蛟龙四班

常见排序算法

#1029、合并数组

题目描述

有两个有序的数组, $nums_1, nums_2$,请你把两个数组合并成一个有序的数组 num 。

输入格式

第一行两个整数 n, m ,代表两个数组的长度

第二行 n 个数，表示数组 num_1

第二行 m 个数，表示数组 num_2

保证 $2 \leq n + m \leq 1000000$

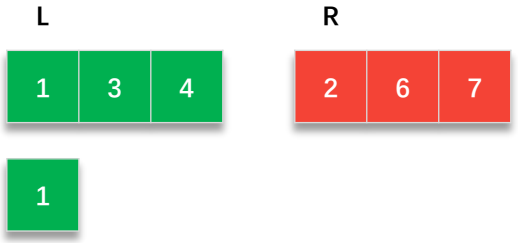
输出格式

一行一个数

合并后重新排序将会超时

时间复杂度需要$O(n + m)$

#1029、合并数组



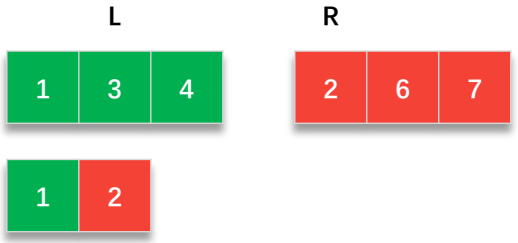
1、

L为数组a第一个元素下标,R为数组b第一个元素下标

比较$a[L]$ 和 $b[R]$大小

取出$a[L]$(较小)放入数组c

L递增1,考虑a数组下一个元素

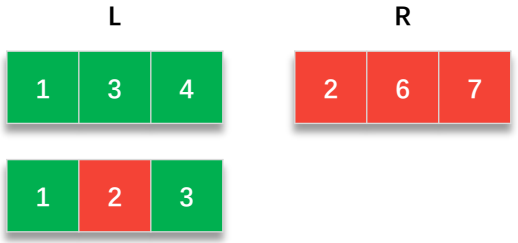


2、

比较$a[L]$ 和 $b[R]$大小

取出b$[R]$(较小)放入数组c

R递增1,考考虑b数组下一个元素

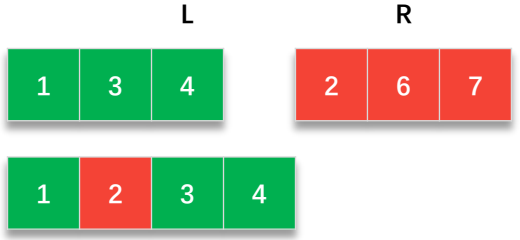


3、

比较$a[L]$ 和 $b[R]$大小

取出a$[L]$(较小)放入数组c

L递增1,考虑a数组下一个元素

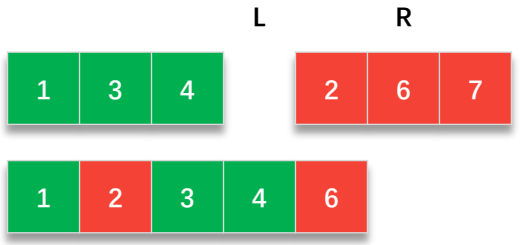


4、

比较$a[L]$ 和 $b[R]$大小

取出a$[L]$(较小)放入数组c

L递增1,考虑a数组下一个元素

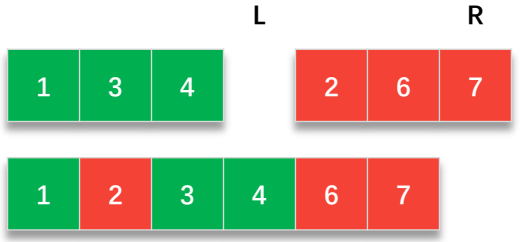


5、

a数组为空

取出b$[R]$放入数组c

R递增1,考虑b数组下一个元素



6、

a数组为空

取出b$[R]$放入数组c

b数组为空,全部取完结束

#1029、合并数组


```cpp
#include <bits/stdc++.h>
using namespace std;
int a[1000005], b[1000005], c[1000005], n, m;
int main()
{
  scanf("%d%d", &n, &m);
  for (int i = 1; i <= n; i++)
    scanf("%d", a + i);
  for (int i = 1; i <= m; i++)
    scanf("%d", b + i);
  int L = 1, R = 1, pos = 1;
  while (L <= n || R <= m)
    if (R > m || (L <= n && a[L] <= b[R]))   // 当b数组取完 或者  a数组没取完并且 a[L]不小于b[R]时
      c[pos++] = a[L++];  //取出 a[L]放入 c 数组
    else
      c[pos++] = b[R++];
  for (int i = 1; i <= m + n; i++)
    printf("%d\n", c[i]);
  return 0;
}
```

#1732、归并

题目描述

所谓的**归并**，就是把多个有序数组合并为一个有序数组的操作，本题就是要实现这个操作。
某中学初一有 n 个班级，第 i 个班级有 a_i 位同学，现在每个班的同学都已经在各自班级门口按身高从小到大排成了一队，现在作为精通编程的你要指挥这 n 条队伍中的所有人排成一队，排成的新队伍需要满足 3 个条件：

- 按照身高从小到大顺序；
- 如果身高相同，按照班级编号小的在前，例如 1 班身高 160 的同学需要排在 2 班身高 160 的同学前面；
- 如果班级编号也相同，按照他们在原来队伍中的顺序排列，例如 2 位 1 班身高 160 的同学 a 和 b，在原先队伍中 a 在 b 前面，则新队伍中 a 还在 b 前面。

排队操作是这样进行的：
老师给你了这 n 条队伍的信息，你从 1 班到 n 班叫出所有班级队伍的第一位同学，找出其中身高最小的，让他成为新队伍的第一位；
接着该同学原先队伍中的后面一位同学代替他成为那条队伍中的第一位；
接着你继续找出所有队伍第一位中身高最小的同学，让他成为新队伍的第二位.....这样的操作一直进行下去，直到所有同学都进入新队伍

例如：
有三条队伍

```
1班：140,150,160
2班：150,160,165
3班：155,157,166
```

你首先选择 1 班 140，2 班 150，3 班 155 中最小的 1 班 140，然后 1 班 150 同学就代替 140 同学成为第一位，接着你在 1 班 150，2 班 150，3 班 155 同学中选择最小的 1 班 150（相同身高班级编号小优先）.....最后新队伍就变成 140，1 班 150，2 班 150，155，157，1 班 160，2 班 160，165，166，完成排序工作。

现在老师不想知道新队伍中每位同学的身高，他只想知道每位同学的做操优美度，请你将结果汇报给他。

输入格式

输入第一行一个数 n，表示班级的数量；
接下来第二行，n 个数 a_i，表示每个班级的人数；

接下来 n 行，每行 a_i 个数，表示 i 班队伍中的身高顺序 $H[i][j]$，已经按照身高从小到大；

接下来 n 行，每行 a_i 个数，表示 i 班队伍中每位同学的做操优美度 $v[i][j]$，与身高顺序一一对应。

输出格式

输出 1 行，按顺序输出新队伍中每个同学的做操优美度，用空格隔开。

输入样例1

```
3
3 3 3
140 150 160
150 160 165
155 157 166
1 3 5
2 4 5
5 2 4
```

输出样例1

```
1 3 2 5 2 5 4 5 4
```

数据范围

对于 20% 的数据，$n=2, 1 \le a_i \le 1000$；

对于 60% 的数据，$1 \le a[i] \le 100000$；

对于 100% 的数据，$1 \le n \le 4, 1 \le a_i \le 700000, 1 \le H[i][j] \le 1000, 1 \le v[i][j] \le 9$。

#1732、归并

多个有序数组合并成一个有序数组

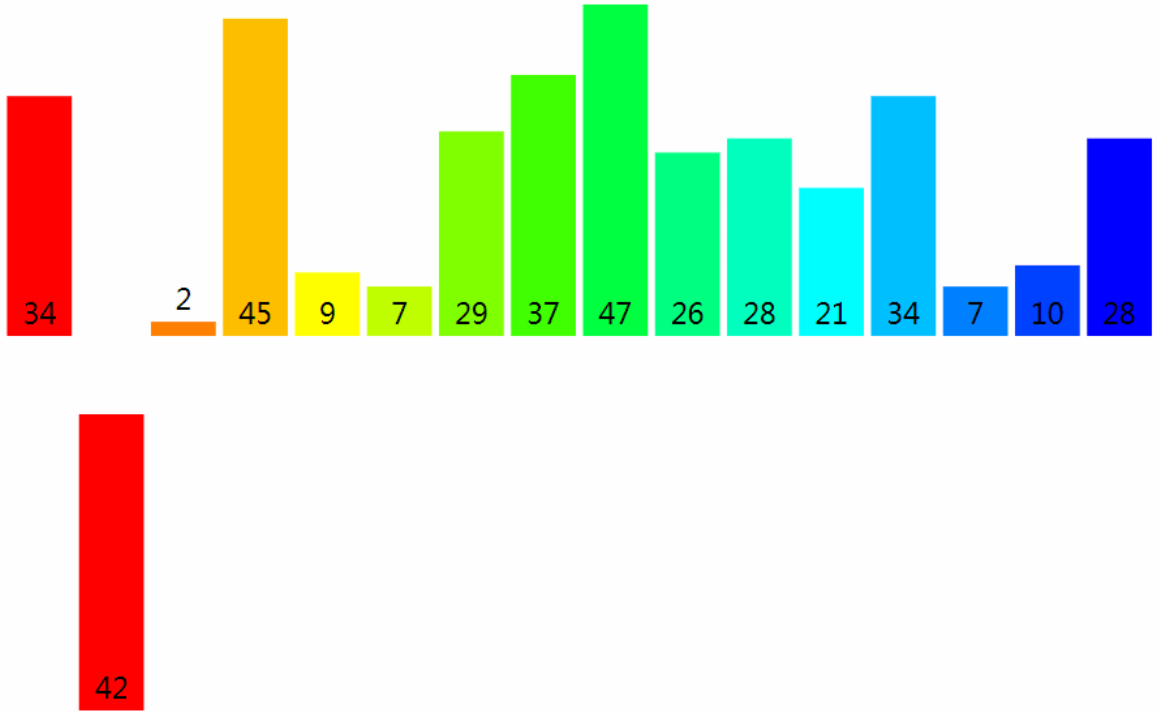
记 $\sum_{i=1}^{n} a_i$ 为T

直接排序进行多关键字排序,时间复杂度$O(T \log T)$，时间复杂度比较高

参考 #1029、合并数组 思路,时间复杂度$O(Tn)$

```
while (cnt++ < total)
{
  int m = INT_MAX, id;
  for (int i = 0; i < n; i++)          //从n个数组中找出最小
    if (nxt[i] < a[i] && m > l[i][nxt[i]].h)
      m = l[i][nxt[i]].h, id = i;
  printf("%d ", l[id][nxt[id]].v);
  nxt[id]++;    //记录每个数组下一次选取的位置
}
```

归并排序



Merge Sort

We copy the elements from the new array back into the original array.

```
split each element into partitions of size 1
recursively merge adjacent partitions
  for i = leftPartIdx to rightPartIdx
    if leftPartHeadValue <= rightPartHeadValue
      copy leftPartHeadValue
    else: copy rightPartHeadValue
copy elements back to original array
```

归并排序（$Merge\ Sort$）是建立在归并操作上的一种有效的排序算法,该算法是采用分治法（$Divide\ and\ Conquer$）的一个典型的应用。

将已有序的子序列合并,得到完全有序的序列；即先使每个子序列有序,再使子序列段间有序。

若将两个有序表合并成一个有序表,称为二路归并。

归并排序

```
int helper[100001],a[100001];
void mergeSort(int l, int r)
{
    if (l >= r)
        return;
    int mid = l + r >> 1;
    mergeSort(l, mid); //递归处理 a[l~mid] 使得有序
    mergeSort(mid + 1, r);//递归处理 a[mid+1 ~ r]得有序
    int x = l, y = mid + 1, pos = 0;
    while (x <= mid || y <= r)   //将  a[l~mid]、a[mid+1 ~ r] 两个有序数组合并到helper数组中
        if ((x <= mid && a[x] <= a[y]) || y > r)
            tmp[pos++] = a[x++];
        else
            tmp[pos++] = a[y++];
    for (int i = l; i <= r; i++) //将 helper 数组中元素覆盖到a[l~r]中
        a[i] = tmp[i - l];
}
```

归并排序分为三个步骤:

- 将数列划分为两部分
- 递归地分别对两个子序列进行归并排序
- 合并两个子序列

不难发现,归并排序的前两步都很好实现,关键是如何合并两个子序列。

注意到两个子序列在第二步中已经保证了都是有序的了,第三步中实际上是想要把两个 **有序** 的序列合并起来。

归并排序时间复杂度

$$f(n) = 2f(\frac{n}{2}) + n$$

其中$f(n)$表示对n个数进行的复杂度;$2f(\frac{n}{2})$表示将n个数分成两部分分别进行归并排序复杂度,n表示两个子过程结束后合并的复杂度

$$f\left(\frac{n}{2}\right) = 2f\left(\frac{n}{4}\right) + \frac{n}{4}$$

$$f\left(\frac{n}{4}\right) = 2f\left(\frac{n}{8}\right) + \frac{n}{8}$$

$$\cdots$$

$$f(\frac{n}{2^{m-1}}) = 2f(\frac{n}{2^m}) + \frac{n}{2^{m-1}}$$

$$f(n) = 2f\left(\frac{n}{2}\right) + n = 2 \times \left(2f\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 2^2 f\left(\frac{n}{2^2}\right) + 2n = \cdots = 2^m f(\frac{n}{2^m}) + nm$$

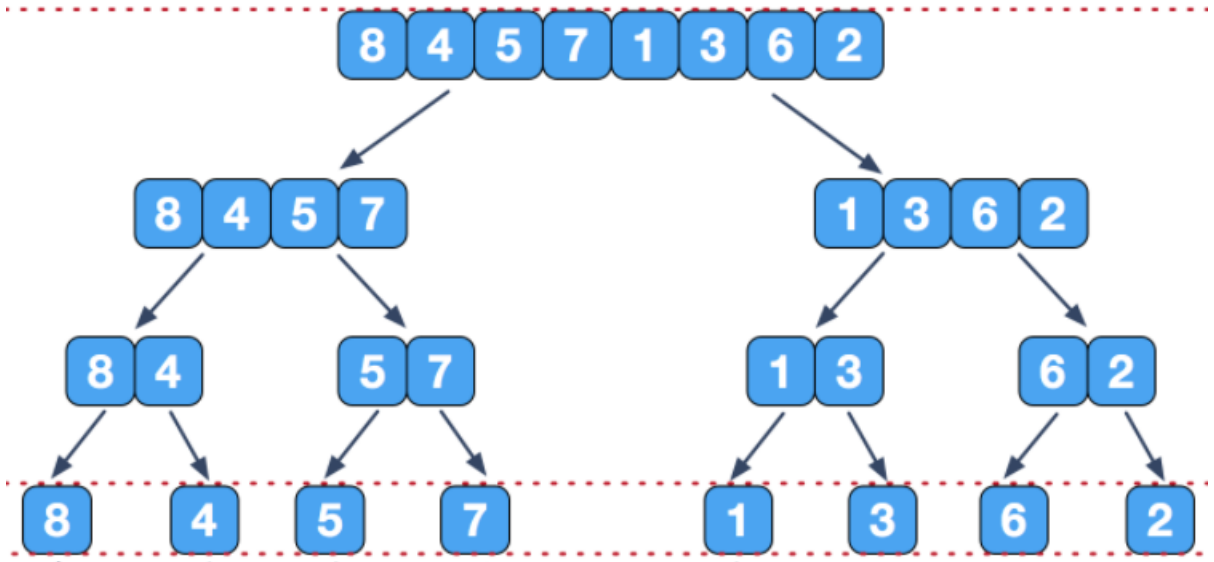
当m足够大时(仅剩一个数时), $\frac{n}{2^m} = 1$,求出 $m = \log_2 n$

$$f(n) = 2^m f\left(\frac{n}{2^m}\right) + mn = 2^{\log_2 n} \times f(1) + n \cdot \log_2 n$$

其中$f(1) = 0$,即$f(n) = n \cdot \log_2 n$

归并排序时间复杂度

不想推式子怎么办？我们直接来可视化时间复杂度：



如图是一个分治排序的递归过程，考虑“一层层”地来计算复杂度。

容易发现,每一层的长度和正好是 n。

一共有多少层呢？每过一层长度都会除一半，所以只有 $\log_2 n$ 层。

综上所述，分治排序的时间复杂度是 $O(n \cdot \log_2 n)$，而且这个复杂度是“满”的。

#322、逆序对数

题目描述

给你一个 n 个数的数组，逆序对定义为:

存在两个整数 $i < j$ 使得 $a_i > a_j$

你需要求出逆序对的个数

输入

第一行 n，代表数组长度

第二行 n 个数，代表 $a_i(1 \leq i \leq n)$

输出

输出逆序对数

数据规模

对于 10% 的数据，$n \leq 100, 1 \leq a_i \leq 100$

对于 40% 的数据，$n \leq 10000, 1 \leq a_i \leq 10^9$

对于 100% 的数据，$n \leq 100000, 1 \leq a_i \leq 10^9$

输入样例

```
4
4 3 2 1
```

输出样例

```
6
```

冒泡排序中

每进行一次交换,实际上减少了一个逆序对

进行冒泡排序,统计交换次数

时间复杂度$O(n^2)$,时间复杂度太高无法通过

#322、逆序对数

在归并排序中,对于每一次合并都会得到两个有序的数组$a[L \sim mid]$、$a[mid+1 \sim R]$。

其中$a[mid+1 \sim R]$中的元素下标都比$a[L \sim mid]$ 大

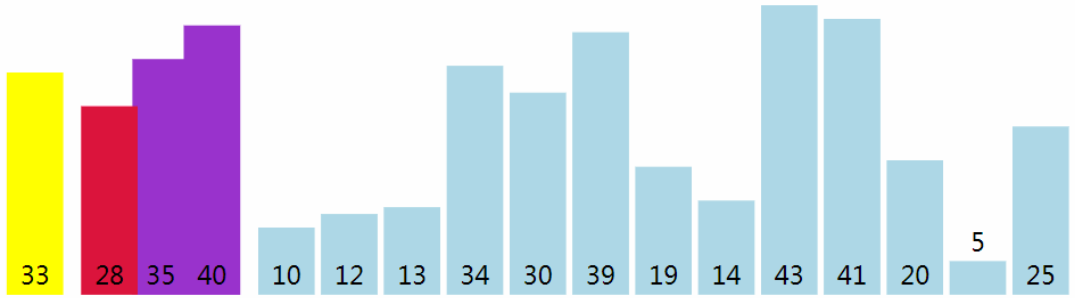
当比较后选择$a[mid+1 \sim R]$中的元素时,$a[L \sim mid]$中剩余元素都可以和其构成逆序对

时间复杂度$O(n\log n)$

对数据离散化后,也可以使用树状数组/线段树等树结构求解


```
int a[100001], helper[100001];
long long n, ans = 0;
void mergeSort(int l, int r)
{
  if (l >= r)
    return;
  int mid = l + r >> 1;
  mergeSort(l, mid),  mergeSort(mid + 1, r);
  int x = l, y = mid + 1, index = 0;
  while (x <= mid || y <= r)
  {
    if ((x <= mid && a[x] <= a[y]) || y > r)
      tmp[index++] = a[x++];
    else
    {
      ans += (mid - x + 1); //累加左半边剩余个数
      helper[index++] = a[y++];
    }
  }
  for (int i = l; i <= r; i++)
    a[i] = helper[i - l];
}
```

快速排序



Quick Sort

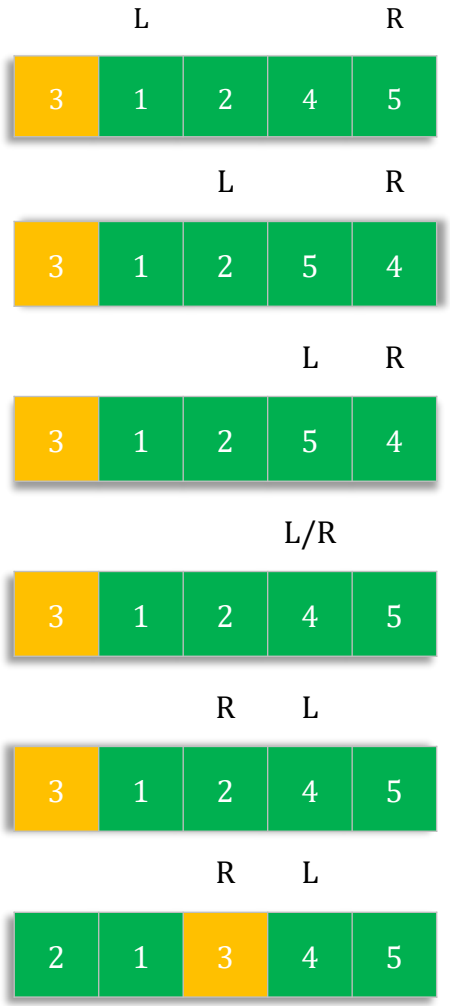
```
28 < 33 (pivot) is true. Swapping index 3 (value = {val}) with element at
storeIndex 1 (value = 40). (Value of storeIndex = {2}).

for each (unsorted) partition
set first element as pivot
  storeIndex = pivotIndex + 1
  for i = pivotIndex + 1 to rightmostIndex
    if element[i] < element[pivot]
      swap(i, storeIndex); storeIndex++
  swap(pivot, storeIndex - 1)
```

快速排序（$Quick\ Sort$）是对冒泡排序的一种改进,由C. A. R. Hoare在1960年提出。

它的基本思想是

通过一趟排序将要排序的数据分割成独立的两部分,其中一部分的所有数据都比另外一部分的所有数据都要小,然后再按此方法对这两部分数据分别进行快速排序,整个排序过程可以递归进行,以此达到整个数据变成有序序列。

快速排序




```
int partition(int l, int r)
{
  int pivot = a[l], L = l + 1, R = r;
  while (L <= R)
    if (a[L] <= pivot)
      L++;
    else
      swap(a[L], a[R--]);
  swap(a[l], a[R]);
  return R;
}
void quickSort(int l, int r)
{
  if (l >= r)
    return;
  int p = partition(l, r); //得到主元位置
  quickSort(l, p - 1);  //递归处理 l ~ p-1 的区间
  quickSort(p + 1, r);  //递归处理 p+1 ~ r 的区间
}
```

快速排序

若一次能将区间平分成两个部分

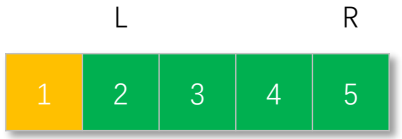
参考归并排序,平均时间复杂度为$O(n\log n)$

当出现已经有序时,上述写法每次将分为长度1和$n-1$两个区间(退化成冒泡排序$O(n^2)$)

当整个数组都是相同值时,退化成$O(n^2)$

快速排序的一些优化

- 选取第一个、最后一个以及中间的元素中的中位数的方法来选择两个子序列的主元。避免升序/降序带来的退化
- 当序列较短时,使用 插入排序 效率更高
- 每趟排序后,将与主元相等的元素聚集在分界元素周围,避免大部分元素都相等带来的退化



快速排序

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, a[100001];
void quickSort(int l, int r)
{
  if (l >= r)
    return;
  int p = a[l + r >> 1], i = l, j = r;
  while (i <= j)
  {
    while (a[i] < p)
      i++;
    while (a[j] > p)
      j--;
    if (i <= j)
      swap(a[i++], a[j--]);
  }
  quickSort(l, j), quickSort(i, r);
}
int main()
{
  scanf("%d", &n);
  for (int i = 1; i <= n; i++)
    scanf("%d", a + i);
  random_shuffle(a + 1, a + 1 + n); // 随机打乱
  quickSort(1, n);
  for (int i = 1; i <= n; i++)
    printf("%d ", a[i]);
  return 0;
}
```

思路源于 荷兰国旗问题

$a[l \sim j]$都不超过主元

$a[i \sim r]$都不小于主元

$a[j+1 \sim i-1]$都等于主元

这样避免大量重复的极端情况

对于升序或降序等情况,可以人为打乱顺序

#1028、快速排序

题目描述

利用快速排序算法将读入的 N 个数从小到大排序后输出。

快速排序是竞赛的必备算法之一。

对于快速排序不是很了解的同学可以自行上网查询相关资料，掌握后独立完成。（$C++$ 选手请不要试图使用 STL，虽然你可以使用 $sort$ 一遍过，但是你并没有掌握快速排序算法的精髓。）

输入格式：

第 1 行为一个正整数 N

第 2 行包含 N 个空格隔开的正整数 a_i，为你需要进行排序的数，数据保证了 A_i ,不超过 10^9。

输出格式：

将给定的 N 个数从小到大输出，一行一个数。

输入样例：

```
5
4 2 4 5 1
```

输出样例：

```
1
2
4
4
5
```

说明

对于 100% 的数据，有 $N \leq 100000$。

#2699、快速选择

题目描述

给定一个长度为 n 的整数数列，以及一个整数 k，请用快速选择算法求出数列从小到大排序后的第 k 个数。

输入格式

第一行包含两个整数 n 和 k。

第二行包含 n 个整数(所有整数均在 $-10^9 \sim 10^9$ 范围内),表示整数数列。

输出格式

输出一个整数，表示数列的第 k 小数。

输入样例

```
5 3
2 4 1 5 3
```

输出样例

```
3
```

数据范围

对于全部的数据 $1 \leq n \leq 7 \times 10^6, 1 \leq k \leq n$

升序排序后输出$a[k]$

时间复杂度$O(n\log n)$,时间复杂度太高无法通过

#2699、快速选择

```cpp
#include <bits/stdc++.h>
using namespace std;
int n, a[7000005], k;
void findKth(int l, int r, int k)
{
  if (l >= r)
    return;
  int p = a[l + r >> 1], i = l, j = r;
  while (i <= j)
  {
    while (a[i] < p)
      i++;
    while (a[j] > p)
      j--;
    if (i <= j)
      swap(a[i++], a[j--]);
  }
  if ((j - l + 1) >= k)
    return findKth(l, j, k);
  return findKth(i, r, k - (j - l + 1));
}
int main()
{
  scanf("%d%d", &n, &k);
  for (int i = 1; i <= n; i++)
    scanf("%d", a + i);
  findKth(1, n, k);
  printf("%d", a[k]);
  return 0;
}
```

可以借助快速排序的思想解决这个问题

考虑快速排序的划分过程,在快速排序的**划分**结束后

数组 $a[L \sim R]$ 被分成了$a[L \sim p]$ 和 $a[p+1 \sim R]$,p为主元的位置(排序后的位置)。

此时可以按照左边元素的个数$p-L+1$和k 的大小关系来判断是目标在左边还是在右边

递归地求解,平均时间复杂度$O(n)$

排序算法分析

算法	最好时间复杂度	最坏时间复杂度	空间复杂度	稳定性
冒泡排序	$O(n)$	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
插入排序	$O(n)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log n)$	$O(n\log n)$	$O(1)$	不稳定
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n)$	稳定
快速排序	$O(n\log n)$	$O(n^2)$	$O(\log n) \sim O(n)$	不稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(k)$	稳定

稳定

如果a原本在b前面,而$a = b$,排序之后a仍然在b的前面

不稳定

如果a原本在b的前面,而$a = b$,排序之后 a 可能会出现在 b 的后面

谢谢观看