

Lecture 30: 插入排序

30.1 插入排序基础概念

30.1.1 基础概念

什么是插入排序

- 思想：
 - 每次将一个待排序的记录按其关键字大小**插入到前面已排好序的子序列**中，直到全部记录插入完成；
 - 插入排序不是插入... 而是将当前序列当中的元素，按照往后插入的方式进行排序；

代码实现

- 直接插入排序

```
//直接插入排序
void InsertSort(int A[], int n){
    int i, j, temp;
    for(i=1; i<n; i++){
        //将各元素插入已排好序的序列中
        if(A[i]<A[i-1]){ //若A[i]关键字小于前驱
            temp=A[i]; //用temp暂存A[i]
            for(j=i-1; j>=0&&A[j]>temp; --j) //检查所有前面已经排好序的元素
                A[j+1]=A[j]; //所有大于temp的元素都向后挪位
            A[j+1]=temp; //复制到插入位置
        }
    }
}
```

- 直接插入排序：带哨兵
 - 实际的数据元素是从 1 开始存储的，0 的位置会空出来；
 - 当前存储的元素会放到 `A[0]` 的位置；
- 优点：
 - 不用每轮循环都判断 `j>=0`

```
//直接插入排序(带哨兵)
void InsertSort(int A[], int n){
    int i, j;
    for(i=2; i<=n; i++){ //依次将A[2]~A[n]插入到前面已排序序列
        if(A[i]<A[i-1]){ //若A[i]关键码小于其前驱，将A[i]插入有序表
            A[0]=A[i]; //复制为哨兵，A[0]不存放元素
            for(j=i-1; A[0]<A[j]; --j) //从后往前查找待插入位置
                A[j+1]=A[j]; //向后挪位
            A[j+1]=A[0]; //复制到插入位置
        }
    }
}
```

算法效率分析

- 空间复杂度：
 - $O(1)$
- 时间复杂度：
 - 概念：
 - 主要来自对比关键字、移动元素；
 - 若有 n 个元素，则需要 $n-1$ 趟处理；
 - 最好情况：
 - 若原本就有序： $O(n)$
 - 若原本时逆序： $O(n^2)$
 - 平均时间复杂度：
 - $O(n^2)$
- 最多比较次数：
 - $O(\frac{n(n-1)}{2})$

对链表的插入排序

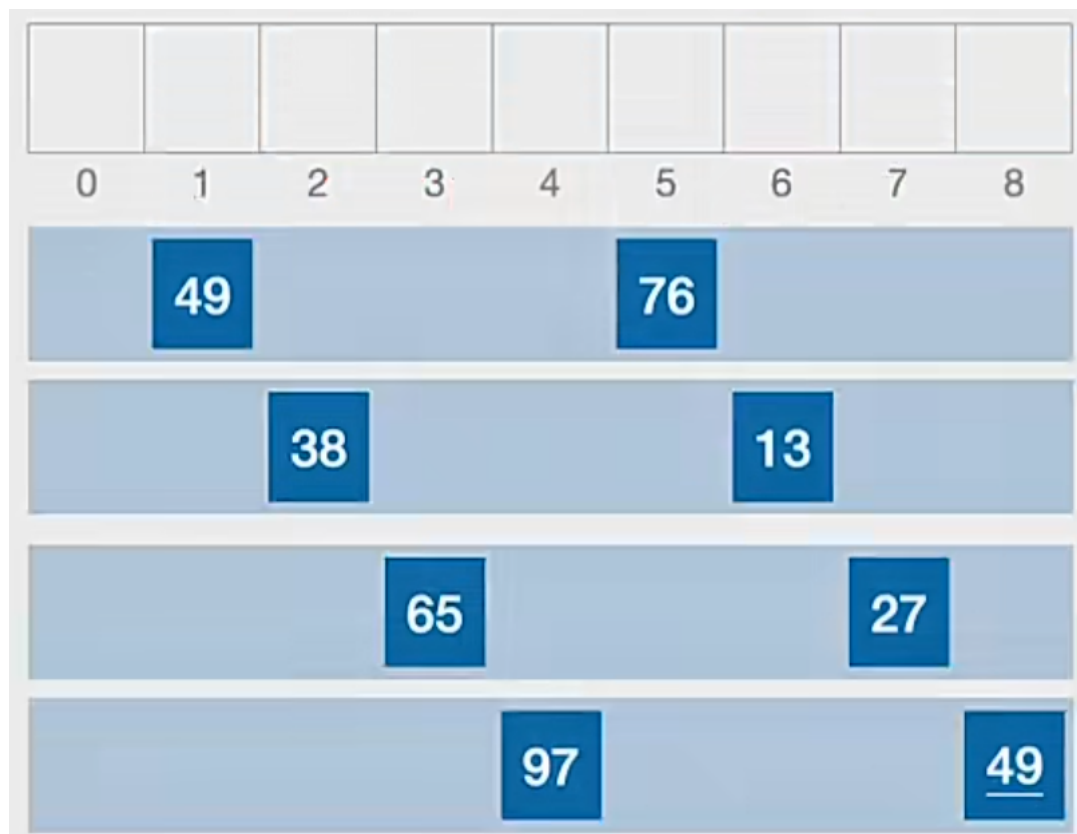
- 概念：
 - 移动元素的次数变少了，但是关键字对比的次数依然是 $O(n^2)$ 数量级整体来看时间复杂度依然是 $O(n^2)$

30.1.2 优化：折半插入排序

概念

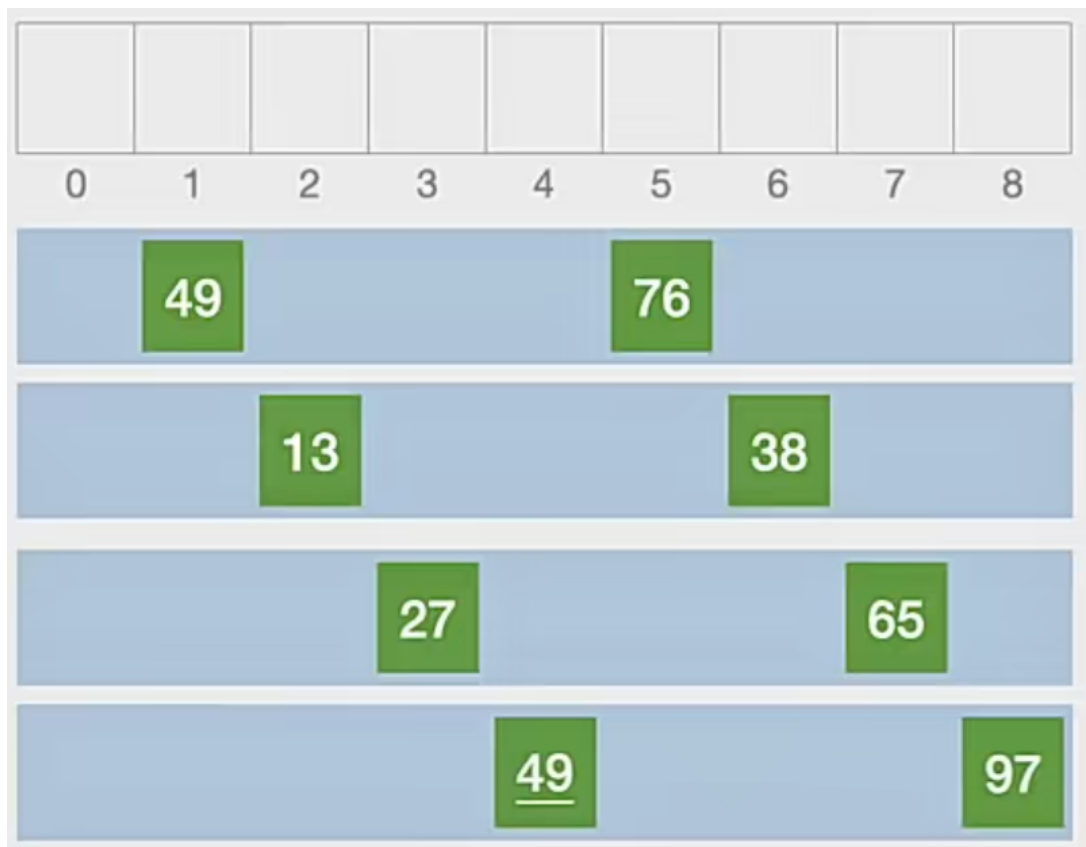
- 思路：
 - 先用折半查找找到应该插入的位置，再移动元素；
- 移动数据：
 - 当 $low > high$ 时折半查找停止，应将 $[low, i-1]$ （或者说是 $[high + 1, i - 1]$ ）内的元素全部右移，并将 $A[0]$ 复制到 low 所指位置；
- 注意：
 - 当 $A[mid] == A[0]$ 时，为了保证算法的“稳定性”，应继续在 mid 所指位置右边寻找插入位置；
- 性能：
 - 比起“直接插入排序”，比较关键字的次数减少了，其比较的次数为 $O(n \log_2 n)$
 - 但是移动元素的次数没变，整体来看排序的时间复杂度依然是 $O(n^2)$
- 适用：
 - 只适用于顺序存储的线性表；
 - 当数据量小的排序表，折半插入排序性能好；
- 图示：插入时，将所有元素右移；

- 但考研当中也会考其他的排序序列 $\rightarrow L[i, i + d, i + 2d, \dots, i + kd]$
- 适用性：
 - 只适用于**顺序表**，**不适用于链表**；
- 举例：
 - 假如当前有 9 个元素；
 - 第一趟： $d_1 = n/2 = 4$
 - 相距距离为 4 的元素在一个子表当中；
 - 处理前



- 处理后：

获取笔记持



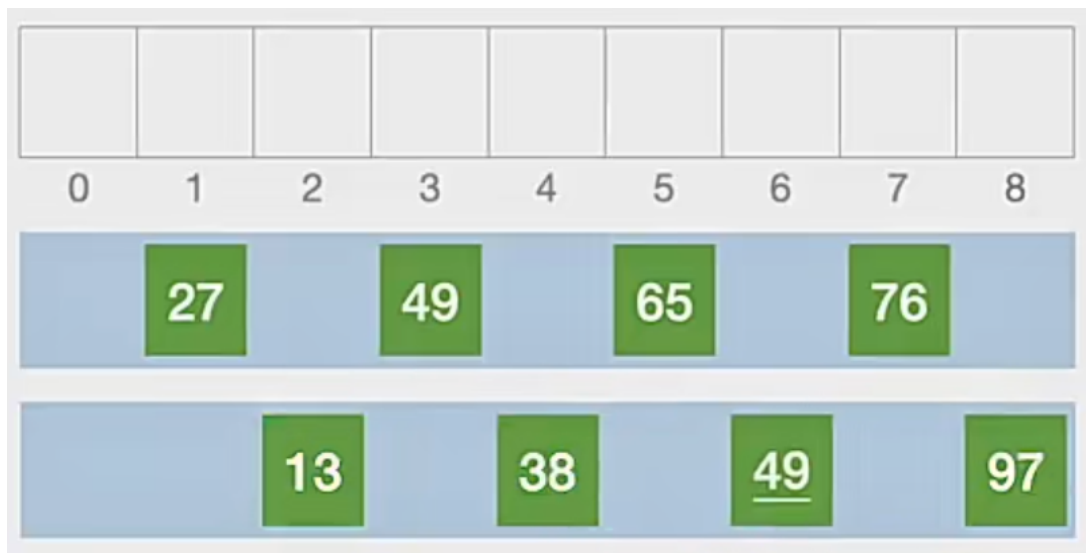
- 第二趟: $d_2 = d_1/2 = 2$



- 执行前:



- 对两个子表执行直接插入排序:



性能分析

- 空间复杂度：
 - $O(1)$
- 时间复杂度：
 - 和增量序列 $d_1, d_2, d_3, \dots$ 的选择有关，目前无法用数学手段证明确切的时间复杂度；
 - 最坏时间复杂度为 $O(n^2)$ ；
 - 当 n 在某个范围内时可达 $O(n^{1.3})$
- 稳定性：
 - 不稳定的；

30.2.2 代码实现

代码

```
//希尔排序
void ShellSort(int A[],int n){
    int d,i,j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d=n/2;d>=1;d=d/2)        //补偿变化
        for(i=d+1;i<=n;++i)
            if(A[i]<A[i-d]){        //需将A[i]插入有序增量子表
                A[0]=A[i];        //暂存再A[0]
                for(j=i-d;j>0&&A[0]<A[j];j-=d)
                    A[j+d]=A[j];  //记录后移，查找插入的位置
                A[j+d]=A[0];      //插入
            }//if
}
```