目录

[第1章 绪论 2](#_Toc169550944)

[第2章 线性表 3](#_Toc169550945)

[第3章 枚举与二分法 4](#_Toc169550946)

[第4章 递归和分治 4](#_Toc169550947)

[第5章 栈和队列 5](#_Toc169550948)

[第6章 二叉树 6](#_Toc169550949)

[第7章 树、森林和并查集 8](#_Toc169550950)

[第8章 字符串 8](#_Toc169550951)

[第9章 动态规划 9](#_Toc169550952)

[第10章 图的遍历和搜索 9](#_Toc169550953)

[第11章 图论基础应用算法 9](#_Toc169550954)

[第12章 排序 10](#_Toc169550955)

[第13章 查找 11](#_Toc169550956)

[第14章 贪心算法 12](#_Toc169550957)

### 第1章 绪论

**算法的时间复杂度及其表示法**

 **常数时间复杂度 O(1)**：算法的执行时间不随输入规模的变化而变化。

 **对数时间复杂度 O(log n)**：算法的执行时间随着输入规模的增加按对数增长。

 **线性时间复杂度 O(n)**：算法的执行时间与输入规模成正比。

 **线性对数时间复杂度 O(n log n)**：如常见的快速排序、归并排序等。

 **平方时间复杂度 O(n²)**：算法的执行时间与输入规模的平方成正比，如冒泡排序、选择排序等。

 **立方时间复杂度 O(n³)**：算法的执行时间与输入规模的立方成正比。

 **指数时间复杂度 O(2^n)**：算法的执行时间随着输入规模呈指数增长。（递归）

 **阶乘时间复杂度 O(n!)**：算法的执行时间随着输入规模的阶乘增长。

**数据的逻辑结构和存储结构**

#### 数据的逻辑结构

数据的逻辑结构指数据元素之间的逻辑关系，主要有以下几种类型：

1. **线性结构**：数据元素之间存在一对一的线性关系。
   1. **非线性结构：树、图、堆、哈希表**
2. **树形结构**：数据元素之间存在一对多的层次关系。
3. **图形结构**：数据元素之间存在多对多的关系。

#### 数据的存储结构

数据的存储结构是指数据在计算机中的表示和存储方式。主要有以下几种：

1. **顺序存储结构**：数据元素按照顺序依次存储在存储单元中，逻辑上相邻的元素在物理位置上也相邻。（连续）（数组）
2. **链式存储结构**：通过指针将数据元素链接起来，不要求逻辑上相邻的元素在物理位置上也相邻。（分散）（链表）
3. **散列存储结构**：根据数据元素的关键字，通过一定的散列函数计算出数据元素的存储地址。

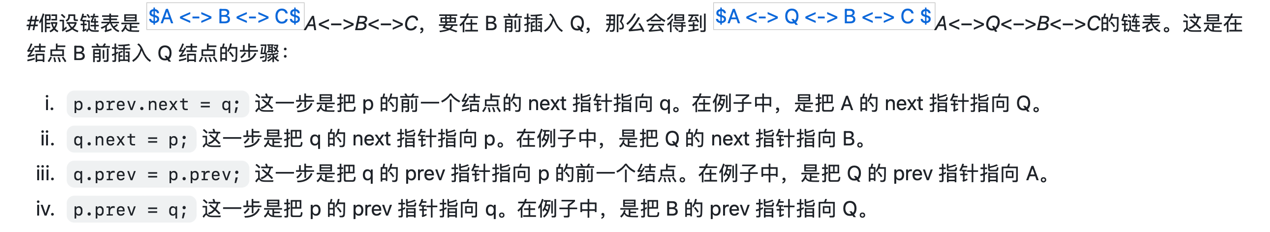
### 第2章 线性表

顺序表：

* 元素按照线性顺序存储，顺序表尾部的增删操作时间复杂度为 O(1)
* 插入新元素，平均需要移动（n/2）
* 合并两个顺序表，至少比较n次

链表：

* 单链表：每个节点包含数据和一个指向下一个节点的指针（插入新节点O(n)）
* 循环单链表：最后一个节点的指针指向头节点，从而形成一个环
* 双链表：每个节点除了指向下一个节点的指针外，还增加一个指向前一个节点的指针

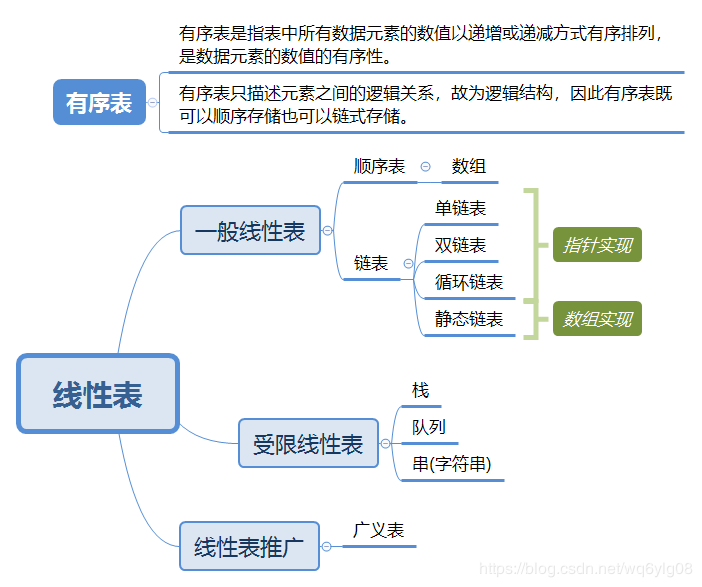


#### 使用顺序表的情况

1. **随机访问性能要求高**：顺序表支持 O(1) 时间复杂度的随机访问。
2. **空间利用率高**：顺序表不需要额外的存储空间来存储指针。
3. **数据量较少**：顺序表在数据量较少时效率较高。

#### 使用链表的情况

1. **频繁的插入和删除操作**：链表在插入和删除操作方面有更好的性能，因为不需要移动大量元素。
2. **动态大小**：链表在存储数据时更加灵活，可以根据需要动态增加和减少大小。
3. **避免内存浪费**：链表可以避免顺序表可能出现的内存浪费（如数组需要预分配空间）。



### 第3章 枚举与二分法

二分法是一种高效的查找和排序算法，用于在有序序列中查找特定元素或确定插入位置。

-通过反复将搜索范围缩小一半来快速定位目标值。

-定义left，right不断缩小范围来寻找目标值

**特点**

* 时间复杂度为 O(logn)，效率高。
* 适用于有序数据。
* 需要能够随机访问数据元素。

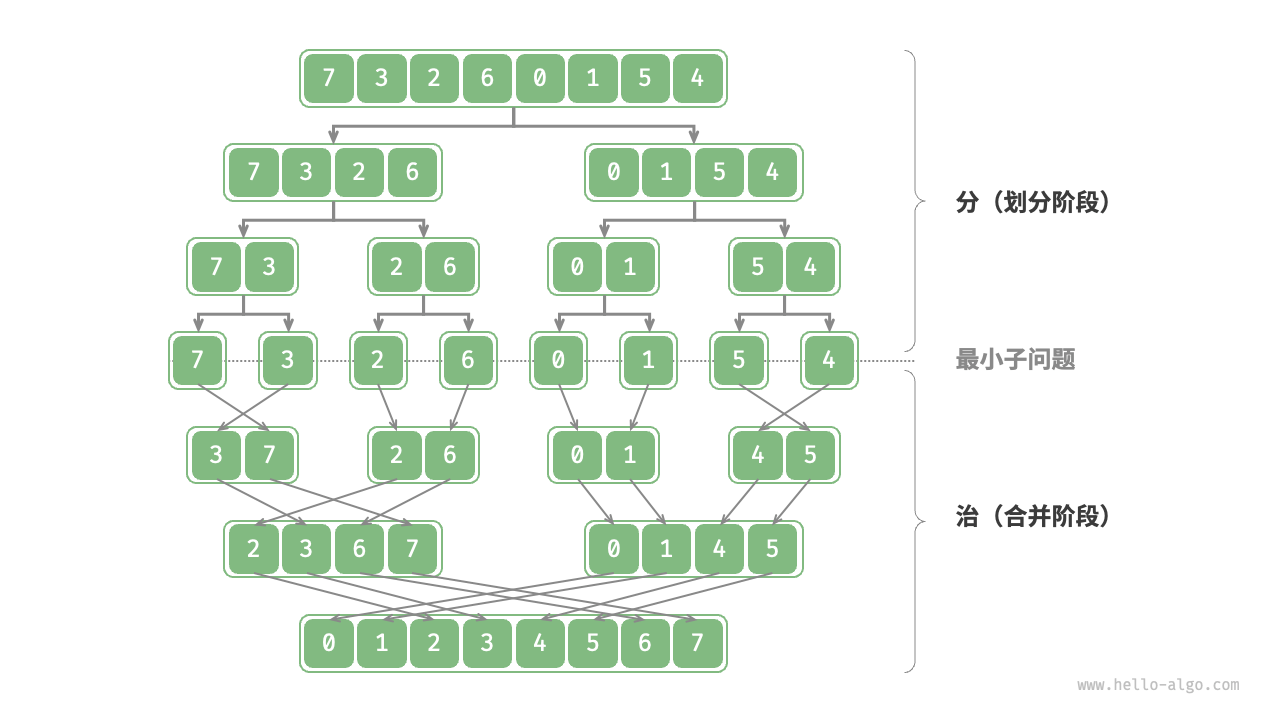
### 第4章 递归和分治

**递归**

需要基准情况来终止递归，否则会导致无限递归

可能导致栈溢出；可能存在重复计算，效率低

**分治**



适用于：问题可以分解、子问题是独立的、子问题可以合并

分：O()

治：O(n)

### 

### 第5章 栈和队列

后序表达式求值（逆波兰表达式）

·从左到右扫描

中序表达式转后序表达式,

·要有优先级（\*/优先于+-）

·准备一个栈存储操作符

·另一个存储数字

四则运算表达式求值

·换成后序表达式后就能计算了

普通队列：先进先出

循环队列：

 **入队**：在队列未满时将元素添加到队列尾部。

 **出队**：在队列不为空时从队列头部移除元素。

* Length = rear – head + 1

### 第6章 二叉树

* 完美二叉树：2^(h+1)– 1个节点 \*h为高度
* 满二叉树：没有度为1的节点
* 完全二叉树：除了叶子节点全部节点都是满的，而且叶子节点有缺，一定在右子树

**二叉树**

* 高度：n-1
* 深度：n（和层级一样）
* 第i层有2^i个节点（找全部节点数量的话要用等比公式）
* 节点数n，边的数目为n-1
* n个节点的非空二叉树最少有层的节点
* 节点总数 = 度为2的节点个数 + 1
* 满二叉树叶节点数目 = 分支节点数目 + 1
* 边 = 节点

**完全二叉树**

* 度数为1的节点为0或1
* 有n个节点的二叉树有(n+1)/2个叶节点
* 下标为i的节点，其父节点是(i-1)//2 \*层次遍历的形式

前序遍历：根结点+左子树+右子树

后序遍历：左子树+右子树+根节点

中序遍历：左子树+根节点+右子树

\*复杂度O(n)

**堆**

·最大堆：

* arr[i]≥arr[2i+1]（如果左子节点存在）
* arr[i]≥arr[2i+2]（如果右子节点存在）

·最小堆

* arr[i]≤arr[2i+1]（如果左子节点存在）
* arr[i]≤arr[2i+2]（如果右子节点存在）

·数组表现形式是层次遍历

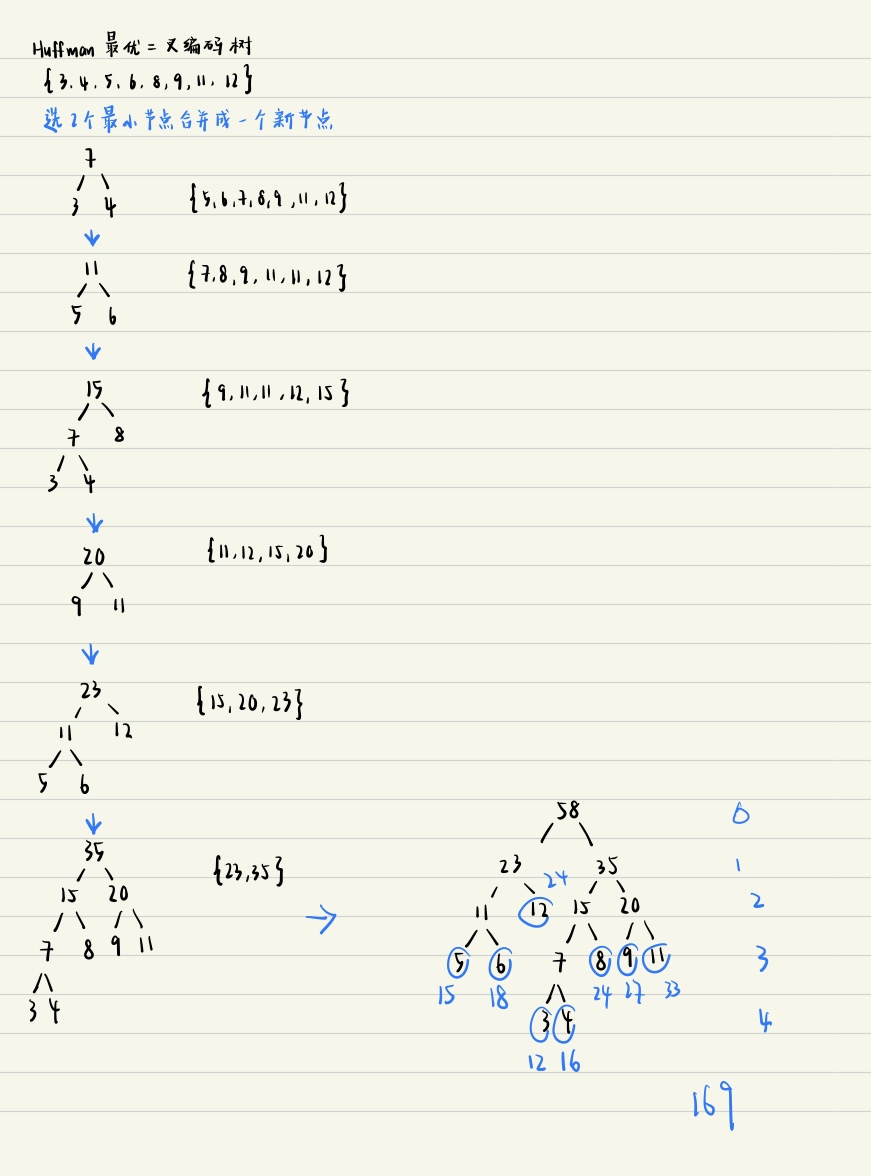
**哈夫曼树**

最优二叉树

·带权路径长度最小的二叉树

·从大到小（根结点是最大权重）

·选两个最小节点合并成新节点



### 

### 第7章 树、森林和并查集

**树**

·每个节点可以有任意多棵不相交的树

·不分左右子树（所以二叉树不是树）

·不能进行中序遍历

·节点度数最多为k，第i层最多k^i个节点

·节点度数最多为k，高为h时，最多有(k^(h+1)-1)/(k-1)

·n个节点有n-1条边

·总节点数 = 2\*边

·所有节点度数 = 总边数 = 总节点数-1

\*无环的无向图

直观表示法：每个节点有一个变量存放数据，加上一个可变长列表存放所有子节点指针

儿子-兄弟表示法：二叉树表示的树

\*树的后序遍历和儿子-兄弟树的中序遍历一致

**森林**

**·**不相交的树的集合

### 第8章 字符串

**暴力匹配算法**

暴力匹配算法是最简单直接的字符串匹配方法，它的基本思想是从主串的每一个可能的位置开始，尝试匹配模式串，如果匹配失败，则将模式串向右移动一位，继续匹配，直到找到匹配的位置或者遍历完主串。

* **算法步骤**：
  1. 从主串的第一个字符开始，依次与模式串的第一个字符进行比较。
  2. 如果匹配成功，则继续比较主串和模式串的下一个字符，直到模式串全部匹配成功。
  3. 如果匹配失败，则主串的位置向右移动一位，再从该位置开始重复步骤1。
* **时间复杂度**：最坏情况下时间复杂度为 O(m⋅n)O，其中m是主串的长度，n是模式串的长度。

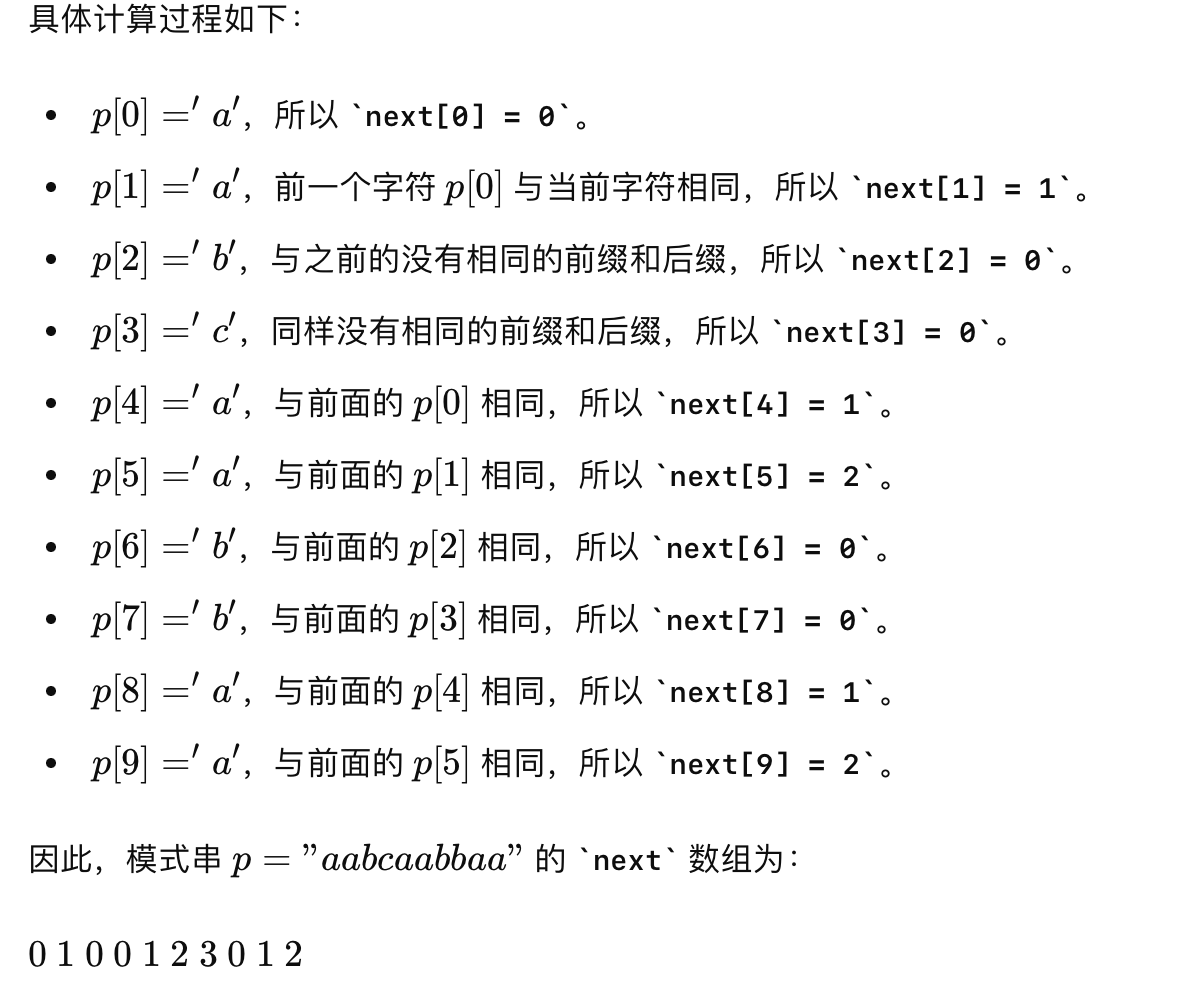
**KMP字符串匹配算法**

KMP算法通过预处理模式串构建Next数组，利用这个数组避免在匹配过程中重复比较已经比较过的字符，从而提高匹配效率。

**基本Next求法**

基本的Next数组求法是指直接根据模式串构建一个Next数组，该数组用于在匹配过程中指导匹配的位置移动。

* **算法步骤**：
  1. 构建Next数组：从模式串的第一个字符开始，依次计算每个位置的最长相同前缀后缀的长度。
  2. 匹配过程中，利用Next数组指导主串和模式串的移动。
* **时间复杂度**：构建Next数组的时间复杂度为O(n)，匹配过程的时间复杂度为O(m+n)，其中m是主串的长度，n是模式串的长度。



代码实现：



### 

### 第9章 动态规划

### 第10章 图的遍历和搜索

**图的概念**

**顶点**

总度数+1

**边**

有向图：(n-1)/2

无向图：n(n-1)/2

顶点度数/2

**强连通有向图**

图中任意两个顶点互相可达

**强连通分量**

图中的一个极大子图，其中的任意两个节点都可以通过有向路径互相到达。强连通分量可以理解为有向图的极大连通子图。（连通分量是无向图）

**邻接矩阵**

适用于数目达到O(n^2)的稠密图

复杂度O(顶点数^2)

**邻接表**

适用于稀疏图

复杂度O(边数+顶点数)

### 第11章 图论基础应用算法

**单源最短路问题的Dijkstra算法**

图中所有边的权重须为非负

采用贪心算法

1. 初始化距离数组，记录源点到每个顶点的当前已知最短距离。
2. 选择距离最短的未访问顶点，更新其邻接顶点的最短距离。
3. 重复以上步骤，直到所有顶点都被访问，或者确定源点到某个顶点的最短路径。

**求每对顶点之间最短路的Floyd算法**

采用动态规划算法

1. 初始化路径矩阵，记录每对顶点之间的直接路径或者无穷大（表示不可达）。
2. 通过不断更新路径矩阵，从而逐步求解所有顶点对之间的最短路径。

**最小生成树Prim算法**

用于在有无图中找到最小生成树（权值最小）

采用贪心算法

1. 选择一个起始顶点作为生成树的一部分。
2. 不断从已经加入生成树的顶点集合中选择与其相邻且权值最小的边加入生成树，直到生成树包含了所有顶点。

**最小生成树Kruskal算法**

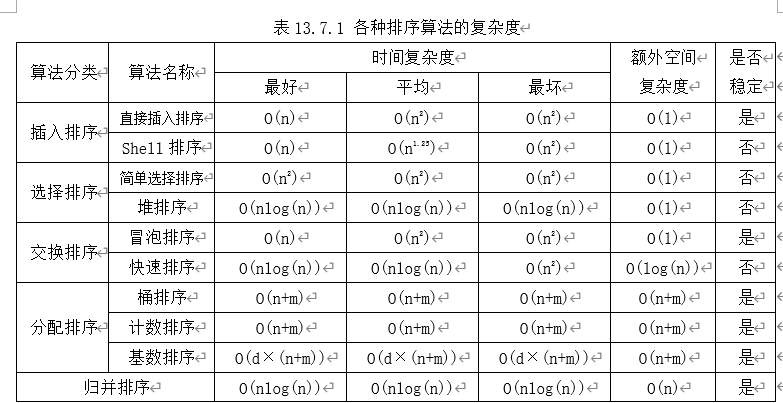
1. 将所有边按权值排序。
2. 逐条选择权值最小的边，若加入该边不形成环，则加入生成树。

**拓扑排序**

1. 计算每个顶点的入度（即指向该顶点的边的数量）。
2. 将入度为0的顶点加入结果序列，并移除其出边。
3. 重复以上步骤，直到所有顶点都加入序列或者无法再移除入度为0的顶点。

### 第12章 排序

**稳定性定义：**对于待排序序列中的任意两个相等的元素，假设它们在原序列中的位置分别是i和j，且i<j，如果第一次排序后它们的相对顺序仍然是i<j，则该排序算法是稳定的。



直接插入排序：每次抽一个出来对比，放到相对应的位置

希尔排序：插入排序的优化，先把列表分成len(list)/2组，一一对比后再进行排序插入

选择排序：数据规模越小越好，先选最大or最小，依次进行选择排序（比较次数：n（n-1）/2

堆排序：先构成大顶堆，将根结点与末尾元素交换，再反复操作（升序）

冒泡排序：一次比较两个元素，如果顺序错误就交换

快速排序：挑选基准值，把小于基准值的元素放到前面去，递归的对小于基准值元素的子序列和大于基准值元素的子序列排序。（相同数字不定）最后，（从右开始找小于基准值的元素），（从左开始找大于基准值的元素）交换两个的位置

桶排序：遍历数组，把每个元素分类到各个桶里，根据桶的大小顺序排序，再合并结果

记数排序：\*只用于非负数和较小的数据，统计每个数值出现多少次存在桶里

基数排序：先跟着最后一位数分配到对应的桶，再根据前一位数分类，进行排序

归并排序：不断分成每组剩两个元素，再逐个合并比较

### 第13章 查找

**平均查找长度ASL的计算**

**= 查找概率x比较次数**

1. **无序表的查找概率**：
   * 如果元素是均匀分布的，即每个元素被查找的概率相等，那么查找概率可以简单地用 p=1/n​，其中n是表中元素的总数。
2. **有序表的查找概率**：
   * 对于有序表，通常要根据具体的查找算法和具体问题来计算。例如，二分查找中每次查找的成功概率可以通过元素分布和二分法的性质来计算。
3. **散列表的查找概率**：
   * 在散列表中，查找概率通常依赖于哈希函数的设计和键的分布。如果哈希函数均匀地将键映射到表的槽位，且负载因子适中，那么每个槽位的查找概率大致相等。
4. **无序表的平均比较次数**：
   * 在无序表中，平均比较次数取决于查找元素在表中的位置分布和查找算法。如果元素均匀分布，则平均比较次数为(n+1)/2。
5. **有序表的比较次数**：
   * 对于有序表，常见的查找算法有顺序查找和二分查找：
     + **顺序查找**：平均比较次数为(n+1)/2​。
     + **二分查找**：平均比较次数为，其中n是表中元素的总数。
6. **散列表的比较次数**：
   * 在散列表中，比较次数通常是通过哈希函数计算出的槽位后，进一步进行的比较次数。在理想情况下，如果散列函数均匀且负载因子适中，平均比较次数可以近似为常数。

**二分查找,二叉查找树**

* 二分查找是一种高效的查找算法，适用于已经排好序的列表。它通过将待查找区间不断缩小一半的方式来查找目标值。
* 时间复杂度为 O()，其中n是元素个数。

二叉查找树是一种二叉树结构，每个节点最多有两个子节点，且左子树的节点值小于当前节点，右子树的节点值大于当前节点。

 **插入操作**：将新节点插入到二叉查找树中，保持二叉查找树的有序性。

 **查找操作**：按照二叉查找树的性质进行查找，平均时间复杂度为 O()，最坏情况下可能退化为 O(n)。

**散列表(线性探查，双散列探查是重点，影响散列表效率的因素)**

* 散列表是一种通过哈希函数将关键字映射到表中一个位置（索引）的数据结构，以实现快速插入、删除和查找操作。

**冲突处理**：

* **线性探查**：发生冲突时，依次向后探查，直到找到空槽。
* **双散列探查**：使用第二个哈希函数来计算探查步长，避免线性探查中的聚集效应。

**影响效率的因素**：

* **哈希函数的选择**：影响键的分布和冲突率。
* **负载因子**：哈希表中已存储元素的数量与哈希表大小的比率，过高会增加冲突。
* **冲突解决策略**：直接影响查找性能和插入性能。

### 第14章 贪心算法