1、设 **X[0:n-1]**和 **Y[0:n-1]**为两个数组，每个数组中的 **n** 个均已经排好序，试设计一个 **O(log(n))**的算法，找出 **X** 和 **Y** 中 **2n** 个数的中位数，并进行复杂性分析。

答：算法思想：首先找出 X 的中位数 a=X[$\left\lceil\frac{n}{2}\right\rceil$]与 Y 的中位数 b=Y[$\left\lfloor\frac{n}{2}\right\rfloor$]，比较 a 与 b 的大小，有以下三种情况：

(1)若 a=b，则中位数为 a。

(2)若 a<b，则一定有 X[0]＜X[1]<…< a<…<b<Y[$\left\lfloor\frac{n}{2}\right\rfloor$+1]<…< Y[n-1]。即中位数一定不在 X 中比 a 小的部分和 Y 中比 b 大的部分。这样可在 X 的子数组 X[$\left\lceil\frac{n}{2}\right\rceil$:n-1]和 Y 的子数组 Y[0: $\left\lfloor\frac{n}{2}\right\rfloor$]中继续寻找中位数。

(3)若 a＞b，则一定有 Y[0]＜…<b<…<a<X[$\left\lceil\frac{n}{2}\right\rceil$+1]<…< X[n-1]。即中位数一定不在 Y 中比 b 小的部分和 X 中比 a 大的部分。这样可在 X 的子数组 X[0: $\left\lceil\frac{n}{2}\right\rceil$]和 Y 的子数组 Y[$\left\lfloor\frac{n}{2}\right\rfloor$:n-1]中继续寻找中位数。

递归调用查找中位数的函数，直到序列长度为 1，得到的即为所求的中位数。由于每次将问题化为原问题规模的 1/2，因此算法的时间复杂度为 O(logn)。

伪代码描述：

Input:　　有序数组 X[0:n-1]和 Y[0:n-1]。

Output：　X 和 Y 中 2n 个数的中位数。

FindMediam(X,Y,startX,endX,startY,endY)

1　if endX-startX=0

2　　then return $\frac{X[startX]+Y[startY]}{2}$

3　a←X[$\left\lfloor\frac{startX+endX}{2}\right\rfloor$]

4 b←Y[$\left\lceil\frac{startY+endY}{2}\right\rceil$]

5 if a=b

6 then return a

7 else if a>b

8 return FindMediam(X,Y, $\left\lfloor\frac{startX+endX}{2}\right\rfloor$,endX,startY, $\left\lceil\frac{startY+endY}{2}\right\rceil$)

9 else

return FindMediam(X,Y,startX, $\left\lfloor\frac{startX+endX}{2}\right\rfloor$, $\left\lceil\frac{startY+endY}{2}\right\rceil$,endY)

调用函数 FindMediam(X,Y,0,n-1,0,n-1)即可求出结果。

时间复杂度分析：算法 1-7 步的时间复杂度为 O(1)，第 8 步递归调用该函数，问题规模变为原来的一半，递推公式为 T(n)=T(n/2)+ O(1)。根据 Master 定理可得此问题的时间复杂度为 O(logn)。

2、设 **A[1 : n]**是由不同实数组成的数组，如果 **i<j** 且 **A[i]>A[j]**，则称实数对**(A[i],A[j])**是该数组的一个反序。如，若 **A=[3,5,2,4]**，则该数组存在 **3** 个反序**(3,2)**、**(5,2)**和**(5,4)**。反序的个数可以用来衡量一个数组的无序程度。设计一个分治算法（要求时间复杂度严格低于 $\mathbf{n^2}$），计算给定数组的反序个数。答案要求包含以下内容：

(1)用简明的自然语言表述算法的基本思想；

(2)用伪代码描述算法；

(3)分析算法的时间复杂度。

答：

(1) 算法思想：根据改变 Merge-Sort 的中 Merge 的过程来统计反序个数。在合并过程中，设两个待合并串分别为 L 和 R，比较 L[i]和 R[j]，

若 L[i]>R[j]，说明 R[j]比 L 中剩余的大于 L[i]的元素都小，并与这些元素构成反序。在合并排序算法的运行过程中，统计反序个数，排序算法结束时，可得出 A 中的反序个数。

(2)伪代码描述：

```
ISMerge(A,p,q,r,count)
1   n1←q-p+1
2   n2←r-q
3   create arrays L[1:n1+1] and R[1: n2+1]
4   for i←1 to n1
5       do L[i]←A[p+i-1]
6   for j←1 to n2
7       do R[j]←A[q+j]
8   L[n1+1]←Max
9   R[n2+1]←Max
10  i←1
11  j←1
12  for k←p to r
13      do if L[i]≤R[j]
14          then A[k]←L[i]
15               i←i+1
16          else A[k]←R[i]
17               j←j+1
```

18 count←count+n_2-i+1

ISMerge-Sort(A,p,r,count)

1 if(p<r)

2 then q←⌊(p + r)/2⌋

3 ISMerge-Sort (A,p,q, count)

4 ISMerge-Sort(A,q+1,r,count)

5 ISMerge(A,p,q,r,count)

Main(A,1,n)

1 count←0

2 ISMerge-Sort(A,1,n,count)

3 return count

(3)由于此算法是在 Merge-Sort 算法中，Merge 阶段增加了计数过程，该步骤复杂度为 O(1)，因此，总得算法时间复杂度与合并排序算法相同，为 O(nlogn)。

3、给定一个由 **n** 个实数构成的集合 **S** 和另一个实数 **x**，判断 **S** 中是否有两个元素的和为 **x**。试设计一个分治算法求解上述问题，并分析算法的时间复杂度。

答：算法思想：首先对集合 S 内的 n 个实数进行合并排序，接着对排序后 S 内的每一个元素 S[i]，求得 z=x-S[i], 利用二分查找法在 S 中寻找 z, 若找到则输出结果，否则不存在。

伪代码描述：

Input：一个由 n 个实数构成的集合 S,记为 S[1:n]

Output: S 中是否有两个元素的和为 x 的结果

```
Search(S,x)
1   Merge-Sort(S,1,n)
2   for i←1 to n
3       do  z←x-S[i]
4           if Binary-Search(S,1,n,z)>0
5               then return Exist
6   return notExist
Binary-Search(S,start,end,key)
1   if start>end||key<S[start]||key>S[end]
2       then return 0
3   medium←⌊(start+end)/2⌋
4   if key=S[medium]
5       then return medium
6       else if key<S[medium]
7           return Binary-Search(S,start, medium-1,key)
8           else return Binary-Search(S, medium+1, end,key)
```

时间复杂度分析：第 1 步为合并排序，时间复杂度为 O(nlogn)，2-5 步为一个复杂度为 n 的循环，循环内部执行二分查找算法，复杂度为 O(logn)，因此 2-5 步整体的时间复杂度为 O(nlogn)，第 6 步复杂度为 O(1)。因此，该算法的整体时间复杂度为 O(nlogn)。

4、设单调递增有序数组 **A** 中的元素被循环右移了 **k** 个位置，如**<35;**

42; 5; 15; 27; 29>被循环右移两个位置（**k = 2**）得到**<27; 29; 35; 42; 5; 15>**。

(1)假设 **k** 已知，给出一个时间复杂度为 **O(1)**的算法找出 **A** 中的最大元素。

(2)假设 **k** 未知，设计一个时间复杂度为 **O(logn)**的算法找出 **A** 中的最大元素。

答：设组 A 中共有 n 个元素，记为 A[1:n]。观察该数组知除了最大元素外，右侧相邻元素均大于左侧元素。

(1)算法思想：若要找出最大元素，只需要令 m=k%n，若 m=0,则最大元素为 A[n];若 m≠0，则 A[m]即为所求，时间复杂度为 O(1)。

Input:A[1:n];

Output:max(A[i]),1≤i≤n;

FindMax(A,k, n)

1　m=k%n;

2　if m=0

3　　then return A[n];

4　　else return A[m];

(2)算法思想：若 k 未知，记 a=A[1],b=A[n],c=A[$\left\lceil\frac{n+1}{2}\right\rceil$]。比较 a、b、c 大小，有如下三种情况：

1 若 a<c<b，则说明该序列已经单调递增排列，最大元素为 A[n]。

2 若 c<a 且 c<b，说明最大元素在序列的前半部分，对 A[1: $\left\lceil\frac{n+1}{2}\right\rceil$-1]递归求解。

3 若 c>a 且 c>b，说明最大元素在序列的后半部分，对 A[$\left\lceil\frac{n+1}{2}\right\rceil$:n]递归求解。

伪代码描述：

FindMax(A,start,end)

1 a←A[start],b←A[end],c←A[$\left\lceil\frac{start+end}{2}\right\rceil$]

2 if a<c<b

3 then return A[end]

4 else if c<a&&c<b

5 FindMax(A,start, $\left\lceil\frac{start+end}{2}\right\rceil$-1)

6 else if c>a&&c>b

7 FindMax(A, $\left\lceil\frac{start+end}{2}\right\rceil$,end)

调用函数 FindMax(A,1,n)，即可得到结果。

时间复杂度分析：1-4 步和 6 步时间复杂度为 O(1)，第 5、7 步递归调用该函数，问题规模变为原来的一半，递推公式为 T(n)=T(n/2)+O(1)。根据 Master 定理可得此问题的时间复杂度为 O(logn)。

5、设 **M** 是一个 **m×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。分析算法的时间复杂性。

算法思想：设 M 内的元素为 p_{ij}, $1 \le i \le n, 1 \le j \le m$,

1 对于矩阵$M_{(a:b)(i:j)}$首先判断p_{ai}和p_{bj}与 x 是否相等,

若相等，输出元素位置, 结束这个矩阵的计算;

若$p_{ai} > x$或$p_{bj} < x$, 则 M 中不存在 x, 算法结束;

若$p_{ai} < x < p_{bj}$，则 x 在 M 中可能存在，继续步骤 2;

2 将矩阵 $M_{(a:b)(i:j)}$分为四个子矩阵，$M_{(a:\frac{a+b}{2})(i:\frac{i+j}{2})}$，$M_{\left(a:\frac{a+b}{2}\right)\left(\frac{i+j}{2}+1:j\right)}$，$M_{(\frac{a+b}{2}+1:b)(i:\frac{i+j}{2})}$，$M_{(\frac{a+b}{2}+1:b)(\frac{i+j}{2}+1:j)}$，对每个矩阵执行步骤(1)

当所有的矩阵都计算结束，则算法结束。

伪代码：

Input: m×n 的矩阵 M，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，元素 x

Output: 输出 M 中所有与 x 相等的元素位置

```
printX(M, a, b, i, j, x)
1   if(a>b||i>j)
2       return
3   if(M[a][i]==x)
4       print (a,i)
5   else if(M[b][j]==x)
6       print (b, j)
7   else if(M[a][i]<x< M[b][j])
8       printX(M, a, (a+b)/2, i, (i+j)/2, x)
9       printX(M, a, (a+b)/2, (i+j)/2 + 1, j, x)
10      printX(M, (a+b)/2 + 1, b, (i+j)/2 + 1, j, x)
11      printX(M, (a+b)/2 + 1, b, i, (i+j)/2, x)
```

时间复杂度分析：算法最坏情况下，一次划分后仍然可以去掉四个子矩阵其中之一，即$T(mn) = 3T\left(\frac{mn}{4}\right) + O(1)$。则算法的时间复杂度为

$O\left((mn)^{\log_4 3}\right) = O((mn)^{0.6})$。