1、有个大小不同的杯子和与之匹配的*n*个杯盖, 你可以尝试一个杯子和一个杯盖是否匹配, 尝试结果有三种：（1）杯子太大；（2）匹配成功；（3）杯盖太大. 请设计一个分治算法完成所有杯子和杯盖的匹配, 算法的时间复杂性用匹配尝试的次数来衡量。

（1）叙述算法设计思路. (请叙述以下方面)

边界条件：当只有一个杯子和一个杯盖时，匹配杯子和杯盖

Divide：从杯子集合中随机选择一个杯子x, 将x与所有杯盖进行匹配，把结果为杯子太大的杯盖放入G1, 把结果为杯盖太大的杯盖放入G2, 找到和x匹配成功的杯盖记为y。将y与x以外所有杯子进行匹配，把结果为杯盖太大的杯子放入B1, 把匹配结果为杯子太大的杯盖放入B2。

Conquer：递归地对B1和G1、B2和G2进行匹配.

Merge：返回B1和G1、B2和G2的匹配结果以及(x, y).

（2）写出算法伪代码.

Match(B, G)

Input: 杯子集合B, 杯盖集合G, |B|=|G|=n, B中每个杯子都只和G中一个杯盖匹配成功

Output: {(x, y)| x∈B, y∈G, x与y匹配成功}

1. if |B|=1

2. return BG;

3. 初始化B1, B2, G1, G2, M=; y=NIL;

4. 随机选择杯子集合B中的一个元素x;

5. for G

6. if g和x的匹配结果为“杯子太大”

7. G1=G1{g};

8. else if g和x的匹配结果为“杯盖太大”

9. G2=G2{g};

10. else

11. y=g;

12. M=M{(x, y)};

13. for B/{x}

14. if y和b的匹配结果为“杯盖太大”

15. B1=B1{b};

16. else if g和x的匹配结果为“杯子太大”

17. B2=B2{b};

18. if B1

19. M=MMatch(B1, G1);

20. if B2

21. M=MMatch(B2, G2);

22. return M;

（3）分析算法的时间复杂性.

记算法时间复杂性T(n). 将x与所有杯盖匹配代价为*O*(*n*); 将y与x以外所有杯盖匹配的代价为*O*(*n*); 综上, 划分代价为*O*(*n*). Conquer 代价为T(|B1|)+T(*n*-1-|B1|), 其中|B1|为0、1、…、*n*-1的概率都是1/*n*. Combine（Merge）代价不计（算法的时间复杂性用匹配尝试的次数来衡量）. 总体代价与快速排序相同, 最坏情况为*O*(*n*2); 期望为O(*n*log*n*).

2、和为两个数组，每个数组中的个均已经排好序，试设计一个的分治算法，找出*X*和*Y*中个数的中位数。

（1）叙述算法设计思路.

**需找到*X*和*Y*合并后排序第*n*和第*n*+1的元素.** 将*X*划分为和, 将*Y*划分为和. 这样和中共有*n*个元素, 和中共有*n*个元素. 与此同时, 和中元素数量相等, 和中元素数量相等. 比较和:

Case 1: 如果, 则*X*和*Y*合并后排序第*n*个元素必为, 并且第个元素为;

Case 2: 如果, 则在*X*和*Y*合并后排序至少为*n*, 而且 在*X*和*Y*合并后排序小于*n*. 进一步比较和:

Case 2.1: 如果, 则在*X*和*Y*合并后排序为*n*, 而的排序为;

Case 2.2: 如果, 则在*X*和*Y*合并后排序至少为, 此时中元素排序位置均小于并且中元素排序均大于, 由此可知*X*和*Y*合并后排序第*n*和第*n*+1的元素在和中仍为中位数. 原始问题转化为在和中找到中位数.

Case 3: 如果, 则在*X*和*Y*合并后排序至少为*n*, 而且 在*X*和*Y*合并后排序小于*n*. 进一步比较和:

Case 3.1: 如果, 则在*X*和*Y*合并后排序为*n*, 而的排序为;

Case 3.2: 如果, 则在*X*和*Y*合并后排序至少为, 此时中元素排序均小于并且中元素排序均大于, 由此可知*X*和*Y*合并后排序第*n*和第*n*+1的元素在和中仍为中位数. 原始问题转化为在和中找到中位数.

（2）写出算法伪代码.

FindMedium(, )

1. ;

2. if

3. 按大小合并和到数组;

4. return , ;

5. if

6. return , ;

7. else if

8. if

9. return , ;

10. else

11. return FindMedium();

12. else

13. if

14. return , ;

15. else

16. return FindMedium();

运行FindeMedium(, )求解原始问题.

（3）分析算法的时间复杂性.

, .

3、对于两个二维数据元素和，如果(1) , 或者 (2) , ，则支配，记为。二维数据集合的定义如下：。本部分内容计算二维数据集合的。设计基于分治的二维数据Skyline求解算法。

（1）叙述算法设计思路. (请叙述以下方面)

边界条件: 数据集中只有一个点，则；

Divide:

1. 按二维键值(*x*, *y*)使用线性时间算法计算的中位数，两个点比较大小时首先比较值，若值相等时比较值（注: |*D*|为偶数时求第|*D*|/2个数作为中位数）；用将划分为两个大小最多相差1的子集合和；

Conquer: 递归地在和中计算和；

Merge:

1. 扫描，若中存在*x*值等于的点，计算，否则；若中存在*x*值大于的点，计算，否则；

2. 初始化，对于中任一点:

Case 1: ，如果且，;

Case 2: ，如果且，;

（注：中的点*x*值都不大于，因此只需要讨论和两种情况。）

（2）写出算法伪代码.

1. if

2. 比较D中每个点对, 并返回Skyline集合;

3. D中第的点（点与点比较时首先比较x值, x值相同时比较y值）;

4. , ;

5. for

6. if

7. ;

8. if

9. ;

10. if

11. for to

12. ;

13. if

14. for to

15. ;

16. ;

17. ;

18. ;

19. for

20. if and

21. ;

22. else if and

23. ;

24. for

25. if and and

26. ;

27. if and and

28. ;

29. return ;

（3）分析算法的时间复杂性.

边界条件处理代价为*O*(1);

Divide代价为*O*(*n*);

Merge代价中计算和的代价O(||)，初始化的代价*O*(||)，扫描中每个点并更新的代价为*O*(||)，因此Merge代价为*O*(*n*);

算法代价的递归方程为T(*n*)=2T(*n*/2)+*O*(*n*)，T(*n*)=*O*(*n*log*n*)。