数据结构与算法（八）

张铭 主讲

采用教材：张铭，王腾蛟，赵海燕 编写
高等教育出版社，2008. 6 （“十一五”国家级规划教材）
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg

大 纲

- 8.1 **排序问题的基本概念**
- 8.2 插入排序 (Shell 排序)
- 8.3 选择排序 (堆排序)
- 8.4 交换排序
  - 8.4.1 冒泡排序
  - 8.4.2 快速排序
- 8.5 归并排序
- 8.6 分配排序和索引排序
- 8.7 排序算法的时间代价
- 内排序知识点总结

8.1 基本概念

- 序列 (Sequence)：线性表
  - 由记录组成
- 记录 (Record)：结点，进行排序的基本单位
- 关键码 (Key)：唯一确定记录的一个或多个域
- 排序码 (Sort Key)：作为排序运算依据的一个或多个域

什么是排序？

- 排序
  - 将序列中的记录按照排序码顺序排列起来
  - 排序码域的值具有不减（或不增）的顺序
- 内排序
  - 整个排序过程在内存中完成

排序问题

- 给定一个序列 $R = \{ r_1, r_2, \ldots , r_n \}$
  - 其排序码分别为 $k = \{ k_1, k_2, \ldots , k_n \}$
- 排序的目的：将记录按排序码重排
  - 形成新的有序序列 $R' = \{ r'_1, r'_2, \ldots , r'_n \}$
  - 相应排序码为 $k' = \{ k'_1, k'_2, \ldots , k'_n \}$
- 排序码的顺序
  - 其中 $k'_1 \le k'_2 \le \ldots \le k'_n$，称为不减序
  - 或 $k'_1 \ge k'_2 \ge \ldots \ge k'_n$，称为不增序

正序 vs. 逆序

- “正序”序列：待排序序列正好符合排序要求
- “逆序”序列：把待排序序列逆转过来，正好符合排序要求
- 例如，要求不升序列
  - 08 12 34 96
  - 96 34 12 08

正序！

逆序！

排序的稳定性

- 稳定性
  - 存在多个具有相同排序码的记录
  - 排序后这些记录的相对次序保持不变
- 例如，
  - 34 12 34′ 08 96
  - 08 12 34 34′ 96

稳定！

- 稳定性的证明——形式化证明

排序的稳定性

- 稳定性
  - 存在多个具有相同排序码的记录
  - 排序后这些记录的相对次序保持不变
- 例如，
  - 34　12　34′　08　96
  - 08　12　34′　34　96

不稳定！

- 不稳定性的证明——反例说明

排序算法的衡量标准

- 时间代价：记录的比较和移动次数
- 空间代价
- 算法本身的繁杂程度

45	34	78	12

思考

1. 排序算法的稳定性有何意义？

2. 为何需要考虑“正序”与“逆序”序列？

大 纲

- 8.1 排序问题的基本概念
- **8.2 插入排序 (Shell 排序)**
- 8.3 选择排序 (堆排序)
- 8.4 交换排序
  - 8.4.1 冒泡排序
  - 8.4.2 快速排序
- 8.5 归并排序
- 8.6 分配排序和索引排序
- 8.7 排序算法的时间代价
- 内排序知识点总结

8.2 插入排序

- 8.2.1 直接插入排序
- 8.2.2 Shell 排序

插入排序动画

45	34	78	12	*34'*	32	29	64

插入排序算法

12	34	45	78	34′

```
template <class Record>
void ImprovedInsertSort (Record Array[], int n){
//Array[] 为待排序数组，n 为数组长度
    Record TempRecord;              // 临时变量
    for (int i=1; i<n; i++){        // 依次插入第 i 个记录
        TempRecord = Array[i];
        //从 i 开始往前寻找记录 i 的正确位置
        int j = i-1;
        //将那些大于等于记录 i 的记录后移
        while ((j>=0) && (TempRecord < Array[j])){
            Array[j+1] = Array[j];
            j = j - 1;
        }
        //此时 j 后面就是记录 i 的正确位置，回填
        Array[j+1] = TempRecord;
    }
}
```

8.2 插入排序

算法分析

- 稳定
- 空间代价：Θ(1)
- 时间代价：
  - 最佳情况：n-1 次比较，2(n-1) 次移动，Θ(*n*)
  - 最差情况： Θ(n^2)
    - 比较次数为 $$\sum_{i=1}^{n-1} i = n(n-1)/2 = \Theta(n^2)$$
    - 移动次数为 $$\sum_{i=1}^{n-1} (i+2) = (n-1)(n+4)/2 = \Theta(n^2)$$
  - 平均情况：Θ(n^2)

8.2.2 Shell排序

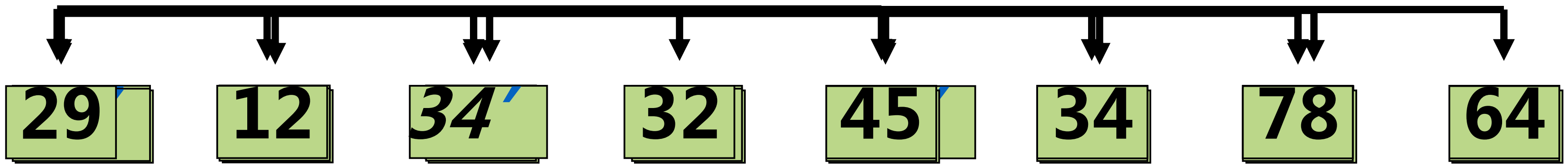
- 直接插入排序的两个性质：
  - 在最好情况（序列本身已是有序的）下时间代价为 $\Theta(n)$
  - 对于短序列，直接插入排序比较有效
- Shell 排序有效地利用了直接插入排序的这两个性质

Shell排序算法思想

- 先将序列转化为若干小序列，在这些小序列内进行插入排序
- 逐渐增加小序列的规模，而减少小序列个数，使得待排序序列逐渐处于更有序的状态
- 最后对整个序列进行扫尾直接插入排序，从而完成排序

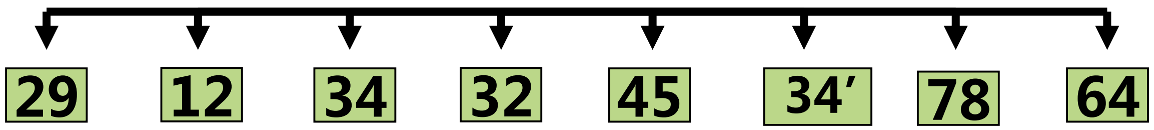
8.2.2 Shell排序

Shell排序动画



“增量每次除以2递减”的Shell 排序

```
template <class Record>
void ShellSort(Record Array[], int n) {
// Shell排序，Array[]为待排序数组，n为数组长度
    int i, delta;
    // 增量delta每次除以2递减
    for (delta = n/2; delta>0; delta /= 2)
        for (i = 0; i < delta; i++)
            // 分别对delta个子序列进行插入排序
            //“&”传 Array[i]的地址，数组总长度为n-i
            ModInsSort(&Array[i], n-i, delta);
    // 如果增量序列不能保证最后一个delta间距为1
    // 可以安排下面这个扫尾性质的插入排序
    // ModInsSort(Array, n, 1);
}
```



8.2.2 Shell排序

针对增量而修改的插入排序算法

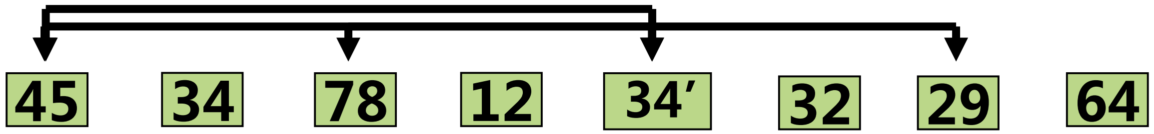
```
template <class Record> // 参数delta表示当前的增量
void ModInsSort(Record Array[], int n, int delta) {
   int i, j;
   // 对子序列中第i个记录，寻找合适的插入位置
   for (i = delta; i < n; i += delta)
         // j以dealta为步长向前寻找逆置对进行调整
         for (j = i; j >= delta; j -= delta)  {
             if (Array[j] < Array[j-delta]) // 逆置对
                   swap(Array, j, j-delta);// 交换
             else break;
         }
}
```

算法分析

- 不稳定
- 空间代价：$\Theta(1)$
- 时间代价
  - 增量每次除以2递减，$\Theta(n^2)$
- 选择适当的增量序列
  - 可以使得时间代价接近 $\Theta(n)$

Shell 排序选择增量序列

- 增量每次除以2递减
  - 效率仍然为 $\Theta(n^2)$
- 问题：选取的增量之间并不互质
  - 间距为 2^{k-1} 的子序列，都是由那些间距为 2^k 的子序列组成的
  - 上一轮循环中这些子序列都已经排过序了，导致处理效率不高



Hibbard 增量序列

- Hibbard 增量序列
  - $\{2^k - 1, 2^{k-1} - 1, \ldots, 7, 3, 1\}$
- Shell(3) 和 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

Shell最好的代价

- 呈 2^p3^q 形式的一系列整数：
  - 1, 2, 3, 4, 6, 8, 9, 12
- $\Theta(n \log_2 n)$

思考

- 1. 插入排序的变种
  - 发现逆序对直接交换
  - 查找待插入位置时，采用二分法

- 2. Shell 排序中增量作用是什么？增量为2和增量为3的序列，哪个更好？为什么？

- 3. Shell 排序的每一轮子序列排序可以用其他方法吗？

大 纲

- 8.1 排序问题的基本概念
- 8.2 插入排序 (Shell 排序)
- **8.3 选择排序 (堆排序)**
- 8.4 交换排序
  - 8.4.1 冒泡排序
  - 8.4.2 快速排序
- 8.5 归并排序
- 8.6 分配排序和索引排序
- 8.7 排序算法的时间代价
- 内排序知识点总结

8.3 选择排序

- 8.3.1 直接选择排序
  - 依次选出剩下的未排序记录中的最小记录
- 8.3.2 堆排序
  - 堆排序：基于最大堆来实现

直接选择排序动画

45	34	78	12	*34′*	32	29	64

直接选择排序

```
template <class Record>
void SelectSort(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++) {
        // 首先假设记录i就是最小的
        int Smallest = i;
        // 开始向后扫描所有剩余记录
        for (int j=i+1;j<n; j++)
            // 如果发现更小的记录，记录它的位置
            if (Array[j] < Array[Smallest])
                Smallest = j;
        // 将第i小的记录放在数组中第i个位置
        swap(Array, i, Smallest);
    }
}
```

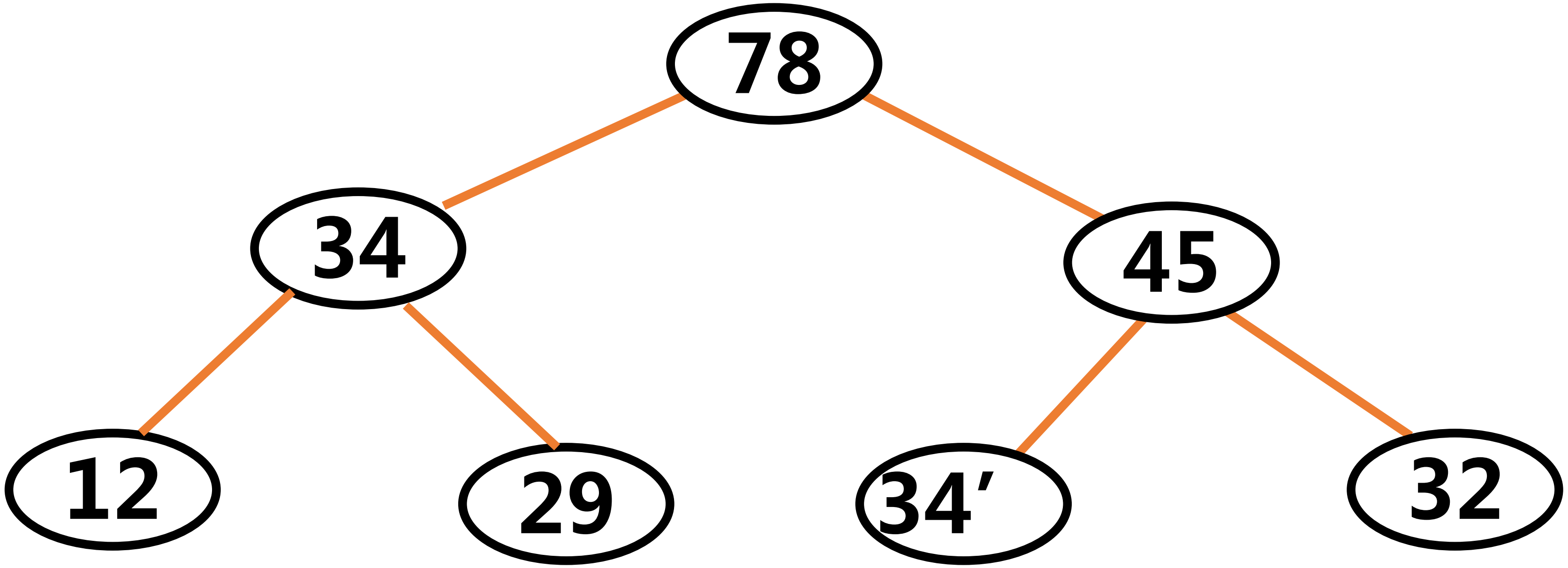
直接选择排序性能分析

- 不稳定
- 空间代价：$\Theta(1)$
- 时间代价
  - 比较次数：$\Theta(n^2)$
  - 交换次数：n-1
  - 总时间代价：$\Theta(n^2)$

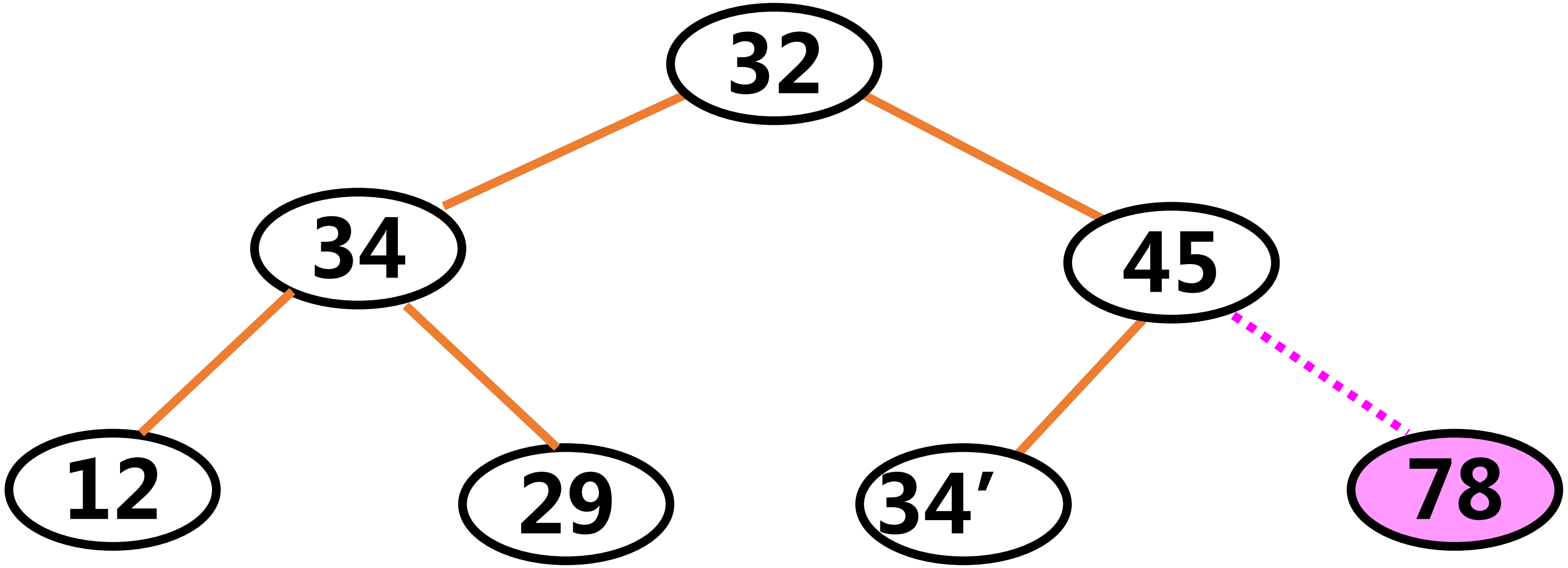
8.3.2 堆排序

- 选择类内排序
  - 直接选择排序：直接从剩余记录中线性查找最大记录
  - 堆排序：基于最大堆来实现，效率更高
- 选择类外排序
  - 置换选择排序
  - 赢者树、败方树

最大堆排序过程示意图



最大堆排序过程示意图



堆排序算法

```
template <class Record>
void sort(Record Array[], int n){
   int i;
   // 建堆
   MaxHeap<Record> max_heap
         = MaxHeap<Record>(Array,n,n);
   // 算法操作n-1次，最小元素不需要出堆
   for (i = 0; i < n-1; i++)
      // 依次找出剩余记录中的最大记录，即堆顶
      max_heap. RemoveMax();
}
```

算法分析

- 建堆：$\Theta(n)$
- 删除堆顶: $\Theta(\log n)$
- 一次建堆，n 次删除堆顶
- 总时间代价为 $\Theta(n\log n)$
- 空间代价为 $\Theta(1)$

思考

- 直接选择排序为什么不稳定？怎么修改一下让它变稳定
- 改写堆排序算法，发现逆序对直接交换

大 纲

- 8.1 排序问题的基本概念
- 8.2 插入排序 (Shell 排序)
- 8.3 选择排序 (堆排序)
- **8.4 交换排序**
  - **8.4.1 冒泡排序**
  - **8.4.2 快速排序**
- 8.5 归并排序
- 8.6 分配排序和索引排序
- 8.7 排序算法的时间代价
- 内排序知识点总结

8.4 交换排序

- 8.4.1 冒泡排序
- 8.4.2 快速排序

8.4.1 冒泡排序

- 算法思想
  - 不停地比较相邻的记录，如果不满足排序要求，就交换相邻记录，直到所有的记录都已经排好序
- 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
  - 避免不必要的比较

- 冒泡排序之舞

  http://v.youku.com/v_show/id_XMjU4MTg3MTU2.html

冒泡排序动画

45	34	78	12	*34'*	32	29	64

冒泡排序算法

```
template <class Record>
void BubbleSort(Record Array[], int n) {
    bool NoSwap;                    // 是否发生了交换的标志
    int i, j;
    for (i = 0; i < n-1; i++) {
        NoSwap = true;                  // 标志初始为真
        for (j = n-1; j > i; j--){
            if (Array[j] < Array[j-1]) {    // 判断是否逆置
                swap(Array, j, j-1);        // 交换逆置对
                NoSwap = false              // 发生了交换，标志变为假
            }
            if (NoSwap)                     // 没发生交换，则已完成排好序
                return;
        }
    }
}
```

算法分析

- 稳定
- 空间代价：Θ(1)
- 时间代价分析
  - 比较次数
    - 最少：$\Theta(n)$
    - 最多：

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

  交换次数最多为 $\Theta(n^2)$，最少为 0，平均为 $\Theta(n^2)$
- 时间代价结论
  - 最大，平均时间代价均为 $\Theta(n^2)$
  - 最小时间代价为 $\Theta(n)$：最佳情况下只运行第一轮循环

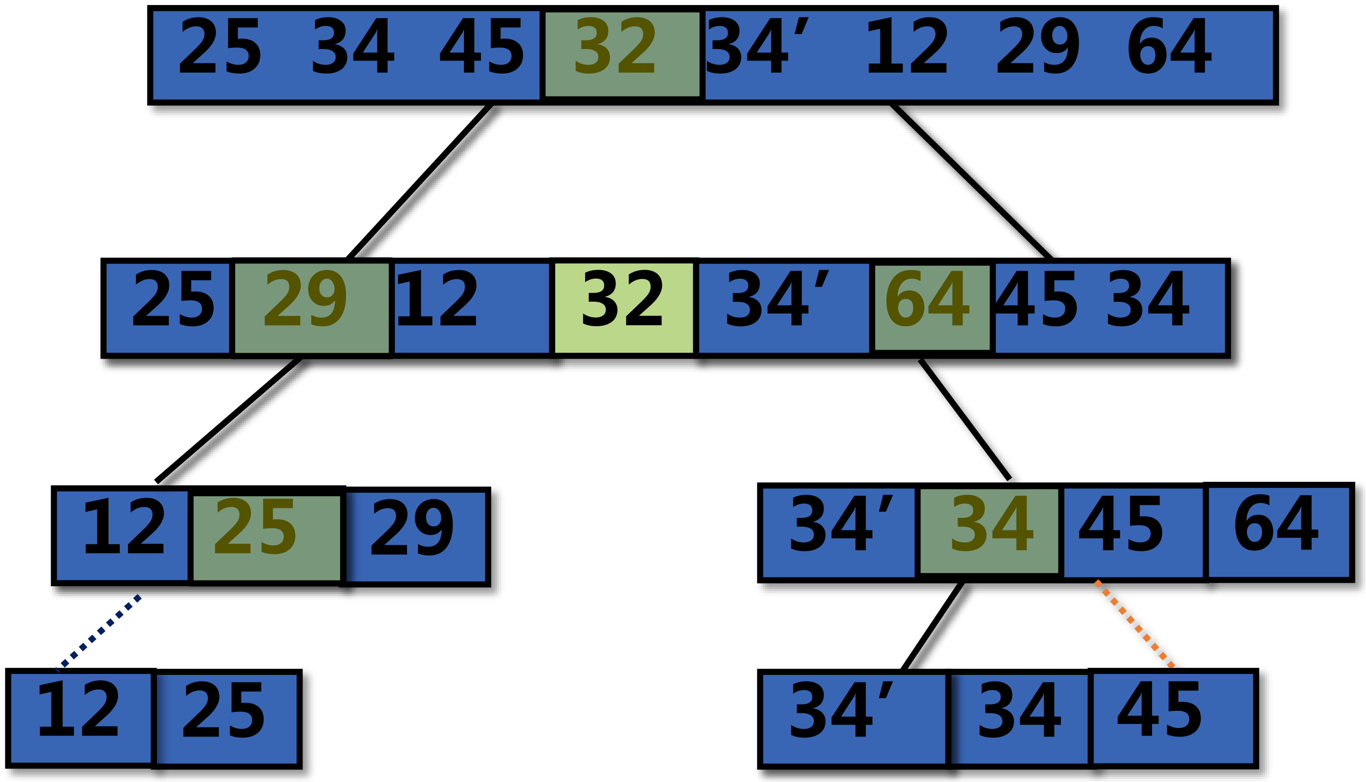
8.4.2 快速排序

- 算法思想
  - 选择轴值 (pivot)
  - 将序列划分为两个子序列 L 和 R，使得 L 中所有记录都小于或等于轴值，R 中记录都大于轴值
  - 对子序列 L 和 R 递归进行快速排序
- 20世纪十大算法
  - Top 10 Algorithms of the Century
  - 7. 1962 London 的 Elliot Brothers Ltd 的 Tony Hoare 提出的快速排序
- 基于分治法的排序：快速、归并

分治策略的基本思想

- 分治策略的实例
  - BST 查找、插入、删除算法
  - 快速排序、归并排序
  - 二分检索
- 主要思想
  - 划分
  - 求解子问题 (子问题不重叠)
  - 综合解

快速排序分治思想



最终排序结果： 12 25 29 32 34’ 34 45 64

轴值选择

- 尽可能使 L, R 长度相等
- 选择策略：
  - 选择最左边记录
  - 随机选择
  - 选择平均值

8.4.2 快速排序

一次分割过程

25	34	45	32	34’	12	29	64
i							j

- 选择轴值并存储轴值
- 最后一个元素放到轴值位置
- 初始化下标 i, j，分别指向头尾
- i 递增直到遇到比轴值大的元素，将此元素覆盖到 j 的位置；j 递减直到遇到比轴值小的元素，将此元素覆盖到 i 的位置
- 重复上一步直到 i==j，将轴值放到 i 的位置，完毕

8.4.2 快速排序

快速排序算法

```
template <class Record>
void QuickSort(Record Array[], int left, int right)  {
// Array[]为待排序数组，  left,right分别为数组两端
     if (right <= left)              // 只有0或1个记录，就不需排序
          return;
     int pivot = SelectPivot(left, right);   // 选择轴值
     swap(Array, pivot, right);                    // 轴值放到数组末端
     pivot = Partition(Array, left, right); // 分割后轴值正确
     QuickSort(Array, left, pivot-1);          // 右子序列递归快排
     QuickSort(Array, pivot +1, right);    // 右子序列递归快排
}
int SelectPivot(int left, int right) {
   // 选择轴值，  参数left,right分别表示序列的左右端下标
     return (left+right)/2;                    // 选中间记录作为轴值
}
```

分割函数

```
template <class Record>
int Partition(Record Array[], int left, int right) {
// 分割函数，分割后轴值已到达正确位置
    int l = left;               // l 为左指针
    int r = right;              // r 为右指针
    Record TempRecord = Array[r];  // 保存轴值
    while (l != r) {            // l, r 不断向中间移动，直到相遇
        // l 指针向右移动，直到找到一个大于轴值的记录
        while (Array[l] <= TempRecord && r > l)
            l++;
        if (l < r)  {           // 未相遇，将逆置元素换到右边空位
            Array[r] = Array[l];
            r--;                // r 指针向左移动一步
        }
```

8.4.2 快速排序

```
        // r 指针向左移动，直到找到一个大于轴值的记录
        while (Array[r] >= TempRecord && r > l)
            r--;
        if (l < r)  {                // 未相遇，将逆置元素换到左空位
            Array[l] = Array[r];
            l++;                         // l 指针向右移动一步
        }
    }  //end while
    Array[l] = TempRecord;   // 把轴值回填到分界位置 l 上
    return l;                          // 返回分界位置l
}
```

时间代价

- 长度为n的序列，时间为 T(*n*)
  - T(0) = T(1) =1
- 选择轴值时间为常数
- 分割时间为 *cn*
  - 分割后长度分别为 *i* 和 *n*-*i*-1
  - 左右子序列 T(*i*) 和 T(*n*-*i*-1)
- 求解递推方程

$$T(n)=T(i)+T(n-1-i)+cn$$

最差情况

$$T(n)=T(n-1)+cn$$
$$T(n-1)=T(n-2)+c(n-1)$$
$$T(n-2)=T(n-3)+c(n-2)$$
$$\cdots$$
$$T(2)=T(1)+c(2)$$

- 总的时间代价为：

$$T(n)=T(1)+c\sum_{i=2}^{n} i=\Theta(n^2)$$

8.4.2 快速排序

最佳情况

$$T(n) = 2T(n/2) + cn$$

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

$$\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + c$$

$$\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + c$$

$$\cdots$$

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

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

$$T(n) = cn\log n + n = \Theta(n\log n)$$

等概率情况

- 也就是说，轴值将数组分成长度为 0 和 n-1, 1 和 n-2，… 的子序列的概率是相等的，都为 1/n
- T(i) 和 T(n-1-i) 的平均值均为

$$T(i) = T(n-1-i) = \frac{1}{n}\sum_{k=0}^{n-1} T(k)$$

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

$$nT(n) = (n+1)T(n-1) + 2cn - c$$

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

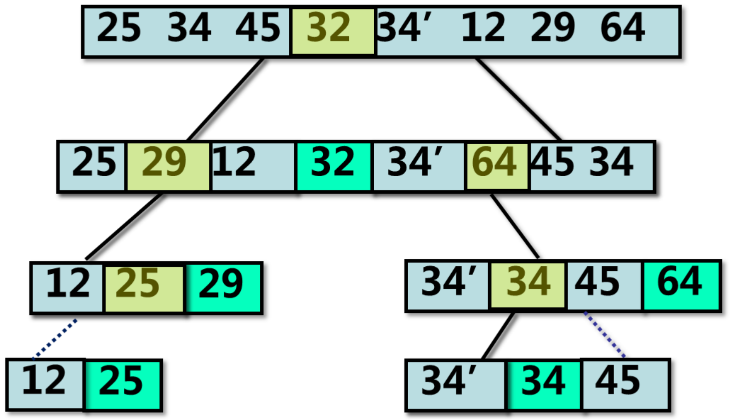
快速排序算法分析

- 最差情况：
  - 时间代价：$\Theta(n^2)$
  - 空间代价：$\Theta(n)$
- 最佳情况：
  - 时间代价：$\Theta(n\log n)$
  - 空间代价：$\Theta(\log n)$
- 平均情况：
  - 时间代价：$\Theta(n\log n)$
  - 空间代价：$\Theta(\log n)$

8.4.2 快速排序

思考

- 冒泡排序和直接选择排序哪个更优
- 快速排序为什么不稳定
- 快速排序可能的优化
  - 轴值选择 RQS
  - 小子串不递归（阈值 28？ ）
  - 消除递归（用栈，队列？）



数据结构与算法

谢谢聆听

国家精品课“数据结构与算法”
http://www.jpk.pku.edu.cn/pkujpk/course/sjjg/

张铭，王腾蛟，赵海燕
高等教育出版社，2008. 6。“十一五”国家级规划教材