8.4.1 冒泡排序

算法思路 将待排序的记录数组R[0]到R[n-1]，由前向后扫描，依次对相邻的两个记录R_i和R_{i+1}进行比较，如果R_i大于R_{i+1}，则交换两个记录，否则不交换。经过第一趟n-1次比较交换后，关键字值最大的记录将移到最后单元位置。接着对记录R[0]到R[n-2]进行第二趟冒泡，将第二个小的记录放在数组的倒数第二个位置，依此类推，重复此过程直到所有的记录都已有序为止。

8.4.1 冒泡排序

冒泡排序演示

原始序列：15 13(1) 9 46 4 18 13(2) 7

比较： j-1 j

交换： 13(1) 15 9 46 4 18 13(2) 7

比较： j-1 j

交换： 13(1) 9 15 46 4 18 13(2) 7

比较： j-1 j

交换： 13(1) 9 15 4 46 18 13(2) 7

j-1 j

8.4.1 冒泡排序

冒泡排序演示

交换： 13(1) 9 15 4 46 18 13(2) 7

比较： j-1 j

交换： 13(1) 9 15 4 18 46 13(2) 7

比较： j-1 j

交换： 13(1) 9 15 4 18 13(2) 46 7

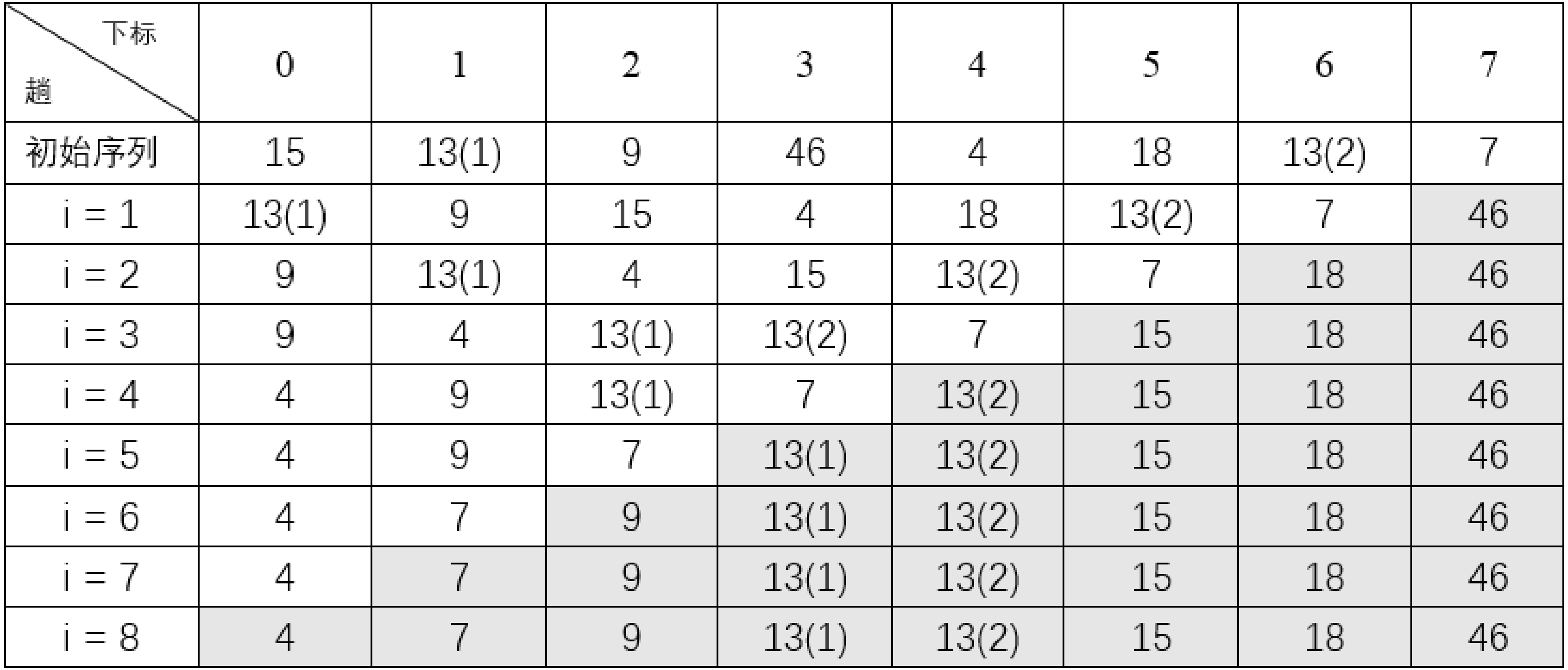
比较： j-1 j

交换： 13(1) 9 15 4 18 13(2) 7 46

第一趟排序： 13(1) 9 15 4 18 13(2) 7 46

8.4.1 冒泡排序

表 8-4 冒泡排序每趟结果



下标 趟	0	1	2	3	4	5	6	7
初始序列	15	13(1)	9	46	4	18	13(2)	7
i = 1	13(1)	9	15	4	18	13(2)	7	46
i = 2	9	13(1)	4	15	13(2)	7	18	46
i = 3	9	4	13(1)	13(2)	7	15	18	46
i = 4	4	9	13(1)	7	13(2)	15	18	46
i = 5	4	9	7	13(1)	13(2)	15	18	46
i = 6	4	7	9	13(1)	13(2)	15	18	46
i = 7	4	7	9	13(1)	13(2)	15	18	46
i = 8	4	7	9	13(1)	13(2)	15	18	46

算法8-8

```
void BubbleSort(SortArr *sortArr)
{
        int i,j;
        int hasSwap = 0; // 标志，用于检测内循环是否还有数据交换
        for(i = 1; i < sortArr->cnt; i++)
        {
                hasSwap = 0; //每趟开始重新设置交换标志为0
                //注意j是从后往前循环 ,数组的下标是0到cnt-1
                for(j = sortArr->cnt - 1; j >= i; j--)
                {
                        //若前者大于后者
                if(sortArr->recordArr[j-1].key>sortArr->recordArr[j].key)
                 {
                        Swap(sortArr, j, j-1); //交换
                        hasSwap = 1; //有交换发生，则设置交换标志为1
                 }
                }
                if (!hasSwap) //本趟没有发生交换
                        break;
        }
}
```

8.4.2 快速排序-quicksort

比枢轴小的元素 **枢轴** **比枢轴大的元素**

例如：关键字序列

52 49 80 36 14 58 61 97 23 75

调整为：

23 49 14 36 52 58 61 97 80 75

先宏观调整再微观调整

8.4.2 快速排序

设枢轴记录的关键字存放在temp变量，

设两个指针i和j,初值分别是一个序列的第一个和最后一个记录的位置：

1.从j所指位置由后向前搜索直到第一个关键字小于temp的记录和枢轴记录交换，

2.从i所指位置起由前向后搜索，找到第一个关键字大于temp的记录和枢轴记录互相交换，

3.重复交替1和2，直到i=j为止

8.4.2 快速排序-举例

快速排序演示

	x ||							
j向左扫描：	13(1)	15	9	18	4	46	13(2)	7
	↑ i							↑ j
第一次交换：	7	15	9	18	4	46	13(2)	13(1)
i向右扫描：	7	15	9	18	4	46	13(2)	13(1)
		↑ i						↑ j
第二次交换：	7	13(1)	9	18	4	46	13(2)	15
j向左扫描：	7	13(1)	9	18	4	46	13(2)	15
		↑ i					↑ j	

8.4.2 快速排序-举例

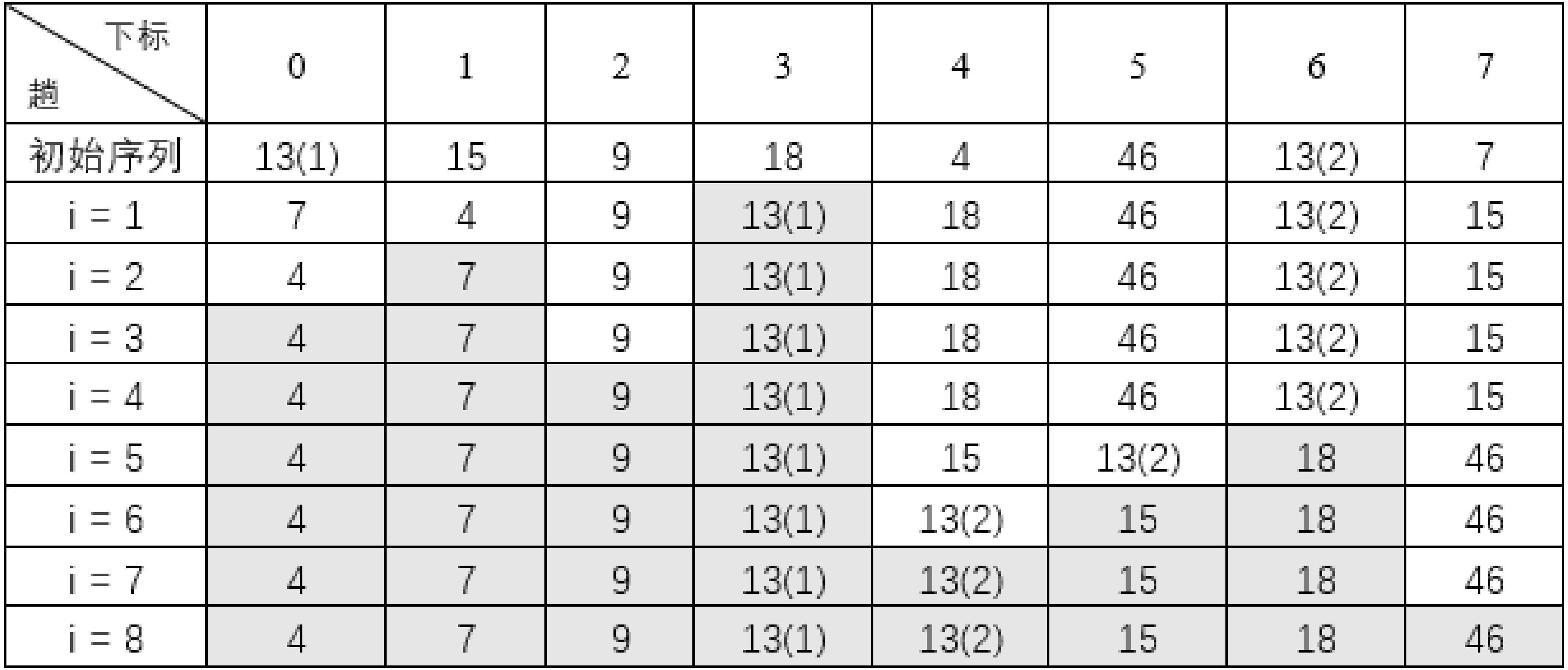
快速排序的演示

j向左扫描：	7	13(1)	9	18	4	46	13(2)	15
		↑ i					↑ j	
第三次交换：	7	4	9	18	13(1)	46	13(2)	15
i向右扫描：	7	4	9	18	13(1)	46	13(2)	15
			↑ i		↑ j			
第四次交换：	7	4	9	13(1)	18	46	13(2)	15
第一趟完成：	7	4	9	13(1)	18	46	13(2)	15
				↑ ↑ i j				

	i							j
初始序列	13(1)	15	9	18	4	46	13(2)	7
	i							j
j 向左扫描	13(1)	15	9	18	4	46	13(2)	7
		i						j
第一次交换	7	15	9	18	4	46	13(2)	13(1)
		i						j
i 向右扫描	7	15	9	18	4	46	13(2)	13(1)
		i					j	
第二次交换	7	13(1)	9	18	4	46	13(2)	15
		i			j			
j 向左扫描	7	13(1)	9	18	4	46	13(2)	15
			i		j			
第三次交换	7	4	9	18	13(1)	46	13(2)	15
				i	j			
i 向右扫描	7	4	9	18	13(1)	46	13(2)	15
				i j				
第四次交换	7	4	9	13(1)	18	46	13(2)	15

8.4.2 快速排序

表 8-5 快速排序每趟结果（加底纹的记录表示排序好的记录）



下标 趟	0	1	2	3	4	5	6	7
初始序列	13(1)	15	9	18	4	46	13(2)	7
i = 1	7	4	9	13(1)	18	46	13(2)	15
i = 2	4	7	9	13(1)	18	46	13(2)	15
i = 3	4	7	9	13(1)	18	46	13(2)	15
i = 4	4	7	9	13(1)	18	46	13(2)	15
i = 5	4	7	9	13(1)	15	13(2)	18	46
i = 6	4	7	9	13(1)	13(2)	15	18	46
i = 7	4	7	9	13(1)	13(2)	15	18	46
i = 8	4	7	9	13(1)	13(2)	15	18	46

课堂练习：写出一趟排序后的结果

	x							
	| |							
初始关键字：	27	38	13	49	76	97	65	50
	↑	↑	↑	↑ ↑	↑	↑	↑	↑
	i	i	i	i j	j	j	j	j

完成一趟排序： (27 38 13) 49 (76 97 65 50)

分别进行快速排序： (13) 27 (38) 49 (50 65) 76 (97)

快速排序结束： 13 27 38 49 50 65 76 97

算法分析

最坏情况: $C_{\max} = \sum_{i=1}^{n-1}(n-i) = \frac{n}{2}(n-1) \approx \frac{n^2}{2}$

1,2,3,4,5,6,7

最好情况

$$
\begin{aligned}
&C(n) \\
&\le n + 2C(n/2) \\
&\le n + 2[n/2 + 2C(n/2^2)] = 2n + 4C(n/2^2) \\
&\le 2n + 4[n/4 + 2C(n/2^3)] = 3n + 8C(n/2^3) \\
&\le \ldots\ldots \\
&\le kn + 2^k C(n/2^k) \\
&= n\log_2^n + nC(1) \\
&= O(n\log_2^n)
\end{aligned}
$$

算法分析

空间复杂度: 算法是递归算法，需要一个栈空间，栈的大小取决于递归调用的深度，最坏不超过n，最好情况是logn

稳定性： 不稳定的排序