qA. ICPC Tutorial

題目簡述：

判斷輸入規模 n 的情況下，採用時間複雜度 t 的演算法，在每秒可執行 m 個指令的電腦下可否在一秒內執行完畢。若可輸出 ”AC” 否則輸出”TLE” 。

t = 1 , O(n!)
t = 2 , O(2^n)
t = 3 , O(n^4)
t = 4 , O(n^3)
t = 5 , O(n^2)
t = 6 , O($n\log_2 n$)
t = 7 , O(n)

解法：

根據 t 對應的數學公式代入 n ， 判斷結果是否 ≦ m
t 若為 1..4 用迴圈跑，每次檢查目前執行數是否已確定答案為 ”TLE“

留意：

- 要用 long long
- 計算 $n\log_2 n$ 要算無條件進位，因為條件式 ≦ m

My AC Code：

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/a.cpp

```
#include <cstdio>
#include <cmath>
using namespace std;

int main(){
   long long m , n;
   int t;
   scanf("%lld%lld%d",&m,&n,&t);
   long long cnt = 1LL;
   switch(t){
      case 1:
         while( n && cnt<=m ){
            cnt = cnt*n;
            --n;
         }
         break;
      case 2:
         while( n && cnt<=m ){
            cnt <<= 1;
```

```
            --n;
        }
        break;
    case 3:
        for(int i=0 ; i<4 && cnt<=m ; ++i)
            cnt = cnt*n;
        break;
    case 4:
        for(int i=0 ; i<3 && cnt<=m ; ++i)
            cnt = cnt*n;
        break;
    case 5:
        cnt = n*n;
        break;
    case 6:
        cnt = ceil(n*log2(n));
        break;
    default:
        cnt = n;
        break;
  }
  if( cnt<=m ) printf("AC");
  else printf("TLE");
  return 0;
}
```

qB. 2048

題目簡述：

在一個 4 x 4 的方格內，每個格子有一個正整數（二的次方數）或為空（用 0 表示）。有四種不同的操做（上，下，左，右），會讓每個非空的格子同時往該方向移動直到撞到邊緣或其他格子；若兩個同樣數字的格子狀在一起，會變成一個格字其數字為兩者的和，並繼續往該方向移動。合併後的格子無法再與其他格子合併。

給定當前的狀態及一個方向的操作，輸出執行操作後的狀態

解法：

實作『往上操作』(up) 跟『順時鐘旋轉』(rotate)

『往上操作』 up：把會合併的格子處理完在全部推到最上面

處理合併
從上到下掃
若該格非空則往下找同一 col 第一個出現的非空格子
若找到了則將其刪除並把此格乘2
如此，每個格子只會合併一次
若有三個一樣的數字在同一 col，較上方的兩格會先合併

全部往上推
上到下掃
每個空格子往下找同一 col第一個出現非空的格子
若找到了把它刪掉存到這格
若找不到則這格為 0

往左操作：rotate -> up -> rotate -> rotate
往下操作：rotate -> rotate -> up -> rotate
往右操作：rotate -> rotate -> rotate -> up

My AC Code：

souce code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/b.cpp

```
#include <cstdio>
#include <cstring>
using namespace std;

int ori[4][4];
int tmp[4][4];

void rotate(){
	for(int i=0 ; i<4 ; ++i)
		for(int j=0 ; j<4 ; ++j)
```

```
                        tmp[i][j] = ori[3-j][i];
        for(int i=0 ; i<4 ; ++i)
                for(int j=0 ; j<4 ; ++j)
                        ori[i][j] = tmp[i][j];
}

void up(){
        for(int i=0 ; i<4 ; ++i)
                for(int j=0 ; j<4 ; ++j){
                        if( ori[i][j]==0 ) continue;
                        for(int k=1 ; i+k<4 ; ++k)
                                if( ori[i+k][j]!=0 ){
                                        if( ori[i+k][j] == ori[i][j] )
                                                ori[i+k][j] = 0 , ori[i][j] <<= 1;
                                        break;
                                }
                }
        for(int i=0 ; i<4 ; ++i)
                for(int j=0 ; j<4 ; ++j){
                        int b = 0;
                        for(int k=0 ; i+k<4 ; ++k)
                                if( ori[i+k][j] ){
                                        b = ori[i+k][j];
                                        ori[i+k][j] = 0;
                                        break;
                                }
                        ori[i][j] = b;
                }
}

void left(){
        rotate(); up(); rotate(); rotate(); rotate();
}
void right(){
        rotate(); rotate(); rotate(); up(); rotate();
}
void down(){
        rotate(); rotate(); up(); rotate(); rotate();
}

int main(){
        for(int i=0 ; i<4 ; ++i)
                for(int j=0 ; j<4 ; ++j)
                        scanf("%d",&ori[i][j]);

        int way;
```

```
        scanf("%d",&way);
        if( way==0 ) left();
        else if( way==1 ) up();
        else if( way==2 ) right();
        else down();

        for(int i=0 ; i<4 ; ++i){
                for(int j=0 ; j<4 ; ++j)
                        printf("%d ",ori[i][j]);
                printf("\n");
        }
        return 0;
}
```

qC. VisuaAlgo Online Quiz

題目簡述：

求最多 10000 个點，最多 200000 條有向邊（權重介於1～99）的圖，一源點到一終點的最短路共有幾種走法。不保證連通。

解法：

因為都是正邊，所以用 dijkstra + priority_queue 可以保證每次拿出來的點必已經建構好最短路，且之後不可能有其他點可以讓這點更接近原點（否則那點應該先被拿出來）。同時表示，若有紀錄到此點的方法數，到此點的最短路走法不會再改變。

因此，在鬆弛時處理方法數

若讓目標點更近原點，則更新其方法數為到目前點的方法數

若不會讓目標點更近或更遠，則將其方法數加上到目前點的方法數

My AC Code：

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/c.cpp

```cpp
#include <cstdio>
#include <algorithm>
#include <vector>
#include <queue>
#define PII pair<int,int>
#define INF 1023456789
using namespace std;

// cost , nextId
vector<PII> g[10004];
int sD[10004];
int way[10004];
bool cnted[10004];

struct myCmp{
	bool operator()(const int&l , const int&r) const{
		return sD[l] > sD[r];
	}
};

int main(){
	int v , e;
	scanf("%d%d",&v,&e);
	for(int i=0,u,v,w ; i<e ; ++i){
		scanf("%d%d%d",&u,&v,&w);
		g[u].push_back( {w,v} );
```

```
        }
        int s,t;
        scanf("%d%d",&s,&t);

        for(int i=0 ; i<v ; ++i)
                sD[i] = INF;

        priority_queue< int , vector<int> , myCmp > myPQ;
        myPQ.push( s );
        sD[s] = 0;
        way[s] = 1;
        while( !myPQ.empty() && myPQ.top()!=t ){
                int nowId = myPQ.top();
                myPQ.pop();
                if( cnted[nowId] ) continue;
                cnted[nowId] = true;
                for(auto &v : g[nowId]){
                        int nextId = v.second;
                        int w = v.first;
                        if( sD[nextId] > sD[nowId] + w ){
                                sD[nextId] = sD[nowId] + w;
                                way[nextId] = way[nowId];
                                myPQ.push( nextId );
                        }
                        else if( sD[nextId] == sD[nowId] + w ){
                                way[nextId] += way[nowId];
                        }
                }
        }
        printf("%d\n",way[t]);
        return 0;
}
```

qD. Grid MST

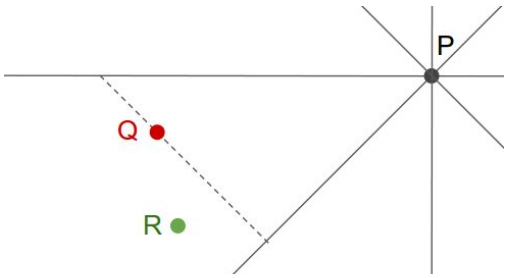
題目簡述：

[0,1000] x [0,1000] 的座標平面上，給定 n ≤ 1e5 个格子點，點可能重疊，點之間的距離為曼哈頓距離。求這 n 个點的 minimum spanning tree。

解法：

最多 n^2 條邊，記憶體裝不下也跑不完，所以要利用格子點之間曼哈頓距離的特性來刪邊。

一個點用『米』字切八個區域，這個點只要跟這八個區域中最靠近自己的點建邊就可以了。見下圖，虛線為跟 P 的等曼哈頓距離線，Q是P 此區域的最近點



若 MST 存在邊 PR，我們可以用如下方式讓 MST 更好：

先證明 PR 必是三邊中的最長邊，PQ < PR 顯而易見。虛線上兩個端點跟 P 洽形成曼哈頓距離下三邊等距的三角形，因此當 P 通過虛線時，虛線上任何一點都可以在小於等於 PQ 的距離走到該點，故 QR 必小於等於 PR。

結論 PR 是最長邊。

斷開 PR，考慮 Q 跟 P 還是 R 連通而將 QR 或 QP 放入 MST。

如此我們得到更好的 MST。

為了簡單實作，每次每個點只考慮上圖中的那個區域，做完後所有點再繞原點轉 45 度就可以除理完所有區域了。旋轉時可以 newX = x+y , newY = -x+y 直接放大 $2^{0.5}$ 並不影響找最近點的結果，只是原本的位置要另外存用來算距離。

實做找每個點找八點鐘區域的最近點時。先把每個點按先比 (x-y) 在比 x 從小到大排好，然後用一個 map[x+y] => id 維護該等距線最靠近目前掃到 x-y =k 這條線的點。

因為 grid 範圍很小，所以每次拿出一個點時，掃過 map 中所有的點 (≤ 2000個)，找出 map 中 y 值比自己小最靠近我的點，跟他建邊。然後把自己丟入 map 中。建邊複雜度 O(2000 * 1e5 * 8) 為了壓常數在9秒內跑完，其實每個點只要掃過下方的點就行了 O(2000 * 1e5 * 4) 如此可以勉強過關。

這個地方也可以用 Treap 實作，y 當 key 值，維護區間 x+y 的最大值。如此每次可以在 log2000 把 y 比自己小的樹切下來問最大的 x+y 建邊。O(log2000 * 1e5 * 4)

最後總共 4n 條邊，跑 Kruskal 或 Prim 得到 MST 的花費

參考自：https://goo.gl/lvJEU4

My AC Code：

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/d.cpp

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <map>
#include <algorithm>
using namespace std;

struct Point{
	int x,y,id;
	int vX,vY,lb2rt;
	Point(const int &_x,const int &_y,const int &_id)
	:x(_x) , y(_y) , id(_id) , vX(_x) , vY(_y) , lb2rt(_x-_y){}
};
void rotate(Point &v){
	int tmpx = v.vX , tmpy = v.vY;
	v.vX = tmpx + tmpy;
	v.vY = -tmpx + tmpy;
	v.lb2rt = v.vX - v.vY;
}
int dis(const Point &l,const Point &r){
	return abs(l.x - r.x) + abs(l.y - r.y);
}
vector< Point > V;

struct Edge{
	int u , v , len;
	Edge(const int &_u,const int &_v)
	:u(_u) , v(_v){
		len = dis(V[_u] , V[_v]);
	}
	bool operator < (const Edge &rth) const{
		return len < rth.len;
	}
};

vector< Edge > E;
bool grid[1004][1004];
```

```
vector< int > ids;
bool myCmp(const int &l,const int &r){
	return V[l].lb2rt < V[r].lb2rt ||
		(V[l].lb2rt==V[r].lb2rt && V[l].vX<V[r].vX);
}

void scan(){
	sort( ids.begin() , ids.end() , myCmp );

	map<int,int> line;
	for(auto &id : ids){
		int nowLine = V[id].vX + V[id].vY;
		int minId = -1;
		for(auto &l : line){
			if( l.first >= nowLine ) break;
			if( V[l.second].vY > V[id].vY ) continue;
			if( minId==-1 || dis(V[l.second] , V[id]) < dis(V[minId] , V[id]) )
				minId = l.second;
		}
		if( minId != -1 )
			E.push_back( Edge(minId , id) );
		line[nowLine] = id;
	}
}
void rotateAll(){
	for(auto &v : V) rotate(v);
}
void buildEdge(){
	for(int i=0 ; i<V.size() ; ++i)
		ids.emplace_back( i );
	for(int i=0 ; i<4 ; ++i){
		scan();
		rotateAll();
	}
}

int root[100004];
int findRoot(int from){
	if( root[from]==from ) return from;
	return root[from] = findRoot( root[from] );
}
int kruskal(){
	for(int i=0 ; i<100004 ; ++i)
		root[i] = i;
	int sumDis = 0;
```

```
        sort(E.begin() , E.end());
        for(auto &e : E){
                int a = findRoot(e.u);
                int b = findRoot(e.v);
                if( a==b ) continue;
                sumDis = sumDis + e.len;
                root[b] = a;
        }
        return sumDis;
}

int main(){
        int N;
        scanf("%d",&N);
        for(int i=0 ,x,y ; i<N ; ++i){
                scanf("%d%d",&x,&y);
                if( !grid[x][y] ){
                        V.emplace_back( Point(x , y , V.size()) );
                        grid[x][y] = true;
                }
        }
        buildEdge();
        printf("%d\n",kruskal());
        return 0;
}
```

qE. Rectangle Land

題目簡述：

在 x: [-1e6 , 1e6] y: [-1e6 , ye6] 的範圍內，給定一些訊號臺覆蓋的矩形範圍及強度，一點的強度為所有覆蓋到此點的強度相加。訊號臺數量 ≤ 1e5

求在所有點中強度的最大值及最大強度的總面積。

解法：

x 方向用掃描線左掃到右，y 方向建 segment tree 維護資訊跟答案。

將所有訊號範圍拆成左右界丟入掃描線的事件中，左界代表要在其覆蓋的 y 區間增加強度，右界則要減少強度。全部事件用 x 位置排序，同一 x 位置應先處理右界，否則本來沒重疊的會算成重疊。實作上因為整數除法無法整除時，正的會往小，負的會往大，為避免 segment tree 實作可能會出的問題，故將所有座標都 +1e6 位移到非負數。

setment tree 要能做區間修改跟區間查詢，更新操作有加跟減。每個節點要維護區間最大值跟區間有此最大值的個數，區間加減法可直接加減在最大值上且對最大值個數沒有影響。

面積需要紀錄一區間最近更新的 x 位置跟那些有最大值的 y 『左界 x 的總和』（以下稱 sumX）。

利用 sumX 可以計算最大訊號的面積：

最大值個數 * 目前 x 位置 - 左界 x 的總和

此公式是整理自 ∑ nowx - lastX 對於此區間所有有最大值的 y

sumX 的維護要靠最近更新的 x 位置：

在更新一整塊區間時，標記最近更新的 x 位置。

push 時，若一區間有標記最近更新的 x 位置，則讓此區間的 sumX 變成 最大值個數 * 最近更新的 x 位置，並取消最近更新標記。

pull 時，sumX 的值依據左子區間 大於、小於、等於 右子區間，更新成 左子區間的 sumX 、右子區間的 sumX、左右子區間的 sumX 和。

如此，在執行減法前（掃到右界）先 query 該區間以更新答案即可。

My AC Code：

source code

https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/e.cpp

```
#include <cstdio>
#include <cstdlib>
```

```cpp
#include <vector>
#include <algorithm>
using namespace std;

typedef pair<int,long long> PIL;

struct event{
	int x , top , btn , diff;
	event(int _x , int _top , int _btn , int _diff)
	:x(_x) , top(_top) , btn(_btn) , diff(_diff) {}
	bool operator < (const event &rth) const{
		return x < rth.x || (x == rth.x && diff < rth.diff);
	}
};

struct node{
	node *l = nullptr , *r = nullptr;
	int lb , rb;
	bool tag = false;
	int diff = 0 , lastX;
	int maxS = 0 , maxN;
	long long sumX = 0LL;
	node(){}
	node(const int &_lb,const int &_rb)
	:lb(_lb) , rb(_rb){
		maxN = _rb - _lb + 1;
	}
};

inline void push(node *nd){
	if( !nd->tag ) return;
	nd->maxS += nd->diff;
	nd->sumX = 1LL * nd->maxN * nd->lastX;
	if( nd->l ){
		nd->l->diff += nd->diff;
		nd->l->lastX = nd->lastX;
		nd->l->tag = true;
		nd->r->diff += nd->diff;
```

```
                nd->r->lastX = nd->lastX;
                nd->r->tag = true;
        }
        nd->diff = 0;
        nd->tag = false;
}
inline void pull(node *nd){
        if( !nd->l ) return;
        push(nd->l);
        push(nd->r);
        nd->maxS = max( nd->l->maxS , nd->r->maxS );
        nd->maxN = nd->sumX = 0LL;
        if( nd->l->maxS == nd->maxS )
                nd->maxN += nd->l->maxN , nd->sumX += nd->l->sumX;
        if( nd->r->maxS == nd->maxS )
                nd->maxN += nd->r->maxN , nd->sumX += nd->r->sumX;
}

PIL query(node *nd,int tl,int tr , int X){
        push(nd);
        if( tl==nd->lb && tr==nd->rb ){
                return {nd->maxS , 1LL* nd->maxN*X - nd->sumX};
        }
        int mid = (nd->lb + nd->rb)>>1;
        if( tr<=mid )
                return query(nd->l , tl,tr , X);
        if( tl >mid )
                return query(nd->r , tl,tr , X);
        PIL tmp1 = query(nd->l , tl,mid , X);
        PIL tmp2 = query(nd->r , mid+1,tr , X);
        if( tmp1.first > tmp2.first ) return tmp1;
        if( tmp2.first > tmp1.first ) return tmp2;
        return {tmp1.first , tmp1.second + tmp2.second};
}
void rangeOP(node *nd,int tl,int tr , int diff , int X){
        push(nd);
        if( tl==nd->lb && tr==nd->rb ){
                nd->diff += diff;
```

```
                nd->lastX = X;
                nd->tag = true;
                return;
        }
        int mid = (nd->lb + nd->rb)>>1;
        if( tr<=mid ){
                rangeOP(nd->l , tl,tr , diff,X);
        }
        else if( tl >mid ){
                rangeOP(nd->r , tl,tr , diff,X);
        }
        else{
                rangeOP(nd->l , tl,mid , diff,X);
                rangeOP(nd->r , mid+1,tr , diff,X);
        }
        pull(nd);
}

node mem[4100000] , *memtop;
node* build(int l,int r){
        *memtop = node(l,r);
        node *now = memtop++;
        if( l==r ) return now;
        int mid = (l+r)>>1;
        now->l = build(l , mid);
        now->r = build(mid+1 , r);
        return now;
}

inline void init(){
        node *i = mem;
        while( i!=memtop ){
                i->tag = false;
                i->maxS = i->diff = 0;
                i->maxN = i->rb - i->lb + 1;
                i->sumX = 0LL;
                ++i;
        }
```

```
}

int main(){
	memtop = mem;
	build(0 , 2000004);

	int C;
	scanf("%d",&C);
	while( C-- ){
		int n;
		scanf("%d",&n);
		vector< event > es;
		for(int i=0 , x1,y1,x2,y2,s ; i<n ; ++i){
			scanf("%d%d%d%d%d",&x1,&y1,&x2,&y2,&s);
			x1 += 1000000 , x2 += 1000000;
			y1 += 1000000 , y2 += 1000000;
			es.emplace_back( event(x1,y2,y1+1, s) );
			es.emplace_back( event(x2,y2,y1+1,-s) );
		}
		sort( es.begin() , es.end() );
		long long maxSig = 0LL , maxArea = 0LL;
		for(auto &v : es){
			if( v.diff > 0 )
				rangeOP(mem , v.btn,v.top , v.diff , v.x);
			else{
				PIL qq = query(mem , v.btn,v.top , v.x);
				if( qq.first == maxSig )
					maxArea += qq.second;
				else if( qq.first > maxSig ){
					maxSig = qq.first;
					maxArea = qq.second;
				}
				rangeOP(mem , v.btn,v.top , v.diff , v.x);
			}
		}
		printf("%lld %lld\n",maxSig , maxArea);

		if( C ){
```

```
                    init();
                }
            }
        }
```

qF. Triangles

題目簡述：

在 [0 , 1e6) x [0 , 1e6) 的範圍內給定 N≤1e6 個不同的點，不同點可在同位置。求任三不同點所圍出的面積平方和，輸出到小數點後第四位。

解法：

定義 f(i , j , k) 為點 i , j , k 所為出的面積平方

則答案為 $\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} f(i,j,k) / 6$

a = xi , b = yi

c = xj , d = yj

e = xk , f = yk

f(i , j , k) = [(c - a)(f - b) - (e - a)(d - b)] ^{2} / 4

= (cf - af - bc + ad + be - de) ^{2} 展開整理

- 利用 $\sum x+y = \sum x + \sum y$

  $\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} f = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} f1 + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} f2 + \ldots.$

  其中 f1 , f2 …. 是 f 展開後的各項

- 利用 $\sum_{i=1}\sum_{j=1} ij = \sum_{i=1} i \sum_{j=1} j$

  可以發現每一項 fi 都是可以預先在 n 次內算好的

整理結果很漂亮

A = sigma(x)

B = sigma(y)

AA = sigma(x^2)

BB = sigma(y^2)

AB = sigma(xy)

tmp1 = AA*BB*N + AB*A*B*2

tmp2 = AB*AB*N + BB*A*A + AA*B*B

答案就是 (tmp1 - tmp2) / 4

此題答案可到 40 位數以上，需要自己實作大數

減法拉到最後算，所以只要實作較大正整數減正整數

除法也拉到最後算，所以只要實作除以 4 即可

My AC Code :

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/f.cpp

```cpp
#include <cstdio>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;

struct BigNum{
	vector<int> val;
	int sz() const{
		return val.size();
	}
	int get(int id) const{
		return id<sz() ? val[id] : 0;
	}
	void add(int id,int v){
		while( id>=sz() ) val.push_back(0);
		val[id] += v;
	}

	BigNum(){ val.push_back(0); }
	BigNum(long long n){
		while( n ){
			val.push_back( n%10000LL );
			n /= 10000LL;
		}
		if( !sz() ) val.push_back(0);
	}
	void fix(){
		int c = 0;
		for(int i=0 ; i<sz() ; ++i){
			val[i] += c;
			c = val[i] / 10000;
			val[i] %= 10000;
```

```
        }
        if( c ) val.push_back( c );
}
BigNum operator + (const BigNum &rth) const{
        BigNum tmp;
        int to = max(sz() , rth.sz());
        for(int i=0 ; i<to ; ++i)
                tmp.add( i , get(i) + rth.get(i) );
        tmp.fix();
        return tmp;
}
BigNum operator * (const BigNum &rth) const{
        BigNum tmp;
        for(int i=0 ; i<sz() ; ++i){
                for(int j=0 ; j<rth.sz() ; ++j)
                        tmp.add( i+j , get(i)*rth.get(j) );
                tmp.fix();
        }
        return tmp;
}
BigNum operator + (const long long &rth) const{
        BigNum tmp = rth;
        tmp = tmp + *this;
        return tmp;
}
BigNum operator * (const long long &rth) const{
        BigNum tmp = rth;
        tmp = tmp * (*this);
        return tmp;
}
BigNum operator - (const BigNum &rth) const{
        BigNum tmp = *this;
        for(int i=0 ; i<tmp.sz() ; ++i){
                if( tmp.get(i)<0 || tmp.get(i)<rth.get(i) ){
                        tmp.add(i+1 , -1);
                        tmp.add(i , 10000);
                }
                tmp.add(i , -rth.get(i));
```

```
            }
            tmp.fix();
            return tmp;
        }
};

int main(){
        int N;
        scanf("%d",&N);

        long long x , y;
        long long sigX=0LL , sigY=0LL , sigXX=0LL , sigYY=0LL , sigXY = 0LL;
        for(int i=0 ; i<N ; ++i){
            scanf("%lld%lld",&x,&y);
            sigX += x , sigY += y;
            sigXX += x*x , sigYY += y*y;
            sigXY += x*y;
        }

        BigNum A=sigX , B=sigY , AA=sigXX , BB=sigYY , AB=sigXY;
        BigNum tmp1 = AA*BB*N + AB*A*B*2LL;
        BigNum tmp2 = AB*AB*N + BB*A*A + AA*B*B;
        BigNum ans = tmp1 - tmp2;

        while( ans.sz()>1 && ans.val.back()==0 )
            ans.val.pop_back();

        if( ans.val.back()/4 || ans.sz()==1 )
            printf("%d",ans.val.back()/4);
        int last = ans.val.back()%4;
        for(int i=ans.sz()-2 ; i>=0 ; --i){
            printf("%04d",(last*10000 + ans.get(i)) / 4);
            last = (last*10000 + ans.get(i)) % 4;
        }
        putchar('.');
        printf("%04d\n",last*2500);
        return 0;
}
```

qH. Panda Chess

題目簡述：

N 位玩家舉行了 M 場一對一的比賽決定排名高低，規則是：贏家的排名永遠比輸家的高，輸入保證有唯一且符合規定的排名順序。
每位玩家的 ID 是由 D 個 0..9 數字組成，輸入皆用識別碼代表該玩家。
給定 M 場比賽的結果。
給一錯誤的排名，問至少要幾個操作才能改成正確的排名。
只能從排名插入或移除識別碼，各算一個操作。

解法：

每位玩家建節點，存他的正確排名跟贏誰。
建邊階段，每次讀入的 ID 字串後轉成 long long，用 map 建立 ID 跟節點的對應。
用輸贏建邊拓墣排序，得到所有人的正確排序。
讀入待更正錯誤排序，若讀入的 ID 不存在則跳過，否則則將此玩家的正確排名放入陣列尾端。最後求此陣列的最長嚴格遞增子序列(LIS)。只有 LIS 上的排名是可以不用改的，其他都要刪掉並插入正確的玩家。此為最佳答案，因為若要保留非 LIS 上的玩家，勢必要
答案即是 (N - LIS's length) * 2

因為 N最大100000，因此此題的 LIS 要用建表紀錄 長度 => 此長度最後一位的排名。每個長度最後一位的排名越高（數字小）越好，因為這樣會讓之後的玩家更容易接在自己後面。長度越長，最後一位的排名越低。利用此單調性可進行二分搜。然而此題線上測資不用二分搜跑起來跟二分搜一樣快。

My AC Code：

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/h.cpp

```cpp
#include <cstdio>
#include <string>
#include <cstring>
#include <map>
#include <queue>
using namespace std;

inline long long strToLL(const char *str){
	long long re = 0LL;
	for(int i=0 ; str[i]!='\0' ; ++i){
		re *= 10LL;
		re += str[i]-'0';
	}
	return re;
}
```

```
struct node{
	long long id;
	int score;
	vector<int> next;
	int inE = 0;
	node(){}
	node(const int &_id):id(_id){}
};
vector<node> g;

int main(){
	int N,M,D;
	scanf("%d%d%d",&N,&M,&D);

	map< long long , int > hs;
	char tmp[15];
	for(int i=0 ; i<M ; ++i){
		long long aid , bid;
		scanf("%s",tmp); aid = strToLL(tmp);
		scanf("%s",tmp); bid = strToLL(tmp);

		if( hs.find(aid)==hs.end() ){
			g.emplace_back( node(aid) );
			hs[aid] = g.size() - 1;
		}
		if( hs.find(bid)==hs.end() ){
			g.emplace_back( node(bid) );
			hs[bid] = g.size() - 1;
		}

		int ra = hs[aid] , rb = hs[bid];
		g[ra].next.emplace_back( rb );
		++g[rb].inE;
	}

	queue<int> myQ;
	for(int i=0 ; i<g.size() ; ++i)
		if( g[i].inE == 0 ){
			myQ.push( i );
			break;
		}
	int nowScore = 0;
	while( !myQ.empty() ){
		int nowAt = myQ.front();
		myQ.pop();
```

```
                g[nowAt].score = ++nowScore;
                for(auto &nid : g[nowAt].next)
                        if( --g[nid].inE == 0 )
                                myQ.push( nid );
        }

        vector<int> len;
        for(int i=0 ; i<N ; ++i){
                scanf("%s",tmp);
                long long nowID = strToLL(tmp);
                if( hs.find(nowID)==hs.end() ) continue;

                int nowRank = g[ hs[nowID] ].score;
                auto it = lower_bound(len.begin() , len.end() , nowRank);
                if( it==len.end()  ) len.emplace_back( nowRank );
                else *it = min(*it , nowRank);
        }
        int tlen = len.size();
        printf("%d\n",(N-tlen)*2);
        return 0;
}
```

ql. Pivot

題目簡述：

有一個陣列 A 有 n 個不同的整數，定義可當 pivot 的元素為：

大於所有左邊的元素

小於所有右邊的元素

給定陣列 A (2 ≤ n ≤ 10 0000)，問 pivot 數量

解法：

初始設所有點都不可能為答案

從左掃到右，邊紀錄左邊看過最大的數

若目前的元素大於左邊最大的數，則標記此點可能為答案

從右掃到左，邊紀錄右邊看過最小的數

若目前的元素大於右邊最小的數，則標記此點不可能為答案

最後統計可能為答案的數量即可

Hack??

此方法若用自定義的無限大當最大最小的初始值

要留意最左邊為 -INT_MAX-1 跟最右邊的元素為 INT_MAX 的情況

從左掃到右時，最左元素必可能是答案（因左邊沒有元素比自己大）

從右掃到左時，最右元素必可能是答案（因右邊沒有元素比自己小）

然而，如下測資

3

-2147483648 0 2147483647

在本機用兩種 code 跑出來的答案分別為

1

3 (正解)

在 judge 上都過了

My AC Code：

source code

https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/i.cpp

```
#include <cstdio>
#include <climits>
using namespace std;

int a[100004];
bool possible[100004];
```

```
int main(){
	int n;
	scanf("%d",&n);
	for(int i=0 ; i<n ; ++i)
		scanf("%d",&a[i]);
	int biggest = -INT_MAX-1;
	for(int i=0 ; i<n ; ++i)
		if( a[i] >= biggest )
			possible[i] = true , biggest = a[i];
	int smallest = INT_MAX;
	for(int i=n-1 ; i>=0 ; --i){
		if( a[i] > smallest )
			possible[i] = false;
		else
			smallest = a[i];
	}
	int ans = 0;
	for(int i=0 ; i<n ; ++i)
		if( possible[i] ) ++ans;
	printf("%d",ans);
	return 0;
}
```

qJ. Animal Classification

題目簡述：

有 N 種動物，兩位動物學家用各自的方法把他們放在二元樹的葉節點上。每個節點代表一種分類集合，含有所有後代葉節點上的動物。
用(,)還有數字給定兩棵二元樹，求共有幾個共同的分類集合。

解法：

用分類集合所有的數字和、積、個數代表該集合。
先把兩顆樹的所有分類集合算出來：開一個 Stack 存代表集合的那三個數字，讀到數字就 push 只有一個元素的集合進去，讀到) 就把 Stack 最後面兩個拿出來合併（和跟個數相加、積相乘模1e9+7）再丟回去。過程中出現過的即是全部的分類集合，共 2N-1 個存到陣列裡。
兩個陣列分別 sort 就可以在 O(N) 時間裡把兩個陣列的交集算出來，交集的元素個數即是答案。

My AC Code：

source code
https://github.com/sunset1995/NCTU_Prac0915_Singapore_Preliminary/blob/master/j.cpp

```
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;

struct data{
	long long sum , pro , sz;
	data& operator += (const data &rth){
		sum += rth.sum;
		pro = (pro*rth.pro) % 1000000007;
		sz += rth.sz;
	}
	bool operator < (const data &rth) const{
		if( sum<rth.sum ) return true;
		if( sum>rth.sum ) return false;
		if( pro<rth.pro ) return true;
		if( pro>rth.pro ) return false;
		if( sz<rth.sz ) return true;
		if( sz>rth.sz ) return false;
		return false;
	}
	bool operator == (const data &rth) const{
		return sum==rth.sum && pro==rth.pro && sz==rth.sz;
	}
```

```
};

inline bool isNum(const char &c){
	return c<='9' && c>='0';
}

void getLine(vector< data > &st){
	vector< data > myStack;
	int tmp = 0;
	int inCh = getchar();
	while( inCh!='\n' ){
		if( isNum(inCh) ){
			tmp *= 10;
			tmp += inCh-'0';
		}
		else if( tmp ){
			myStack.push_back( {tmp,tmp,1} );
			tmp = 0;
		}
		if( inCh==')' ){
			data added = myStack.back();
			myStack.pop_back();
			myStack.back() += added;
			st.push_back( myStack.back() );
		}
		inCh = getchar();
	}
}

int main(){
	int N;
	scanf("%d",&N);
	vector< data > setAlice , setBob;

	getchar();
	getLine( setAlice );
	getLine( setBob );
	sort(setAlice.begin() , setAlice.end());
	sort(setBob.begin() , setBob.end());

	auto itA = setAlice.begin();
	auto itB = setBob.begin();
	int ans = N;
	while( itA!=setAlice.end() && itB!=setBob.end() ){
		if( *itA == *itB ){
			++ans , ++itA , ++itB;
```

```
                }
                else if( *itA < *itB )
                        ++itA;
                else
                        ++itB;
        }
        printf("%d\n",ans);

        return 0;
}
```