递归与分治

算法设计与分析

武汉大学
国家网络安全学院
李雨晴

递归的概念

例1　阶乘函数

阶乘函数可递归地定义为：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & n > 0 \end{cases}$$

边界条件

递归方程

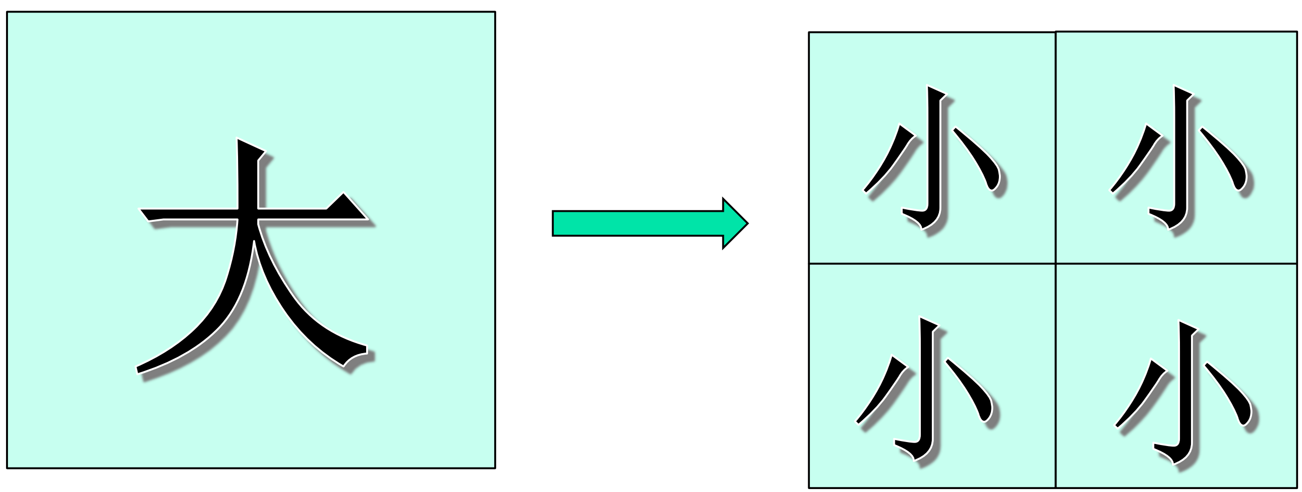
边界条件与递归方程是递归函数的二个要素，递归函数只有具备了这两个要素，才能在有限次计算后得出结果。

递归小结

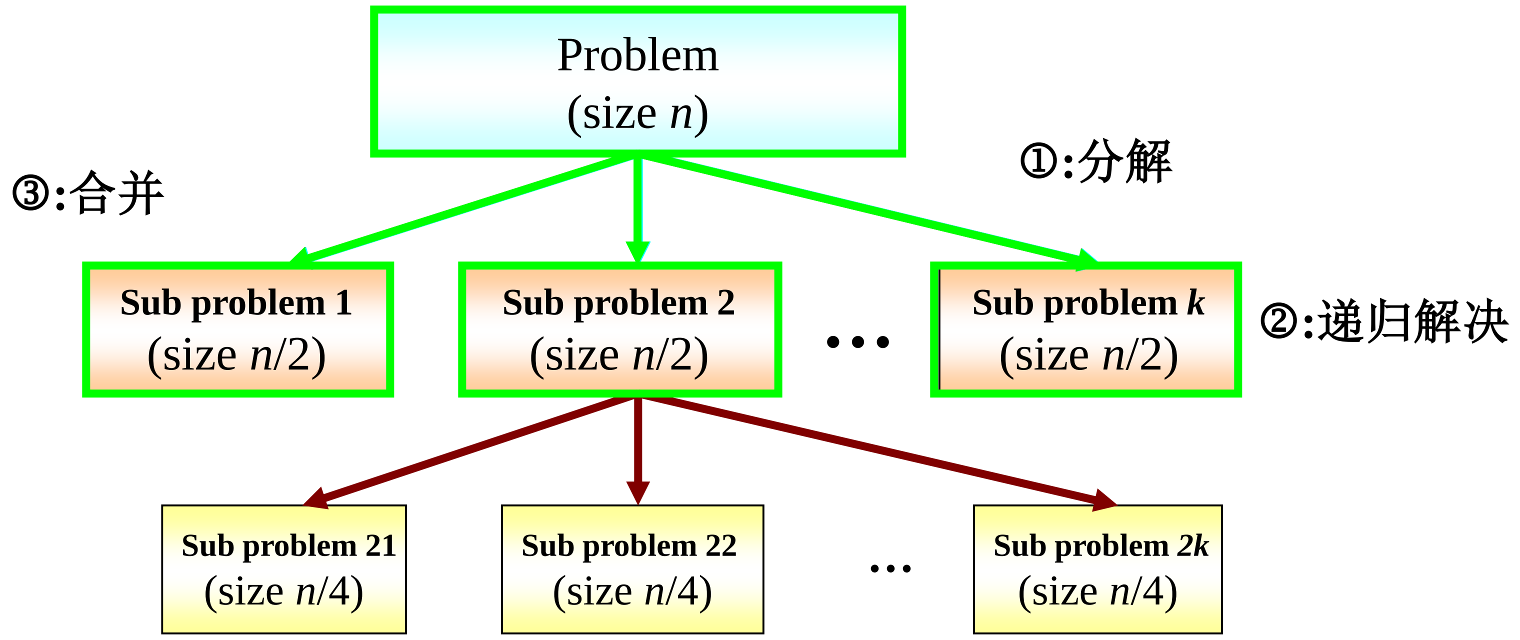
- 优点： 结构清晰，可读性强，而且容易用数学归纳法来证明算法的正确性，因此它为设计算法、调试程序带来很大方便。
- 缺点： 递归算法的运行效率较低，无论是耗费的计算时间还是占用的存储空间都比非递归算法要多。

分治策略
Divide and Conquer

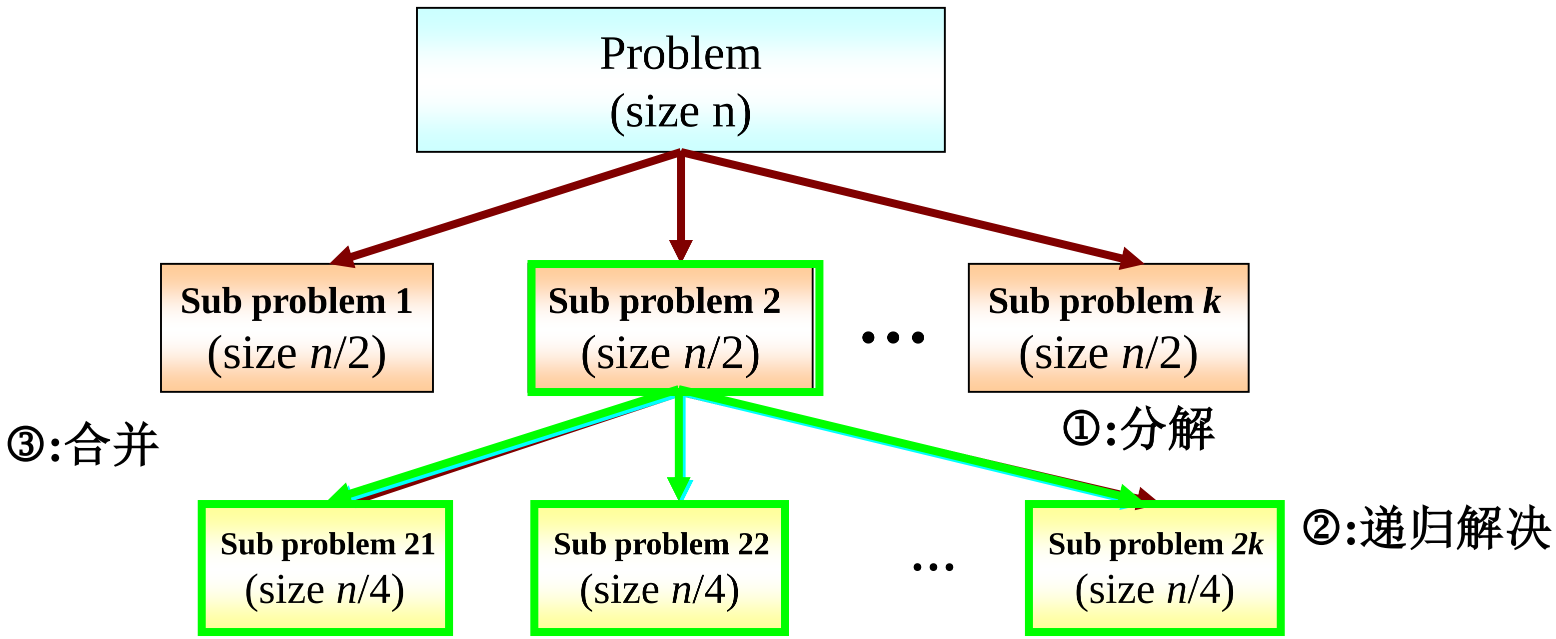
分治策略的思想



分治策略的思想


Problem
(size n)
①:分解
③:合并
Sub problem 1
(size n/2)
Sub problem 2
(size n/2)
…
Sub problem k
(size n/2)
②:递归解决
Sub problem 21
(size n/4)
Sub problem 22
(size n/4)
…
Sub problem 2k
(size n/4)

分治策略的思想


Problem
(size n)
Sub problem 1
(size n/2)
Sub problem 2
(size n/2)
…
Sub problem k
(size n/2)
①:分解
③:合并
Sub problem 21
(size n/4)
Sub problem 22
(size n/4)
…
Sub problem 2k
(size n/4)
②:递归解决

分治策略的思想

- **划分**：把规模较大的问题(n)分解为若干(通常为2)个规模较小的子问题(<n)， 这些子问题相互独立且与原问题同类; (该子问题的规模减小到一定的程度就可以容易地解决)
- **治理**：依次求出这些子问题的解
- **组合**：把这些子问题的解组合起来得到原问题的解。

由于子问题与原问题是同类的，故分治法可以很自然地应用递归。

分治算法形式

- 如果实例I规律是小的，则直接求解，否则继续做下一步
- 把实例I分割成p个大小几乎相同的子实例I_1, I_2... I_p，对每个子实例I_j, 1≤j≤p，递归调用算法，并得到个p部分解
- 组合 p个部分解的结果得到原实例I的解，返回实例 I的解

分治算法

- 引例：二分搜索
- 合并排序
- 寻找中项和第k小元素
- 快速排序
- 矩阵乘法
- 大整数相乘

自底向上合并排序

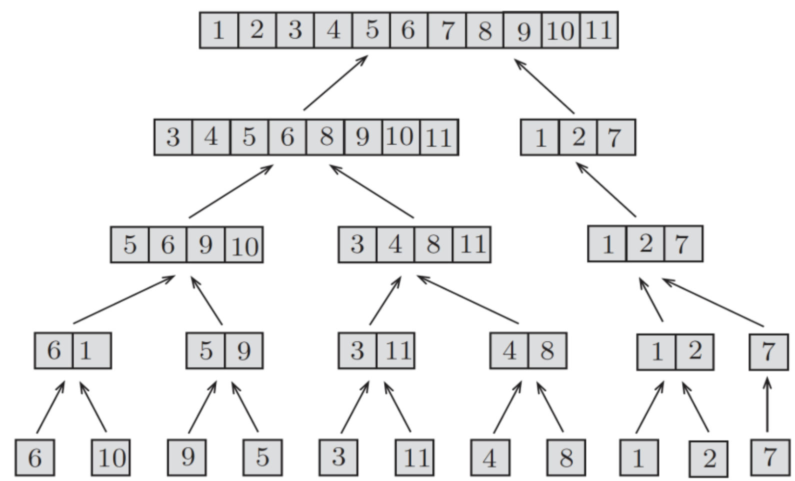
- n不是2的幂示例

算法 1.6 BOTTOMUPSORT

输入： n 个元素的数组 $A[1\cdots n]$。

输出： 按非降序排列的数组 $A[1\cdots n]$。

```
t ← 1
while t < n
    s ← t;   t ← 2s;   i ← 0
    while i + t ≤ n
        MERGE (A, i + 1, i + s, i + t)
        i ← i + t
end while
    if i + s < n then MERGE (A, i + 1, i + s, n)
end while
```



合并Merge

```
Algorithm: MERGE(A, p, q, r)
输入：数组A[p...q]和A[q+1...r], 各自按升序排列
输出：将A[p...q]和A[q+1...r]合并成一个升序排序的新数组
1. s←p; t←q+1; k←p; {s, t, p 分别指向A[p...q],A[q+1...r]和B}
2. while s≤q and t≤r
3.    if A[s] ≤A[t] then
4.        B[k]←A[s]
5.        s ←s+1
6.    else
7.        B[k]←A[t]
8.        t←t+1
9.    end if
10. k←k+1
11.end while
12.   if s=q+1 then B[k...r] ←A[t...r]
13.   else B[k...r] ←A[s...q]
14. end if
15. A[p...q] ←B[p...q]
```

合并排序Mergesort

- Using 划分-治理-组合， we can obtain a merge-sort algorithm
  - 划分：将n个元素的数组分成两个元素个数为n/2的子数组.
  - 治理：递归的解决子数组.
  - 组合：将两个排序好的子数组合并成一个数组.

合并排序Mergesort

例：给定数组A [1…8]=

8	4	3	1	6	2	9	7

1. 将其分分成左右两个子数组:

8	4	3	1

6	2	9	7

2. 对子数组进行排序(可采用任何排序方法) 。

3. 对排序后的子数组进行合并 :两个已排序的子数组用A[p…q]和A[q+1…r]表示. 设两个指针s和t，初始时各自指向A[p]和A[q+1],再设一空数组B[p…q, q+1…r] 做暂存器，比较元素A[s]和A[t], 将较小者添加到B , 然后移动指针,

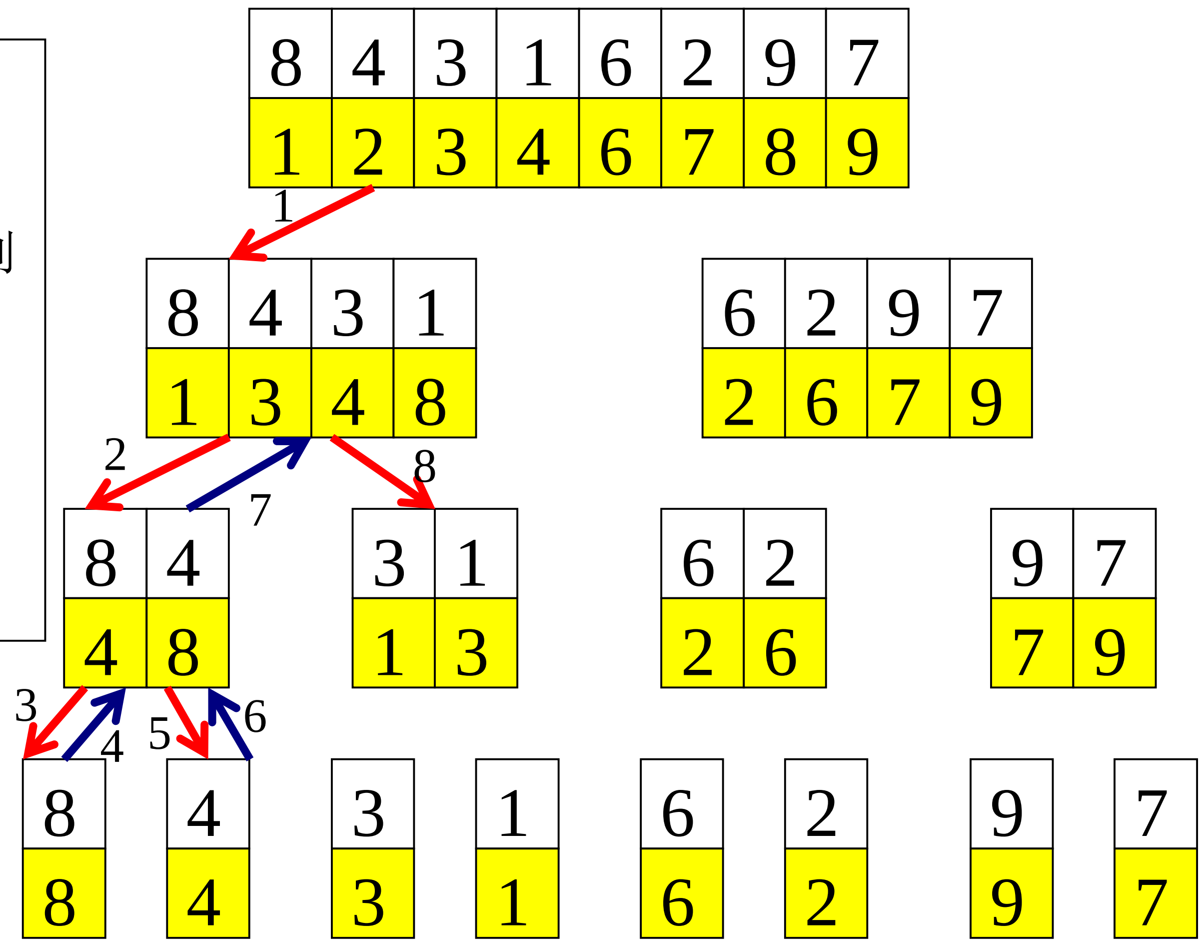
   若A[s]较小，则s+1，否则t+1,

   直到s=q+1 或 t=r+1 为止

   将剩余元素A[t…r] 或 A[s…q] 拷贝到数组B, 然后令A←B.

合并排序Mergesort

Algorithm: MERGESORT(A, low, high)
输入：待排序数组A[low,...high]
输出：A[low…high]按非降序排列
1. if low<high then
2. mid←⌊(low+high)/2⌋
3. MERGESORT(A, low, mid)
4. MERGESORT(A, mid+1, high)
5. MERGE(A, low, mid, high)
6. end if



调用顺序：前序遍历
处理顺序：后序遍历

时间复杂度分析

- 如果两数组大小分别为$\lfloor n/2 \rfloor$和$\lceil n/2 \rceil$，则比较的次数是$\lfloor n/2 \rfloor$到n-1

最小比较次数

$$C(n)=\begin{cases}0 & if\ \ n=1\\ 2C(n/2)+n/2 & if\ \ n\ge 2\end{cases} \Longrightarrow C(n)=\frac{n\log n}{2}$$

最大比较次数

$$C(n)=\begin{cases}0 & if\ \ n=1\\ 2C(n/2)+n-1 & if\ \ n\ge 2\end{cases} \Longrightarrow C(n)=n\log n-n+1$$

较大比较次数

$$
\begin{aligned}
C(n) &= 2C(n/2) + n - 1 \\
&= 2(2C(n/2^2) + n/2 - 1) + n - 1 \\
&= 2^2 C(n/2^2) + n - 2 + n - 1 \\
&= 2^2 C(n/2^2) + 2n - 2 - 1 \\
&\vdots \\
&= 2^k C(n/2^k) + kn - 2^{k-1} - 2^{k-2} - \cdots - 2 - 1 \\
&= 2^k C(1) + kn - \sum_{j=0}^{k-1} 2^j \\
&= 2^k \times 0 + kn - (2^k - 1) \quad (\text{等式 } 2.10) \\
&= kn - 2^k + 1 \\
&= n \log n - n + 1
\end{aligned}
$$

合并排序算法复杂度

- 算法mergesort对于一个n个元素的数组排序所需要的时间是$\Theta(n \log n)$ 空间是$\Theta(n)$

寻找中项和第*k*小元素

- 给定已排好序(非降序)的数组A[1…n],中项是指其“中间”元素。若n为奇数，则中项为数组中的第(n+1)/2个元素；若n为偶数，则存在两个中间元素，分别为第n/2和第n/2+1个元素，在这种情形下，我们取第n/2个元素作为中项；综上，中项为第$\lceil n/2 \rceil$个最小元素。

寻找中项和第*k*小元素

- 寻找中项的一个直接的方法：先排序，后取中项。显然，该方法的时间复杂度至少为Ω(nlogn)。能否找到更为高效的方法？
- 寻找中项是寻找第k小元素的一个特列。如果能解决寻找第k小元素的问题，那么当k=⌈n/2⌉时，解决的就是寻找中项问题。
- 回顾二分搜索：以中间元素为基准抛弃部分元素，不断减小问题规模。

寻找中项和第k小元素

- 数组A[1,…,n]中第k小元素的特点（充要条件）

对任意$x \in A$，定义三个集合

$A_1=\{a \mid a<x\}$,　　$A_2=\{a \mid a=x\}$,

$A_3=\{a \mid a>x\}$

那么，x为第k小元素当前仅当$|A_1|<k$且$|A_1|+|A_2| \geq k$

寻找中项和第k小元素

- 解决思路：
  - 如果数组A中元素的个数(问题规模)小于一个阈值，那么采用直接的方法(先排序，后查找)寻找第k小元素
  - 否则，
    - (1)找出一个候选值mm
    - (2) 求出mm对应的A_1, A_2, 和A_3
    - (3) 判断mm是否满足第k小元素的条件，如果否，继续在下一个候选值进行上述操作：三种情况

$$|A_1| \geqslant k$$

$$|A_1| < k \text{ and } |A_1| + |A_2| \geqslant k$$

$$|A_1| + |A_2| < k$$

寻找中项和第k小元素

- 解决思路：

(1)找出一个候选值*mm*

令 $q = \lfloor p/5 \rfloor$。将 A 分成 q 组，每组 5 个元素。如果 5 不整除 p，则排除剩余的元素。

将 q 组中的每一组单独排序，找出中项。所有中项的集合为 M。

然后，选出q个中项的中项

1 14	6 09	05	03	02	12	01	17	20	04	36
2 22	7 10	06	11	25	16	13	24	31	07	27
3 28	8 23	38	15	40	19	18	43	32	35	08
4 29	9 39	50	26	53	30	41	46	33	49	21
5 45	10 44	52	37	54	53	48	47	34	51	

寻找中项和第*k*小元素

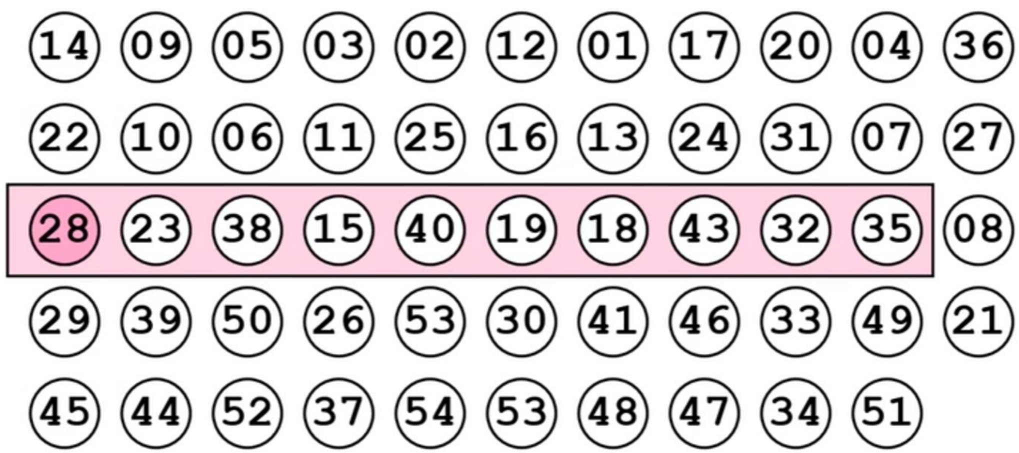
- 解决思路：

(1)找出一个候选值*mm*

令 $q=\lfloor p/5 \rfloor$。将 A 分成 q 组，每组 5 个元素。如果 5 不整除 p，则排除剩余的元素。

将 q 组中的每一组单独排序，找出中项。所有中项的集合为 M。

然后，选出q个中项的中项



寻找中项和第k小元素

算法 6.4 SELECT

输入： n 个元素的数组 $A[1\cdots n]$和整数 k，$1\leqslant k\leqslant n$。

输出： A 中的第 k 小元素。

1. *select*$(A,1,n,k)$

过程 *select*$(A, low, high, k)$

1. $p \leftarrow high - low + 1$
2. **if** $p<44$ **then** 将 A 排序 **return** $(A[k])$
3. 令 $q=\lfloor p/5 \rfloor$。将 A 分成 q 组，每组 5 个元素。如果 5 不整除 p，则排除剩余的元素。
4. 将 q 组中的每一组单独排序，找出中项。所有中项的集合为 M。
5. $mm \leftarrow select(M,1,q,\lceil q/2 \rceil)$ {mm 为中项集合的中项}
6. 将 $A[low\cdots high]$分成三组
   $A_1=\{a \mid a<mm\}$
   $A_2=\{a \mid a=mm\}$
   $A_3=\{a \mid a>mm\}$
7. **case**
   $|A_1|\geqslant k$: **return** $select(A_1,1,|A_1|,k)$
   $|A_1|+|A_2|\geqslant k$: **return** mm
   $|A_1|+|A_2|<k$: **return** $select(A_3,1,|A_3|,k-|A_1|-|A_2|)$
8. **end case**

为方便演示，设阈值为6。现要寻找下面数组A中的第13小元素：
A={8,33,17,51,57,49,35,11,25,37,14,3,2,13,52,12,6,29,32,54,5,16,22,23,7}

{8,33,17,51,57} {49,35,11,25,37} {14,3,2,13,52} {12,6,29,32,54} {5,16,22,23,7}

{8,17,33,51,57} {11,25,35,37,49 } {2, 3,13,14,52} {6, 12,29,32,54} {5,7,16,22,23}

M={33,35,13,29,16} mm=29

因为k=13<|A_1|=15

A_1={8,17,11,25,14,3,2,13,12,6,5,16,22,23,7} ~~A_2={29}~~ ~~A_3={33,51,57,49,35,37,52,32,54}~~

{8,17,11,25,14} {3,2,13,12,6} {5,16,22,23,7}

{8,11,14,17,25} {2,3,6,12,13} {5,7,16,22,23}

M={14,6,16} mm=14 →

M={14,6,16}　mm=14

A={8,17,11,25,14,3,2,13,12,6,5,16,22,23,7}

A_1={8, 11,~~3,2~~,13,12,6,5,7}

~~A_2={14}~~

A_3={17,25,16,22,23}

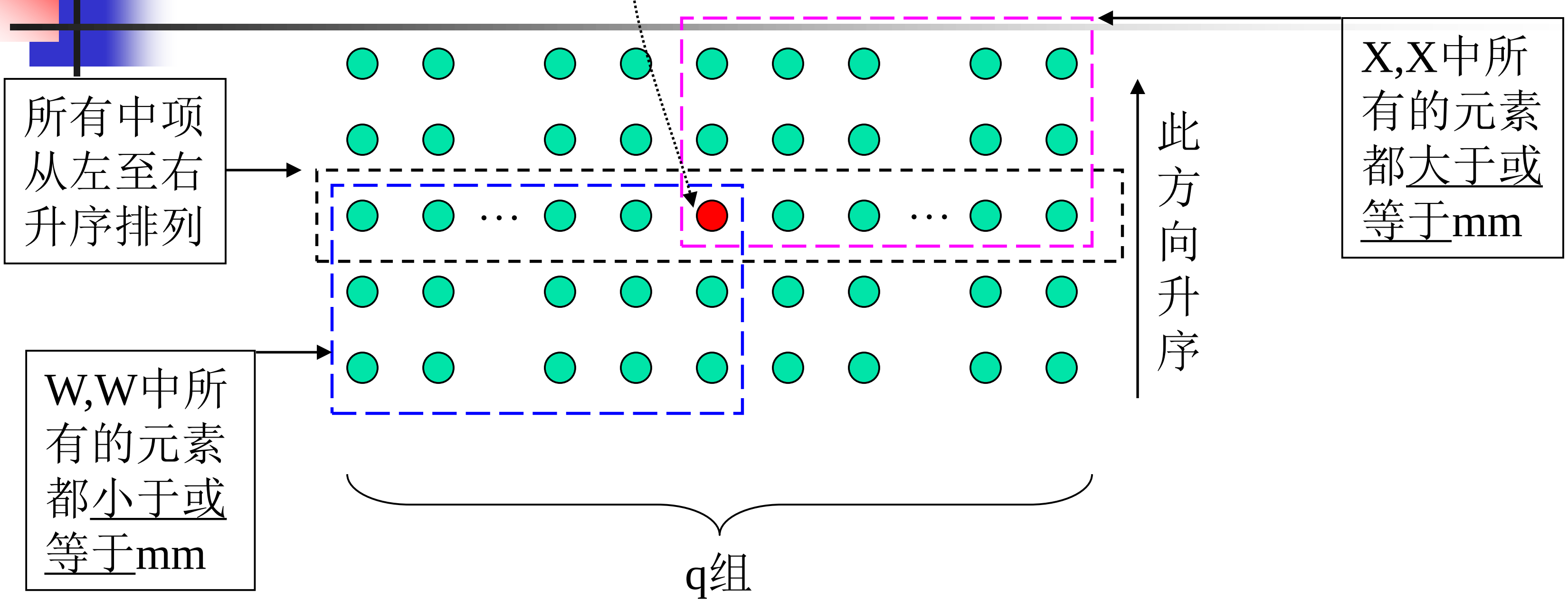
因为$|A_3|$<p=6　阈值

因为k=13>$|A_1|+|A_2|$＝9＋1=10

{16,17, 22,23,25,}

return A[3]　因为(3=13-9-1)

时间复杂性分析



- A_1^+表示A中<u>小于或等于</u>mm的元素集,A_1是A中<u>严格小于</u>mm的元素集。
- A_3^+表示A中<u>大于或等于</u>mm的元素集,A_3是A中<u>严格大于</u>mm的元素集。
- 因为 A_1^+至少与W同样大(为什么？)，所以$|A_1^+| \geq 3\lceil \lfloor n/5 \rfloor /2 \rceil \geq 3/2 \lfloor n/5 \rfloor$，所以，$|A_3| \leq n - 3/2 \lfloor n/5 \rfloor \leq n - 3/2 ((n-4)/5) = 0.7n+1.2$
- 由对称性，我们有$|A_1| \leq 0.7n+1.2$

时间复杂性分析

- 至此，我们为A1和A3中的元素个数建立了一个上界：即小于mm的元素个数或是大于mm的元素的个数均不超过0.7n+1.2。
- T(n)表示从n个元素中选择第k小元素所需要耗费的时间。
- Step 1，2耗费时间均为Θ(1)。
- Step 3耗费时间为Θ(n);Step 4耗费时间为Θ(n)。
- Step 5耗费时间为$T(\lfloor n/5 \rfloor)$。
- Step 6耗费时间为Θ(n)
- Step 7耗费时间至多为T(0.7n+1.2)。下面设法去掉其中的常数1.2。假设$0.7n+1.2 \le \lfloor 0.75n \rfloor$,那么当$0.7n+1.2 \le 0.75n-1$，即当$n \ge 44$时，$0.7n+1.2 \le \lfloor 0.75n \rfloor$成立。此时，Step 7耗费时间之多为$T(\lfloor 0.75n \rfloor)$.

$$T(n) \le \begin{cases} c & \text{if } n < 44 \\ T(\lfloor n/5 \rfloor) + T(\lfloor 3n/4 \rfloor) + \Theta(n) & \text{if } n \ge 44 \end{cases}$$

$$\big\downarrow$$

$$T(n) = \Theta(n)$$

定理2.7

定理2.7

定理 2.7 设 b, c_1, c_2 是非负常数，那么递推式

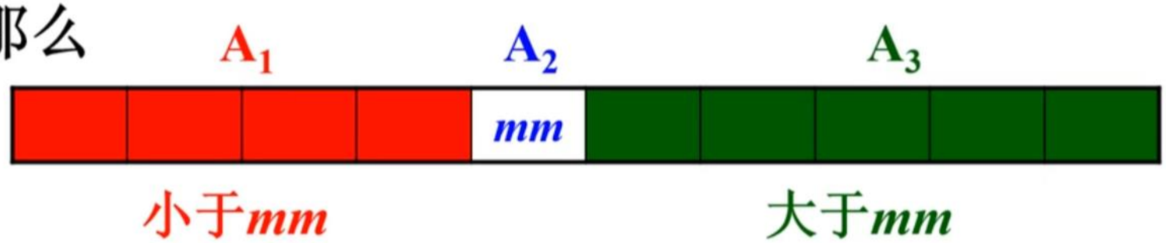
$$f(n)=\begin{cases}0 & 若\ n=0\\ b & 若\ n=1\\ f(\lfloor c_1 n\rfloor)+f(\lfloor c_2 n\rfloor)+bn & 若\ n\geqslant 2\end{cases}$$

的解是

$$f(n)=\begin{cases}O(n\log n) & 若\ c_1+c_2=1\\ \Theta(n) & 若\ c_1+c_2<1\end{cases}$$

再看一下A1、A2和A3

将A[low,…,*high*]分成三块，使得小于*mm*的在左边，等于*mm*的在中间，大于*mm*的右边，那么



- 你能看出什么？
- 在不经意间，**我们对数组A做了什么？**
- 再提示一点：**我们对这个*mm*做了什么？**

快速排序

快速排序是一个非常流行而且高效的算法，其平均时间复杂度为 $\Theta(n\log n)$. 其优于合并排序之处在于它在原位上排序，不需要额外的辅助存贮空间(合并排序需 $\Theta(n)$的辅助空间)。

Charles A. R. Hoare 1960 年发布了使他闻名于世的快速排序算法(Quicksort)，这个算法也是当前世界上使用最广泛的算法之一，当时他供职于伦敦一家不大的计算机生产厂家。1980 年，Hoare 被授予图灵奖，以表彰其在程序语言定义与设计领域的根本性的贡献。在 2000 年，Hoare 因其在计算机科学和教育方面的杰出贡献被英国皇家封为爵士。

快速排序

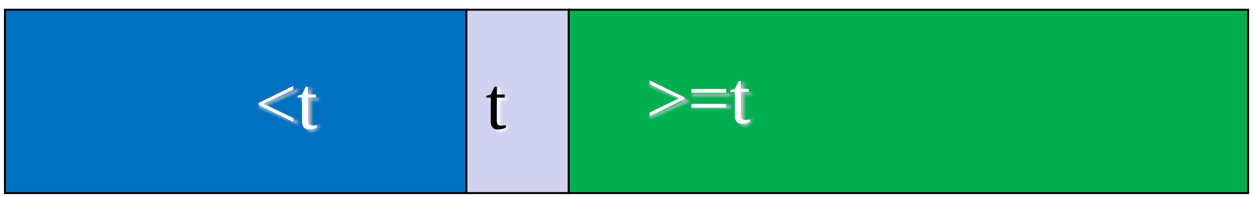
[0] [1] [2] [3] [4] [5]

[0] [1] [2] [3] [4] [5]

快速排序思想

1）寻找一个中心元素（通常为第一个数）

2）将小于中心点的元素移动至中心点之前，大于中心点的元素移动至中心点之后



3）对上步分成的两个无序数组段重复1）和 2）操作直到段长为1

快速排序

以21为中心元素

划分可得：

以06、49为中心元素

划分可得：

快速排序

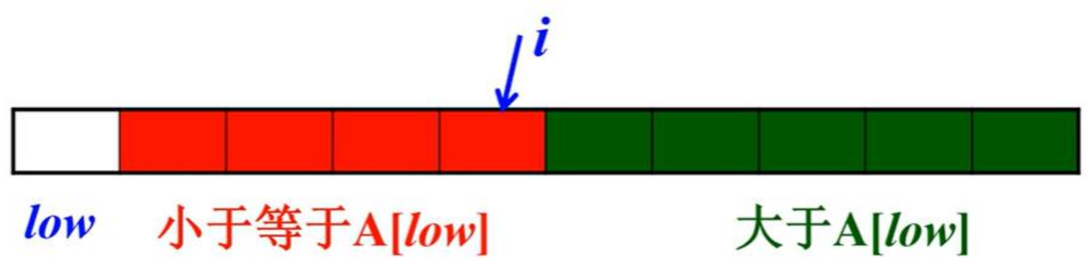
- 选取中心元素的问题
  - 选取第一个数为中心元素
- 如何划分的问题
- 如何重复步骤1和2将所有数据排序
  - 使用递归

快速排序

- **需要解决的问题（Split划分算法）**
  - 当已知中心元素的前提下，怎样将其他元素划分好？

（即：大于中心点在之后，小于中心点在之前）

- **解法一**
  - 从左向右遍历A[low+1,…, high]，大于中心元素不交换，小于就和第一个大于的元素交换
  - A[low]和最后一个小于它的元素交换



快速排序

算法 6.5 SPLIT

输入： 数组 $A[low \cdots high]$。

输出： (1) 如有必要，输出按上述描述的重新排列的数组 A；

(2) 划分元素 $A[low]$的新位置 w。

```
i ← low
x ← A[low]
for j ← low + 1 to high
    if A[j] ≤ x then
        i ← i + 1
        if i ≠ j then 互换 A[i]和 A[j]
    end if
end for
互换 A[low]和 A[i]
w ← i
return A 和 w
```

快速排序

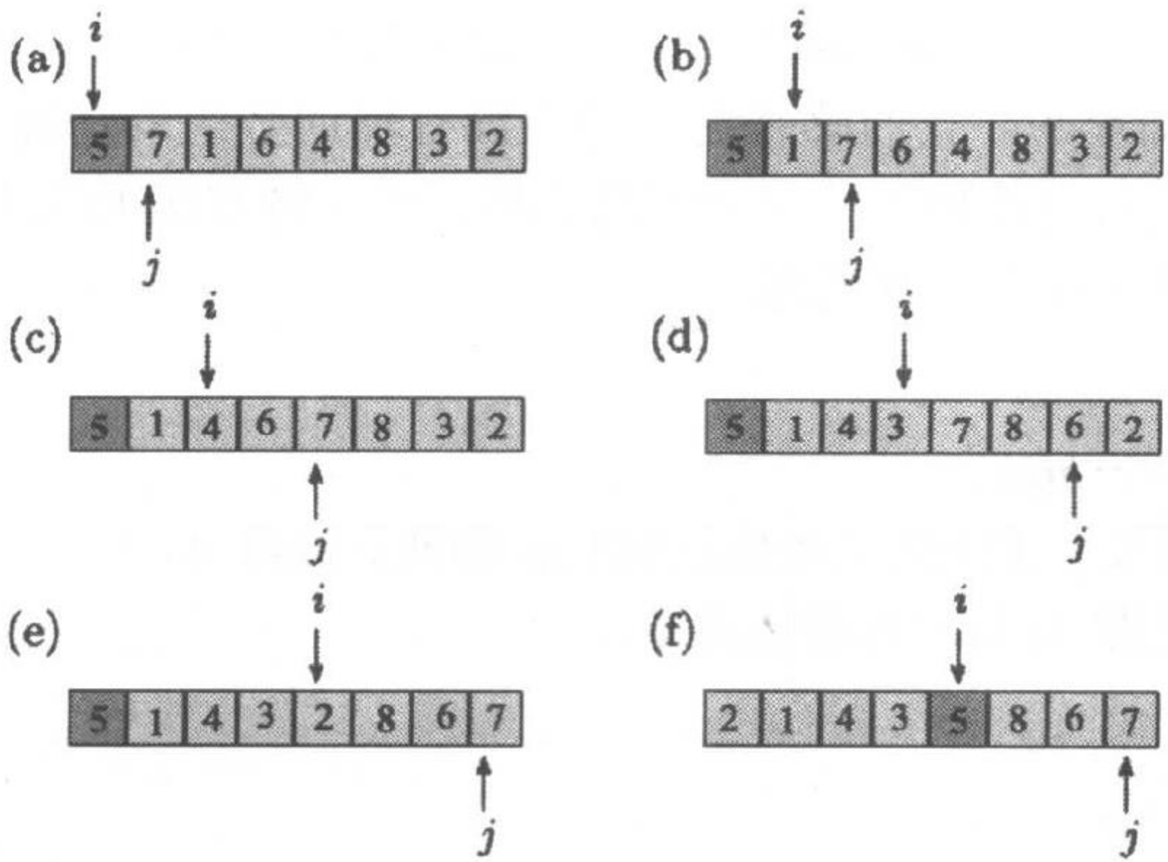


图 6.4 用 SPLIT 算法划分序列数的例子

快速排序

- **需要解决的问题（Split划分算法）**
  - 当已知中心元素的前提下，怎样将其他元素划分好？

（即：大于中心点在之后，小于中心点在之前）

- **解法二**
  - 左边向右找第一个大于等于中心点的数字
  - 右边向左找第一个小于等于中心点的数字
  - 两个数字交换

左右重合

快速排序实例讲解

i=0	j=5

[0] [1] [2] [3] [4] [5]

快速排序实例讲解

i=1	j=5

[0] [1] [2] [3] [4] [5]

快速排序实例讲解

i=1	j=4

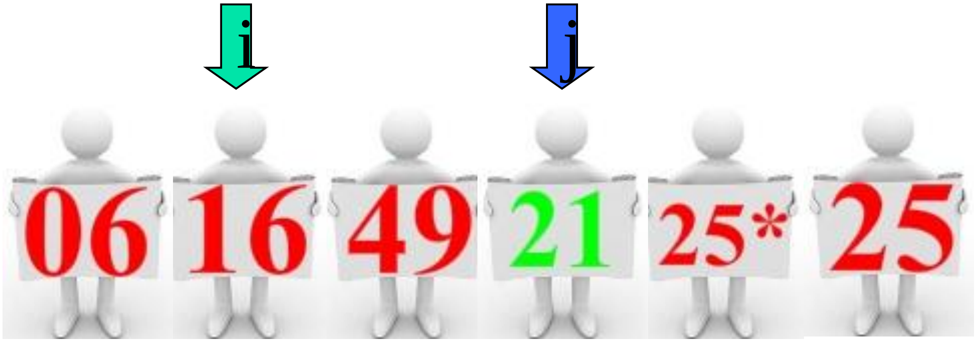
快速排序实例讲解

i=1	j=3

[0] [1] [2] [3] [4] [5]

快速排序实例讲解

i=2	j=3

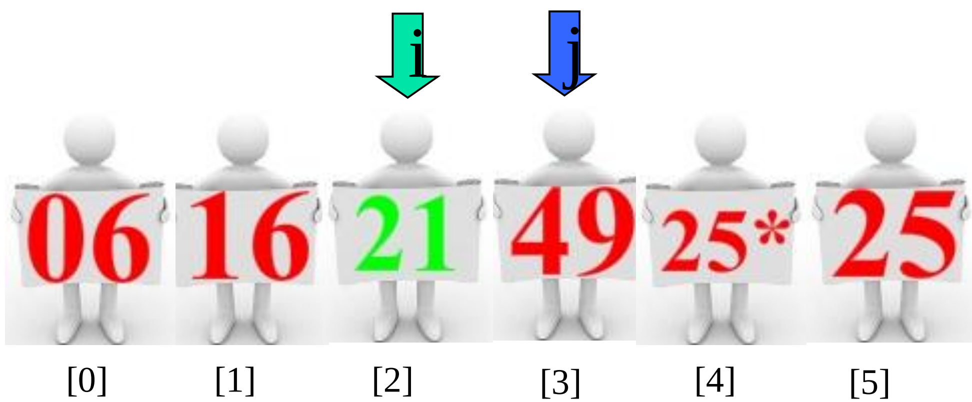


[0] [1] [2] [3] [4] [5]

快速排序实例讲解

i=2	j=2

算法终止



快速排序

```
Algorithm: SPLIT(A[low…high])
i ←low
j←high
x←A[low]
while(i< j)
    while(i<j&&A[i]<x)
        i=i+1
    while(i<j &&A[j]>x)
        j=j-1
    互换A[i]和A[j]
w←i
return A和w
```

快速排序

- 对原数组进行划分
- 对划分后的左、右子数组进行递归调用

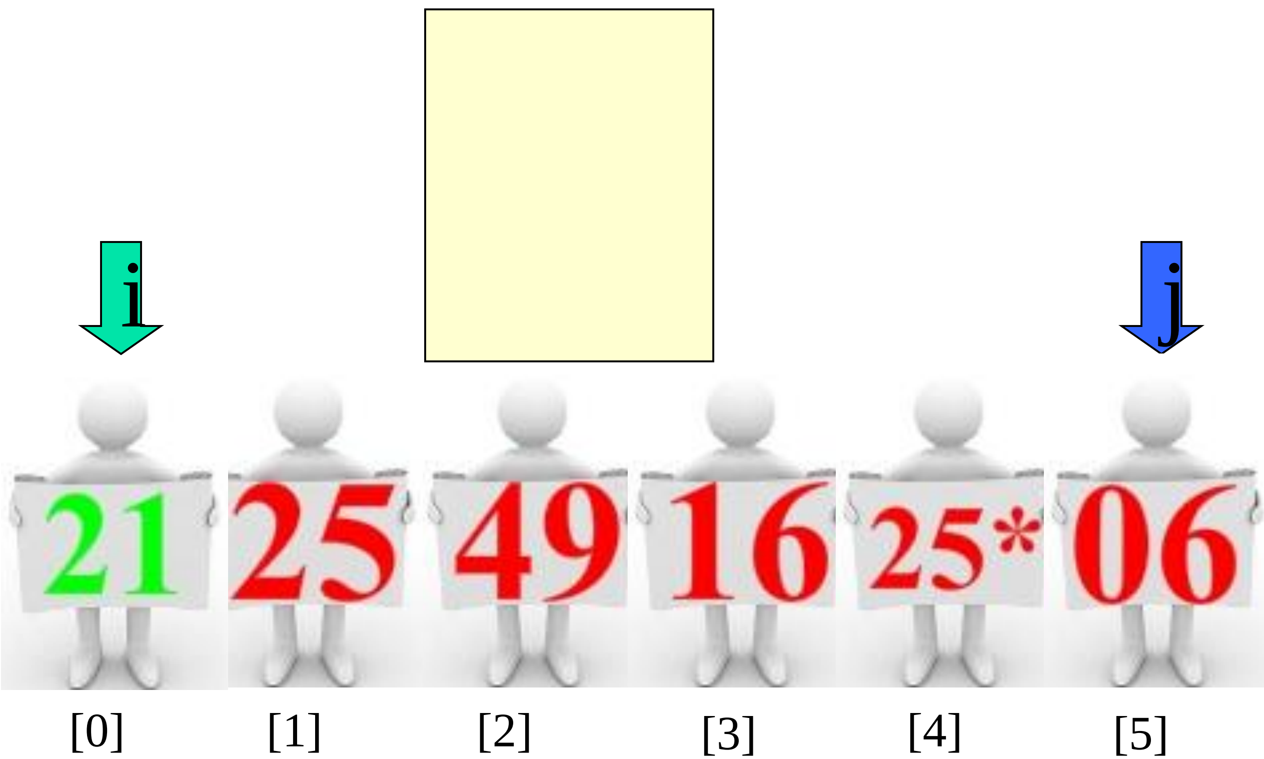
```
Algorithm: QUICKSORT(A[low…high])
输入: n个元素的数组A[low…high]
输出：按非降序排列的数组A[low…high]
if low<high then
    w ← SPLIT(A[low…high]) {w为基准元素A[low]的新位置}    划分
    quicksort(A, low, w-1)
    quicksort(A, w+1, high)
end if
```

请同学们思考

该算法有没有可以改进的地方

通过动画，可以看出每次中心元素都要交换。根据划分的思想最后位置一定是中心元素

可以申请一个变量保存中心元素，以避免交换



i=0	j=5
i=1	j=5
i=1	j=4
i=1	j=3
i=2	j=3
i=2	j=2

算法终止

程序填空

left,right用于限定要排序数列的范围,temp即为中心元素

```
i=left;j=right;int temp=a[left];
do
{       //从右向左找第1个小于中心元素的位置j
        while( [ ] > [ ] && i<j)      j--;
        if(i<j)
        {       a[ ] = a[ ];
                i++;
        }
```

当前元素小于中心元素结束循环时，应当在中心元素的左边

移至左边

程序填空

```
        //从左向右找第1个大于中心元素的位置i
        while(a[i]<temp && i<j )      i++;
        if(i<j)
        {        a[j]=a[i];
                 j--;
        }
}while(i<j);
[          ]//将中心元素t填入最终位置
w=i;
```

排序方法对比

- 冒泡排序 $\Theta(n^2)$
- 选择排序$\Theta(n^2)$
- 插入排序$\Theta(n^2)$
- 合并排序$\Theta(n \log n)$
- 堆排序$\Theta(n \log n)$
- **快速排序?**

时间复杂度分析

理想情形：每次SPLIT后得到的左右子数组规模相当，因此有：

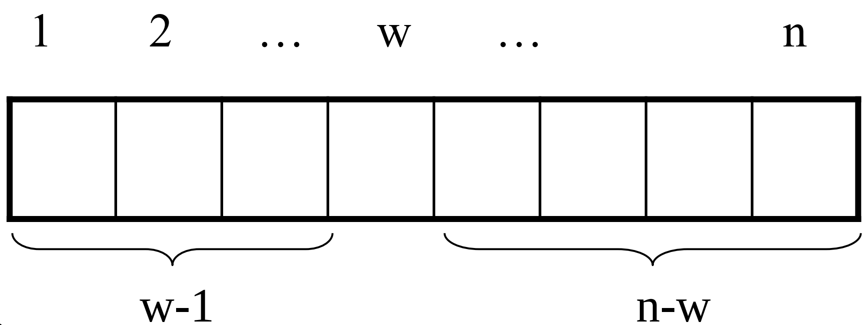
$$T(n)=\begin{cases} O(1) & if\ \ n=1 \\ 2T(n/2)+\Theta(n) & if\ \ n>1 \end{cases} \quad \Longrightarrow \quad T(n)=\Theta(n\log n)$$

最差情形(已经排好序或是逆序的数组)：每次SPLIT后，只得到左或是右子数组，因此有：

$$T(n)=\begin{cases} O(1) & if\ \ n=1 \\ T(n-1)+\Theta(n) & if\ \ n>1 \end{cases} \quad \Longrightarrow \quad T(n)=\Theta(n^2)$$

平均情形：

我们用 $C(n)$ 表示对一个n个元素的数组进行快速排序所需要的总的比较次数。



因此，我们有：

$$C(n)=(n-1)+\frac{1}{n}\sum_{w=1}^{n}\left(C(w-1)+C(n-w)\right)$$

$$\because \sum_{w=1}^{n}C(n-w)=C(n-1)+C(n-2)+\cdots+C(0)=\sum_{w=1}^{n}C(w-1)$$

$$\therefore C(n)=(n-1)+\frac{2}{n}\sum_{w=1}^{n}C(w-1)$$

$$n \cdot C(n) = n(n-1) + 2\sum_{w=1}^{n} C(w-1) \ldots\ldots (a)$$

n-1替换n

$$(n-1)C(n-1) = (n-1)(n-2) + 2\sum_{w=1}^{n-1} C(w-1) \ldots\ldots (b)$$

(a)-(b),并适当变换

$$\frac{C(n)}{n+1} = \frac{C(n-1)}{n} + \frac{2(n-1)}{n(n+1)}$$

令$D(n) = \frac{C(n)}{n+1}$

$$D(n) = D(n-1) + \frac{2(n-1)}{n(n+1)}, D(1) = 0$$

$$D(n) = 2\sum_{j=1}^{n} \frac{j-1}{j(j+1)} = 2\sum_{j=1}^{n} \frac{2}{(j+1)} - 2\sum_{j=1}^{n} \frac{1}{j}$$

$$= 4\sum_{j=2}^{n+1} \frac{1}{j} - 2\sum_{j=1}^{n} \frac{1}{j} = 2\sum_{j=1}^{n} \frac{1}{j} - \frac{4n}{n+1} = \Theta(\log n)$$

$$\therefore C(n) = (n+1)D(n) = \Theta(n \log n)$$

使用分治策略的算法设计模式

```
divide_and_conquer(P)
{
    if(|P|<=n0)
        direct_process(P);  //解决小规模的问题
    else
    {
        divide P into smaller subinstances P1,P2,…,Pa ;  //分解问题
        for(int i=1;i<=a;i++)
            yi=divide_and_conquer(Pi);  //递归地解各子问题
        merge(y1,y2,…,ya);  //将各子问题的解合并为原问题的解
    }
}
```

分治算法的时间复杂度分析

- 从分治法的一般设计模式可以看出，用它设计出的算法通常可以是递归算法。因而，算法的时间复杂度通常可以用递归方程来分析。
- 假设算法将规模为n的问题分解为a(a>=1)个规模为n/b(b>1)的子问题解决。分解子问题以及合并子问题的解耗费的时间为s(n)，则算法的时间复杂度可以递归表示为：

$$T(n)=\begin{cases} c & , \ n<=n_0 \\ aT(n/b)+s(n) & , \ n>n_0 \end{cases}$$

- 回顾Master Theorem

Master Theorem

设$a \geq 1$，$b>1$为常数。$s(n)$为一给定的函数，$T(n)$递归定义如下：

$$T(n) = a \cdot T(n/b) + s(n)$$

并且$T(n)$有适当的初始值。那么，当n充分大时，有：

(1) 若存在$\varepsilon>0$，使得$s(n)=\Omega(n^{\log_b^a+\varepsilon})$ 成立，并且存在$c<1$，使得 $a \cdot s(n/b) \leq c \cdot s(n)$，那么有$T(n)=\Theta(s(n))$

(2) 若$s(n)=\Theta(n^{\log_b^a})$，那么$T(n)=\Theta(n^{\log_b^a} \cdot \log n)$

(3) 若存在$\varepsilon>0$，使得$s(n)=O(n^{\log_b^a-\varepsilon})$ 成立，那么有$T(n)=\Theta(n^{\log_b^a})$

Master Theorem

- https://www.youtube.com/watch?v=2H0GKdrIowU

Theorem 5.1 *Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form*

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

then for n a power of b,

1. *if* $\log_b a < c$, $T(n) = \Theta(n^c)$,
2. *if* $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,
3. *if* $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.