0、改变一下 RadixSort 的操作，得到排序算法 HabitSort，按照我们可能的习惯，首先用 Counting Sort 排序最高位，然后按照最高位的取值将数据划分成不同的组，再递归地用 HabitSort 排序各个组内元素，继而得到最终的排序。这样修改后的 HabitSort 算法的复杂性还是 $O(d(n+k))$吗？

解：排序最高位的代价是 $O(n+k)$，最高位最多有 k 种不同的取值。因此，需要将所有数据分组到 k 组中，每组的最高位相同，分组代价为 $O(n)$，将分组代价计入排序最高位后，排序最高位代价仍为 $O(n+k)$。

随后，对每组分别根据次高位进行排序，第 i 组的排序代为 $O(n_i+k)$，其中 n_i 为第 i 组中元素数量，排序 k 组的次高位，代价为$\sum_{i=1}^{k} O(n_i+k) = O\left(\sum_{i=1}^{k}(n_i+k)\right) = O(n+k^2)$。根据前两位对数据分组，分组代价 $O(n)$，将分组代价计入排序最高位后，排序次高位代价仍为$O(n+k^2)$。

以此类推，排序第 3 高位时最多有 k^2 个分组，排序代价为$O(n+k^3)$；

排序第 j 高位时最多有 k^{j-1} 各分组，排序代价为$O(n+k^j)$；

排序最低位时最多有 k^{d-1} 各分组，排序代价为$O(n+k^d)$。

当然，想要让排序次高位到最低位时都有上述数量的分组，需要输入中有足够多的不同数字，这也对 n 的大小有了要求。

当$n=\theta(k^{d-0.5})$时，可以使得各个位的排序过程中产生上述数量的分组，而且排序最低位时每个分组中有$k^{0.5}$个元素（而非常数）。此时，排序所有位的代价为:$T(n) = O\left(\sum_{i=1}^{d}(n+k^d)\right) = O\left(dn+\frac{k^{d+1}-1}{k-1}\right) = O(dn+k^{d+1})$。

当$k=\Omega(\sqrt[d]{d})$时，$T(n) \neq O(d(n+k))$。

1、有n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。

解：（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 B×G;
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 ∈G
6. if g 和 x 的匹配结果为“杯子太大”

```
        G1=G1∪{g};
    else if g 和 x 的匹配结果为“杯盖太大”
        G2=G2∪{g};
    else
        y=g;
        M=M∪{(x, y)};
for ∀b ∈B/{x}
    if y 和 b 的匹配结果为“杯盖太大”
        B1=B1∪{b};
    else if g 和 x 的匹配结果为“杯子太大”
        B2=B2∪{b};
if B1≠ ∅
    M=M∪Match(B1, G1);
if B2≠ ∅
    M=M∪Match(B2, G2);
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、对于两个二维数据元素$p=(x_p,y_p)$和$q=(x_q,y_q)$，如果(1) $x_p \geq x_q$，$y_p > y_q$ 或者 (2) $x_p > x_q$，$y_p \geq y_q$，则p支配q，记为$p \rightarrow q$。二维数据集合D的$Skyline$定义如下：$SKL(D)=\{p|p \in D, \nexists q \in D, q \rightarrow p\}$。本部分内容计算二维数据集合的$Skyline$。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D中只有一个点p，则$SKL(D)=\{p\}$;

Divide:

1. 按二维键值(x, y)使用线性时间算法计算D的中位数$p(x_m,y_m)$，两个点比较大小时首先比较x值，若x值相等时比较y值（注: $|D|$为偶数时求第$|D|/2$ 个数作为中位数）；用p将D划分为两个大小最多相差 1 的子集合D_L和D_R；

Conquer: 递归地在D_L和D_R中计算 $SKL(D_L)$和$SKL(D_R)$；

Merge:

1. 扫描$SKL(D_R)$，若$SKL(D_R)$中存在 x 值等于x_m的点，计算$y_1=max\{y|(x,y) \in SKL(D_R), x=x_m\}$，否则$y_1=-\infty$；若$SKL(D_R)$中存在 x 值大于x_m的点，计算$y_2=max\{y|(x,y) \in SKL(D_R), x>x_m\}$，否则$y_2=-\infty$；

2. 初始化$SKL(D) \leftarrow SKL(D_R)$，对于$SKL(D_L)$中任一点$p(x,y)$:

Case 1: $p.x<x_m$，如果$p.y>y_1$且$p.y>y_2$，$SKL(D)=SKL(D) \cup \{p\}$;

Case 2: $p.x=x_m$，如果$p.y=y_1$且$p.y>y_2$，$SKL(D)=SKL(D) \cup \{p\}$;

（注：$SKL(D_L)$中的点 x 值都不大于x_m，因此只需要讨论$p.x<x_m$和$p.x=x_m$两种情况。）

（2）写出算法伪代码.

$SKL(D)$

if $|D| < 3$
    比较 D 中每个点对，并返回 Skyline 集合;
$p \leftarrow$D 中第$\lfloor |D|/2 \rfloor$的点（点与点比较时首先比较 x 值, x 值相同时比较 y 值）;
D_L，D_R，$S \leftarrow \emptyset$;
for $\forall q \in D$
    if $q < p$
        $D_L = D_L \cup \{q\}$;
    if $q > p$
        $D_R = D_R \cup \{q\}$;
if $|D_L| < \lfloor |D|/2 \rfloor$
    for $j = 1$ to $\lfloor |D|/2 \rfloor - |D_L|$
        $D_L = D_L \cup \{(p.x, p.y)\}$;
if $|D_R| < \lceil |D|/2 \rceil$
    for $j = 1$ to $\lceil |D|/2 \rceil - |D_R|$
        $D_R = D_R \cup \{(p.x, p.y)\}$;
$S \leftarrow SKL(D_R)$;
$L \leftarrow SKL(D_L)$;
y_1，$y_2 = -\infty$;
for $\forall q \in S$
    if $q.x = p.x$ and $q.y > y_1$
        $y_1 = q.y$;
    else if $q.x > p.x$ and $q.y > y_2$
        $y_2 = q.y$;
for $\forall q \in L$
    if $q.x = p.x$ and $q.y = y_1$ and $q.y > y_2$
        $S = S \cup \{q\}$;
    if $q.x < p.x$ and $q.y > y_1$ and $q.y > y_2$
        $S = S \cup \{q\}$;
return S;

（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y_1和y_2的代价 $O(|SKL(D_R)|)$，初始化$SKL(D) \leftarrow SKL(D_R)$的代价 $O(|SKL(D_R)|)$，扫描$SKL(D_L)$中每个点并更新$SKL(D)$的代价为 $O(|SKL(D_L)|)$，因此 Merge 代价为 $O(n)$;

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

4、$X[0:n-1]$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$O(\log n)$的分治算法，找出 X 和 Y 中$2n$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$X_L = X[0:\lfloor n/2 \rfloor - 1]$

和$X_R = X[\lfloor n/2 \rfloor : n-1]$，将 Y 划分为$Y_L = Y[0:\lceil n/2 \rceil - 1]$和$Y_R = Y[\lceil n/2 \rceil : n-1]$. 这样$X_L$和$Y_L$中共有 n 个元素，X_R和Y_R中共有 n 个元素. 与此同时，X_L和Y_R中元素数量相等，X_R和Y_L中元素数量相等. 比较$X[\lfloor n/2 \rfloor - 1]$和$Y[\lceil n/2 \rceil - 1]$:

Case 1: 如果$X[\lfloor n/2 \rfloor - 1] = Y[\lceil n/2 \rceil - 1]$，则 X 和 Y 合并后排序第 n 个元素必为$Y[\lceil n/2 \rceil - 1]$，并且第$n+1$个元素为$\min\{X[\lfloor n/2 \rfloor], Y[\lceil n/2 \rceil]\}$;

Case 2: 如果$X[\lfloor n/2 \rfloor - 1] > Y[\lceil n/2 \rceil - 1]$，则$X[\lfloor n/2 \rfloor - 1]$在 X 和 Y 合并后排序至少为 n，而且$Y[\lceil n/2 \rceil - 1]$ 在 X 和 Y 合并后排序小于 n. 进一步比较$X[\lfloor n/2 \rfloor - 1]$和$Y[\lceil n/2 \rceil]$:

Case 2.1: 如果$X[\lfloor n/2 \rfloor - 1] \le Y[\lceil n/2 \rceil]$，则$X[\lfloor n/2 \rfloor - 1]$在 X 和 Y 合并后排序为 n，而$\min\{X[\lfloor n/2 \rfloor], Y[\lceil n/2 \rceil]\}$的排序为$n+1$;

Case 2.2: 如果$X[\lfloor n/2 \rfloor - 1] > Y[\lceil n/2 \rceil]$，则$X[\lfloor n/2 \rfloor - 1]$在 X 和 Y 合并后排序至少为$n+1$，此时Y_L中元素排序位置均小于n并且X_R中元素排序均大于$n+1$，由此可知X和Y合并后排序第 n 和第 n+1 的元素在X_L和Y_R中仍为中位数. 原始问题转化为在X_L和Y_R中找到中位数.

Case 3: 如果$X[\lfloor n/2 \rfloor - 1] < Y[\lceil n/2 \rceil - 1]$，则 $Y[\lceil n/2 \rceil - 1]$在 X 和 Y 合并后排序至少为 n，而且$X[\lfloor n/2 \rfloor - 1]$ 在 X 和 Y 合并后排序小于 n. 进一步比较$Y[\lceil n/2 \rceil - 1]$和$X[\lfloor n/2 \rfloor]$:

Case 3.1: 如果$Y[\lceil n/2 \rceil - 1] \le X[\lfloor n/2 \rfloor]$，则$Y[\lceil n/2 \rceil - 1]$在 X 和 Y 合并后排序为 n，而$\min\{X[\lfloor n/2 \rfloor], Y[\lceil n/2 \rceil]\}$的排序为$n+1$;

Case 3.2: 如果$Y[\lceil n/2 \rceil - 1] > X[\lfloor n/2 \rfloor]$，则$Y[\lceil n/2 \rceil - 1]$在 X 和 Y 合并后排序至少为$n+1$，此时X_L中元素排序均小于n并且Y_R中元素排序均大于$n+1$，由此可知X和Y合并后排序第 n 和第 n+1 的元素在X_R和Y_L中仍为中位数. 原始问题转化为在X_R和Y_L中找到中位数.

（2）写出算法伪代码.

FindMedium($X[l_X : u_X]$, $Y[l_Y : u_Y]$)

$m \leftarrow u_X - l_X + 1$;
if $m < 3$
    按大小合并$X[l_X : u_X]$和$Y[l_Y : u_Y]$到数组$M[0:2m-1]$;
    return $M[m-1]$, $M[m]$;
if $X[l_X + \lfloor m/2 \rfloor - 1] = Y[l_Y + \lceil m/2 \rceil - 1]$
    return $X[l_X + \lfloor m/2 \rfloor - 1]$, $\min\{X[l_X + \lfloor m/2 \rfloor], Y[l_Y + \lceil m/2 \rceil]\}$;
else if $X[l_X + \lfloor m/2 \rfloor - 1] > Y[l_Y + \lceil m/2 \rceil - 1]$
    if $X[l_X + \lfloor m/2 \rfloor - 1] \le Y[l_Y + \lceil m/2 \rceil]$
        return $X[l_X + \lfloor m/2 \rfloor - 1]$, $\min\{X[l_X + \lfloor m/2 \rfloor], Y[l_Y + \lceil m/2 \rceil]\}$;
    else
        return FindMedium($X[l_X : l_X + \lfloor m/2 \rfloor - 1], Y[l_Y + \lceil m/2 \rceil : u_Y]$);
else
    if $Y[l_Y + \lceil m/2 \rceil - 1] \le X[l_X + \lfloor m/2 \rfloor]$
        return $Y[l_Y + \lceil m/2 \rceil - 1]$, $\min\{X[l_X + \lfloor m/2 \rfloor], Y[l_Y + \lceil m/2 \rceil]\}$;
    else
    return FindMedium($X[l_X + \lfloor m/2 \rfloor : u_x], Y[l_Y : l_Y + \lceil m/2 \rceil - 1]$);

运行 FindeMedium($X[0:n-1]$, $Y[0:n-1]$)求解原始问题.

（3）分析算法的时间复杂性.

$T(n) \leq T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(1),\ T(n) = O(\log n).$