- 学习路线：（1）体系班1-40节（2）进阶班leetcode高频题目（49-78节）（3）进阶班经典题目（4）大厂刷题
- 学习思路：（1）自己能给新人讲会这道题（2）题目核心点二三句话的总结

复杂度、对数器、二分法

复杂度

- 常数（固定）时间操作：1.加减乘除；2.寻址操作（eg：数组寻址）
- 拆分算法流程：拆分出来的所有行为都是常数时间的操作
- 额外空间复杂度
- 常数项优化

对数器

- 一种检验算法正确性的思路
- 对数器的意思是，用随机样本来测试已写的未知正确性的算法A
- 前提是有个容易实现（最好是已知正确性）的算法B，对比A、B的结果

二分法

- 二分查找

```
trick：数组中点
min = (lo + hi)/2
min = lo + (hi-lo)/2  这种写法更好，防止lo+hi相加溢出
min = lo + (hi-lo)>>1
```

- 局部最小值

异或运算

- 异或：无进位相加，不要进位的相加

```
7:  0000 0111
13: 0000 1101
    0000 1010
```

```
0 ^ N = N
N ^ N = 0
```

- 异或满足交换律和结合律，异或跟顺序没什么关系，无进位相加嘛！
- 应用一：两数交换

```
a = a ^ b
b = a ^ b
a = a ^ b
```

- 注意：上述异或交换前提条件是，a,b指向内存的不同两块区域，如果a,b指向同一区域的数，那么就刷没了

```
int[] arr = {1,2,3}
int i = 0
int j = 0
arr[i] = arr[i] ^ arr[j]
arr[j] = arr[i] ^ arr[j]
arr[i] = arr[i] ^ arr[j]
最后打印 arr[i] arr[j]都为0
```

基础数据结构

- 链表：单链表翻转、双链表翻转、删除链表中某个value
- 链表一定要有指针的概念，一般都会有个可移动的辅助指针
- 指针链表的赋值要注意：如果我对pre或者cur有结构性的删除更改时，是对head指针的指向有影响的，但是pre=pre.next这种，是对head节点没啥影响的

```
Node pre = head;
Node cur = head;
```

- 两个栈实现队列
- 两个队列实现栈
- 拓展：BFS、DFS用栈、队列去实现
- 任何递归都可以改成非递归的形式
- 递归算法分析：写/读递归代码，一定要有系统栈的概念，方法压栈（同时保存数据）
- 递归算法分析：也可以直接画一层层调用的逻辑图
- 新手分析递归脑图一定要画
- Master公式，这玩意不知道怎么推理来的，记下来有什么用?
- Hash表增删改查都是O(1)复杂度，无论数据量大小。其实还是跟样本数据量有点关系的，比如，增删改查一个字符串，"abcxxxxx"，那么时间复杂度和字符串的长度有关o(k)
- HashMap只有key就相当于是HashSet
- TreeMap有序表（接口名）：红黑树、AVL、sb树、跳表。增删改查复杂度：O(logN)
- TreeMap传入的Key如果不是Integer、String这种类型的话，而是自定义对象的话，那是需要比较器的，不然怎么知道怎么排序

归并排序相关

- 迭代实现归并的思路可以看出来时间复杂度确实是O(N)*log(N)

- merge sort把比较这事变得有序，这样就能省好多时间
- 单调性，不回退
- 两个有序块，两指针，保证两指针只滑动一次，不回退，这是归并相关题目的关键
- LeetCode 327 思路关键：其实脑海里有根数轴就很好理解

> 假设 0-i 的整体和是x，求目标以i结尾的子数组有多少个在[lo,hi]上 == > 等同于求i之前所有前缀和中有多少个前缀和在[x-hi,x-lo]上

> 还有一点就是一个数组的任意区间和，可以转换为rangeSum数组的前后两个值相减

```
long[] rangeSum = new long[arr.length];
int sum = 0;
for (int i = 0; i < arr.length; i++) {
    sum += arr[i];
    rangeSum[i] = sum;
}
```

随机快速排序

- 荷兰国旗问题
- 随机快排的空间复杂度：o(logN)
- 注意随机快排和快排是有区别的

比较器

- 记住：返回负数第一个参数在前面、返回正数第二个参数在前面、返回0无所谓
- 有序表：TreeMap TreeMap特性：如果key比较判定相同，则有去重功能，后面put的kv，就put不进去，不会加重复的key

堆和堆排序

- heap -> priority queue（优先级队列）
- 前置概念：完全二叉树
- 数组的位置去填充完全二叉树居然还有这规律：

> i 的左孩子位置：2*i+1

> i 的右孩子位置：2*i+2

> i 的父亲位置：(i-1)/2

- 堆：（1）完全二叉树
- 大根堆：以任何一个节点为根节点的树，根节点的值最大
- 小根堆：以任何一个节点为根节点的树，根节点的值最小
- 往数组中填数的时候构建堆结构

- JAVA里面有这种数据结构 PriorityQueue （堆），默认是小根堆。自定义一个逆序的比较器，变成大根堆
- heapInsert/heapify 都是log(N)的复杂度，想象二叉树的树高
- 堆排序：1、建堆；2、堆数据交换

加强堆

- 小经验：LeetCode 评测能过的指令条数约$10^{8\sim10}9$，JAVA花费时间大致2-4s，那么就可以看评测的数据量来推测用什么时间复杂度的算法
- 反向索引表：在数组里，我们通常是由index -> value，没有从value ->index，由value -> index这个就是反向索引表

前缀树

- Trie Tree
- AC自动机 -> 前缀树+KMP

桶排序

计数排序

- Count Sort，桶排序的一种，利用容器的思想
- 基于比较的排序：冒泡、插入、选择、归并、快排、堆、希尔
- 不基于比较的排序（必须是数据范围特殊）：桶排序（计数），时间复杂度可以达到O(N)

基数排序

- 非负数，十进制，都属于桶排序
- RadixSort实现方式有一种非常trick的实现方式，不用桶，用一个出现次数的累加和数组就能实现

```
021,010,111,022,011,012
count[1,3,2,0,0,0,0,0,0,0]     //个位数出现次数的累加和
count'[1,4,6,6,6,6,6,6,6,6]     //前面数组的前缀和，意义：个位数<=的数一共有几个
```

排序算法的稳定性

- 稳定性：指同样大小的样本再排序后不会改变相对次序