

# 1 基本算法

## 1.1 尺取法 (双指针)

用以解决序列的区间问题, 一般有两个要求:

1. 序列是有序的, 需要先对序列进行排序
2. 问题与序列的区间有关, 操作两个或多个指针  $i, j$  表示区间

在 Python 中, 用 while 实现较为方便

扫描方向:

反向扫描, 左右指针:  $i, j$  的方向相反;

同向扫描, 快慢指针:  $i, j$  的方向相同, 但是扫描速度一般不同, 可以形成一个大小可变的滑动窗口

### 1.1.1 反向扫描

找指定和的整数对

问题: 输入  $n$  ( $n \leq 100000$ ) 个整数, 放在数组  $a[]$  中. 找出其中的两个数, 它们之和等于整数  $m$ . (假定肯定有解).

输入:

第 1 行是数组  $a[]$ , 第 2 行是  $m$

Sample input

21 4 5 6 13 65 32 9 23

28

Sample output

5 23

---

```
1 # 哈希 复杂度为  $O(n)$ , 但是需要较大的哈希空间
2 a = list(map(int, input().split()))
3 m = int(input())
4 s = set(a)
5 outed = set()
6 for item in s:
7     if m - item in s and item not in outed:
8         print(item, m - item)
9         outed.add(m-item)
```

---

---

```
1 # 尺取法 复杂度为  $O(n \log_2 n)$ , 其中, 排序的复杂度为  $O(\log_2 n)$ , 检查的复杂度为  $O(n)$ 
2 a = list(map(int, input().split()))
3 a.sort()
4 m = int(input())
5
6 # 双指针
7 i, j = 0, len(a) - 1
8 while (i < j):
9     s = a[i] + a[j]
10    #  $s < m$ :  $i$  增加 1, 之后的  $s \geq$  当前  $s$ 
11    if s < m:
12        i += 1
13    elif s > m:
14        j -= 1
15    else:
16        print("{} {}".format(a[i], a[j]))
17        i += 1
```

---

### 判断回文串

输入: 第 1 行输入测试实例个数, 之后每行输入一个字符串

输出: 是回文串输出 yes, 不是输出 no

---

```
1 n = int(input())
2 for i in range(n):
3     s = str(input())
4     i, j = 0, len(s) - 1
5     while i < j:
6         flag = False
7         if s[i] == s[j]:
8             flag = True
9         else:
10            flag = False
11            break
12        i += 1
13        j -= 1
14    if (flag):
15        print('yes')
16    else:
17        print('no')
```

---

#### 1.1.2 同向扫描

使用尺取法产生滑动窗口

寻找区间和

给定一个长度为  $n$  的正整数数组  $a[]$  和一个数  $s$ , 在数组中找一个区间, 使得该区间的数组元素之和等于  $s$ .

输出区间的起点和终点位置

第 1 行输入数组长度  $n$ , 第 2 行输入数组, 第 3 行为  $s$

Sample input

15

6 1 2 3 4 6 4 2 8 9 10 11 12 13 14

6

Sample output

0 0

1 3

5 5

6 7

初始值  $i = j = 0$

如果  $\text{sum} = s$ : 输出一个解,  $\text{sum}$  减去  $a[i]$ ,  $i++$

如果  $\text{sum} < s$ :  $j++$ ,  $\text{sum} + a[j]$

如果  $\text{sum} > s$ :  $\text{sum} - a[i]$ ,  $i++$

---

```
1 n = int(input())
2 a = list(map(int, input().split()))
3 s = int(input())
4
5 sum = a[0]
6 i, j = 0, 0
7 while i < n and j < n:
8     if sum == s:
9         print("{} {}".format(i, j))
10        sum -= a[i]
11        i += 1
12        j += 1
13        sum += a[j]
14    elif sum < s:
15        j += 1
16        sum += a[j]
17    elif sum > s:
18        sum -= a[i]
19        i += 1
```

---

## 数组去重

给出一个数组, 输出去除重复元素之后的数组

---

```
1 # 哈希, 数据多或者数值过大时需要占用大量的空间
2 a = list(map(int, input().split()))
3 s = set(a)
4 unique_a = list(s)
5 print(unique_a)
```

---

```
1 # 尺取法
2 a = list(map(int, input().split()))
3 # a排序, 使得相同元素排列在一起
4 a.sort()
5 n = len(a)
6 # 双指针均从0开始
7 i, j = 0, 0
8 # j始终指向无重复元素部分的最后一个元素
9 while i < n and j < n:
10     # 若i和j指向的元素不同, j++, 将i指向的元素复制到j上
11     # EX: 1 2(j) 3(i) 3
12     # -> 1 2 3(j, i) 3
13     # EX: 1 2 3(j) 3 4(i)
14     # -> 1 2 3 3(j) 4(i)
15     # -> 1 2 3 4(j) 4(i)
16     if a[i] != a[j]:
17         j += 1
18         a[j] = a[i]
19     i += 1
20 unique_a = a[0:j+1]
21 print(unique_a)
```

---

## 找相同数对

洛谷 P1102

给出一串数字和一个数字  $C$ , 要求计算出所有  $A - B = C$  的数对的个数 (不同位置的数字一样的数对算不同的数对)

输入: 2 行, 第 1 行输入整数  $n$  和  $C$ , 第 2 行输入  $n$  个整数

输出: 满足  $A - B = C$  的数对的个数

Sample Input

6 3

8 4 5 7 7 4

Sample Output

5

---

```
1 n, c = map(int, input().split())
2 a = list(map(int, input().split()))
3 a.sort()
4
5 i, j, k = 0, 0, 0
6 ans = 0
7
8 for i in range(n):
9     # j, k指向相同元素区间的起点和终点后1个元素
10    # 寻找的对象是区间内的元素 - a[i] = C
11    while j < n - 1 and a[j] - a[i] < c:
12        j += 1
13    while k < n and a[k] - a[i] <= c:
14        k += 1
15    if a[j] - a[i] == c and a[k-1] - a[i] == c and k - 1 >= 0:
16        ans += k - j
17 print(ans)
```

---

## 1.2 二分法

### 1.2.1 Python 二分搜索库 bisect

---

```
1 from bisect import *
2 def fun(find, x, bias=0):
3     global a
4     index = find(a, x) + bias
5     print("index: {}, element: {}".format(index, a[index]))
6
7 global a
8 a = [1, 2, 4, 4, 4, 5]
9 # target: x
10 x = 4
11
12 # first > x
13 fun(bisect_right, x)
14 # first >= x
15 fun(bisect_left, x)
16 # first = x
17 fun(bisect_left, x)
18 # last = x
19 fun(bisect_right, x, bias=-1)
20 # last <= x
21 fun(bisect_right, 3, bias=-1)
22 # last < x
23 fun(bisect_left, x, bias=-1)
24 # count x in a monotonic array
25 # slow
26 print(a.count(x))
27 # fast, using binary search
28 print(bisect_right(a, x) - bisect_left(a, x))
```

---

#### output

---

```
1 index: 5, element: 5
2 index: 2, element: 4
3 index: 2, element: 4
4 index: 4, element: 4
5 index: 1, element: 2
6 index: 1, element: 2
7 3
8 3
```

---

### 1.2.2 整数二分

需要注意终止边界和左右区间问题, 避免漏解和死循环

#### mid 的计算

---

```
1 # 适用单调递增序列的后继问题
2 mid = l + (r - l) // 2 # 相当于计算出的mid向下取整, 计算的是左中位数
3 # 适用单调递增序列的前驱问题
4 mid = l + (r - l + 1) // 2 # 相当于计算出的mid向上取整, 计算的是右中位数
```

---

### 在单调递增序列中寻找 $x$ 或 $x$ 的后继

在单调递增序列中寻找第一个  $x$  的位置, 若没有  $x$ , 则寻找比  $x$  大的第一个数的位置, 即寻找第一个  $\geq x$  的位置

---

```

1 # 左闭右开 [0, n)
2 l, r = 0, n
3 while l < r:
4     mid = l + (r - l) // 2
5     if a[mid] >= x:
6         r = mid
7     else:
8         l = mid + 1
9 return l

```

---

### 在单调递增序列中寻找 $x$ 或 $x$ 的前驱

在单调递增序列中寻找第一个  $x$  的位置, 若没有  $x$ , 则寻找比  $x$  小的第一个数的位置, 即寻找第一个  $\leq x$  的位置

---

```

1 # 左开右闭 (-1, n-1]
2 l, r = -1, n - 1
3 while l < r:
4     mid = l + (r - l + 1) // 2
5     if a[mid] <= x:
6         l = mid
7     else:
8         r = mid - 1
9 return l

```

---

### 寻找 minimum

---

```

1 while l < r:
2     mid = l + (l - r) // 2
3     if check(mid):
4         # reduce
5         r = mid
6     else:
7         # enlarge
8         l = mid + 1
9 # 若  $r = mid - 1$ , 则当  $best = mid$  时可能遗漏最优解

```

---

## 寻找指定和的整数对

输入  $n$  ( $n \leq 100000$ ) 个整数, 找出其中的两个数, 使它们之和等于整数  $m$ , 假设肯定有解

---

```
1 from bisect import *
2
3 a = list(map(int, input().split()))
4 m = int(input())
5 n = len(a)
6 a.sort()
7
8 # ver. 1
9 for i in range(n-1):
10     # bisearch, a[k] = m - a[i]
11     l, r = i+1, n
12     x = m - a[i]
13     while l < r:
14         mid = l + (r - l) // 2
15         if a[mid] >= x:
16             r = mid
17         else:
18             l = mid + 1
19     if a[l] == x:
20         print("{} {}".format(a[i], a[l]))
21
22 # ver. 2
23 for i in range(n-1):
24     # bisearch, a[k] = m - a[i]
25     x = m - a[i]
26     # search from a[i+1] to a[end-]
27     p = bisect_left(a, x, lo=i+1, hi=n)
28     if a[p] == x:
29         print("{} {}".format(a[i], a[p]))
```

---

### 1.2.3 整数二分 最大值最小化

#### 序列划分: 二分 + 贪心

给定一个序列, 如  $\{2, 2, 3, 4, 5, 1\}$ , 将其划分为  $m$  个连续子序列  $S_1, S_2, S_3$ , 每个子序列至少有一个元素, 使得每个子序列的和的最大值最小

EX.  $m = 3$

划分为  $(2, 2, 3), (4, 5), (1)$  子序列和分别为 7, 9, 1, 最大值为 9

划分为  $(2, 2, 3), (4), (5, 1)$  子序列和为 7, 4, 6, 最大值为 7, 优于前一个

---

```

1 # Input:
2 # array
3 # m: amount of subarray
4 # Output:
5 # x: minimum of maximum of sum(all possible subarrays)
6 # subarrays
7
8 from bisect import *
9
10 a = list(map(int, input().split()))
11 m = int(input())
12 n = len(a)
13 l, r = max(a), sum(a)
14 subs = []
15
16 while l < r:
17     mid = l + (r - l) // 2
18     # greedy divide subarray
19     # each sum(subarray) <= mid
20     flag = False
21     idx = 0
22     subs = []
23     for i in range(m):
24         sum = 0
25         while idx < n and sum + a[idx] <= mid:
26             sum += a[idx]
27             idx += 1
28         if idx == n or sum + a[idx] > mid:
29             subs.append(idx-1)
30     # judge
31     if subs[-1] < n-1:
32         flag = False
33     else:
34         flag = True
35     # binary control
36     if flag:
37         # reduce
38         r = mid
39     else:
40         # enlarge
41         l = mid + 1
42
43 print('minimum sum: ', l)
44 for i in range(m):
45     if i == 0:
46         print(a[ : subs[0]+1])
47     else:
48         print(a[subs[i-1]+1 : subs[i]+1])

```

---



### 1.2.4 整数二分 最小值最大化

#### 洛谷 P1824 进击的奶牛

在一条很长的直线上, 指定  $n$  个坐标点. 有  $c$  头牛, 每头牛占据一个坐标点, 求相邻两头牛之间距离的最大值

Input:

第 1 行输入:  $n\ c$

第 2 行开始每行输入: 一个整数, 表示每个点的坐标

---

```

1 n, c = map(int, input().split())
2 x = []
3 for i in range(n):
4     x.append(int(input()))
5 x.sort()
6
7 def check(x, dis, c):
8     i = 1
9     last = 0
10    c -= 1
11    while c > 0 and i < len(x):
12        while i < len(x) and x[i] - x[last] < dis:
13            i += 1
14        if i < len(x) and x[i] - x[last] >= dis:
15            c -= 1
16            last = i
17            i += 1
18    if c == 0:
19        return True
20    else:
21        return False
22
23 l, r = 0, x[-1] - x[0]
24 ans = 0
25 while l < r:
26     mid = l + (r - l) // 2
27     if check(x, mid, c):
28         ans = mid
29         l = mid + 1
30     else:
31         r = mid
32 print(ans)

```

---

### 1.2.5 实数二分

**for 控制:** 过大的 for 次数会超时, 过小的会导致精度不够答案错误. 一般取 100, 但是循环体内计算量大的时候容易超时, 可以缩减到 50

**while 控制:** while 需要设计精度 eps, 过小的 eps 会超时, 过大的会导致精度不够答案错误

---

```

1 # 精度, 可以调整
2 eps = 1e-7
3 while r - l > eps:
4     # for ver.
5     # epoch = 100 # 轮次
6     # for i in range(epochs):
7         mid = l + (r - l) / 2.0
8         if check(mid):
9             # reduce range
10            r = mid
11        else:
12            # enlarge range
13            l = mid
14 return l

```

---

## 分蛋糕 poj 3122

$m+1$  个人分  $n$  个半径不同的蛋糕, 要求每个人分得的蛋糕重量一致, 且必须是可以切出来的一整块, 每个人能分到的最大蛋糕是多少.

Input:

第一行: 1 个整数, 表示测试用例个数

对每个测试, 第一行输入  $n, m$ . 第二行输入  $n$  个整数, 表示每个蛋糕的半径

Output:

对于每个测试, 输出一个答案, 保留 4 位小数

可以将问题建模为最小值最大化问题, 用面积代替重量

保留小数

---

```
1 a = 1.13456 # float
2 ans = format(a, '.4f') # 四舍五入保留4位小数
```

---



---

```
1 def check(mid, area, f):
2     sum = 0
3     for i in range(len(area)):
4         sum += int(area[i] / mid)
5     if sum >= f:
6         return True
7     else:
8         return False
9
10 eps = 1e-5
11 pi = 3.1415926
12 T = int(input())
13 for t in range(T):
14     n, f = map(int, input().split())
15     cakes = list(map(float, input().split()))
16     maxx = 0
17     area = []
18     for i in range(n):
19         area.append(pi * cakes[i] * cakes[i])
20         if area[i] > maxx:
21             maxx = area[i]
22     l, r = 0, maxx
23     while r - l > eps:
24         mid = l + (r - l) / 2.0
25         if check(mid, area, f):
26             l = mid
27         else:
28             r = mid
29     ans = format(l, '.4f')
30     print(ans)
```

---

## 1.3 三分法

用于求取单峰函数的极值. 通过在  $[l, r]$  内取两个点 **mid1, mid2**, 将函数分为三段

---

```
1 k = (r - l) / 3.0
2 mid1, mid2 = l + k, r - k
```

---

### 三分法模板 洛谷 P3382

给出一个  $N$  次多项式函数, 保证在区间  $[l, r]$  内存在一点  $x$ , 使得  $x$  是函数在区间上的极大值, 求出  $x$

Input:

第一行输入  $n$ :  $N \mid r$

第二行输入:  $N + 1$  个实数, 表示从高到低各项的系数

Output:

$x$ , 四舍五入保留 5 位小数

---

```
1 n, l, r = map(float, input().split())
2 n = int(n)
3 cons = list(map(float, input().split()))
4 def f(x, n, cons):
5     sum = 0
6     for i in range(1, n+2):
7         sum += cons[-i] * pow(x, i-1)
8     return sum
9 eps = 1e-6
10 while r - l > eps:
11     k = (r - l) / 3.0
12     mid1, mid2 = l + k, r - k
13     if f(mid1, n, cons) < f(mid2, n, cons):
14         l = mid1
15     else:
16         r = mid2
17 ans = format(l, '.5f')
18 print(ans)
```

---

**三分法函数洛谷 P1883**

给定  $n$  个二次函数  $f_1(x), f_2(x), \dots, f_n(x)$  (均形如  $ax^2 + bx + c$ ) , 设  $F(x) = \max\{f_1(x), f_2(x), \dots, f_n(x)\}$ , 求  $F(x)$  在区间  $[0, 1000]$  上的最小值。

输入格式

输入第一行为正整数  $T$ , 表示有  $T$  组数据。

每组数据第一行一个正整数  $n$ , 接着  $n$  行, 每行 3 个整数  $a, b, c$ , 用来表示每个二次函数的 3 个系数, 注意二次函数有可能退化成一个。

输出格式

每组数据输出一行, 表示  $F(x)$  的在区间  $[0, 1000]$  上的最小值。答案精确到小数点后四位, 四舍五入。

输入输出样例 1

输入 1

2

1

2 0 0

2

2 0 0

2 -4 2

输出 1

0.0000

0.5000

说明/提示

对于 50% 的数据,  $n \leq 100$ 。

对于 100% 的数据,  $T < 10$ ,  $n \leq 10^4$ ,  $0 \leq a \leq 100$ ,  $|b| \leq 5 \times 10^3$ ,  $|c| \leq 5 \times 10^3$ 。

---

```

1 def cal(cons, x):
2     ans = -float('inf')
3     for a, b, c in cons:
4         ans = max(ans, a * x ** 2 + b * x + c)
5     return ans
6
7 eps = 1e-9
8 T = int(input())
9 for t in range(T):
10    n = int(input())
11    cons = []
12    for i in range(n):
13        a, b, c = map(float, input().split())
14        cons.append([a, b, c])
15    l = 0
16    r = 1000
17    while r - l > eps:
18        margin = (r - l) / 3.0
19        mid1 = l + margin
20        mid2 = r - margin
21        f1 = cal(cons, mid1)
22        f2 = cal(cons, mid2)
23        if f1 <= f2:
24            r = mid2
25        else:
26            l = mid1
27    ans = cal(cons, l)
28    print(format(ans, '.4f'))

```

---

## 1.4 排序与排列

### 1.4.1 排序

#### Python 自定排序 key

key function:

Input:  $x, y$

Output: 1:  $x > y$   
0:  $x = y$   
-1:  $x < y$

---

```
1 from functools import cmp_to_key
2 def cmp(x, y):
3     # compare x and y
4     # return result
5 sort_key = cmp_to_key(cmp)
```

---

#### 洛谷 P1093

P1093 [NOIP 2007 普及组] 奖学金

题目背景

NOIP2007 普及组 T1

题目描述

某小学最近得到了一笔赞助，打算拿出其中一部分为学习成绩优秀的前 5 名学生发奖学金。期末，每个学生都有 3 门课的成绩：语文、数学、英语。先按总分从高到低排序，如果两个同学总分相同，再按语文成绩从高到低排序，如果两个同学总分和语文成绩都相同，那么规定学号小的同学排在前面，这样，每个学生的排序是唯一的。

任务：先根据输入的 3 门课的成绩计算总分，然后按上述规则排序，最后按排名顺序输出前五名名学生的学号和总分。

注意，在前 5 名同学中，每个人的奖学金都不相同，因此，你必须严格按上述规则排序。例如，在某个正确答案中，如果前两行的输出数据（每行输出两个数：学号、总分）是：

7 279

5 279

这两行数据的含义是：总分最高的两个同学的学号依次是 7 号、5 号。这两名同学的总分都是 279（总分等于输入的语文、数学、英语三科成绩之和），但学号为 7 的学生语文成绩更高一些。

如果你的前两名的输出数据是：

5 279

7 279

则按输出错误处理，不能得分。

输入格式

共  $n + 1$  行。

第 1 行为一个正整数  $n \leq 300$ ，表示该校参加评选的学生人数。

第 2 到  $n + 1$  行，每行有 3 个用空格隔开的数字，每个数字都在 0 到 100 之间。第  $j$  行的 3 个数字依次表示学号为  $j - 1$  的学生的语文、数学、英语的成绩。每个学生的学号按照输入顺序编号为  $1 \sim n$ （恰好是输入数据的行号减 1）。

保证所给的数据都是正确的，不必检验。

输出格式

共 5 行，每行是两个用空格隔开的正整数，依次表示前 5 名学生的学号和总分。

输入输出样例 1

输入 1

```
6
90 67 80
87 66 91
78 89 91
88 99 77
67 89 64
78 89 98
```

输出 1

```
6 265
4 264
3 258
2 244
1 237
```

---

```
1 # 洛谷P1093
2
3 # 自定义比较函数 cmp -> key
4 # x: (id, c, m, e, sum)
5 # Input: x, y
6 # Output:
7 # 1: x > y; 0: x = y; -1: x < y
8 from functools import cmp_to_key
9 def cmp(x, y):
10     if x[4] > y[4]:
11         return 10
12     elif x[4] < y[4]:
13         return -1
14     else:
15         if x[1] > y[1]:
16             return 1
17         elif x[1] < y[1]:
18             return -1
19         else:
20             if x[0] < y[0]:
21                 return 1
22             else:
23                 return -1
24 sort_key = cmp_to_key(cmp)
25
26 n = int(input())
27 stu = []
28 for id in range(1, n+1):
29     c, m, e = map(int, input().split())
30     summ = c + m + e
31     stu.append([id, c, m, e, summ])
32 rst = sorted(stu, key=sort_key, reverse=True)
33 for i in range(5):
34     print('{} {}'.format(rst[i][0], rst[i][-1]))
```

---

### 1.4.2 排列

#### Python 生成全排列

---

```
1 from itertools import permutations
2 a = [1, 2, 3]
3 # 全排列
4 per = permutations(a)
5 for item in per:
6     print(item)
7 # Output:
8 (1, 2, 3)
9 (1, 3, 2)
10 (2, 1, 3)
11 (2, 3, 1)
12 (3, 1, 2)
13 (3, 2, 1)
14
15 #  $n$ 取 $m$ 排列
16 per = permutations(a, 2)
17 for item in per:
18     print(item)
19 # Output:
20 (1, 2)
21 (1, 3)
22 (2, 1)
23 (2, 3)
24 (3, 1)
25 (3, 2)
```

---

## 蓝桥杯 2016 省赛 Cpp A 组 T6

有加减乘除 4 种运算:

$$() + () = ()$$

$$() - () = ()$$

$$() \times () = ()$$

$$() \div () = ()$$

每个括号代表 1-13 中的某个数字, 但不能重复, 一共有多少种方案

---

```
1 # 蓝桥杯2016省赛C++A组第6题
2
3 a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
4 n = len(a)
5 vis = [False] * n
6 b = [None] * n
7 ans = 0
8
9 def permu(s, t):
10     global ans
11     # 输出
12     if s == 12 and b[9] == b[10] * b[11]:
13         ans += 1
14     return
15     # 剪枝
16     if s == 3 and b[0] + b[1] != b[2]:
17         return
18     if s == 6 and b[3] - b[4] != b[5]:
19         return
20     if s == 9 and b[6] * b[7] != b[8]:
21         return
22     # 全排列
23     for i in range(t):
24         if not vis[i]:
25             vis[i] = True
26             b[s] = a[i]
27             permu(s+1, t)
28             vis[i] = False
29
30 permu(0, n)
31 print(ans)
```

---



## 1.5 分治法

将问题分解为多个独立的子问题, 子问题的规模大致相等. 分治法常能够显著降低算法的时间复杂度.

分治法经典应用: 汉诺塔, 归并排序, 快速排序等

### 汉诺塔 蓝桥 1512

Input: 两个整数 N M. N 为要移动的盘子数, M 为最少移动步骤的第 M 步

Output:

第一行: #No: a->b. No 为移动的盘子编号, a->b 为从 a 杆移动到 b 杆, 取值为 {A, B, C}

第二行: 一个整数, 最少移动步数

---

```
1 step = 0
2 n, m = map(int, input().split())
3
4 def hanoi(x, y, z, n):
5     global step, m
6     if n == 1:
7         step += 1
8         if step == m:
9             print('#{}: {}->{}'.format(m, x, z))
10    else:
11        hanoi(x, z, y, n-1)
12        step += 1
13        if step == m:
14            print('#{}: {}->{}'.format(m, x, z))
15        hanoi(y, x, z, n-1)
16
17 hanoi('A', 'B', 'C', n)
18 print(step)
```

---

**逆序对 hdu 4911**

输入一个序列, 交换任意两个元素, 不超过  $k$  次交换之后, 最少的逆序对有多少个.

逆序对:  $a_i, a_j$ , when  $1 \leq i < j < n$  and  $a_i > a_j$

Input:

第一行:  $n, k$

第二行:  $n$  个整数

Output: 最少的逆序对数量

---

```
1 # hdu 4911
2
3 n, k = map(int, input().split())
4 a = list(map(int, input().split()))
5 count = 0
6 b = [0]*10000
7
8 def merge(l, mid, r):
9     global count, a, b
10    i, j = l, mid+1
11    t = 0
12    while i <= mid and j <= r:
13        if a[i] > a[j]:
14            b[t] = a[j]
15            t, j = t+1, j+1
16            count += mid - i + 1
17        else:
18            b[t] = a[i]
19            t, i = t+1, i+1
20    while i <= mid:
21        b[t] = a[i]
22        t, i = t+1, i+1
23    while j <= r:
24        b[t] = a[j]
25        t, j = t+1, j+1
26    for i in range(t):
27        a[l+i] = b[i]
28
29 def merge_sort(l, r):
30     if l < r:
31         mid = (l + r) // 2
32         merge_sort(l, mid)
33         merge_sort(mid+1, r)
34         merge(l, mid, r)
35
36 merge_sort(0, len(a)-1)
37 if count <= k:
38     print(0)
39 else:
40     print(count - k)
```

---