W(A)=6$ 。請注意到序列中有兩個9都在2之前，因此有兩個 $(9, 2)$ 反序對，也就是說，不同位置的反序對都要計算，不管兩對的內容是否一樣。請撰寫一個程式，計算一個數列 A 的反序數量 $W(A)$ 。
- 其實把merge-sort的程式稍加修改就可以計算反序數量
 - 請自行參照講義
- 學競程的選手通常不喜歡以分治來解，因為他們的腦海裡有許多資料結構。這一題從前往後掃，對於每個數 $A[i]$ 只要能求出在 i 之前有多少大於它的數就可以了，因此他們可能會自然使用線段樹來做。

低地距離(APCS202010)

- 有 $2n$ 座碉堡排成一行，這些碉堡的高度是成對出現，相同高度的碉堡一定恰有兩座。對於每一對相同高度的碉堡，考古隊定義這一對碉堡的「低地距離」為：位於它們之間且高度較低的碉堡數量。請計算這一群碉堡的低地距離總和。
- 假設碉堡的高度依序為 $(1, 4, 3, 2, 3, 1, 2, 4)$
 - 低地距離總和為 $0 + 1 + 1 + 5 = 7$
- **提示：**本題有多種解法，其中一種是將碉堡高度的序列依照高度分成兩個子序列，再以分而治之的策略遞迴求解。另外一種解法是對每一個位置 i ，計算出在 i 之前而高度小於 h_i 的碉堡個數。

分治主架構

```
// 計算序列a[]的低地距離總和，數字範圍是[low,up]
int dist (a[], low, up)
    if n < 3 then return 0; // 只有一對，沒有距離
    mid = (low + up)/2; // 數字大小的中間值
    // <= mid的稱為小數字，否則稱為大數字
    將 a 拆成小數字與大數字的序列 small與large;
    計算 ans = 小數字介於大數對之間的總數;
    ans = ans + dist(small, low, mid); // 遞迴
    ans = ans + dist(large, mid+1, up); // 遞迴
end dist
```

```
05 LL sol(int a[], int n, int low, int up) {
06     if (n<=2) return 0; // terminal case
07     int mid = (low+up)/2; // pivot
08     int small[n], large[n], n_small=0, n_large=0;
09     int between[up-low+1]; // if between pair of low+i
10     for (int i=0; i<=up-low; i++) between[i]=0;
11     LL d=0; // small in between large pair
12     int num=0; // between how many large pair
13     for (int i=0; i<n; i++) {
14         if (a[i] <= mid) { // small part
15             small[n_small++] = a[i]; // put into small
16             d += num; // small between big-pair
17         } else { // large part
18             large[n_large++] = a[i]; // put into large
19             if (between[a[i]-low]) { // already appeared
20                 between[a[i]-low] = 0;
21                 num--;
22             } else { // first appearance
23                 between[a[i]-low] = 1;
24                 num++;
25             }
26         }
27     }
28     // recursively solve small and large parts
29     d += sol(small, n_small, low, mid);
30     d += sol(large, n_large, mid+1, up);
31     return d;
32 }
33 int main() {
```

「每一個DP背後都有一個dag，DP需要dag指引方向，如同視障者需要dog。」
(Dynamic programming, Bottom-up)

「設計DP以尋找遞迴式開始，以避免遞迴來完成，除非，除非你準備小抄。」
(Dynamic programming, Top-down memoization)

動態規劃

基本原理

- DP與分治有個相同之處，都是將問題劃分成子問題，再由子問題的解合併成最後的解，其中子問題是指相同問題比較小的輸入資料。所以，設計DP的方法從找出遞迴式開始，設計DP的演算法通常包含下面幾個步驟：
 - 1. 定義子問題。
 - 2. 找出問題與子問題之間的(遞迴)關係。
 - 3. 找出子問題的計算順序來避免以遞迴的方式進行計算。
- 如果沒有進行第三個步驟，而只是以遞迴的方式寫程式，就會變成與第一章相同的純遞迴方式，純遞迴往往會遭遇到非常大的效率問題，所以也可以說DP就是要改善純遞迴的效率問題的技術

Top-down memoization

- 標準DP算法的步驟基本上要先找出遞迴式，然後找出計算順序來避免遞迴。這算是標準的DP，也稱為Bottom-up的DP。
- Top-down memoization
 - 開一個表格當作小抄，用來記錄計算過的結果，表格初值設為某個不可能的值，用以辨識是否曾經算過。
 - 在遞迴之前，先偷看一下小抄是否曾經算過，如果小抄上有答案(已經算過)，則直接回傳答案。
 - 如果小抄上沒有，就遞迴呼叫，但計算完畢回傳前，要把它記在小抄上。

Example

```
1 //ch6 stair, top-down DP
2 #include <stdio>
3 long long F[100]={0}; // memo, 0 if not computed
4 long long stair(int n) {
5     // if (n<3) return n; not necessary
6     if (F[n]>0) return F[n]; // check memo
7     F[n] = stair(n-1)+stair(n-2); // record to memo
8     return F[n];
9 }
10 int main() {
11     int n;
12     scanf("%d", &n);
13     F[1]=1, F[2]=2;
14     printf("%lld\n", stair(n));
15     return 0;
16 }
17
```


1d/0d DP

- P-6-2. 不連續的表演酬勞
 - 非負整數序列中挑選不相鄰的最大總和
- 把每天可以獲得的酬勞放在一個一維陣列 $p[]$ 中
- 子問題定義為： $dp[i]$ 是前 i 天可以獲得的最大報酬。
 - 把第 i 天選或不選
 - 如果第 i 天不選，前 i 天的獲利就是前 $i-1$ 天的獲利， $dp[i]=dp[i-1]$ ；
 - 如果第 i 天要選，可以獲得 $p[i]$ ，但是第 $i-1$ 就不可選了，因此最大的獲利是 $p[i]+dp[i-2]$ 。
 - 在第 i 天選或不選之間挑最大的，
 - $dp[i] = \max(dp[i-1], p[i]+dp[i-2])$ 。

2D0D DP

- P-6-7. LCS Longest Common Subsequence
- 我們定義 $lcs[i][j]$ 為：s1 的前 i 個字元與 s2 的前 j 個字元的 LCS 長度
 - 如果 $s1[i] \neq s2[j]$ ：既然兩者不相等，可以刪掉 s1 的最後一個字母 $s1[i]$ 或者刪掉 $s2[j]$ ，因為要找越長越好的，所以兩者取其大， $lcs[i][j] = \max(lcs[i-1][j], lcs[i][j-1])$ 。
 - 如果 $s1[i] = s2[j]$ ：既然對中，不對白不對，把其中一個刪除不會更好，因此可以得到 $lcs[i][j] = lcs[i-1][j-1] + 1$ 。

```

8      int n=strlen(s1), m=strlen(s2);
9      int lcs[2][N]={0}, int main::m=1;
10     for (int i=0; i<m; i++) {
11         for (int j=1; j<=n; j++)
12             lcs[to][j] = (s2[i]==s1[j-1])? lcs[from][j-1]+1: \
13                 max(lcs[to][j-1], lcs[from][j]);
14         swap(from, to); // int temp=from; from=to; to=temp;
15     }
16     printf("%d\n", lcs[from][n]); // final result at [from]

```

0/1-knapsack

- P-6-9. 大賣場免費大搬家
 - 你抽中了大賣場的周年慶的抽獎活動，在不超過總重量 W 的限制下，你可任意挑選商品免費帶走。現場一共 n 項商品，每項商品有它的重量與價值，每項商品只可以選或不選，不可以拆開只拿一部份。請計算可以獲得的最大價值總和。
- 令 $d[i][j]$ 是考慮前 i 項物品且重量不超過 j 的最佳解(最大可以獲得價值)
 - 假設第 i 項物品的重量是 $w[i]$ ，而價值是 $p[i]$
 - $w[i] > j$ ：也就是根本不可能挑選， $d[i][j] = d[i-1][j]$ 。
 - 第 i 項物品的重量不超過 j ，但是選擇不放：跟前一個情形一樣。
 - 挑選第 i 項物品：既然已經挑選了第 i 項，那麼前 $i-1$ 項中挑選的重量不超過 $(j - w[i])$ ，因此， $d[i][j] = p[i] + d[i-1][j-w[i]]$ 。

1d1d LIS (APCS 2021.01.Q4)

- P-6-15. 一覽衆山小
 - 在一個數列中找到一個最長的遞增子序列，所以這個問題被稱為 LIS(longest Increasing subsequence)
- 陣列 $\text{last}[L]$ = 「長度 L 的最小可能結尾」
 - 這個陣列的元素是單調遞增的
 - 在第 i 個回合計算 $\text{lis}(i)$ 時，
 - 找到 $L = \max(j: \text{last}[j] < S[i])$ ，
 - 於是我們得到 $\text{lis}(i) = L + 1$ ， $\text{last}[L + 1] = \min(\text{last}[L + 1], S[i])$
 - L 是小於 $S[i]$ 的最大的那一個 $\text{last}[]$ ，也就是說除非 L 是 $\text{last}[]$ 的最後一個元素，否則 $\text{last}[L + 1] \geq S[i]$

```
1 // P-6-15 LIS, using STL
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main() {
6     int n, si;
7     scanf("%d", &n);
8     vector<int> last;
9     for (int i=0; i<n; i++) {
10         scanf("%d", &si);
11         auto it = lower_bound(last.begin(), last.end(), si);
12         if (it==last.end()) last.push_back(si);
13         else *it=si;
14     }
15     printf("%d\n", (int)last.size());
16     return 0;
```

- P-6-17. 切棍子
- 有一台切割棍子的機器，每次將一段棍子會送入此台機器時，我們可以選擇棍子上的切割位置，機器會將此棍子在指定位置切斷，而切割成本是該段棍子的長度。現在有一根長度 L 的棍子，上面標有 n 個需要切割的位置座標，因為不同的切割順序會影響切割總成本，請計算出最小的切割總成本。
- 例如 $L=10$ ，三個切割點的座標是 $(2, 4, 7)$ 。如果切割順序是 $(2, 4, 7)$ ，則第一次切的成本是10，第二次的成本是8，第三次成本6，總成本 $10+8+6=24$ 。如果切割順序改成 $(4, 2, 7)$ ，第一次切的成本是10，切成長度4與6的兩段，第二次的成本是4，第三次成本6，總成本 $10+4+6=20$ 。

P-6-17. 切棍子

- 定義 $\text{cost}(i, j)$ 是第 i 點到第 j 點之間這段的最低切割成本，為了方便，以座標0與 L 當作第0與第 $n+1$ 點，所以我們要計算的就是 $\text{cost}(0, n+1)$
 - 對任意 $i < j$ ， $\text{cost}(i, j)$ 就是一個子問題，所以我們有 $O(n^2)$ 個子問題
 - 對於 $j=i+1$ ， $\text{cost}(i, j)=0$ ，因為中間沒有要切割的點；
 - 否則 $j>i+1$ ，假設第一刀切在 k ，剩下兩段的成本就是 $\text{cost}(i, k)$ 與 $\text{cost}(k, j)$ 。
- DP的思維：既然不知道 k 是多少，那就全部都算吧！所以我們可以得到
$$\text{cost}(i, j) = \begin{cases} 0 & \text{if } j = i + 1 \\ \min_{i < k < j} \{ \text{cost}(i, k) + \text{cost}(k, j) \} + p[j] - p[i] & \text{otherwise} \end{cases}$$
- 其中 $p[j]$ 與 $p[i]$ 是兩點的座標。我們注意等號右邊的座標間格 $(k-i)$ 與 $(j-k)$ 都會小於左邊的 $(j-i)$ ，所以它是區間由大到小的遞迴


```
8 // min cost of [i,j]
9 int cost(int i, int j) {
10     if (memo[i][j]>=0) return memo[i][j];
11     int mincost = oo;
12     for (int k=i+1; k<j; k++)
13         mincost = min( mincost, cost(i,k) + cost(k,j));
14     mincost += p[j] - p[i];
15     return memo[i][j]=mincost;
16 }
17
18 int main() {
19     int n,L;
20     scanf("%d%d", &n, &L);
21     for (int i=1; i<=n; i++)
22         scanf("%d",&p[i]);
23     p[0]=0, p[n+1]=L;
24     for (int i=0;i<n+2;i++)
25         for (int j=i+1;j<n+2;j++)
26             memo[i][j]=-1;
27     for (int i=0; i<n+1; i++)
28         memo[i][i+1] = 0;
29     printf("%d\n", cost(0,n+1));
30     return 0;
```


基本圖論演算法(僅列綱要)

APCS層級的圖論演算法

- 圖的基本知識
 - vertex, edge, neighbor, degree
 - directed and undirected graph
 - weighted and unweighted graph
 - path and cycle
 - connected component
- 資料結構 (how to find neighbors of a vertex)
 - adjacency matrix and adjacency list
- BFS
- DFS
- Directed Acyclic Graph and Topological sort
 - find shortest/longest distances on a DAG

「樹上的問題大部分都可以用DP解決，一個變數不夠，就用兩個。」

「好想抖著拖鞋在樹上打扣，從上往下看，好多問題都變簡單了。」

樹上演算法(摘要)

樹上演算法

- 基本知識
 - tree and forest
 - root, parent, child, leaf, ancestor, descendant
 - binary tree, complete binary tree, binary search tree (BST)
- DFS, BFS, and bottom-up traversal
- 樹是天生的DP，樹是天生的分治
- 經典題
 - diameter, radius, center, median
 - weighted/unweighted independent set, domination set
 - matching

競技程式 -- Beyond APCS

要學的很多，但其實到這階段已經
具備自學能力了



IOI and ICPC

- IOI 有範圍限制(IOI syllabus)而ICPC沒有
 - 簡單來說，競程常見的範圍不在IOI的範圍內的包括
 - max flow (linear programming)
 - 進階數論
 - 複數、高維幾何與三角函數
 - 解遞迴
- 但有些高中競賽也沒規定必須在IOI範圍

起碼的配備

- STL中的裝備
 - priority queue, (multi)set/map, unordered_(multi)set/map
 - lower_bound之類的binary search
 - bitset
- 其他重要資料結構
 - disjoint set (union and find)
 - binary index tree (BIT, Fenwick tree), 線段樹
 - Range Minimum/Maximum Query (RMQ)
 - Lowest Common Ancestor (LCA)

- 進階DP與常用優化
 - 斜率優化
 - totally monotonic (Monge)(四角不等式)
- Graph algorithm
 - 算距離的：Dijkstra, Bellman-Ford, Floyd-Warshall
 - minimum spanning tree: Kruskal, Prim
 - Bipartite matching
- String algorithm
 - KMP
 - suffix array

謝謝