

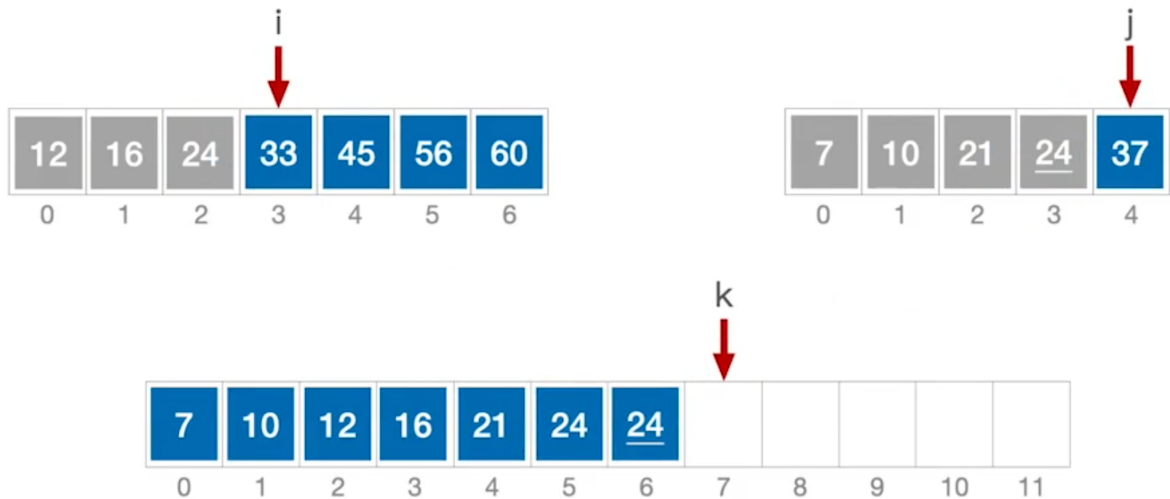
Lecture 33: 归并排序与基准排序

33.1 归并排序

33.1.1 基础概念

概念

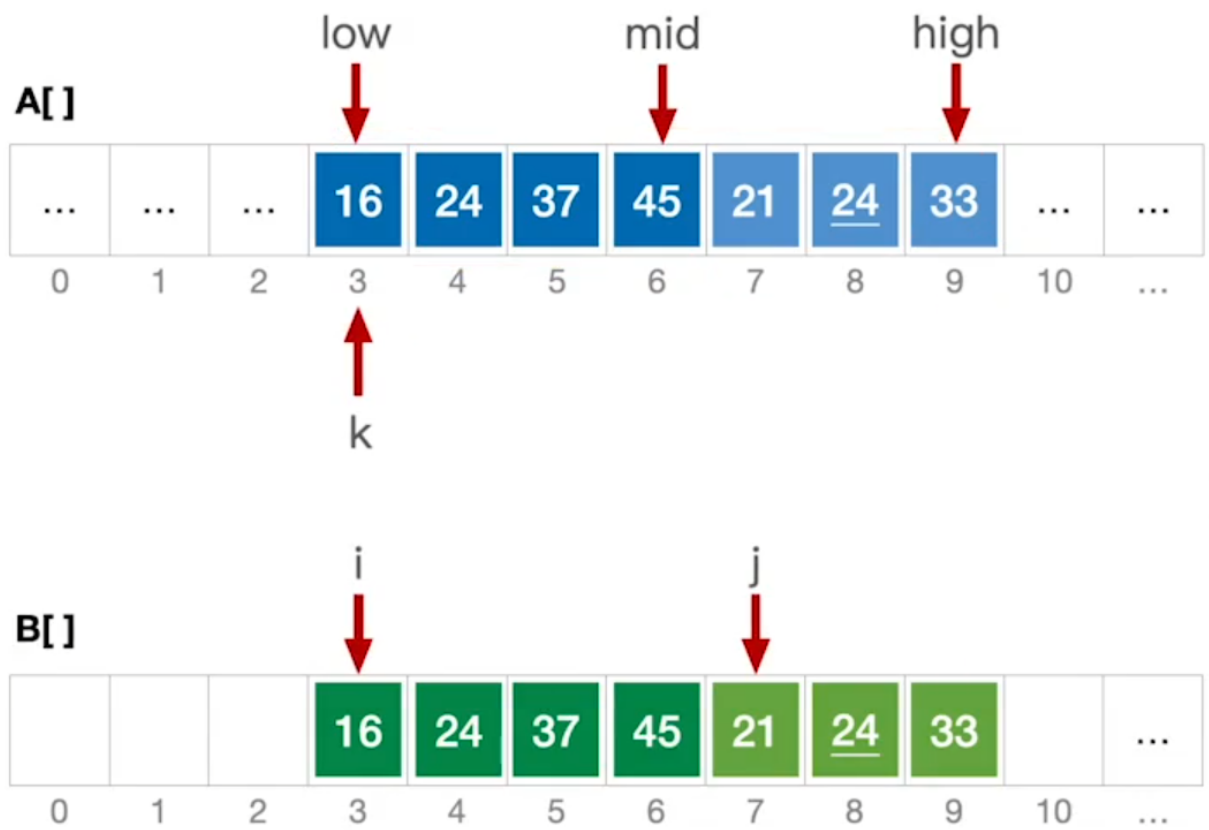
- 归并：
 - 把两个或多个已经有序的序列合并成一个；
- 实现：
 - 对比 i、j 所指元素，选择更小的一个放入 k 所指位置；
 - 只剩一个子表未合并时，可以将该表中剩余元素全部加到总表；
- 多路归并：
 - 二路归并 -> 每选出一个元素需对比关键字 1 次
 - 四路归并 -> 每选出一个元素需对比关键字 3 次 -> 对比 p1、p2、p3、p4 所指元素，选择更小的一个放入 k 所指位置；
 - 通常使用二路归并；
- 核心操作：
 - 把数组内的两个有序序列归并为一个；
- 图示：



33.1.2 代码实现

代码实现

- 指针图示：



```
int *B(int *)malloc(n*sizeof(int)); //辅助数组B
```

```
//A[low...mid]和A[mid+1...high]各自有序，将两个部分归并
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k]; //将A中所有元素复制到B中
    for(i=low,j=mid+1,k=i;i<=mid&& j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++]; //将较小值复制到A中
        else
            A[k]=B[j++];
    }//for
    while(i<=mid)
        A[k++]=B[i++];
    while(j<=high)
        A[k++]=B[j++];
}
```

- 分别对左右两部分递归的进行排序;

```
//完整代码(加上下面这一段)
void MergeSort(int A[],int low,int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A,low,mid); //对左半部分归并排序
        MergeSort(A,mid+1,high); //对右半部份归并排序
    }
}
```

```
    Merge(A, low, mid, high); //归并
} //if
}
```

33.1.3 效率分析

概念

- 概念：
 - 二路归并的归并树在形态上 就是一颗倒立的二叉树；
- 二叉树性质：
 - 二叉树的第 h 层最多有 2^{h-1} 个结点；
 - 若树高为 h ，则应满足

$$n \leq 2^{h-1}$$

- 总趟数：
 - 递归总趟数 = $\lceil \log_2 n \rceil$
- 每趟归并的时间复杂度：
 - $O(n)$
- 算法时间复杂度：
 - $O(n \log_2 n)$
- 空间复杂度：
 - 数组的大小 $\rightarrow O(n)$

33.2 基数排序

33.2.1 基础概念

基础概念

- 定义：
 - 假设长度为 n 的线性表中每个结点 a_j 的关键字 d 元组 $(k_j^{d-1}, k_j^{d-2}, k_j^{d-3}, \dots, k_j^1, k_j^0)$ 组成；
 - 其中： $0 \leq k_j^i \leq r-1$ ($0 \leq j < n, 0 \leq i \leq d-1$)， r 称之为基数；
- 过程：
 1. 初始化：设置 r 个空队列， $Q_{r-1}, Q_{r-2}, \dots, Q_0$
 2. 分配/收集：对 d 个关键字位分别做“分配”和“收集”
 - 分配：顺序扫描各个元素，若当前处理的关键字位= x ，则将元素插入 Q_x 队尾
 - 收集：把 $Q_{r-1}, Q_{r-2}, \dots, Q_0$ 各个队列中的结点依次出队并链接
- 图示：
 - 其中：百位为**最高位关键字**，个位为**最低位关键字**；

初始序列:



第一趟按“个位”分配、收集: 得到按“个位”递减排序的序列

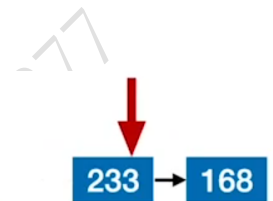


第二趟按“十位”分配、收集: 得到按“十位”递减排序的序列, “十位”相同的按“个位”递减排序

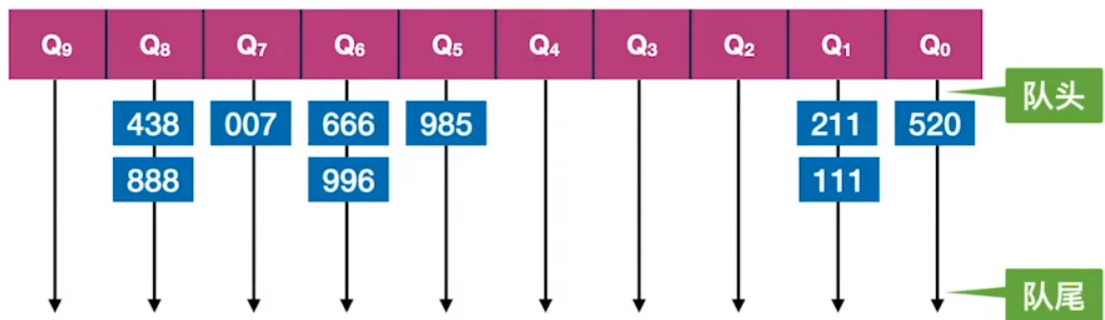


分配过程

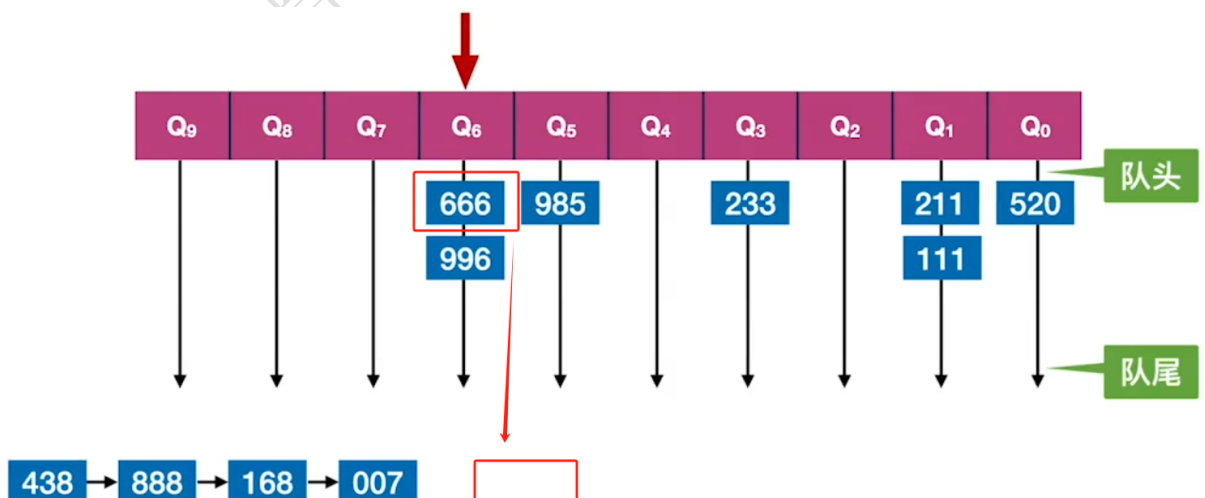
- 第一趟 -1 : 以 个位 进行分配



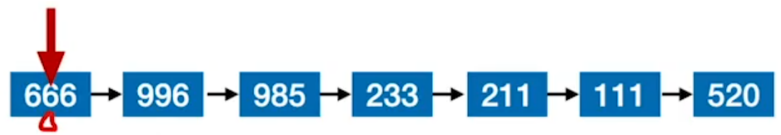
第一趟: 以“个位”进行“分配”



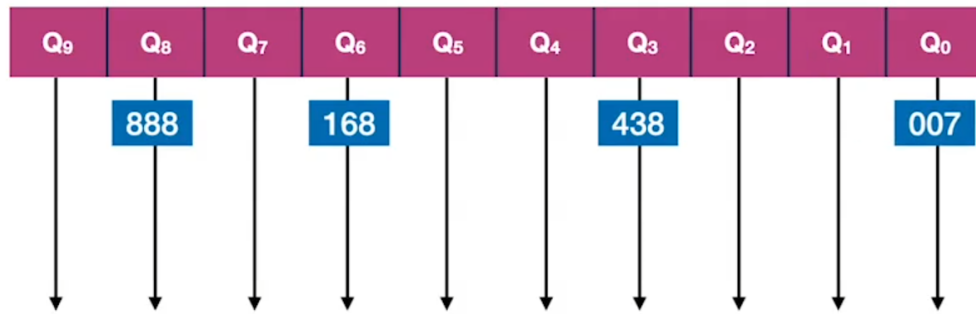
- 第一趟 - 2: 收集
 - 效果: 第一趟“收集”结束, 得到按“个位”递减排序的序列



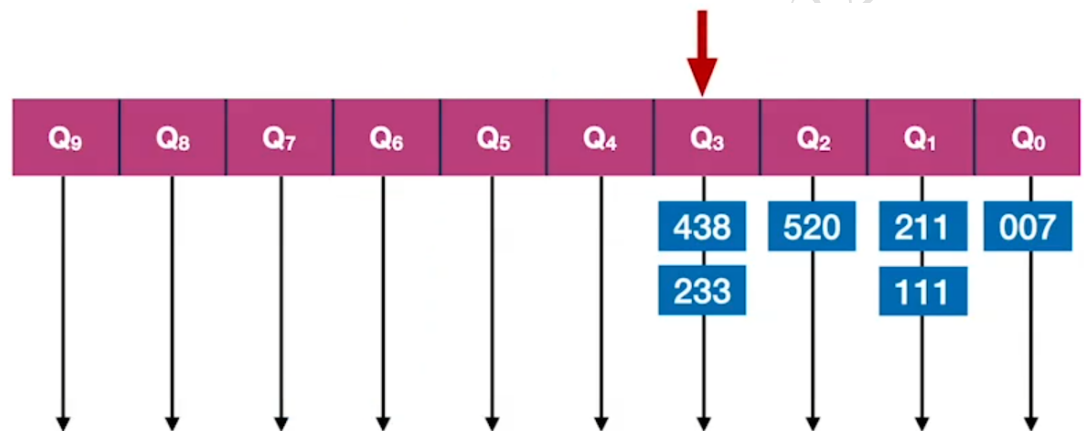
- 第二趟 - 1: 分配 -> 以 十位 进行分配
 - 因为第一趟的效果, 当十位数相同时, 总是个位越大的越先入队



第二趟：以“十位”进行“分配”



- 第二趟 - 2：收集
 - 第二趟“收集”结束 -> 得到按“十位”递减排序的序列，“十位”相同的按“个位”递减排序



- 第三趟 - 1：分配 -> 以 百位 进行分配
 - “十位”越大的越先入队，分配过程一致；
- 第三趟 - 2：收集
 - 得到结果：对百位进行排列，因此得到百位递减的序列。当百位相同时，又是按照十位递减进行排列。当十位相同时，按照个位递减进行排列；



33.2.2 效率分析

概念

- 实现：
 - 基于链式实现；
 - 使用指针连接在一起；
- 空间复杂度：
 - 空间复杂度 = $O(r)$
- 时间复杂度：

- 时间复杂度 = $O(d(n + r))$
- 稳定性:
 - 稳定性;

应用

- 1. 数据元素的关键字可以方便地拆分为 d 组, 且 d 较小
 - 反例: 给五个人的身份证号排序
- 2. 每组关键字的取值范围不大, 即 r 较小
 - 给中文人民排序 $\rightarrow$ 排序的种类非常多, r 很大;
- 3. 数据元素个数 n 较大
 - 给十亿人排序;

获取笔记持续更新可加微信: WandD2277