算法分析

- 算法是稳定的
- 空间代价：Θ(1)，交换操作需要一个辅助空间
- 时间代价
  - 最佳情况（正序）：n-1次比较，0次交换，Θ(n)复杂度
  - 最差情况（逆序）：比较和交换次数为

$$\sum_{i=1}^{n-1} i = n(n-1)/2 = \Theta(n^2)$$

  - 平均情况：$\Theta(n^2)$

实验表明：当记录数量n较小时，直接插入排序是一种高效的排序算法！

算法分析

- 算法是稳定的
- 空间代价：Θ(1)的临时空间
- 时间代价
  - 比较次数

$$\sum_{i=1}^{n-1}(n-i)=n(n-1)/2=\Theta(n^2)$$

  - 交换次数最多为$\Theta(n^2)$，最少为0，平均为$\Theta(n^2)$。
  - 最大，最小，平均时间代价均为$\Theta(n^2)$。

优化的冒泡排序

- 优化思路
  - 检查每次冒泡过程中是否发生过交换，如果没有，则表明整个数组已经排好序了，排序结束。
  - 结束条件：不再有元素交换
- 时间代价：
  - 最小时间代价为$\Theta(n)$：最佳情况下只运行第一轮循环
  - 平均情况下时间代价仍为$\Theta(n^2)$

直接选择排序算法

```
void StraightSelectSorter (Record Array[], int n) {
    // 依次选出第i小的记录，即剩余记录中最小的那个
    for (int i=0; i<n-1; i++){
        int Smallest = i;                    // 首先假设记录i就是最小的
        for (int j=i+1;j<n; j++)             //开始向后扫描所有剩余记录
            if ( Array[j] < Array[Smallest])
                Smallest = j;                // 如发现更小记录，记录其位置
        swap(Array, i, Smallest);            //将第i小的记录放在第i个位置
    }
}
```

与冒泡排序的关系：（1）冒泡排序从后面两两交换冒出最小的；而直接选择排序是直接找到最小的和第一个进行交换！（2）前者是稳定的，后者是不稳定的！

4、简单排序算法的时间代价对比

比较次数	直接插入排序	改进的插入排序	二分法插入排序	冒泡排序	改进的冒泡排序	选择排序
最佳情况	Θ(n)	Θ(n)	Θ(nlog n)	Θ(n2)	Θ(n)	Θ(n2)
平均情况	Θ(n2)	Θ(n2)	Θ(nlog n)	Θ(n2)	Θ(n2)	Θ(n2)
最差情况	Θ(n2)	Θ(n2)	Θ(nlog n)	Θ(n2)	Θ(n2)	Θ(n2)

移动次数	直接插入排序	改进的插入排序	二分法插入排序	冒泡排序	改进的冒泡排序	选择排序
最佳情况	0	$\Theta(n)$	$\Theta(n)$	0	0	$\Theta(n)$
平均情况	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
最差情况	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$

总代价	直接插入排序	改进的插入排序	二分法插入排序	冒泡排序	改进的冒泡排序	选择排序
最佳情况	$\Theta(n)$	$\Theta(n)$	$\Theta(n\log n)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
平均情况	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
最差情况	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$

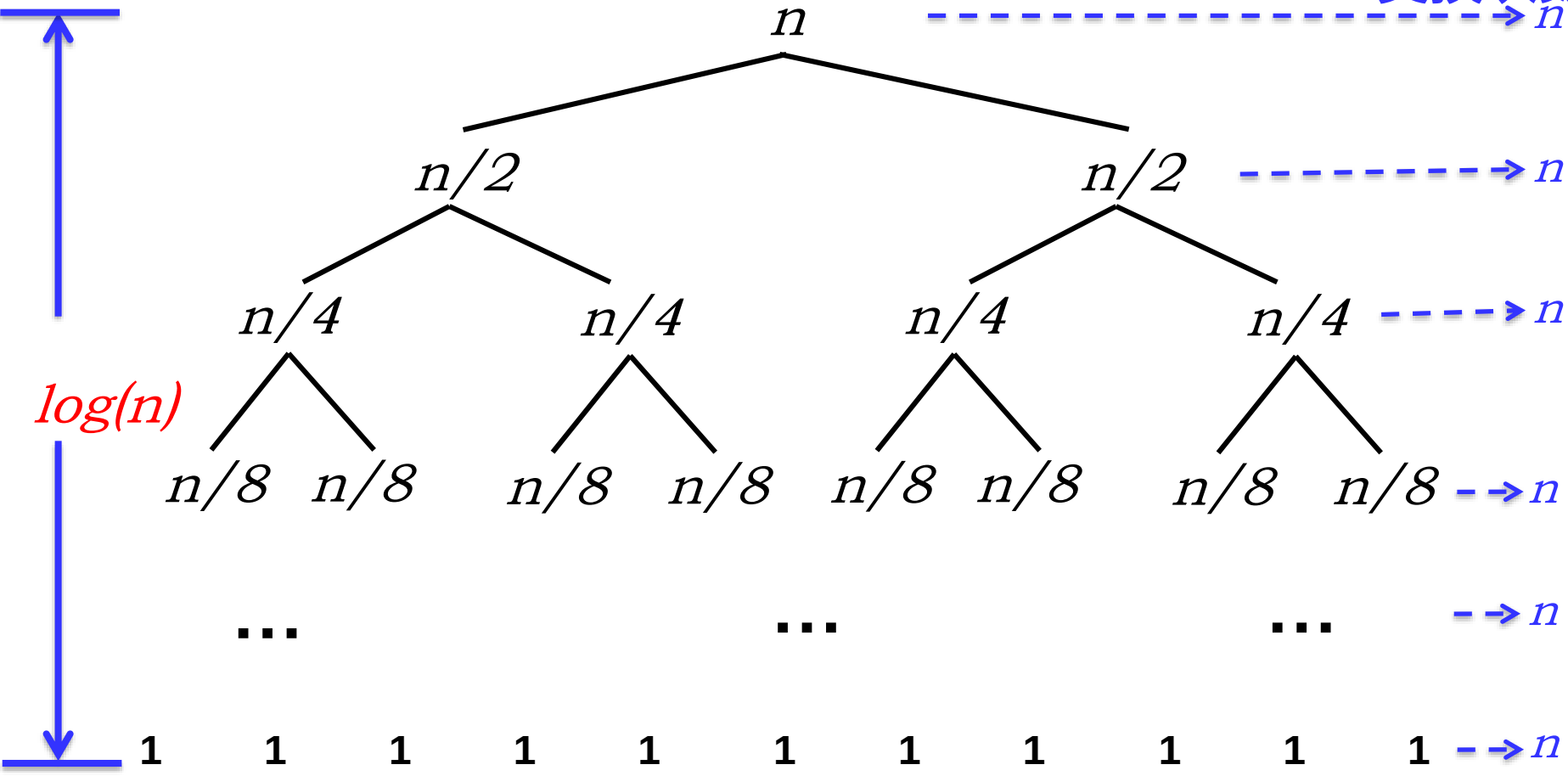
分析

- 增量序列可有各种取法：任何正整数的递减序列d1, d2, …dt, 只要 d1<n, dt=1, 原则上都可作为希尔排序的增量序列
- Hibbard增量序列
  - $\{2^k -1，2^{k-1} -1，…，7，3，1\}$，
  - Hibbard增量序列的Shell排序的效率可以达到$\Theta(n^{3/2})$
- 选取其他增量序列还可以更进一步减少时间代价
- 希尔排序是一种不稳定的排序方法

算法分析(1)

➢ **最佳性能：** $T(n)= O(n\lg(n))$

每层比较交换次数

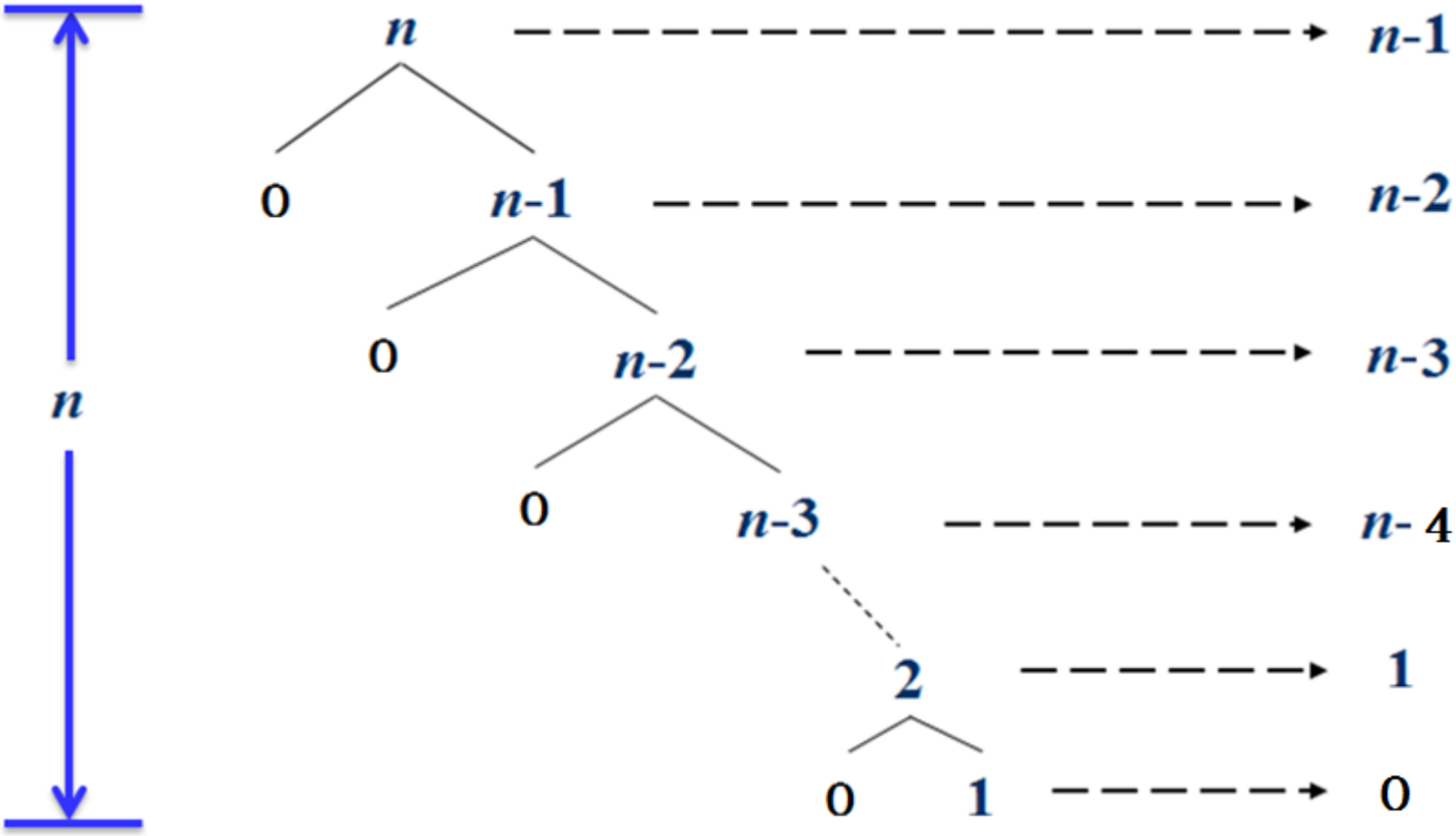


算法分析(2)

- **最差性能：** $T(n)= O(n^2)$

每层比较

交换次数



算法分析(3)

- **平均性能：** $T(n)= O(\text{nlogn})$

$$\mathrm{T(n)} = \frac{1}{n}\sum_{i=0}^{n-1}\big(T(i)+T(n-i-1)\big)+cn = \frac{2}{n}\sum_{i=0}^{n-1}T(i)+cn$$

$$\mathrm{nT(n)} = 2\sum_{i=0}^{n-1}T(i)+cn^2$$

$$\mathrm{nT(n)} - (n-1)T(n-1) = 2T(n-1)+2cn-c$$

$$\mathrm{nT(n)} = (n+1)T(n-1)+2cn$$ 公式两侧除于n(n+1)

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

归并排序性能分析

- 容易看出，对 n 个记录进行归并排序的时间复杂度为 O(nlogn)。即：
  - 每一趟归并的时间复杂度为 O(n)，
  - 总共需进行⌈logn⌉趟。
  - 归并排序需要附加一倍的存储量O(n)
    - 是辅助存储量最多的一种排序方法
- 是稳定排序算法

算法分析

- 非稳定性排序
- 建堆：$\Theta(n)$
- 删除一次堆顶重新建堆：$\Theta(\log n)$
- 一次建堆，n次删除堆顶，总时间代价为$\Theta(n\log n)$
- 理论上，堆排序最佳、最差、平均情况下的时间代价均为$\Theta(n\log n)$
- 辅助空间代价：$\Theta(1)$

算法分析

- 时间代价
  - 统计计数：$\Theta(m+n)$
  - 总时间代价：$\Theta(m+n)$
- 空间代价
  - 需要m个计数器，n个临时空间
  - 总的空间代价：$\Theta(m+n)$
  - 适用于m相对于n很小的情况
- 稳定算法

对m的讨论

- m的取值决定了算法的复杂性
  - 当m为Θ(*n*)数量级时，时间代价为Θ(Θ(*n*)+*n*)，还是Θ(*n*)。
  - 当m为Θ(*n*log *n*)或Θ(n^2)时，时间代价变成Θ(*n*log *n*)或Θ(n^2)。
- 因此，桶式排序只适合m很小的情况

m非常大时如何处理?

算法分析

- **空间代价：**
  - **临时数组,n**
  - **r个计数器**
  - **$\Theta(n+r)$**
- **时间代价**
  - **桶式排序：$\Theta(r+n)$**
  - **d次桶式排序**
  - **总的时间复杂性：$\Theta(d \cdot (r+n))$**

算法分析

- 空间代价
  - n个记录指针空间
  - r个子序列的头尾指针
  - O(n + r)
- 时间代价
  - 不需移动记录，只需修改next指针
  - O(d·(n+r))
- 时间代价O(d*n)，线性复杂性吗?
  - 实际上还是O(nlogn)
  - 没有重复编码的情况，需要n个不同的编码来处理他们
  - 也就是说，$d \geq \log_r^n$ 即O(nlogn)

```
            IndexArray[j] = j;  // 因为是正确归位，索引j就是自身
            j = k;                          //j换到循环链中的下一个，继续处理
        }
        Array[j]=TempRec;      // 第i大元素正确入位
        IndexArray[j]=j;          // 因为是正确归位，索引j就是自身
    }
}
```

在调整算法中，i下标是无回溯的for循环，while循环中的调整处理也都一次到位，每个元素最多参与一轮调整。因此，整个调整算法的时间代价为O(n)。空间代价显然为O(1)。

8.7 各种排序算法的理论和实验时间代价

算法	最大时间	平均时间	最小时间	辅助空间代价	稳定性
直接插入排序	Θ(n^2)	Θ(n^2)	Θ(n)	Θ(1)	稳定
二分法插入排序	Θ(n^2)	Θ(n^2)	Θ(nlog^n)	Θ(1)	稳定
冒泡排序	Θ(n^2)	Θ(n^2)	Θ(n^2)	Θ(1)	稳定
改进的冒泡排序	Θ(n^2)	Θ(n^2)	Θ(n)	Θ(1)	稳定
选择排序	Θ(n^2)	Θ(n^2)	Θ(n^2)	Θ(1)	不稳定
Shell排序(3)	Θ(n3/2)	Θ(n3/2)	Θ(n3/2)	Θ(1)	不稳定
快速排序	Θ(n^2)	Θ(nlog^n)	Θ(nlog^n)	Θ(log^n)	不稳定
归并排序	Θ(nlog^n)	Θ(nlog^n)	Θ(nlog^n)	Θ(n)	稳定
堆排序	Θ(nlog^n)	Θ(nlog^n)	Θ(nlog^n)	Θ(1)	不稳定
桶式排序	Θ(n+m)	Θ(n+m)	Θ(n+m)	Θ(n+m)	稳定
基数排序	Θ(d·(n+r))	Θ(d·(n+r))	Θ(d·(n+r))	Θ(n+r)	稳定

8.8 排序问题的界

- **Lower Bound**
  - 解决排序问题能达到的最佳效率，即使尚未设计出算法
- **Upper Bound**
  - 指已知最快算法所达到的最佳渐进效率

- 排序问题的下限应该在$\Omega(n)$到$O(n \log n)$之间