簡介、STL 資料結構及 algorithm 初階

Ting.H

2017 年 9 月 14 日

1 未來的任務

1.1 主要任務

時間	任務名稱	地點	簡介
9/12	校內資訊能競初賽	建中	前天 (?)
9/29	校內資訊能競複賽	建中	取十幾人成為校隊
10/27	北市賽	台北	一二等獎 (10 人) 進入全國賽
12 月	全國賽	台灣	一二等獎 (10 人) 進入 TOI 一階
2 月	校隊補選	建中	遞補已進 TOI 一階的名額
3 月	TOI 入營考	師大分部	前 20 名進入 TOI 一階
3,4 月	TOI 一階	師大分部	14 天兩次考試，選出 12 人進入 TOI 二階
4 月	TOI 二階	師大分部	4 人成為國手
5 月	APIO	師大分部	由 TOI 二階選手參加
9 月	IOI	Japan Tokyo	為國爭光

1.2 次要任務

時間	任務名稱	地點	簡介
11 月	北市軟體競賽	大安高工	初賽是筆試、複賽是上機
11,12 月	NPSC	台大	同校三人組隊報名。有初賽和決賽，每校最多三隊進入決賽。去年獎品好像不錯。

1.3 OJ

簡稱	網址	介紹
TIOJ	tioj.infor.org	全名是 TIOJ INFOR ONLINE JUDGE，是建中的 OJ 據說已有幾十年的歷史，有很多好題，但當然也有不少爛題。
UVa	uva.onlinejudge.org	可稱為史上第一個 OJ，題目多到寫不完，裡面還有不錯的題單。但如果有英文障礙，應該就會很吃力。
ZJ	zerojudge.tw	有一些 UVa 題和入營考題以及不少水題。
CF	codeforces.com	一段時間就有比賽，可以爬世界排名，但時間通常是半夜。
POI	szkopul.edu.pl	波蘭資奧的網站，裡面有很多不錯的題目，好像不能用 c++11。

1.4 網路資源

名稱	網址	介紹
演算法筆記	http://www.csie.ntnu.edu.tw/~u91029/	有很多競賽會用到的演算法的介紹，但是裡面有些錯，所以如果你覺得有問題，那就應該是他寫錯了
c++ reference	http://www.cplusplus.com/reference/	有 C++ 內建程式庫的介紹，有點難懂，但十分詳細。
資訊之芽	www.csie.ntu.edu.tw/~sprout/algo2016/	這是一個課程，網站上有講義、課堂 PPT、回家作業等，也有一些課堂影片，可以去看看。

2 複雜度介紹

2.1 Big O 函數

在介紹複雜度之前，我們先引進一個函數，Big O。Big O 的定義如下:

$$f(N) = O(g(N)), if\ \exists N_0, c > 0, \forall N > N_0, |f(N)| \leq c|g(N)|$$

這個的意思是說存在一個常數 c，在 N 足夠大時，讓 $f(x)$ 不大於 $g(x)$ 的 c 倍。
對於一個多項式如 $f(x) = x^2 + 2x$，可以是 $O(x^2)$，因為當 $x > 2$ 時，$x^2 + 2x$ 不大於 x^2 的 2 倍。當然也可以寫成 $O(x^3)$、$O(2x^2)$ 等等但一般而言我們都會寫 $O(x^2)$。
仔細想一下就會發現 Big O 是複雜度的上界，而還有另外兩個複雜度的符號 $\Omega(N)$ 和 $\Theta(N)$，定義如下:

$$f(N) = \Omega(g(N)), if\ \exists N_0, c > 0, \forall N > N_0, |f(N)| \geq c|g(N)|$$
$$f(N) = \Theta(g(N)), if\ f(N) = O(g(N))\ and\ f(N) = \Omega(g(N))$$

2.2 時間複雜度

時間複雜度，就是描述一個演算法要執行的時間的函數。而我們通常會以「進行基本運算的次數」來估算。而當我們要計算複雜度時，我們主要關心的是它的成長速度，所我們會使用前面提及的 Big O。

Algorithm 1: Bubble Sort

```
template<typename T>
void bubble_sort(T arr[],int n){
    for(int i = 0; i < n-1; i++)
        for(int j = 0; j < n-1-i; j++)
            if(arr[j] > arr[j+1])
                swap(arr[j], arr[j+1]);
}
```

這是氣泡排序的演算法，當 $i = 0$ 時，要執行 $n - 1$ 次比較。從 $i = 0$ 到 $i = n - 2$，總共要執行 $n(n - 1)/2$ 次比較，時間複雜度為 $O(n^2)$。在上方這段程式碼中我們可以發現在最糟的情況或是已經排好的數列之複雜度皆是 $O(N^2)$，若是經過一點優化後，最優複雜度就會是 $O(N)$。

知道了如何估計複雜度之後，要怎麼知道所寫的程式碼是否會在時間內完成而不會 TLE，若你是 C 或 C++ 的使用者，你可以假設一秒可以進行 10^8 次運算。但如果你的程式常數很大或很小時，就需要估一下常數的影響。

2.3 空間複雜度

與時間複雜度差不多，可以用來估計一個演算法所需的空間。看一下題目所限制的 MB 數就應該能自行估計了。

3 C++ 內建資料結構

3.1 資料結構

資料結構，就是用來儲存資料的結構。陣列就是一種資料結構，但如果遇到一些較麻煩的題目時，資料結構可以幫助你 (有些資料結構不是內建的)。以下就講一些常用的內建資料結構吧。

這些資料結構在「標準模板庫」(Standard Template Library，簡稱 STL) 之中，是一個 C++ 的程式庫。這些都在 std 底下。

3.2 Iterator

假設現在有個容器 C，裡面已經裝了一些資料，如果我要遍歷 C 中的所有元素，那要如何做到呢? 如果這個容器是 set 等等它們是沒有下標的，為了處理這個問題，C++STL 為每個容器提供一個成員型別，Iterator(迭代器)。

我們可以用「指標」的概念來理解「迭代器」(實際上，指標算是一種迭代器)。假設現在有個迭代器 it，如果要存取 it 所指向的內容，那就是在前面加上星號 (*it)，與指標相同。以下有迭代器的三種分類:

1. 隨機存取迭代器：這種迭代器能夠和整數的加減法，往後移 x 項、往前移 x 項皆可。當然也可以遞增 (++) 和遞減 (--)，可以把指標當作這種迭代器。
2. 雙向迭代器：只能做遞增 (++) 和遞減 (--) 的運算，也就是後一項和前一項。
3. 單向迭代器：只能做遞增 (++) 的運算，也就是後一項。

迭代器依照使用方式可分兩種:

1. 輸入迭代器：當要讀取迭代器所指向的內容時，迭代器就為輸入迭代器。所有的迭代器皆可當作輸入迭代器。
2. 輸出迭代器：當要直接更改迭代器所指向的內容時，迭代器為輸出迭代器。除了常數迭代器，其他的都可當作輸出迭代器。

C++ 內建的容器有兩種迭代器，一般的迭代器與逆向迭代器。假設有一個容器型別 C 為 c，宣告時為 C::iterator 和 C::reverse_iterator，前者是從前迭代到後，後者是從後迭代到前。若要表示迭代器的頭尾，前者是 c.begin()、c.end()，後者是 c.rbegin()、c.rend()。一件重要的事，*c.end()、*c.rend() 是不存在的。

3.3 vector

vector 是動態陣列，也就是說可以自由增加長度，不像一般的陣列在宣告時就確定了。以下假設變數名為 v：

1. 標頭檔：<vector>
2. 建構式：vector<T> v(size_type a,const T& b)：一開始 v 會被 a 個 b 填滿，若只有指定 a，那 b 會是 T 的預設值。若什麼都沒有指定，v 會是一個空的 vector。複雜度 $O(a)$。
3. v[i]：v 的第 i 項，複雜度 $O(1)$。
4. v.size()：回傳 v 目前的長度，複雜度 $O(1)$。
5. v.push_back(T a)：在 v 的結尾加一個 a，均攤複雜度 $O(1)$。
6. v.pop_back()：刪除 v 的最末項，若 v 為空，會發生無法預期的結果。複雜度 $O(1)$。
7. v.empty()：回傳一個 bool，表示 v 是否為空的，複雜度 $O(1)$。
8. v.clear()：清空 v，但原本的空間不會被釋放掉。複雜度 $O(size)$。
9. v.resize(size_type a,const T& b)：強制將 v 的長度變為 a，若比原本長，則後面加 b 直到長度為 a，若比原本短則將多出的部分捨去。
10. v.reserve(size_type n)：預留放 n 個 T 的空間，若 $n < size$，此函數不造成任何影響。

3.4 string

等價於 basic_string<char>，string 的用法很像 vector<char>，但因為字串很常用，所以有經過一些優化。以下假設變數名為 s (t 為 string)：

1. 標頭檔：<string>
2. s=t：讓 s 變得跟 t 一樣，複雜度不明，但通常是 $O(size_s + size_t)$。
3. s+=t：在 s 的尾端加上 t，複雜度通常是 $O(size_s + size_t)$。
4. s.c_str()：本函式會回傳跟 s 一樣的 C 式字串。在 C++11 中保證複雜度是 $O(1)$。
5. s (比大小或相等的符號) t：回傳比較 s,t 字典序的結果。複雜度通常是 $O(max(size_s, size_t))$。
6. cin>>s：輸入字串至 s，直到讀到空白。
7. cout<<s：輸出字串 s。
8. getline(cin,s,char c)：輸入字串至 s，直到讀到字元 c。未指定時，c 是換行符號。

除了上方所寫的之外，vector 有的 string 都有。

3.5 deque

位於 <deque> 標頭檔。可視為可以在最前面加東西刪東西的 vector。設變數名為 d，d.pop_front()，d.push_front(a) 就是在最前面刪、加東西。但時間和空間複雜度很爛，沒事不要用 deque。

3.6 list

list 是個「雙向鏈結結構」，也就是說對於 list 中的任何一項，都可以 $O(1)$ 知道它的前一項和後一項。但若是要存取第 i 項時的複雜度是 $O(i)$。以下假設變數名為 l：

1. 標頭檔：<list>
2. 建構式：list<T> l(size_type a,const T& b)：同 vector。
3. l.push_front(T a),l.push_back(T a),l.pop_front(),,l.pop_back()：同 deque。
4. l.size()：回傳 l 有幾項，C++11 保證複雜度 $O(1)$。
5. l.empty()：回傳一個 bool，代表 l 是否是空的。複雜度 $O(1)$。
6. l.insert(iterator it,size_type n,T a)：在 it 指的那項的前面插入 n 個 a 並回傳指向 a 的迭代器。複雜度 $O(n)$。
7. l.erase(iterator it)：把 it 所指的那項刪去並回傳指向之後那項的迭代器。複雜度 $O(1)$。
8. l.erase(iterator first,iterator last)：把 [first,last) 指到的東西全部刪掉，回傳 last。複雜度與砍掉的數量呈線性關係。
9. l.splice(iterator it,list& x,iterator first,iterator last)：first 和 last 是 x 的迭代器。此函式會把 [first,last) 只到的東西從 x 中剪下並加到 it 所指的那項的前面。x 會因為這項函式而改變。若未指定 last，那只會將 first 所指的東西移到 it 前方。複雜度與轉移個數呈線性關係。

3.7 Container adaptor

這個東西主要是接收一個容器，然後改造它，成為新的容器。Container adaptor 通常不會有迭代器，以下介紹三個常用的 container adaptor。

3.8 stack

可以想像成一疊書，每次只能在最上面放置或拿走一本書。也就是「後進先出」(LIFO)。stack 有兩個型別參數 T 和 C，T 是內容物的型別，C 是所採用的容器。stack 能使用的容器有 vector、deque 和 list。以下假設變數名為 s：

1. 標頭檔：<stack>
2. 建構式：stack<T,C> s(C& a)：s 一開始會有一份 a 的複製品。C 預設為 deque<T>。
3. s.size(),s.empty()：同 vector。
4. s.top()：存取最後一個進入 s 的元素，複雜度 $O(1)$。
5. s.push(T a)：將一個元素加入 s 中，複雜度 $O(1)$。
6. s.pop()：將最後一個進入 s 的元素移除，複雜度 $O(1)$。

3.9 queue

可以想像為排隊的人群，有可能是新的一個人來排在隊伍的尾端，或是最前面一個人結完帳離開隊伍。也就是「先進先出」(FIFO)。queue 有兩個型別參數 T 和 C，T 是內容物的型別，C 是所採用的容器。queue 能使用的容器有 deque 和 list。以下假設變數名為 q：

1. 標頭檔：<queue>
2. 建構式：queue<T,C> q(C& a)：q 一開始會有一份 a 的複製品。C 預設為 deque<T>。
3. q.size(),q.empty()：同 vector。
4. q.front()：存取第一個進入 q 的元素，複雜度 $O(1)$。
5. q.back()：存取最後一個進入 q 的元素，複雜度 $O(1)$。
6. q.push(T a)：將一個元素加入 q 中，複雜度 $O(1)$。
7. q.pop()：將第一個進入 q 的元素移除，複雜度 $O(1)$。

3.10 priority_queue

priority_queue 利用幾個內建函式實現 binary heap 結構，它可以維持最頂的元素永遠是最大的。priority_queue 有三個型別參數 T、Con 和 Cmp。T 是內容物的型別，Con 是所採用的容器，Cmp 是比大小的依據。priority_queue 能使用的容器有 vector 和 deque。Cmp 的預設值是 less<T>，此時的 priority_queue 是最大堆，若改成 greater<T>，則 priority_queue 為最小堆。以下假設變數名為 pq：

1. 標頭檔：<queue>
2. 建構式：priority_queue<T,Con,Cmp> pq：會有一個空的 priotiry_queue。Con 預設為 vector<T>。
3. 建構式：priority_queue<T,Con,Cmp> pq(iterator first,iterator last)：會有一個 priotiry_queue 內含 [first,last) 所指到的東西。

4. pq.size(),pq.empty()：同 vector。
5. pq.top()：回傳 pq 中最大 (最小) 的元素，複雜度 $O(1)$。
6. pq.push(T a)：將 a 加入 pq 中，複雜度 $O(\log size)$。
7. pq.pop()：將 pq 中最大 (最小) 的元素移除，複雜度 $O(\log size)$。

若你已經確定 pq 內要放什麼東西，那直接寫在建構式的複雜度比較好，只有 $O(size)$，但如果是一個一個塞進去的話，複雜度是 $O(size \log size)$。一般情況下其實沒差，但知道一下還是比較好。

3.11 pair

pair 的目的是把兩個變數綁在一起 (可以是不同型別)。以下假設變數名為 p：

1. 標頭檔：<utility>
2. 建構式：pair<A,B> p(A a,B b)：將 A,B 兩型別綁在一起，第一項為 a，第二項為 b。
3. p=s：若 s 也是同型別，則 p 會變得跟 s 一樣。
4. p(比較大小或相等的符號)s：若 s 也是同型別，則會比較兩者的大小，先比第一項再比第二項。
5. p.first、p.second：存取第一項、第二項。

make_pair 是一個非成員函式。make_pair 的好處是不用指明第一項和第二項的型別，用法是 make_pair(A a,B b) 函式會回傳一個 pair。另外，C++ 還有 <tuple>，可以自行研究。

3.12 set

set 實現了紅黑樹，也就是說可以 $O(\log n)$ 插入、刪除或查詢一個值是否在其中。特別的是，裡面的元素是不會重複的，因此我們會稱元素的值為鍵值 (Key)。以下假設變數名為 s：

1. 標頭檔：<set>
2. 建構式：set<K> s：建構一個空的 s，複雜度 $O(1)$。
3. s.size(),s.empty()：同 vector。
4. s.insert(K k)：在 s 中放入一個件值為 k 的元素，若本來就有，則什麼事都不會做，複雜度 $O(\log size)$。
5. s.erase(iterator first,iterator last)：刪除 [first,last)，若沒有指定 last 則只刪除 first，複雜度與刪除的個數呈線性。

6. s.erase(K k)：刪除鍵值為 k 的元素，並回傳刪除的個數。複雜度 $O(\log n)$。
7. s.find(K k)：回傳指向鍵值為 k 的迭代器，若 k 值不存在，則回傳 s.end()。複雜度 $O(\log n)$。
8. s.count(K k)：回傳有幾個鍵值為 k 的元素，複雜度 $O(\log n)$。
9. s.lower_bound(K k)：回傳迭代器指向第一個鍵值大於等於 k 的項。複雜度 $O(\log n)$。
10. s.upper_bound(K k)：回傳迭代器指向第一個鍵值大於 k 的項。複雜度 $O(\log n)$。

3.13 map

map 可以 $O(\log n)$ 插入、刪除或查詢一個鍵值對應的值。map 有兩個型別參數 K 和 T，K 是鍵值的型別，T 是對應的值的型別。map 中每一個元素其實是 pair<K,T>，所以迭代器指向的東西是一個 pair。以下假設變數名為 m：

1. 標頭檔：<map>
2. 建構式：map<K,T> m：建構一個空的 m，複雜度 $O(1)$。
3. m.size()、m.empty()、m.erase(iterator first,iterator last)、m.erase(K k)、m.find(K k)、m.count(K k)、m.lower_bound(K k)、m.upper_bound(K k)：同 set。
4. m[k]：存取鍵值 k 對應的值，若 k 沒有對應的值，會插入一個元素，使 k 對應到預設值並回傳之。複雜度 $O(\log n)$。
5. m.insert(pair<K,T> k)：若沒有鍵值為 k.first 的值，插入一個鍵值為 k.first 的值對應到 k.second，並回傳一個 pair，first 是指向剛插入的元素的迭代器、second 是 true；若已經有了，回傳一個 pair，first 是指向鍵值為 k.first 的元素的迭代器，second 是 false。

3.14 multiset,multimap

在 <set>,<map> 中分別有 multiset 和 multimap，與前面的差不多，唯一的差別在於 multiset 和 multimap 中鍵值可以重複出現，由於 multimap 中一個鍵值可能對應到不同的值，所以不支援下標。multiset 和 multimap 有一個函式 equal_range(K k)，會回傳一個 iterator 的 pair，第一項代表 lower_bound(k)，第二項代表 upper_bound(k)。這兩項迭代器之間的項就是那些鍵值是 k 的項。set 跟 map 也有這個函式，但其實沒什麼用。

3.15 unordered_(multi)set,unordered_(multi)map

unordered_(multi)set 和 unordered_(multi)map 優化掉 set 和 map$O(\log n)$ 的複雜度，這兩個分別在在標頭檔 <unordered_set> 和 <unordered_map> 詳細的可以自己查，以下只寫一些重點:

1. unordered 系列的迭代器為單向迭代器。
2. unordered 系列沒有將所有項依鍵值排序 (這也是它名字的由來)，因此迭代器在遍歷容器時不會依鍵值的大小順序遍歷。
3. 因為沒有排序，所以當然沒有 lower_bound、upper_bound。
4. 比起 (multi)map 和 (multi)set，unordered 系列的期望複雜度少一個 log。

在宣告變數 (建構式) 時，你可以指定 bucket 至少有幾個。如果鍵值是內建型別而且你沒有指定 bucket 至少有幾個，那麼它會自己幫你搞定。

3.16 bitset

你可以將它想像為一堆 0 和 1 的陣列，也就是說 bitset 大小是固定的。但每一項只用到 1bit 的空間，並且 bitset 的位元運算是被優化過的，對常數優化及空間壓縮有不錯的效用。以下假設變數名為 b：

1. 標頭檔：<bitset>
2. 建構式 bitset<N> b(a)：用 a 初始化一個長度為 N 的 bitset。這裡 a 可以是 unsigned long、string 或 C 式字串。如果沒有指定 a，或者如果 b 有一些地方沒被 a 初始化，那些地方預設為 0。
3. b.count()：回傳 b 有幾個位元是 1。複雜度 $O(N)$。
4. b.size()：回傳 b 有幾個位元。複雜度 $O(1)$。
5. b(位元運算)：不管是一元還是二元的位元運算都可以。如果是兩個 bitset 的二元位元運算，兩個 bitset 的長度需一致。複雜度 $O(N)$。
6. b[a]：存取第 a 位。複雜度 $O(1)$。
7. b.set()：將所有位元設為 1。複雜度 $O(N)$。
8. b.reset()：將所有位元設為 0。複雜度 $O(N)$。
9. b.flip()：將所有位元的 0、1 互換 (反白)。複雜度 $O(N)$。
10. b.to_string()：回傳一個字串和 b 的內容一樣。複雜度 $O(N)$。
11. b.to_ulong()：回傳一個 unsigned long 和 b 的內容一樣 (在沒有溢位的範圍內)。複雜度 $O(N)$。

bitset 不是容器，而且它也沒有迭代器。通常而言，如果要估計常數的話，相較於直接使用陣列，空間是 1/8、count 約是 1/6、位元運算約是 1/30。當然這些都不是絕對的。
要注意的是，上述的複雜度沒有明文規定，不過通常是如此。

3.17 習題

知道這些還不夠，要實際練習才有用。

1. (ZJ b298)(vector)
2. (ZJ b291)(vector,map,string)
3. (UVa 514)(ZJ c123)(stack)
4. (TIOJ 1993)(bitset)
5. (TIOJ 1613)(priority_queue)
6. (TIOJ 1809)(set)
7. (TIOJ 1168)(priority_queue,bitset/vector)

4 <algorithm>

C++ 裡給了一個標頭檔 <algorithm>，裡面有很多種函式，但這些函式並不知道它會接收到的東西長怎樣，此時迭代器就發揮了功勞。
以下大約介紹一些常用的函式:

4.1 排序

1. sort(iter first,iter last,Cmp cmp)：將 [first,last) 依照 cmp 排序，使得若 a 嚴格地在 b 前面，則 cmp(a,b) 為真。這裡要求 iter 是隨機存取迭代器，而 cmp 是一個接受兩個參數，回傳一個 bool 的函數指標或函數物件。不指定 cmp 時會由小排到大。複雜度 $O(Cn\log n)$，其中 C 是 cmp 的複雜度。
2. stable_sort：跟 sort 一樣，然而保證如果有兩項 a 跟 b 的值一樣且 a 一開始在 b 前面，那麼最後 a 也會在 b 前面。一般來說，有額外空間的話時間複雜度 $O(n\log n)$、額外空間複雜度 $O(n)$。但是如果沒辦法找到額外的空間，時間複雜度會變成 $O(n\log^2 n)$。
3. nth_element(iter first,iter nth,iter last,Cmp cmp)：將排序後應該會在第 nth 位置的元素 x 移到第 nth 個位置，並且讓應該在 x 前面的所有元素都在 x 前面，反之亦同。參數要求同 sort。平均複雜度 $O(n)$，但未保證最差複雜度。

4.2 對付已排序序列的函式

1. lower_bound(iter first,iter last,T t,Cmp cmp)：跟 set 的 lower_bound 是一樣的意思。若 iter 是隨機存取迭代器，複雜度 $O(\log n)$。若不然，複雜度 $O(n)$。如果 iter 是 set 和 map 系列的迭代器，請直接採用 set 和 map 的成員函式。
2. upper_bound(iter first,iter last,T t,Cmp cmp)：跟 set 的 upper_bound 是一樣的意思。
3. equal_range(iter first,iter last,T t,Cmp cmp)：回傳一個 pair，第一項是 lower_bound，第二項是 upper_bound。
4. merge(iter1 first1,iter1 last1,iter2 first2,iter2 last2,iter3 res,Cmp cmp)：假設 [first1, last1) 和 [first2, last2) 是兩個經 cmp 排序好的序列，將兩個序列合併並排序，輸出至以 res 為首的序列並回傳指向序列末端的迭代器。記得讓 res 後面有足夠的空間。複雜度線性。
5. set_union(iter1 first1,iter1 last1,iter2 first2,iter2 last2,iter3 res,Cmp cmp)：假設 A=[first1,last1) 和 B=[first2, last2) 是兩個經 cmp 排序好的序列，將兩個序列取聯集排序輸出至以 res 為首的序列並回傳指向序列末端的迭代器。其餘同 merge。
6. set_intersection：同 set_union，但此函式取的是交集。
7. set_difference：同 set_union，但此函式取的是餘集 $(A-B)$。
8. set_symmetric_difference：同 set_union，但此函式取的是對稱餘集 $(A \cup B - A \cap B)$。

4.3 字典序

1. next_permutation(iter first,iter last,Cmp cmp)：將 [first,last) 變成原本的某個排序，使得字典序是比原本大的所有排序中最小的。如果成功了，這個函式會回傳 true；否則回傳 false，並將 [first,last) 變成字典序最小的那個排序。
2. prev_permutation(iter first,iter last,Cmp cmp)：將 [first,last) 變成原本的某個排序，使得字典序是比原本小的所有排序中最大的。如果成功了，這個函式會回傳 true；否則回傳 false，並將 [first,last) 變成字典序最大的那個排序。

4.4 習題

1. (ZJ a233) 裸排序題
2. (TIOJ 1617)(IOI 2000)(Interactive)(nth_element)
3. (TIOJ 1907)LIS 最長的嚴格遞增子序列 (sort,upper_bound)