

题目描述

在系统、网络均正常的情况下组织核酸采样员和志愿者对人群进行 **核酸检测** 筛查。

每名采样员的效率不同，采样效率为N人/小时。

由于外界变化，采样员的效率会以M人/小时为粒度发生变化，M为采样效率浮动粒度， $M=N*10\%$ ，输入保证 $N*10\%$ 的结果为整数。

采样员效率浮动规则：采样员需要一名志愿者协助组织才能发挥正常效率，在此基础上，每增加一名志愿者，效率提升1M，最多提升3M；如果没有志愿者协助组织，效率下降2M。

怎么安排速度最快？求总最快检测效率（总检查效率为各采样人员效率值相加）。

输入描述

第一行：第一个值，采样员人数，取值范围[1, 100]；第二个值，志愿者人数，取值范围[1, 500]；  
第二行：各采样员基准效率值（单位人/小时），取值范围[60, 600]，保证序列中每项值计算10%为整数。

输出描述

第一行：总最快检测效率（单位人/小时）

用例

输入	2 2 200 200
输出	400
说明	输入需要保证采样员基准效率值序列的每个值*10%为整数。

题目解析

用例意思是：

有两个采样员，两个志愿者。

两个采样员的正常效率都是200，但是要给每个采样员配一个志愿者才能发挥正常效率。

现在刚好采样员和志愿者是一比一，因此可以发挥出总效率是： $200 + 200 = 400$ 。

如果，我们给一个采样员配两个志愿者，那么该采样员发挥的效率是： $200 + 200 * 10\% = 220$ 。

但是另一名采样员就没有志愿者了，因此发挥不了正常效率， $200 - 200 * 20\% = 160$ ，此时总效率是 $220 + 160 = 380$ 。

因此，总最快效率是400。

需要注意的是，用例中采样员的正常效率只是凑巧相同，很有可能出现一个采样员的正常效率极高，一个采样员的正常效率极低的情况。

我的解题思路如下：

首先分两种情况：

1、志愿者数量少于采样员

2、志愿者数量不少于采样员

对于情况1，我们应该将不多的志愿者优先分配给高效率的采样员，默认一比一分配。

接下来，我们应该考虑，剥夺低效率的采样员的志愿者 给 高效率的采样员，只要 高效率采样员增加的10%的效率 可以大于 低效率采样员减少的20%的效率。


其中还要考虑，高效率的采样员最多可以追加3个志愿者，即最多增加30%的效率。如果最高效率的采样员已经提升30%效率，则第二高效率的采样员称为最高优先级，继续上面剥夺逻辑。

对于情况2，我们应该先按一比一的方式，给每个采样员分配一个志愿者。

然后，如果还多出志愿者的话，则优先分配给高效率的采样员，同样需要注意每个采样员最多追加3个志愿者。

当多出的志愿者分配完后，我们需要考虑剥夺低效率的采样员的志愿者 给 高效率的采样员，只要 高效率采样员增加的10%的效率 可以大于 低效率采样员减少的20%的效率。逻辑同情况1。

根据ant shi网友的指正，上面解析对于志愿者超出采样员数量四倍时的情况考虑不够全面：

 伏城之外 [已关注](#)

👍 2

💬 5

🔖 5

📄 4

🔖 5

[专栏目录](#)

[已订阅](#)

## 题目解析

用例意思是：

有两个采样员，两个志愿者。

两个采样员的正常效率都是200，但是要给每个采样员配一个志愿者才能发挥正常效率。

现在刚好采样员和志愿者是一比一，因此可以发挥出总效率是：200 + 200 = 400。

如果，我们给一个采样员配两个志愿者，那么该采样员发挥的效率是：200 + 200 \* 10% = 220。

但是另一名采样员就没有志愿者了，因此发挥不了正常效率，200 - 200 \* 20% = 160，此时总效率是220 + 160 = 380。

因此，总最快效率是400。

需要注意的是，用例中采样员的正常效率只是凑巧相同，很有可能出现一个采样员的正常效率极高，一个采样员的正常效率极低的情况。

我的解题思路如下：

首先分两种情况：

1、志愿者数量少于采样员

2、志愿者数量不少于采样员

对于情况1，我们应该将不多的志愿者优先分配给高效率的采样员，默认一比一分配。

接下来，我们应该考虑，剥夺低效率的采样员的志愿者 给 高效率的采样员，只要 高效率采样员增加的10%的效率 可以大于 低效率采样员减少的20%的效率。

其中还要考虑，高效率的采样员最多可以追加3个志愿者，即最多增加30%的效率。如果最高效率的采样员已经提升30%效率，则第二高效率的采样员称为最高优先级，继续上面剥夺逻辑。

对于情况2，我们应该先按一比一的方式，给每个采样员分配一个志愿者。

然后，如果还多出志愿者的话，则优先分配给高效率的采样员，同样需要注意每个采样员最追加3个志愿者。

当多出的志愿者分配完后，我们需要考虑剥夺低效率的采样员的志愿者 给 高效率的采样员，只要 高效率采样员增加的10%的效率 可以大于 低效率采样员减少的20%的效率。逻辑同情况1。

根据ant\_shi网友的指正，上面解析对于志愿者超出采样员数量四倍时的情况考虑不够全面：

另外，对于情况2而言，如果采样员：志愿者的比例，超过了1: 4，那么超出4倍采样员范围的志愿者将没有效率提升作用，因此有效志愿者数量最多是四倍的采样员数量。

## JavaScript算法源码

```
1  /* JavaScript Node ACP模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12 })
13
14 if (lines.length === 2) {
15   const [x, y] = lines[0].split(" ").map(Number);
16   const arrX = lines[1].split(" ").map(Number);
17
18   console.log(getResult(arrX, x, y));
19   lines.length = 0;
20 }
21
22 /**
23  *
24  * @param {*} x 采样员人数
25  * @param {*} arr 每个采样员的正常效率
26  * @param {*} y 志愿者人数
27  */
28 function getResult(arr, x, y) {
29   // 按照正常效率降序
30   arr.sort((a, b) => b - a);
31
32   let max = 0;
33
34   // 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
35   if (y < x) {
36     for (let i = 0; i < x; i++) {
37       // 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率，而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
38       max += i < y ? arr[i] : arr[i] * 0.8;
39     }
40
41     let i = 0;
42     let j = y - 1;
43     let count = 0;
44     while (i < j) {
45       // 接下来 我们需要从0~y-1范围，最高效率的采样员上起10%的效率 剥夺下去，最低效率的采样员下降20%的效率
```

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [x, y] = lines[0].split(" ").map(Number);
15     const arrX = lines[1].split(" ").map(Number);
16
17     console.log(getResult(arrX, x, y));
18     lines.length = 0;
19   }
20 });
21
22 /**
23  *
24  * @param {*} x 采样员人数
25  * @param {*} arr 每个采样员的正常效率
26  * @param {*} y 志愿者人数
27  */
28 function getResult(arr, x, y) {
29   // 按照正常效率降序
30   arr.sort((a, b) => b - a);
31
32   let max = 0;
33
34   // 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
35   if (y < x) {
36     for (let i = 0; i < x; i++) {
37       // 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率，而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
38       max += i < y ? arr[i] : arr[i] * 0.8;
39     }
40
41     let i = 0;
42     let j = y - 1;
43     let count = 0;
44     while (i < j) {
45       // 接下来，我们需要比较0~y-1范围，最高效率的采样员上升10%的效率，是否大于，最低效率的采样员下降20%的效率
46       if (arr[i] * 0.1 > arr[j] * 0.2) {
47         // 如果大于，则将效率低的采样员的志愿者分配给效率高的采样员
48         max += arr[i] * 0.1 - arr[j] * 0.2;
49         // 由于一个采样员最多只能提升30%，即除了一个基础志愿者外，最多再配3个志愿者，多配了也没用
50         if (++count === 3) {
51           count = 0;
52           i++;
53         }
54         j--;
55       } else {
56         break;
57       }
58     }
59   }
60   // 如果志愿者 不少于 采样员，那么默认情况下每个采样员都分配一个志愿者
61   else {
62     // 如果志愿者人数超过采样员四倍，则多出来的志愿者就没有作用了
63     if (y >= 4 * x) {
64       y = 4 * x;
65     }
66
67     // 每个采样员都默认发挥正常效率
68     max = arr.reduce((p, c) => p + c);
69
70     // surplus记录每个采样员分配一个志愿者后，还多出来的志愿者
71     let surplus = y - x;
72
73     let i = 0;
74     let j = x - 1;
75     let count = 0;
76
77     // 优先将多出来的志愿者分配给高效率的采样员
78     while (surplus > 0) {
79       max += arr[i] * 0.1;
80       surplus--;
81       if (++count === 3) {
82         count = 0;
83         i++;
84       }
85     }
86
87     // 多出来的志愿者分配完后，则继续考虑剥夺低效率采样员的志愿者给高效率的采样员
88     while (i < j) {
89       if (arr[i] * 0.1 > arr[j] * 0.2) {
90         max += arr[i] * 0.1 - arr[j] * 0.2;
91         if (++count === 3) {
92           count = 0;
```

```

84     }
85   }
86
87   // 多出来的志愿者分配完后，则继续考虑剥夺低效率采样员的志愿者给高效率的采样员
88   while (i < j) {
89     if (arr[i] * 0.1 > arr[j] * 0.2) {
90       max += arr[i] * 0.1 - arr[j] * 0.2;
91       if (++count === 3) {
92         count = 0;
93         i++;
94       }
95       j--;
96     } else {
97       break;
98     }
99   }
100 }
101
102 return max;
103 }

```

优化逻辑后代码

```

1  /* JavaScript Node ACM模式，控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [x, y] = lines[0].split(" ").map(Number);
15     const arrX = lines[1].split(" ").map(Number);
16
17     console.log(getResult(arrX, x, y));
18     lines.length = 0;
19   }
20 });
21
22 /**
23  *
24  * @param {number} x 采样员人数
25  * @param {number} arr 每个采样员的正常效率
26  * @param {number} y 志愿者人数
27  */
28 function getResult(arr, x, y) {
29   // 按照正常效率降序
30   arr.sort((a, b) => b - a);
31
32   let max = 0;
33
34   let i;
35   let j;
36   let count = 0;
37
38   // 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
39   if (y < x) {
40     for (let i = 0; i < x; i++) {
41       // 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率，而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
42       max += i < y ? arr[i] : arr[i] * 0.8;
43     }
44
45     i = 0;
46     j = y - 1;
47   }
48   // 如果志愿者 不少于 采样员，那么默认情况下每个采样员都分配一个志愿者
49   else {
50     // 如果志愿者人数超过采样员四倍，则多出来的志愿者就没有作用了
51     if (y >= 4 * x) {
52       y = 4 * x;
53     }
54
55     // 每个采样员默认发挥正常效率
56     max = arr.reduce((p, c) => p + c);
57
58     // surplus记录每个采样员分配一个志愿者后，还多出来的志愿者
59     let surplus = y - x;
60
61     i = 0;
62     j = x - 1;
63
64     // 优先将多出来的志愿者分配给高效率的采样员
65     while (surplus > 0) {
66       max += arr[i] * 0.1;
67       surplus--;
68       if (++count === 3) {
69         count = 0;
70         i++;

```



伏城之外 已关注



专栏目录

已订阅

```

60
61     i = 0;
62     j = x - 1;
63
64     // 优先将多出来的志愿者分配给高效率的采样员
65     while (surplus > 0) {
66         max += arr[i] * 0.1;
67         surplus--;
68         if (++count == 3) {
69             count = 0;
70             i++;
71         }
72     }
73 }
74
75 // 多出来的志愿者分配完后，则继续考虑抢夺低效率采样员的志愿者给高效率的采样员
76 while (i < j) {
77     // 接下来，我们需要比较高效率的采样员上升10%的效率 是否大于 最低效率的采样员下降20%的效率
78     if (arr[i] * 0.1 > arr[j] * 0.2) {
79         // 如果大于，则将效率低的采样员的志愿者分配给效率高的采样员
80         max += arr[i] * 0.1 - arr[j] * 0.2;
81         // 由于一个采样员最多只能提升30%，即除了一个基础志愿者外，最多再配3个志愿者，多配了也没用
82         if (++count == 3) {
83             count = 0;
84             i++;
85         }
86         j--;
87     } else {
88         break;
89     }
90 }
91
92 return max;
93 }

```

#### Java算法源码

```

1  import java.util.Arrays;
2  import java.util.Scanner;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner sc = new Scanner(System.in);
7
8          int x = sc.nextInt();
9          int y = sc.nextInt();
10
11          Integer[] arrX = new Integer[x];
12          for (int i = 0; i < x; i++) {
13              arrX[i] = sc.nextInt();
14          }
15
16          System.out.println(getResult(arrX, x, y));
17      }
18
19      public static int getResult(Integer[] arr, int x, int y) {
20          // 按照正常效率降序
21          Arrays.sort(arr, (a, b) -> b - a);
22
23          int max = 0;
24          int count = 0;
25          int i;
26          int j;
27
28          // 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
29          if (y < x) {
30              // 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率，而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
31              for (int k = 0; k < x; k++) {
32                  max += k < y ? arr[k] : arr[k] * 0.8;
33              }
34
35              i = 0;
36              j = y - 1;
37          }
38          // 如果志愿者 不少于 采样员，那么就默认情况下每个采样员都分配一个志愿者
39          else {
40              // 如果志愿者人数超过采样员四倍，则多出来的志愿者就没有作用了
41              if (y >= 4 * x) {
42                  y = 4 * x;
43              }
44
45              // 每个采样员都默认发挥正常效率
46              for (Integer val : arr) {
47                  max += val;
48              }
49
50              // surplus记录每个采样员分配一个志愿者后，还多出来的志愿者
51              int surplus = y - x;
52
53              i = 0;
54              j = x - 1;
55
56              // 优先将多出来的志愿者分配给高效率的采样员

```

## Java算法源码

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int x = sc.nextInt();
9         int y = sc.nextInt();
10
11         Integer[] arrX = new Integer[x];
12         for (int i = 0; i < x; i++) {
13             arrX[i] = sc.nextInt();
14         }
15
16         System.out.println(getResult(arrX, x, y));
17     }
18
19     public static int getResult(Integer[] arr, int x, int y) {
20         // 按照正常效率降序
21         Arrays.sort(arr, (a, b) -> b - a);
22
23         int max = 0;
24         int count = 0;
25         int i;
26         int j;
27
28         // 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
29         if (y < x) {
30             // 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率。而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
31             for (int k = 0; k < x; k++) {
32                 max += k < y ? arr[k] : arr[k] * 0.8;
33             }
34
35             i = 0;
36             j = y - 1;
37         }
38         // 如果志愿者 不少于 采样员，那么就默认情况下每个采样员都分配一个志愿者
39         else {
40             // 如果志愿者人数超过采样员四倍，则多出来的志愿者就没有作用了
41             if (y >= 4 * x) {
42                 y = 4 * x;
43             }
44
45             // 每个采样员都默认发挥正常效率
46             for (Integer val : arr) {
47                 max += val;
48             }
49
50             // surplus记录每个采样员分配一个志愿者后，还多出来的志愿者
51             int surplus = y - x;
52
53             i = 0;
54             j = x - 1;
55
56             // 优先将多出来的志愿者分配给高效率的采样员
57             while (surplus > 0) {
58                 max += arr[i] * 0.1;
59                 surplus--;
60                 if (++count == 3) {
61                     count = 0;
62                     i++;
63                 }
64             }
65         }
66
67         // 志愿者分配完后，则继续考虑到夺低效率采样员的志愿者给高效率的采样员
68         while (i < j) {
69             // 如果最高效率的采样员上升10%的效率，是否大于 最低效率的采样员下降20%的效率，那么就值得剥夺
70             if (arr[i] * 0.1 > arr[j] * 0.2) {
71                 max += arr[i] * 0.1 - arr[j] * 0.2;
72
73                 // 由于一个采样员最多只能提升30%，即除了一个基础志愿者外，最多再配3个志愿者，多配了也没用
74                 if (++count == 3) {
75                     count = 0;
76                     i++;
77                 }
78                 j--;
79             } else {
80                 break;
81             }
82         }
83
84         return max;
85     }
86 }
```

## Python算法源码

伏城之外 已关注

2 5 4 专栏目录 已订阅

## Python算法源码

```
1 # 输入获取
2 x, y = map(int, input().split())
3 arr = list(map(int, input().split()))
4
5
6 # 算法入口
7 def getResult(arr, x, y):
8     # 按照正常效率降序
9     arr.sort(reverse=True)
10
11     maxV = 0
12     count = 0
13     i = None
14     j = None
15
16     # 如果志愿者少于采样员，则优先将志愿者分配给正常效率高的采样员
17     if y < x:
18         # 0~y-1范围内高效率的采样员优先获得一个志愿者，因此保持正常效率，而y~x-1范围内的低效率采样员则没有志愿者，效率下降20%
19         for k in range(x):
20             maxV += arr[k] if k < y else arr[k] * 0.8
21
22         i = 0
23         j = y - 1
24     # 如果志愿者 不少于 采样员，那么默认情况下每个采样员都分配一个志愿者
25     else:
26         # 如果志愿者人数超过采样员四倍，则多出来的志愿者就没有作用了
27         if y >= 4 * x:
28             y = 4 * x
29
30         # 每个采样员都默认发挥正常效率
31         for val in arr:
32             maxV += val
33
34         # surplus记录每个采样员分配一个志愿者后，还多出来的志愿者
35         surplus = y - x
36
37         i = 0
38         j = x - 1
39
40         # 优先将多出来的志愿者分配给高效率的采样员
41         while surplus > 0:
42             maxV += arr[i] * 0.1
43             surplus -= 1
44             count += 1
45             if count == 3:
46                 count = 0
47                 i += 1
48
49         # 志愿者分配完后，则继续考虑剥夺低效率采样员的志愿者给高效率的采样员
50         while i < j:
51             # 如果最高效率的采样员上升10%的效率 是否大于 最低效率的采样员下降20%的效率，那么就值得剥夺
52             if arr[i] * 0.1 > arr[j] * 0.2:
53                 maxV += arr[i] * 0.1 - arr[j] * 0.2
54
55                 # 由于一个采样员最多只能提升30%，即除了一个基础志愿者外，最多再配3个志愿者，多配了也没用
56                 count += 1
57                 if count == 3:
58                     count = 0
59                     i += 1
60                 j -= 1
61             else:
62                 break
63
64         return int(maxV)
65
66
67 # 算法调用
68 print(getResult(arr, x, y))
```