



程序设计与算法(二)

算法基础

郭 炜

微信公众号



微博: <http://weibo.com/guoweiofpku>

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄



北京大学
PEKING UNIVERSITY

信息科学技术学院

配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在
<http://openjudge.cn>
“百练”组进行搜索即可提交





分治算法



北京大学
PEKING UNIVERSITY

信息科学技术学院

分治的基本概念



贵州黄果树瀑布

分治的基本概念

- 把一个任务，分成形式和原任务相同，但规模更小的几个部分任务（通常是两个部分），分别完成，或只需要选一部完成。然后再处理完成后的这一个或几个部分的结果，实现整个任务的完成。

分治的生活实例 --称假币

- 16硬币，有可能有1枚假币，假币比真币轻。有一架天平，用最少称量次数确定有没有假币，若有的话，假币是哪一枚。

分治的生活实例 – 称假币

- 8 – 8 一称，发现无假币，或假币所在的那8枚
- 4 – 4 一称
- 2 – 2 一称
- 1 – 1 一称



北京大学
PEKING UNIVERSITY

信息科学技术学院

归并排序



挪威天池

分治的典型应用：归并排序

归并排序是需要格外的存储空间的

复杂度： $O(n(\log(n)))$

- 数组排序任务可以如下完成：
 - 1) 把前半部分排序
 - 2) 把后半部分排序
 - 3) 把两半归并到一个新的有序数组，然后再拷贝回原数组，排序完成。

分治的典型应用：归并排序

```
#include <iostream>
using namespace std;
void Merge(int a[],int s,int m, int e,int tmp[])
{ //将数组a的局部a[s,m]和a[m+1,e]合并到tmp,并保证tmp有序, 然后再拷贝回a[s,m]
  //归并操作时间复杂度:  $O(e-m+1)$ ,即 $O(n)$ 
    int pb = 0;
    int p1 = s,p2 = m+1;
    while( p1 <= m && p2 <= e) {
        if( a[p1] < a[p2])
            tmp[pb++] = a[p1++];
        else
            tmp[pb++] = a[p2++];
    }
```

```

while( p1 <= m)
    tmp[pb++] = a[p1++];
while( p2 <= e)
    tmp[pb++] = a[p2++];
for(int i = 0; i < e-s+1; ++i)
    a[s+i] = tmp[i];
}

void MergeSort(int a[], int s, int e, int tmp[])
{
    if( s < e) {
        int m = s + (e-s)/2;
        MergeSort(a, s, m, tmp);
        MergeSort(a, m+1, e, tmp);
        Merge(a, s, m, e, tmp);
    }
}

```

```
int a[10] = { 13,27,19,2,8,12,2,8,30,89};  
int b[10];  
int main()  
{  
    int size = sizeof(a)/sizeof(int);  
    MergeSort(a,0,size-1,b);  
    for(int i = 0;i < size; ++i)  
        cout << a[i] << ",";  
    cout << endl;  
    return 0;  
}
```

1	4	9	12
---	---	---	----

2	5	8	13
---	---	---	----

--	--	--	--	--	--	--	--

归并排序的时间复杂度

对n个元素进行排序的时间:

$$\begin{aligned}T(n) &= 2 * T(n/2) + a * n \\&= 2 * (2 * T(n/4) + a * n/2) + a * n \\&= 4 * T(n/4) + 2a * n \\&= 4 * (2 * T(n/8) + a * n/4) + 2 * a * n \\&= 8 * T(n/8) + 3 * a * n \\&\dots \\&= 2^k * T(n/2^k) + k * a * n\end{aligned}$$

(a是常数,具体多少不重要)

一直做到 $n/2^k = 1$ (此时 $k = \log_2 n$),

$$\begin{aligned}T(n) &= 2^k * T(1) + k * a * n = 2^k * T(1) + k * a * n = 2^k + k * a * n \\&= n + a * (\log_2 n) * n\end{aligned}$$

复杂度 $O(n \log n)$



北京大学
PEKING UNIVERSITY

信息科学技术学院

快速排序



挪威奥勒松

分治的典型应用：快速排序

- 数组排序任务可以如下完成：
 - 1) 设 $k=a[0]$, 将 k 挪到适当位置, 使得比 k 小的元素都在 k 左边,比 k 大的元素都在 k 右边, 和 k 相等的, 不关心在 k 左右出现均可 ($O(n)$ 时间完成)
 - 2) 把 k 左边的部分快速排序
 - 3) 把 k 右边的部分快速排序

$K = 7$

i							j
7	1	3	8	12	11	2	9

$K = 7$

i						j	
7	1	3	8	12	11	2	9

$K = 7$

i						j	
2	1	3	8	12	11	7	9

$K = 7$

i		j					
2	1	3	8	12	11	7	9

$K = 7$

i			j				
2	1	3	8	12	11	7	9

$K = 7$

i				j			
2	1	3	8	12	11	7	9

$K = 7$

i				j			
2	1	3	7	12	11	8	9

$K = 7$

<i>i</i>				<i>j</i>			
2	1	3	7	12	11	8	9

$K = 7$

i j

2	1	3	7	12	11	8	9
---	---	---	---	----	----	---	---

分治的典型应用：快速排序

```
#include <iostream>
using namespace std;
void swap(int & a,int & b) //交换变量a,b值
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
void QuickSort(int a[],int s,int e)
{
    if( s >= e)
        return;
    int k = a[s];
    int i = s,j = e;
    while( i != j ) {
        while( j > i && a[j] >= k )
            --j;
        swap(a[i],a[j]);
        while( i < j && a[i] <= k )
            ++i;
        swap(a[i],a[j]);
    } //处理完后, a[i] = k
    QuickSort(a,s,i-1);
    QuickSort(a,i+1,e);
}
```

```
int a[] = { 93,27,30,2,8,12,2,8,30,89};  
int main()  
{  
    int size = sizeof(a)/sizeof(int);  
    QuickSort(a,0,size-1);  
    for(int i = 0;i < size; ++i)  
        cout << a[i] << ",";  
    cout << endl;  
    return 0;  
}
```



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题 输出前 m 大的数



冰岛火山口湖

例题：输出前m大的数

描述

给定一个数组包含n个元素，统计前m大的数并且把这m个数从大到小输出。

输入

第一行包含一个整数n，表示数组的大小。 $n < 100000$ 。

第二行包含n个整数，表示数组的元素，整数之间以一个空格分开。
每个整数的绝对值不超过100000000。

第三行包含一个整数m。 $m < n$ 。

输出

从大到小输出前m大的数，每个数一行。

例题：输出前 m 大的数

排序后再输出，复杂度 $O(n\log n)$

用分治处理：复杂度 $O(n+m\log m)$

思路：把前 m 大的都弄到数组最右边，然后对这最右边 m 个元素排序，再输出

关键： $O(n)$ 时间内实现把前 m 大的都弄到数组最右边

例题：输出前m大的数

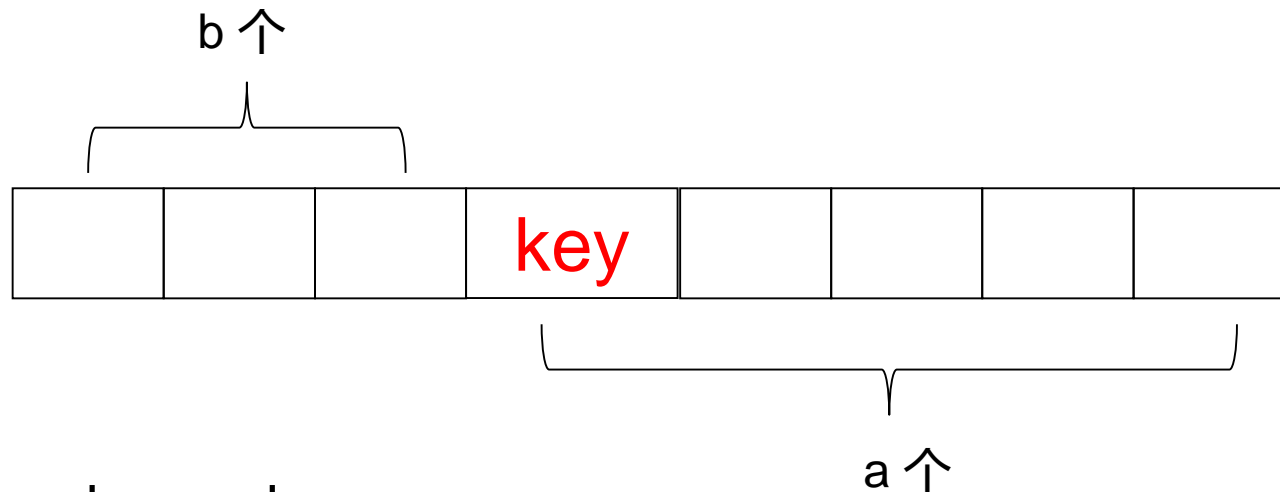
引入操作 `arrangeRight(k)`：把数组 (或数组的一部分) 前k大的都弄到最右边

如何将前k大的都弄到最右边

- 1) 设 $key = a[0]$, 将key挪到适当位置, 使得比key小的元素都在key左边, 比key大的元素都在key右边 (线性时间完成)
- 2) 选择数组的**前部或后部**再进行 `arrangeRight`操作

例题：输出前m大的数

2) 选择数组的**前部或后部**再进行 `arrangeRight`操作



$a = k$

done

$a > k$

对最右边 $a-1$ 个元素再进行 `arrangeRigth(k)`

$a < k$

对左边 b 个元素再进行 `arrangeRight(k-a)`

例题：输出前m大的数

将前m大的都弄到数组最右边的时间：

$$\begin{aligned}T(n) &= T(n/2) + a*n \\&= T(n/4) + a*n/2 + a*n \\&= T(n/8) + a*n/4 + a*n/2 + a*n \\&= \dots \\&= T(1) + \dots + a*n/8 + a*n/4 + a*n/2 + a*n \\&< 2*a*n\end{aligned}$$

即 $O(n)$



北京大学
PEKING UNIVERSITY

信息科学技术学院

例题 求排列的逆序数



冰岛维克镇黑沙滩

例题：求排列的逆序数

考虑 $1, 2, \dots, n$ ($n \leq 100000$)的排列 i_1, i_2, \dots, i_n ，如果其中存在 j, k ，满足 $j < k$ 且 $i_j > i_k$ ，那么就称 (i_j, i_k) 是这个排列的一个逆序。

一个排列含有逆序的个数称为这个排列的逆序数。例如排列 263451 含有8个逆序 $(2, 1), (6, 3), (6, 4), (6, 5), (6, 1), (3, 1), (4, 1), (5, 1)$ ，因此该排列的逆序数就是8。

现给定 $1, 2, \dots, n$ 的一个排列，求它的逆序数。

例题：求排列的逆序数

笨办法： $O(n^2)$

分治 $O(n\log n)$:

- 1) 将数组分成两半，分别求出左半边的逆序数和右半边的逆序数
- 2) 再算有多少逆序是由左半边取一个数和右半边取一个数构成(要求 $O(n)$ 实现)

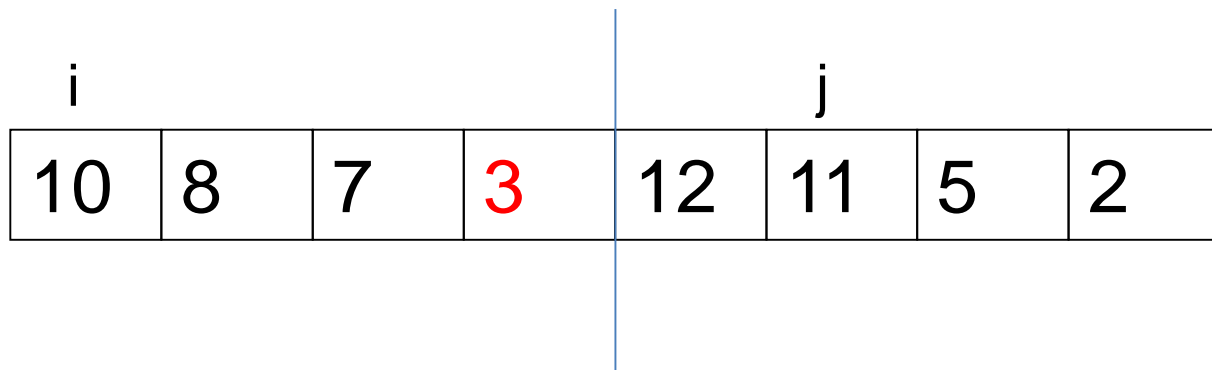
例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数

i				j			
10	8	7	3	12	11	5	2

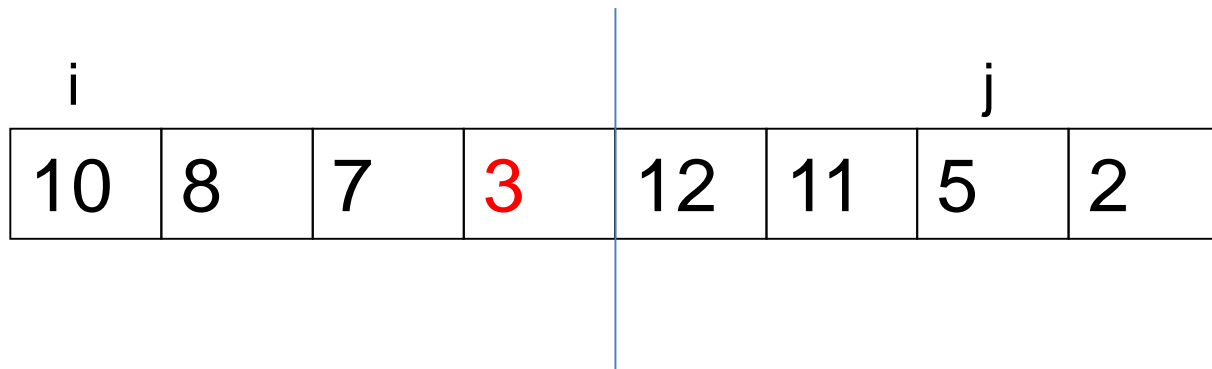
例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数



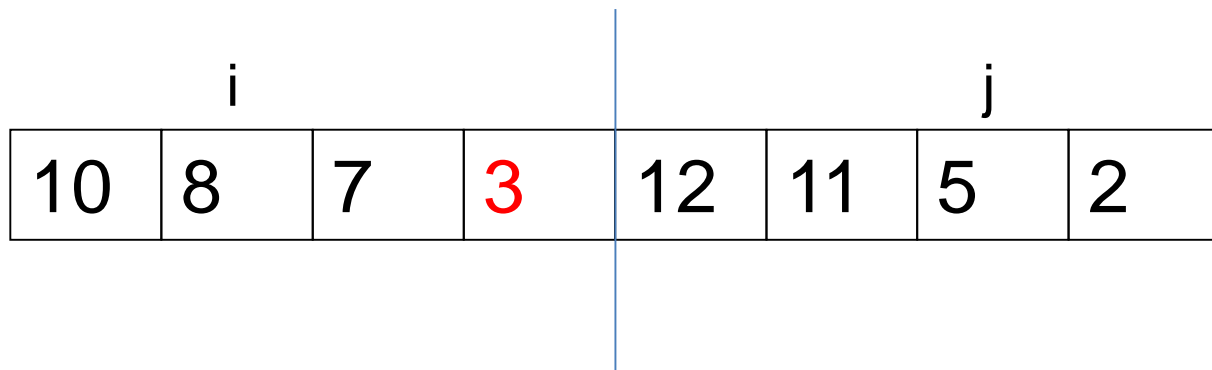
例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数



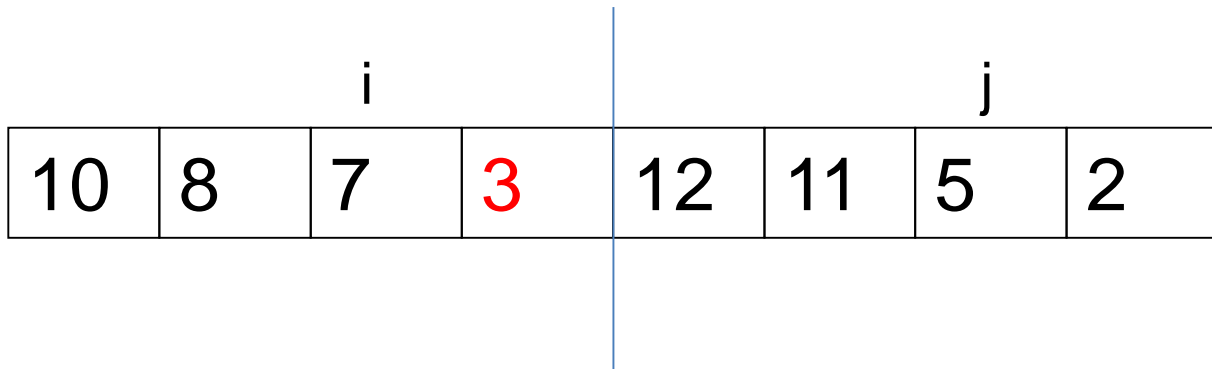
例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数



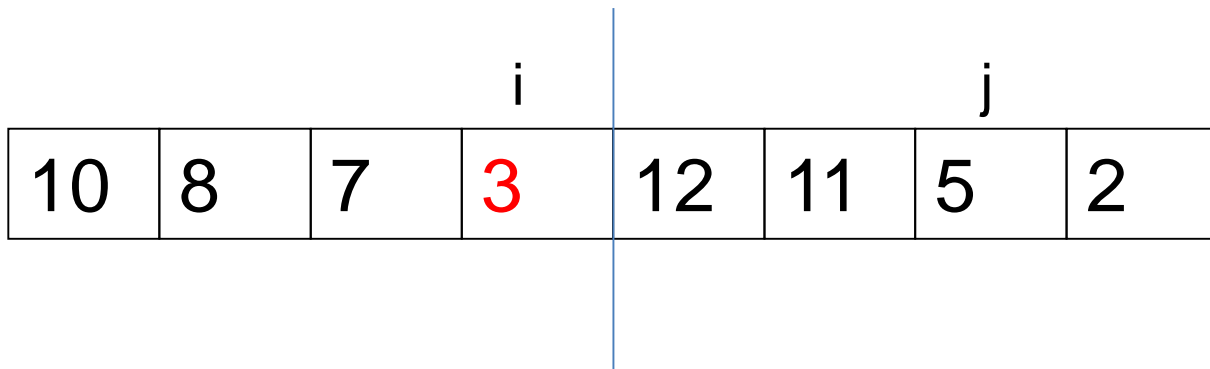
例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数



例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数



例题：求排列的逆序数

2) 的关键：左半边和右半边都是排好序的。比如，都是从大到小排序的。这样，左右半边只需要从头到尾各扫一遍，就可以找出由两边各取一个数构成的逆序个数

<i>i</i>				<i>j</i>			
10	8	7	3	12	11	5	2

例题：求排列的逆序数

总结：

由归并排序改进得到，加上计算逆序的步骤

MergeSortAndCount: 归并排序并计算逆序数



北京大学
PEKING UNIVERSITY

信息科学技术学院

快速幂



冰岛天然玄武岩长城

快速幂

```
int Pow(int a,int b)
{ //快速求 $a^b$  , 复杂度  $\log(b)$ 
    if(b == 0)
        return 1;
    if(b & 1) { //b是奇数
        return a * Pow(a,b-1);
    }
    else {
        int t = Pow(a,b/2);
        return t * t;
    }
}
```

快速幂

```
int Pow(int a,int b)
{
    //快速求 $a^b$  , 复杂度  $\log(b)$ 
    int result = 1;
    int base = a;
    while(b) {
        if( b & 1)
            result *= base;
        base *= base;
        b >>= 1;
    }
    return result;
}
```