第1次作业

作业1：抄写

P10，INSERTIONSORT

P19，MERGESORT

P17，MERGE

作业2：

1.2-2
Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

2≤n≤43

- lg43=5.426
- 43≤8lg43=43.408
- lg43=5.459
- 44≥8lg44=43.672

1.2-3
What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

n≥15

这里lg函数以2为底

- 思考题：2.2-2 选择算法

1） 程序：

```
void SelectSort(RecordType r[], int length)    /*对记录数组r做简单选择排序，length为待排序记录的个数*/
{
   int temp;
   for ( i=0 ; i< length-1 ; i++) //n-1趟排序
   {
      int index=i ;      //假设index处对应的数组元素是最小的
      for (int j=i+1 ; j < length ; j++)     //查找最小记录的位置
         if (r[j].key < r[index].key )
            index=j;
      if ( index!=i)
      {
         temp     = r[i];
         r[i]     = r[index];
         r[index] = temp;
      }
   }
}
```

2）循环不变式：子数组A[1~j-1]已按序排列，且A[1~j-1]中的元素均来自A[1~n] 。

3）最优一个元素已经就位。

4）最好、最坏、平均：$\Theta(n^2)$

3.1-5　证明定理 3.1。

从Ω、O、Θ的定义出发直接证明：

Ω：如果存在两个正常数c和n_0，对于所有的$n \geqslant n_0$，有

$$|f(n)| \geqslant c|g(n)|,$$

则记作$f(n)=\Omega(g(n))$。

O：如果存在两个正常数c和n_0，对于所有的$n \geqslant n_0$，有

$$|f(n)| \leqslant c|g(n)|,$$

则记作 $f(n)=O(g(n))$。

Ω：如果存在正常数c_1，c_2和n_0，对于所有的$n \geqslant n_0$，有

$$c_1|g(n)| \leqslant |f(n)| \leqslant c_2|g(n)|,$$

则记作$f(n)=\Theta(g(n))$

充分性证明：

必要性证明：

2-4 （逆序对） 假设 $A[1..n]$ 是一个有 n 个不同数的数组。若 $i<j$ 且 $A[i]>A[j]$，则对偶 (i, j) 称为 A 的一个**逆序对**(inversion)。

a. 列出数组 ⟨2，3，8，6，1⟩ 的 5 个逆序对。

b. 由集合 $\{1, 2, \cdots, n\}$ 中的元素构成的什么数组具有最多的逆序对？它有多少逆序对？

c. 插入排序的运行时间与输入数组中逆序对的数量之间是什么关系？证明你的回答。

d. 给出一个确定在 n 个元素的任何排列中逆序对数量的算法，最坏情况需要 $\Theta(n\lg n)$ 时间。（提示：修改归并排序。）

a. The inversions are $(1,5)$, $(2,5)$, $(3,4)$, $(3,5)$, $(4,5)$.

b．序列（n，n−1, ⋯, 2, 1）有最多的逆序对。

该序列的逆序对有 $\binom{n}{2} = n(n-1)/2$

c．插入排序

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i+1] = A[i]
        i = i − 1
    A[i+1] = key
```

while循环里的每次交换都是因为key（在A[j]位置）与A[i]是逆序对，通过交换消除逆序对，所以插入排序的运行时间正比于输入数组中的逆序对的数量。

d.类似MergeSort，在Merge的过程中，左侧子序列中任何大于右侧某元素的元素，都形成一个逆序对。所以对n个元素而言，逆序对的个数等于分治后左子序列中的逆序对个数+右子序列中逆序对个数，再加右子序列元素相对于左子序列元素形成的逆序对的个数。

```
COUNT-INVERSIONS(A, p, r)
  inversions = 0
  if p < r
      q = ⌊(p + r)/2⌋
      inversions = inversions + COUNT-INVERSIONS(A, p, q)
      inversions = inversions + COUNT-INVERSIONS(A, q + 1, r)
      inversions = inversions + MERGE-INVERSIONS(A, p, q, r)
  return inversions
```

MERGE-INVERSIONS(A, p, q, r)

```
n_1 = q - p + 1
n_2 = r - q
let L[1 .. n_1 + 1] and R[1 .. n_2 + 1] be new arrays
for i = 1 to n_1
    L[i] = A[p + i - 1]
for j = 1 to n_2
    R[j] = A[q + j]
L[n_1 + 1] = ∞
R[n_2 + 1] = ∞
i = 1
j = 1
inversions = 0
for k = p to r
    if R[j] < L[i]
        inversions = inversions + n_1 - i + 1
        A[k] = R[j]
        j = j + 1
    else A[k] = L[i]
        i = i + 1
return inversions
```

第二次作业

4.1-5 使用如下思想为最大子数组问题设计一个非递归的、线性时间的算法。从数组的左边界开始，由左至右处理，记录到目前为止已经处理过的最大子数组。若已知 $A[1..j]$的最大子数组，基于如下性质将解扩展为 $A[1..j+1]$的最大子数组：$A[1..j+1]$的最大子数组要么是 $A[1..j]$的最大子数组，要么是某个子数组 $A[i..j+1]$($1\leqslant i\leqslant j+1$)。在已知 $A[1..j]$的最大子数组的情况下，可以在线性时间内找出形如 $A[i..j+1]$的最大子数组。

阅读4.1-5题面及以下程序，然后写出你对这个算法的理解。

```
MAX-SUBARRAY-LINEAR(A)
  n = A.length
  max-sum = -∞
  ending-here-sum = -∞
  for j = 1 to n
      ending-here-high = j
      if ending-here-sum > 0
          ending-here-sum = ending-here-sum + A[j]
      else ending-here-low = j
          ending-here-sum = A[j]
      if ending-here-sum > max-sum
          max-sum = ending-here-sum
          low = ending-here-low
          high = ending-here-high
  return (low, high, max-sum)
```

ending-here-sum>0，继续向后扩展到j、计算到j位置的子序列和。不用担心A[j]<0的情况发生，因为即使A[j]<0导致ending-here-sum变小，其前已经找到的更大的子序列和也已经在全局量max-sum中有了记载，这是一个递推的过程，max-sum不受A[j]<0的影响，而一旦找到更大的ending-here-sum，则会在后面的if语句里修正max-sum，从而可以找到和更大的连续子序列。

如果ending-here-sum<=0，则ending-here-sum +A[j]只会不比A[j]更大，甚至还不如A[j]本身大，所以直接调整为“ending-here-low = j；ending-here-sum=A[j]”、ending-here-low==ending-here-high

这一步比较显然，当找到更大的连续子序列和，修正全局的max-sum和下标low、high，以记录当前以求出的最大连续子序列和和下标区间。

4.3-2 证明 $T(n) = T(\lceil n/2 \rceil) + 1$ 的解是O(lgn)

因为包含上取整函数，又要找一个上界，所以我们要用一些技巧：

假设：$T(n) \le c\lg(n-b)$,对$\lceil n/2 \rceil$成立，进而，我们有：

$$
\begin{aligned}
T(n) &\le c\lg(\lceil n/2 \rceil - b) + 1 \\
&\le c\lg(n/2 - b + 1) + 1 \\
&= c\lg\left(\frac{n - 2b + 2}{2}\right) + 1 \\
&= c\lg(n - 2b + 2) - c\lg 2 + 1 \\
&\le c\lg(n - b)
\end{aligned}
$$

最后一个不等号成立需要，$b \ge 2, c \ge 1$

所以，得证！

其它思路：$\lceil n/2 \rceil \le 2n/3$

(4.3-9) 利用改变变量的方法求解递归式 $T(n) = 3T(\sqrt{n}) + \log n$。得到的解应是紧确的。

令 $n=2^m$，$m=\log n$

$T(n)=3T(2^{m/2}) + m$

令 $S(m) = T(n)$

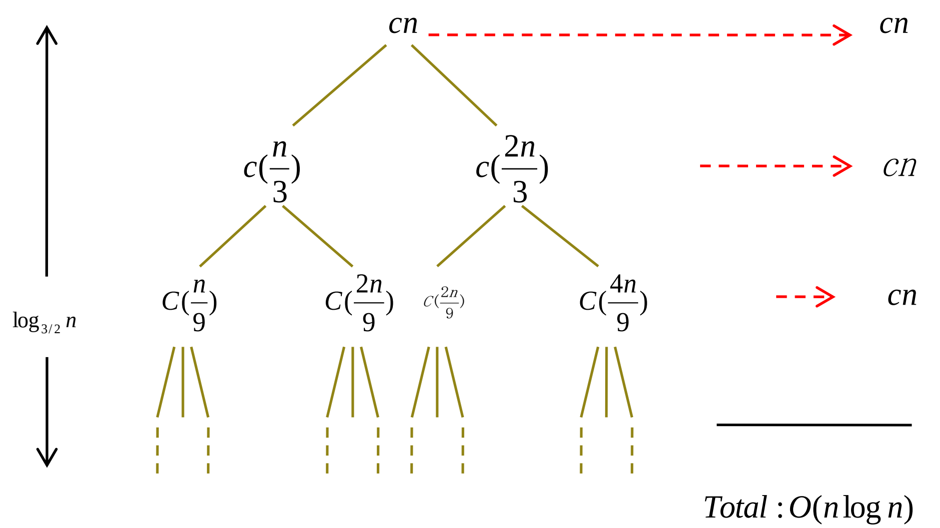
$S(m)=3S(m/2)+m$

则 $T(n) = S(m)=O(m^{\log_2 3})$

$\log_2 3=1.6$

$=O(\log n^{\log_2 3})$ (或$3^{\log_2 \log n}$)

(4.4-6) 对递归式 $T(n)=T(n/3)+T(2n/3)+cn$ 利用递归树证其解是 $\Omega(n\log n)$，其中c是一个常数



讨论其左分支

左分支深度：$\log_3 n$

0~$\log_3 n$每层的代价是cn

故，$T(n)\ge(\log_3 n+1)cn=\Omega(n\log n)$

注：$\log_3 n=\log n/\log 3\approx\log n$

进行一定的证明

(4.5-1) 用主方法来给出下列递归式的紧确渐近界：

b) $T(n) = 2T(n/4) + n^{1/2}$

d) $T(n) = 2T(n/4) + n^2$

根据主方法分情况讨论

b) $T(n) = O(n^{1/2}\log n)$

d) $T(n)=O(n^2)$

(4.5-4) 主方法能否应用于递归式T(n)=4T(n/2)+n^2logn？

为什么？给出此递归式的渐近上界。

这里a=4,b=2，$n^{\log_b a} = n^2$

而 f(n)/ $n^{\log_b a}$ =logn<n $^{\varepsilon}$

不能用主方法

$$
\begin{aligned}
T(n) &= 4T(n/2) + n^2 \log n \\
&= 4(4T(n/4) + (n/2)^2 \log(n/2)) + n^2 \log n \\
&= 4^2 T(n/2^2) + n^2 \log n - n^2 + n^2 \log n \\
&= 4^2 T(n/2^2) + 2n^2 \log n - n^2 \\
&= 4^2(4T(n/2^3) + (n/2^2)^2 \log(n/4)) + 2n^2 \log n - n^2 \\
&= 4^3 T(n/2^3) + n^2 \log n - 2n^2 + 2n^2 \log n - n^2 \\
&= 4^3 T(n/2^3) + 3n^2 \log n - 2n^2 - n^2 \\
&= \ldots \\
&= 4^k T(n/2^k) + kn^2 \log n - n^2 \sum_{i=1}^{k-1} i \\
&= n^2 + kn^2 \log n - n^2(k-1)k/2 \\
&= n^2 + n^2 \log^2 n - (n^2/2) \log^2 n + n^2 \log n/2 \\
&= \mathrm{O}(n^2 \log^2 n)
\end{aligned}
$$

第3次作业

9.1-1：证明：在最坏情况下，找到n 个元素中第二小元素需要n+[**lg**n]–2 次比较。（提示：同时找最小元素。）

以锦标赛方式比较元素——将其两个一组进行比较，然后以同样的方式对**获胜者**进行比较。需要跟踪潜在的赢家所参与的每次“赛事”。

通过n–1 次比较确定最终赢家。而第二小元素就在比赛输于最小元素的[**lg**n] 中——其中每个元素都是在所参与的最后一次赛事中失利。因此要找到最小元素还须[**lg**n]–1次比较。

9.3-1 在算法 SELECT 中，输入元素被分为每组 5 个元素。如果它们被分为每组 7 个元素，该算法仍然会是线性时间吗？证明：如果分成每组 3 个元素，SELECT 的运行时间不是线性的。

1）每组7个元素是可以保证线性时间的。

可以证明，当r=7时，小于mm或大于mm的元素数至少是

2n/7-8个：$4\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{7}\right\rceil\right\rceil-2\right)\geq\frac{2n}{7}-8$,

于是有：$T(n) \le T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n)$

可以证明T(n)=O(n)

2) 每组3个元素不能保证线性时间。

可以证明，当r=3时，**小于mm或大于mm的元素数至少是**

2n/3-4个：$2\left(\left\lceil\frac{1}{2}\left\lceil\frac{n}{3}\right\rceil\right\rceil-2\right)\geq\frac{n}{3}-4$

则有：$T(n) \le T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n)$

而n/3+2n/3+4>n非线性解。

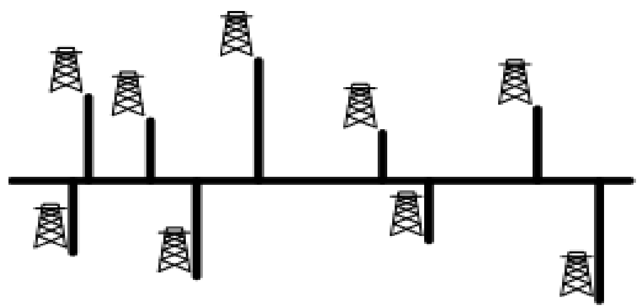
9.3-5：假设你已经有了一个最坏情况下是线性时间的用于求解中位数的“黑箱”子过程。设计一个能在线性时间内解决任意顺序统计量的选择问题算法。

```
SELECT'(A, p, r, i)
  if p == r
      return A[p]
  x = MEDIAN(A, p, r)
  q = PARTITION(x)
  k = q - p + 1
  if i == k
      return A[q]
  elseif i < k
      return SELECT'(A, p, q - 1, i)
  else return SELECT'(A, q + 1, r, i - k)
```

如果i = ⌈n/2⌉，则只需调用一次子过程，显然是线性时间。否则，只需针对所划分成的两部分中的一个调用子过程（视i的大小而定），因此有如下递归式：T(n) = T(n/2)+O(n)

由主定理可知上限为O(n)。

9.3-9：Olay教授是一家石油公司的顾问。这家公司正在计划建造一条从东到西的大型输油管道，这一管道将穿越一个有n 口油井的油田。公司希望每口油井都有一条管道支线沿着最短路径连接到主管道（方向或南或北），如下图所示。给定每口油井的x 和y 坐标，教授应该如何选择主管道的最优位置，使得各支线的总长度最小？证明：该最优位置可以在线性时间内确定。



如果n 是奇数，则选取所有油井y 坐标的中位数,作爲主管道的y坐标，即主管道穿过此油井。这样主管道两侧的油井数目相同。对于任两口油井而言，只要主管道在他们中间通过，那么这两口油井的支线管道总长度是不变的。

如果n是偶数，则需要所有油井y坐标的两个中位数，主管道的y坐标在这两个y坐标中间即可

选择中位数的原因可以证明。由于求解中位数的问题可以在线性时间确定

9-2 （带权中位数） 对分别具有正权重 w_1，w_2，…，w_n，且满足 $\sum_{i=1}^{n} w_i = 1$ 的 n 个互异元素 x_1，x_2，…，x_n 来说，带权中位数 x_k(较小中位数)是满足如下条件的元素：

$$\sum_{x_i < x_k} w_i < \frac{1}{2}$$

和

$$\sum_{x_i > x_k} w_i \leqslant \frac{1}{2}$$

例如，如果元素是 0.1，0.35，0.05，0.1，0.15，0.05，0.2，并且每个元素的权重等于本身(即对所有 $i=1$，2，…，7，都有 $w_i = x_i$)，那么中位数是 0.1，而带权中位数是 0.2。

a. 证明：如果对所有 $i=1$，2，…，n 都有 $w_i = 1/n$，那么 x_1，x_2，…，x_n 的中位数就是 x_i 的带权中位数。

b. 利用排序，设计一个最坏情况下 $O(n\lg n)$时间的算法，可以得到 n 个元素的带权中位数。

c. 说明如何利用像 9.3 节的 SELECT 这样的线性时间中位数算法，在 $\Theta(n)$最坏情况时间内求出带权中位数。

a) 根据带权中位数的定义即可得，当每个数的权重都为1/n时，带权中位数就是排序后位于中间的那个数，也即中位数就是带权中位数。

b）先排序，花O(nlogn)的时间，然后顺序搜索：从排序后的第一元素开始，累加权值sum-w，直到某个元素x_i, $\text{sum-w}_{1\sim i-1}<1/2$, $\text{sum-w}_{1\sim i-1}+w_i \geq 1/2$, 则$x_i$就是该序列的带权中位数.

c）以下程序给出在Θ(n)时间内找带权中位数的算法

WEIGHTED-MEDIAN(X)

if $n == 1$
  return x_1
elseif $n == 2$
  if $w_1 \geq w_2$
    return x_1
  else return x_2
else find the median x_k of $X = \{x_1, x_2, \ldots, x_n\}$
  partition the set X around x_k
  compute $W_L = \sum_{x_i < x_k} w_i$ and $W_G = \sum_{x_i > x_k} w_i$
  if $W_L < 1/2$ and $W_G < 1/2$
    return x_k
  elseif $W_L > 1/2$
    $w_k = w_k + W_G$
    $X' = \{x_i \in X : x_i \leq x_k\}$
    return WEIGHTED-MEDIAN(X')
  else $w_k = w_k + W_L$
    $X' = \{x_i \in X : x_i \geq x_k\}$
    return WEIGHTED-MEDIAN(X')

如果只有两个元素，直接二者的权值比较即可。以权值较大的一个作为带权中位数。

按常规的方法找中位数（普通中位数，不是带权的中位数）。

以x_k为界分为左、右两个子集（左子集的所有元素不大于x_k，右子集的所有元素不小于x_k），然后计算左、右子集的权值之和：W_L和W_G，如果W_L和W_G都小于1/2，则x_k就是要求的带权中位数。

左侧

右侧

否则，则在局部权值和较大的子集：左子集L或右子集R里重复上述过程，但注意，x_k参与下一步在L或R中的搜索，并且在递归进行下一次搜索之前将另一个子集R或L的权值之和W_R或W_L累加到w_k上，这样k元素的新权值就代表了将舍去的另一半子集中所有元素的权值之和，后续计算就可以计算出“全局”权值的和，而不会丢失被舍去的子集的元素的权值。

- 可以证明上述算法的时间是：

$$T(n) = T(n/2+1) + \Theta(n)$$

- 所以总的时间是T(n)= Θ (n).

邮局位置问题的定义如下：给定权重分别为 w_1，w_2，…，w_n 的 n 个点 p_1，p_2，…，p_n，我们希望找到一个点 p(不一定是输入点中的一个)，使得 $\sum_{i=1}^{n} w_i d(p, p_i)$ 最小，这里 $d(a, b)$ 表示点 a 与 b 之间的距离。

d. 证明：对一维邮局位置问题，带权中位数是最好的解决方法，其中，每个点都是一个实数，点 a 与 b 之间的距离是 $d(a, b) = |a-b|$。

d) The property that $\sum_{x_i > x} w_i \le 1/2$ implies that $\sum_{x \le x_i} w_i > 1/2$. This fact, combined with $x - y > 0$ and $\sum_{x > x_i} w_i < 1/2$, yields that $f(y) - f(x) > 0$.

When $x > y$, we again bound the quantity $|y - x_i| - |x - x_i|$ from below by examining three cases:

1. $x_i \le y < x$: Here, $|y - x_i| + |x - y| = |x - x_i|$ and $|x - y| = x - y$, which imply that $|y - x_i| - |x - x_i| = -|x - y| = y - x$.
2. $y \le x_i < x$: Here, $|y - x_i| \ge 0$ and $|x - x_i| \le x - y$, which imply that $|y - x_i| - |x - x_i| \ge -(x - y) = y - x$.
3. $y < x \le x_i$. Here, $|x - y| + |x - x_i| = |y - x_i|$ and $|x - y| = x - y$, which imply that $|y - x_i| - |x - x_i| = |x - y| = x - y$.

Separating out the first two cases, in which $x > x_i$, from the third case, in which $x \le x_i$, we get

$$
\begin{aligned}
f(y) - f(x) &= \sum_{i=1}^{n} w_i (|y - x_i| - |x - x_i|) \\
&\ge \sum_{x > x_i} w_i (y - x) + \sum_{x \le x_i} w_i (x - y) \\
&= (x - y) \left(\sum_{x \le x_i} w_i - \sum_{x > x_i} w_i \right) .
\end{aligned}
$$

The property that $\sum_{x_i > x} w_i \le 1/2$ implies that $\sum_{x \le x_i} w_i > 1/2$. This fact, combined with $x - y > 0$ and $\sum_{x > x_i} w_i < 1/2$, yields that $f(y) - f(x) > 0$.

当x<y时，可类似的进行分类讨论。

邮局位置问题的定义如下：给定权重分别为 w_1，w_2，…，w_n 的 n 个点 p_1，p_2，…，p_n，我们希望找到一个点 p(不一定是输入点中的一个)，使得 $\sum_{i=1}^{n} w_i d(p, p_i)$ 最小，这里 $d(a, b)$ 表示点 a 与 b 之间的距离。

e. 请给出二维邮局位置问题的最好解决方法：其中的点是(x, y)的二维坐标形式，点$a=(x_1, y_1)$与$b=(x_2, y_2)$之间的距离是 Manhattan 距离，即 $d(a, b)=|x_1-x_2|+|y_1-y_2|$。

$$f(x, y) = \sum_{i=1}^{n} w_i \left(|x - x_i| + |y - y_i|\right)$$

$$\begin{aligned}
\min_{x,y} f(x, y) &= \min_{x,y} (g(x) + h(y)) \\
&= \min_{x} \left(\min_{y} (g(x) + h(y)) \right) \\
&= \min_{x} \left(g(x) + \min_{y} h(y) \right) \\
&= \min_{x} g(x) + \min_{y} h(y) .
\end{aligned}$$

故分别选取x坐标和y坐标的带权中位数即为最好的解法。

- 思考题

  分金币（Spreading the Wealth, UVa 11300）

  圆桌旁坐着n个人，每人有一定数量的金币，金币总数能被n整除。每个人可以给他左右相邻的人一些金币，最终使得每个人的金币数目相等。你的任务是求出被转手的金币数量的最小值。比如，n=4，且4个人的金币数量分别为1,2,5,4时，只需转移4枚金币（第3个人给第2个人两枚金币，第2个人和第4个人分别给第1个人1枚金币）即可实现每人手中的金币数目相等。

解：设n个人分别拥有的金币数为$a_1, a_2, \dots, a_n$

最终每个人所拥有金币数为A，则有$A = \sum_{i=1}^{n} a_i / n$

令x_i为i给i+1的金币数(i=1,2,...,n-1) (x_i为负时即为i+1给i金币),

x_n代表n给1的金币数

故有

$$a_1 + x_n - x_1 = A \qquad x_1 = x_n - (A - a_1)$$

$$a_2 + x_1 - x_2 = A \qquad x_2 = x_1 - (A - a_2)$$

$$a_3 + x_2 - x_3 = A \qquad x_3 = x_2 - (A - a_3) = x_1 - (2 * A - a_2 - a_3)$$

$$\dots \dots$$

$$a_n + x_{n-1} - x_n = A \qquad x_n = x_{n-1} - (A - a_n)$$

$$= x_1 - ((n-1)A - \sum_{i=2}^{n} a_i)$$

目标求：$\sum_{i=1}^{n} x_i$ 的最小值；

令 $b_1 = (A - a_2)$

$$b_2 = (2 * A - a_2 - a_3)$$

$$\dots$$

$$b_{n-1} = ((n-1)A - \sum_{i=2}^{n} a_i)$$

结果就是求$|x_1|+|x_1-b_1|+|x_1-b_2|+\ldots+|x_1-b_{n-1}|$的最小值，即取$b_i$的中位数即可。