**第一题：**

**设*X*[0:n-1]和*Y*[0:n-1]为两个数组，每个数组中含有*n*个已排好序的数。试设计一个O(log*n*)时间的分治算法，找出*X*和*Y*的2*n*个数的中位数，并证明算法的时间复杂性为O(log*n*)**

个数为奇数,则处于最中间位置的数

个数为偶数,则中间两个数据的平均数

算法分析：

两个数组，X和Y。设X数组的第一个、中间、最后一个元素分别为lx，mx，rx；设X数组的第一个、中间、最后一个元素分别为ly，my，ry。

先分别求两个数组的中位数mx,my进行比较，如果mx < my，比如X的中间数是3，Y的是5，那么推测两个数组的中间数约等于4，则该中间数对于X来说一定在3的右边，即在mx到rx范围的子数组里面，对于Y来说，中间数一定在5的左边，即在ly到my的子数组里面。比较如下：

if mx < my:

新mx属于[mx, rx]

新my属于[ly, my]

else:

新mx属于[lx, mx]

新my属于[my, ry]

之后再对新的两个子数组进行求中位数比较，一直到子数组元素为1时，进行比较，即可找到原数组X和Y的中位数。

新的mx和my将原来的数组X和Y分别分成了两个子数组，但是在四个子数组中只需使用上式指定的两个子数组即可，所以一次分解之后为原问题规模更小的实例T(n/2)，计算复杂度仍为计算两个数组的中位数（和原问题一样），为T(n/2)。求中间数和中间数比较需要的时间复杂度为O(1),故总的递归式为：

T(n) = T(n/2) + O(1)

由Master定理计算可得时间复杂度为T(n) = O(logn)

第二题：

**有一实数序列，若 且 ，则构成了一个逆序对，请使用分治方法求整个序列中逆序对个数，并分析算法的时间复杂性。**

例如：序列(4,3,2)逆序对有(4,3)，(4,2)，(3,2)共3个

算法分析如下：

参考归并排序，将数组分成两个子数组，再对两个数组进行递归分解，之后合并两个排好序的数组。合并过程中进行求逆序数操作。

设l，mid，r分别为数组的第一个，中间，以及最右边元素。则A[l, mid]为左数组，A[mid+1, r]为右数组，设i和j分别为做右数组的元素，逆序对数为s：

if A[i] < B[j]:

不产生逆序对

else:

s = s + mid-i+1

时间复杂度如下：

分解为两个子数组，规模为n/2，要对两个数组分别求解，故时间复杂度为2T(n/2)，分解时只需找到中位数，故D(n)为O(1)。合并时要遍历两个子数组，故C(n)为O(n)，所以递归方程为T(n) = 2T(n/2) + O(n)

利用Master定理求解，T(n) = O(nlogn)

第三题：

POJ3714：

算法分析：

类似于最近点对，只不过要求两个点要在不同的集合里面。

采用分治法，分为两个集合，求两个集合的最近点对，以及求两个点分别在两个集合的点对。所求的点的位置，一定在于mid-d,mid+d 之间。然后，就在这个区间开始找点，并不断更新d值，最后就可以得到d了。根据题意，返回长度的时候判断是否属于同一集合，若属于同一个集合，就返回一个很大的数

#include <stdio.h>

#include <math.h>

#include <algorithm>

using namespace std;

// 不光有点，还要有集合类别标志

struct Point

{

double x,y;

int flag;

}p[1000001];

int arr[1000001];

double Min(double a,double b)

{

return a<b?a:b;

}

// 求两点之间的距离

double dis(Point a,Point b)

{

return sqrt((a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y));

}

// 根据点横坐标or纵坐标排序

bool cmp\_y( int a,int b)

{

return p[a].y<p[b].y;

}

bool cmp\_x( Point a,Point b)

{

return a.x<b.x;

}

// 求最近点对

double close\_pair( int l,int r )

{

// 如果只剩下两个点，判断是否属于同一集合

if( r==l+1 )

{

if( p[l].flag!=p[r].flag )

return dis( p[l],p[r] );

else return 999999;

}

// 剩下三个点，判断集合类别

else if( r==l+2 )

{

if( p[l].flag==p[l+1].flag )

{

if( p[l].flag==p[l+2].flag ) return 999999;

else return Min( dis(p[l],p[l+2]),dis(p[l+1],p[l+2]) );

}

else

{

if( p[l].flag==p[l+2].flag ) return Min( dis(p[l],p[l+1]),dis(p[l+2],p[l+1]) );

else return Min( dis(p[l],p[l+1]),dis(p[l],p[l+2]) );

}

return Min( dis(p[l],p[r]),Min( dis(p[l],p[l+1]),dis(p[l+1],p[r]) ) );

}

int mid=(l+r)>>1;

double ans=Min(close\_pair(l,mid),close\_pair(mid+1,r));

int i,j,cnt=0;

for(i=l; i<=r; ++i)

if( p[i].x>=p[mid].x-ans && p[i].x<=p[mid].x+ans )

arr[cnt++]=i;

sort(arr,arr+cnt,cmp\_y);

for( i=0; i<cnt ; i++ )

for(j=i+1; j<cnt; j++)

{

if( p[arr[j]].flag != p[arr[i]].flag )

{

if(p[arr[j]].y-p[arr[i]].y>=ans) break;

ans=Min(ans,dis(p[arr[i]],p[arr[j]]));

}

}

return ans;

}

int main()

{

int i,n,t;

scanf("%d",&t);

while( t-- )

{

scanf("%d",&n);

for(i=0;i<n;++i)

{

scanf("%lf%lf",&p[i].x,&p[i].y);

p[i].flag=1;

}

for(;i<n+n;++i)

{

scanf("%lf%lf",&p[i].x,&p[i].y);

p[i].flag=2;

}

sort(p,p+n+n,cmp\_x);

printf("%.3lf\n",close\_pair(0,n+n-1));

}

return 0;

}

第四题：

给定座建筑物，每个建筑物表示为一个矩形，用三元组表示，其中表示建筑左下顶点，表示建筑的右下顶点，表示建筑的高，请设计一个的算法求出这座建筑物的天际轮廓。例如，左下图所示中8座建筑的表示分别为(1,5,11), (2,7,6), (3,9,13), (12,16,7), (14,25,3), (19,22,18), (23,29,13)和(24,28,4)，其天际轮廓如右下图所示可用9个高度的变化(1, 11), (3, 13), (9, 0), (12, 7), (16, 3), (19, 18), (22, 3), (23, 13)和(29,0)表示。另举一个例子，假定只有一个建筑物(1, 5, 11)，其天际轮廓输出为2个高度的变化(1, 11), (5, 0)。



算法分析：

使用分治方法，如果只有一个建筑，那么输出的点一定是建筑的左上顶点和右下顶点，每次将buildings一分为二做递归，当buildings的数量为1时，返回2个点。之后将分开的结果合并，要记录当前left和right的位置，每次选择横坐标较小的点插入结果中，保证结果有序，结果点的横坐标等于原横坐标，至于结果点的纵坐标，应该是取right和left中的较大值，而这个高度的值也应有2个变量实时记录，代表的含义是left和right当前的高度。合并时候点的高度一定不能和上一个点相同，如果多个连续的点纵坐标相同，则结果中只能有第一个点

时间复杂度类似于归并排序，为O(nlogn)