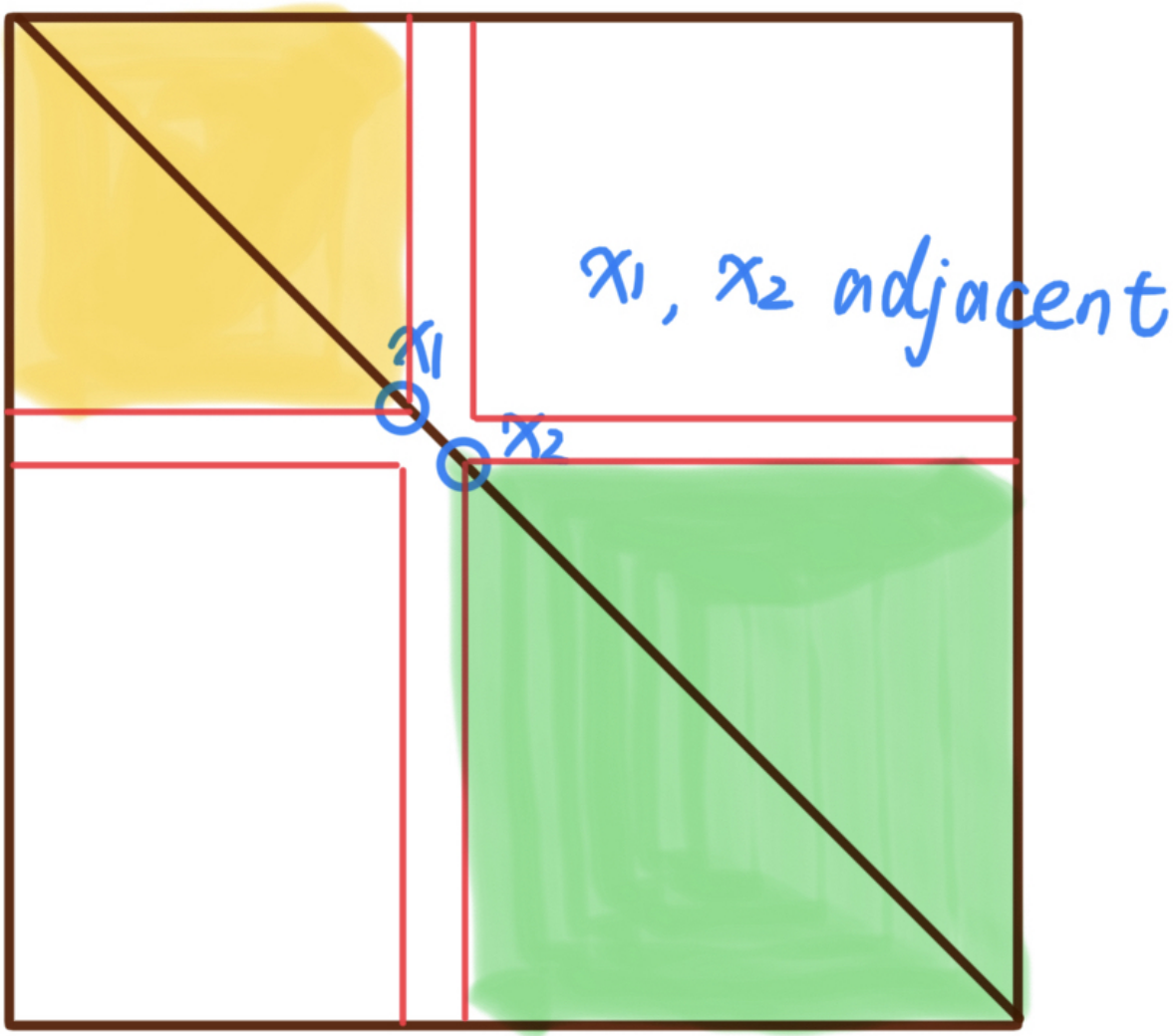
$Homework2$
$name$: 李卓壕
id : 519021911248

1. 设 M 是一个 $m \times n$ 的矩阵，其中每行的元素从左到右单增有序，每列的元素从上到下单增有序。给出一个分治算法计算出给定元素 x 在 M 中的位置或者表明 x 不在 M 中。给出问题的求解思路、伪代码并分析算法的时间复杂度。

1.$Solutions$:

解决思路:
因为这个矩阵按行看从左向右递增有序，按列看从上到下递增有序，所以矩阵的主对角线从左上到右下看也是递增有序的.对于一个递增序列，采用二分查找去查找x，这一步的时间复杂度为$O(logn)$.如果找到了x，则返回x的位置（平凡情况）.否则，一定可以找出对角线上两个相邻的x_1, x_2，满足$x_1 < x < x_2$，(如果x大于这个对角线数列的最大值或者小于对角线数列的最小值，则只能找到一个x_1或是x_2,但是讨论是类似的，即可以把缺省的x_1或x_2视作0或者∞,这样的话，下一次递归就是$n-1$规模的递归)这样的话，进行下一次递归时，可以排除由x_1, x_2作为划分而形成的4个矩形中左上和右下的两个(如下图所示).
事实上，左上的矩阵所有元素永远都比x_1小,右下的矩阵所有元素永远都比x_2大，因此可以排除.
递归调用本算法，对左下和右上两个矩阵做相同操作.
这样就实现了对本问题的分治和实现.
算法的核心部分示意图如下所示:



pseudo code :

```
search(M,x) //函数的定义，行参是一个m*n的矩阵M和待查找的元素x

####创建一个数组保存主对角线上的元素#######

length=min(m,n) //对角线数组的数组长度是m和n中较小的一个
init diag[length] //对角线数组称作diag
for i=0 to length
    diag[i]= M(i,i) //依次写入对角线上的元素

##############

######二分查找########
mid_search(diag[length], x) //二分查找函数的定义

int left= 0, right= length-1, mid= (left+right)/2;
while (left< right)
    if diag[mid]==x return mid
    if dig[mid]>x right=mid
    else left=mid
else return -1  //具体实现二分查找

############

#####递归######

if mid_search(diag[length],x)!= -1
    return (mid,mid) //平凡解，如果在对角线上查找到了x,则返回对角线上的位置，组合成数对就是所求解
//划分
M_leftdown=M(left,0)->M(m,left) //左下的子矩阵
M_rightup=M(0,right)->M(right,n) //右上的子矩阵

if mid_search(diag[length],x)==-1
    search(M_leftdown, x);
    search(M_rightup, x);
else return -1 //如果最后执行了这条语句，表示整个矩阵都没有找到

#############
```

时间复杂度分析：
不妨设$m < n$，创建$diag[length]$：$O(m)$
二分查找：$O(logm)$
划分：$O(mn)$
递归，平均意义上来说：$2T(\frac{mn}{4})$
递归方程：$T(mn) = 2T(\frac{mn}{4}) + O(mn)$
$T(mn) = mn$

2. 给定二维平面上的点集 S 包含 n 个不同的点，任取 3 个点可构成一个三角形 (不考虑三点共线的情况)，其周长可通过汇总三条边的长度来得到。参考最近点对算法，设计一个时间复杂度至多为 $O(n^2)$ 的分治算法，来寻找周长最短的三角形。给出问题的求解思路、伪代码并分析算法的时间复杂度。

2.$Solutions$：
解题思路：
$Divide$：

将S中的点分别按照x和y坐标排序，用x坐标中位数m划分为S_l和S_r两部分，递归求解.
设S_l中周长最短的点对（p_1, p_2, p_3），S_r中周长最短的点对（q_1, q_2, q_3), 令$c = min\{c（p_1, p_2, p_3), c（q_1, q_2, q_3)\}$.
$Merge$：

在m处划分一个矩形区域，取出所有离mid横坐标不超过当前最小值c的一半, 即$m - \frac{c}{2} \leq x \leq m + \frac{c}{2}$的区域内.
假设S_l中的最近点对左侧端点为p_m，从其做$L : x = m$的投影，则右侧两个点必定在以投影点为基准的$\frac{c}{2} \times c$的矩形区域内.（右邻域）
根据抽屉原理，左临界区任意一点p的右邻域至多包含18个点.

事实上，只需要将这个以投影点为基准的$\frac{c}{2} \times c$的矩形区域划分为18块子区域，每个区大小为$\frac{c}{6} \times \frac{c}{6}$，若多于18个点，必有两个位于一个子块中，根据勾股定理，这两个点和相邻子块中的点距离至多为$\frac{(3+\sqrt{5})}{6}c < c$, 这与$d$已经是左右两个子集的分治解冲突.
事实上，只需要证明对于每一个在左邻域的点，只需有限多步就能实现和右邻域的比较即可.
则按照y排序的点，对于左邻域的每一个点，计算右邻域中可能的两个点，至多18个，至多进行C_{18}^2次计算, 如果有比c更小的周长c'，则更新，c'对应的三个点就是所求点.
$pseudocode$：

```
A[0...n-1]= Sort(S,x), B[0...n-1]= Sort(S,y) //初始化，点集S根据横纵坐标排序

ShortestPerimeter(A,B,n) //求最小周长的函数，A、B是两个有序数组，n是点集中点的个数
for i in n-1:
    A[i].z= Pos(B,A[i]) //A[i]在B中的位置
if n<=3 then return P,Q,R, c=Perimeter(P,Q,R) //平凡解，直接返回结果
k<-n/2, s<-0, t<-0 //划分
for i in range(o,k-1): A_L[0:k-1]<- A[0:k-1], B[A[i].z].z<- i
for i in range(k,n-1): A_R[0:n-k-1]<- A[k:n-1], B[A[i].z].z<- i
for j<-0 to n-1 do
    if B[j].z < k then B_L[s]<-B[j], A_L[B[j].z].z<-s, s++
    if B[j].z >= k then B_R[s]<-B[j], A_R[B[j].z-k].z<-t, t++

#####递归######
P1, Q1, R1, c1 <- ShortestPerimeter(A_L, B_L, k)
P2, Q2, R2, c2 <- ShortestPerimeter(A_R, B_R, n-k)
##############

c<- min(c1,c2), (P,Q,R)<- min((P1, P2), (Q1, Q2), (R1, R2))
j<- 0, l<-0, m<- A_L[k-1].x
for i in range(0: k) do:
    if B_L[i].x< m-c/2 then i++; continue  //不在左临界区
    while B_R[j].y < B_L[i].y - c or B_R[j].x > m.x + c do:
        j++;
    while B_R[j].y <= B_L[i].y + c and B_R[j].x <= m.x + c do:
        l<- 0
        while B_R[j+l]<=B_L[i].y+ c and  B_R[j+l].x <= m.x + c
            ++l; // 最多18个
            if Perimeter(B_L[i], B_R[j], B_R[j+l])< c then
                c<-Perimeter(B_L[i], B_R[j], B_R[j+l]); (P, Q, R)<- (B_L[i], B_R[j], B_R[j+l])
        j++;
return (P,Q,R), c;
```

时间复杂度分析：
预处理所需时间：$O(nlogn)$，但是这一部分不参与递归
$Divide$需要的时间：$O(n)$

$Conquer$所需的时间：$2T\ (\frac{n}{2})$

$Merge$所需的时间：之多考察$\frac{n}{2}\times C_{18}^2=77n$次点，复杂度数量级仍为$O(n)$
递归方程：

$$T(n)=\begin{cases}(1),n=2\\2T(\frac{n}{2})+O(n),n\geq 3\end{cases}$$

由主定理可得方程的解为；$T(n)=O(nlogn)$
.
.
$notes$（2021.10.3）:
事实上，本题和课堂上讲述的求点集中距离最短的两个点的距离算法是大同小异的，唯一的区别在于左右邻域变成了$\frac{c}{2}$，而不是d，在B数组中搜索的点的数目从2个变成了3个，此外利用抽屉原理证明经过有限次搜索点的步骤也十分类似，所得到的解不一定是最优的，但一定是有限的，在分析时间复杂度的时候，最后列出的递归方程其实是一样的，这也是为什么时间复杂度和计算最短距离算法一样的原因。
.
$notes$（2021.10.14）：
将会更新LaTeX $pseudo\ code$版本的$hw2.pdf$