

## 第2章 递归与分治策略(2)

# 内容

## 2.1 求解递归式

## 2.2 递归

1. 阶乘函数
2. Fibonacci数列
3. 排列问题
4. 整数划分问题
5. Hanoi塔问题

## 2.3 分治

1. 二分搜索技术
2. 合并排序
3. 快速排序
4. 线性时间选择

# 递归与分治策略总体思想

- 将一个难以直接解决的大问题，分割成一些规模较小的相同子问题。如果这些子问题都可解，并可利用这些子问题的解求出原问题的解，则这种分治法可行。

# 分治法的适用条件

- 分治法所能解决的问题一般具有以下特征：
  1. 该问题的规模缩小到一定的程度就可以容易地解决；
  2. 该问题可以分解为若干个规模较小的相同问题
  3. 利用该问题分解出的子问题的解可以合并为该问题的解；
  4. 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

# 分治法的基本步骤

## **DIVIDE-AND-CONQUER(P)**

//当问题规模小于 $n_0$ 时, 直接求解

if  $|P| \leq n_0$ , **ADHOC**(P), return

//分解问题

divide P into smaller subinstances  $P_1, P_2, \dots, P_a$

//递归的解各子问题

for  $i=1$  to  $a$

$y_i = \mathbf{DIVIDE-AND-CONQUER}(P_i)$

//将各子问题的解合并为原问题的解

return **MERGE**( $y_1, \dots, y_a$ )

# 子问题的规模

- 人们从大量实践中发现，在用分治法设计算法时，最好使**子问题的规模大致相同**。即将一个问题分成大小相等的 $a$ 个子问题的处理方法是行之有效的。
- 这种使子问题规模大致相等的做法是出自一种**平衡 (balancing) 子问题**的思想，它几乎总是比子问题规模不等的做法要好。

## 2.3 分治

1. 二分搜索技术

2. 合并排序

# 1 二分搜索技术

- 问题描述：给定已排好序的 $n$ 个元素 $A[1..n]$ ，在这 $n$ 个元素中找出一特定元素 $key$ 。

查找 $key=51$

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
15	17	18	22	35	51	60	88	90	100



# 两种算法

- 顺序搜索方法

- 没有利用 $n$ 个元素已排好序这个条件
- 最坏情况下,  $T(n) = O(n)$

- 二分搜索方法

- 利用元素间的次序关系, 采用分治策略
- 最坏情况下,  $T(n) = O(\log n)$

# 二分搜索技术

查找key=51

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
15	17	18	22	35	51	60	88	90	100

Left

Mid

Right

由于Key大于Mid位置的数据，修正查找区间范围  
——右半区间：[Mid+1, Right]

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
15	17	18	22	35	51	60	88	90	100

Left

Mid

Right

由于Key小于Mid位置的数据，修正查找区间范围  
——左半区间：[Left, Mid-1]

A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)	A(10)
15	17	18	22	35	51	60	88	90	100

Left

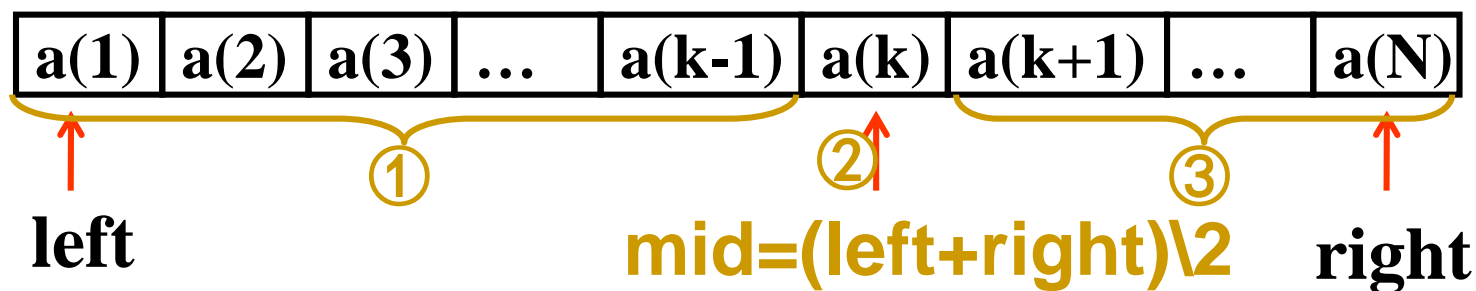
Right

Mid

$Mid = (Left + Right) \setminus 2$

# 二分搜索技术

- 又称折半查找法
- 仅适用于**有序数组**！
- 基本思想：在下标区间 $[left, right]$ 间，用 $mid$ 将区间一分为二，根据 $mid$ 位置元素和 $key$ 的大小关系，确定下一次查找区间



- ①  $a(mid) > key$  则：  $right = mid - 1$
- ②  $a(mid) = key$  则： 找到！
- ③  $a(mid) < key$  则：  $left = mid + 1$

结束条件：

- 找到
- 未找到 ( $left > right$ )

# 二分搜索算法描述

```
1 RECURSIVE-BINARY-SEARCH(A, key, left, right)
2   if left>right or key<A[left] or key>A[right] //在A中找不到key
3     return NULL
4   mid = floor((left + right)/2)
5   if key==A[mid]
6     return mid
7   else if key>A[mid] //在区间A[mid+1..right]中找key
8     return RECURSIVE-BINARY-SEARCH(A, key, mid+1, right)
9   else //在区间A[left..mid-1]中找key
10    return RECURSIVE-BINARY-SEARCH(A, key, left, mid-1)
```

# 二分搜索技术分析

- 满足分治法解决问题的四个特征：
  1. 该问题的规模缩小到一定的程度就可以容易地解决；
  2. 该问题可以分解为若干个规模较小的相同问题；
  3. 分解出的子问题的解可以合并为原问题的解；
  4. 分解出的各个子问题是相互独立的。

# 算法时间复杂度分析

- 每调用一次二分算法，待搜索数组的大小为原来的一半。因此，在最坏情况下，二分算法被调用了  $O(\log n)$  次。
- 每次调用二分算法，运算需要  $\Theta(1)$  时间，因此整个算法在最坏情况下的计算时间复杂度为  $O(\log n)$
- 递归式：

$$T(n) = T(n/2) + \Theta(1)$$

## 2.3 分治

1. 二分搜索技术

2. 合并排序

## 2 合并排序

- a) 递归合并排序
- b) 非递归的合并排序
- c) 折半拆分合并排序
- d) 自然合并排序



## a) 递归合并排序

- 用分治策略实现对 $n$ 个元素进行排序的算法
- 基本思想：
  1. **分解**：将 $n$ 个元素分成各含 $n/2$ 个元素的2个子集合；
  2. **解决**：递归调用合并排序算法，分别对2个子集合进行排序；
  3. **合并**：合并两个已排好序的子集合以得到排序结果。

# 递归合并排序算法描述

```
1  MERGE_SORT(A, p, r)
2  if p < r
3      q = floor((p+r)/2)
4      MERGE_SORT(A, p, q)
5      MERGE_SORT(A, q+1, r)
6      MERGE(A, p, q, r)
```

# 合并函数Merge

A[p:q] {15, 20, 35, 45, 60}

A[q+1:r] {10, 30, 40, 50}

B[p:r] { }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30, 35 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30, 35, 40 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30, 35, 40, 45 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30, 35, 40, 45, 50 }

A[p:q] { 15, 20, 35, 45, 60 }

A[q+1:r] { 10, 30, 40, 50 }

B[p:r] { 10, 15, 20, 30, 35, 40, 45, 50, 60 }

# 合并函数MERGE算法描述

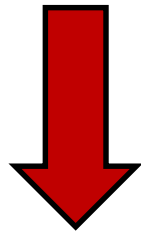
MERGE( $A, p, q, r$ )

```
1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
```

# 递归合并排序

## —算法时间复杂度分析

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



$$T(n) = O(n \log n)$$

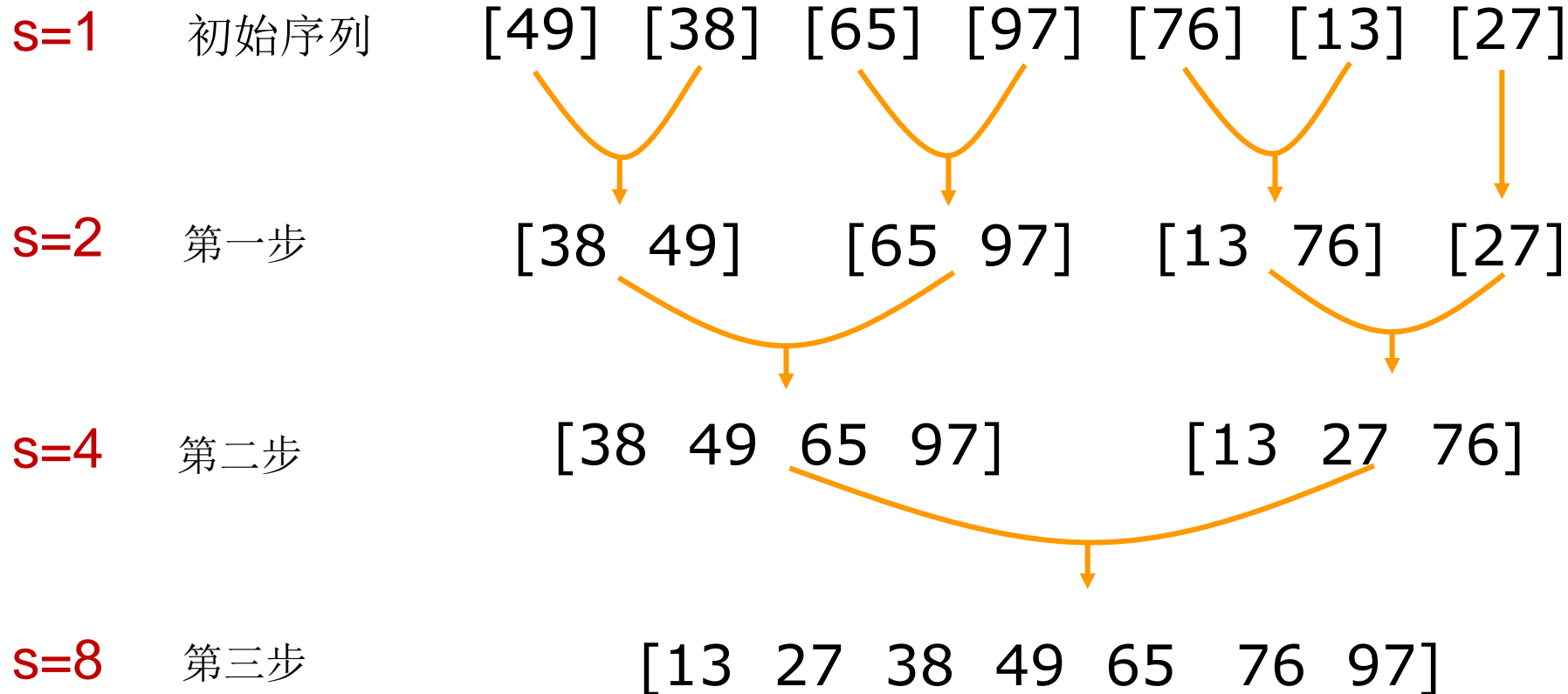
# 递归合并排序——特点

- 辅助空间:  $O(n)$

## b) 非递归的合并排序

- 算法MergeSort的递归过程只是将待排序集合一分为二，直至待排序集合只剩下一个元素为止，然后不断合并两个排好序的数组段。
- 按此机制，可首先将数组A中相邻元素两两配对，用合并算法将它们排序，构成 $n/2$ 组长度为2的排好序的子数组段，然后再将它们排序成长度为4的排好序的子数组段，如此继续下去，直至整个数组排好序。

# 非递归的合并排序图解





## c) 折半拆分合并排序算法描述

1. 设长度 $s$ 为1;
2. 将数组 $A$ 中元素顺序编号, 并把数组 $A$ 拆分成两个数组 $A_1$ 和 $A_2$ , 其中 $A_1$ 中是编号为奇数的元素,  $A_2$ 中是编号为偶数的元素;
3. 按长度 $s$ , 依次按大小合并两数组中对应位置的元素, 再依次复制到数组 $A$ 中;
4. 长度 $s=2*s$ , 连续执行步骤2和3, 直至 $s$ 大于或等于数组 $A$ 中元素个数 $n$ 。

## c) 折半拆分合并排序算法描述

s=1	A	<u>49</u>	<u>38</u>	<u>65</u>	<u>97</u>	<u>76</u>	<u>13</u>	<u>27</u>
		①	②	③	④	⑤	⑥	⑦
	A <sub>1</sub>	49	65	76	27			
	A <sub>2</sub>	38	97	13				
s=2	A	<u>38</u>	<u>49</u>	<u>65</u>	<u>97</u>	<u>13</u>	<u>76</u>	<u>27</u>
		①		②		③		④
	A <sub>1</sub>	<u>38</u>	<u>49</u>	<u>13</u>	<u>76</u>			
	A <sub>2</sub>	<u>65</u>	<u>97</u>	<u>27</u>				
s=4	A	<u>38</u>	<u>49</u>	<u>65</u>	<u>97</u>	<u>13</u>	<u>27</u>	<u>76</u>
				①			②	
	⋮							

## d) 自然合并排序

- 合并排序中，第一步合并的是相邻长度为1的子数组段，这是因为长度为1的子数组段是已排好序的
- 事实上，对于初始给定的数组A，通常存在多个长度大于1的已自然排好序的子数组段。
- 例：
  - [75 55 15 20 85 30 35 10 60 40 50 25 45 80 70 65]
  - [75 55 15 20 31 30 35 32 60 40 50 25 45 80 70 65]

# 自然合并排序算法描述

- 拆分：
  - 对数组进行线性扫描，找出所有排好序的子数组段并进行编号，依次复制到子数组A1和A2中。
  - 分别扫描子数组A1和A2，合并相邻的排好序的子数组段。
- 合并：
  - 合并A1和A2数组中对应位置的排好序的子数组段到A中
- 重复步骤1和2，直到只能拆分成一个子数组为止

# 自然合并排序算法描述

A    75   55   15   20   31   30   35   32   60   40   50   25   45   80   70   65  
          ①     ②            ③            ④            ⑤            ⑥            ⑦            ⑧     ⑨

A<sub>1</sub>    75   15   20   31   32   60   25   45   80   65

A<sub>2</sub>    55   30   35            40   50   70

A    55   75   15   20   30   31   32   35   40   50   60   70   25   45   80   65  
          ①                                    ②                                    ③                                    ④

End