# 一、 数据结构

数组、链表、队列、栈 （链式结构）

堆、树 （树状结构）

哈希

图

## 1. 堆：

大顶堆、小顶堆

### 复杂度

堆排序的复杂度：

建堆(nlog(n)) + 排序(O(n))

**题目：**

1. 选取最大的（最小）top K 元素（top k 个最大的元素）:

1. 剩下最大元素**（小根堆）**， **堆中的元素**是需要采集的：

使用k +1个元素建立**小根堆**，时间复杂度为 O((k+1)log(k+1))

使用 n-k-1 个元素替换根，调整(n-k-1)log(k+1)，最终剩下的k+1个元素是最大的k+1个元素， 然后进行堆排序.(k+1)log(k+1) + O(k+1)

1. 挑选出k个元素**(大根堆)**， **堆顶元素**是需要采集的:

使用n-k+1 个元素建立大根堆O((n-k+1)log(n-k+1))，采集堆顶元素。

用k-1 个元素去替换堆顶元素，然后调整。 O((k-1)log(n-k+1))

对k个元素进行堆排序。O(klogk)+O(k)

2.

## 2. 树

1. 完全二叉树
2. 满二叉树
3. 平衡二叉树

任意两个子树的最大高度差是1

红黑树、AVL（自平衡二叉查找树，例如红黑树）、替罪羊树、Treap、伸展树等

1. 二叉查找树

左子树 < 根 < 右子树

1. 红黑树

平衡二叉树 + 二叉查找树。

因为二叉查找树可能会退化成一个链，查找和插入效率会变的很低，而红黑树的搜索、插入、删除的效率是O(log n)。

1. B树：

B(B-)、B+、 B\*

是一个多路查找平衡树，树的高度比二叉平衡排序树低，所以查找节点时间比较小。每个节点可以用来描述外部存储（例如磁盘块块儿），所以B树适合读写数据块相对比较大的的存储系统，例如文件系统索引、数据库索引。

红黑树的节点就是单个记录，但是磁盘块儿中的记录无法单独拎出来作为单独的节点，所以，红黑树不适用于文件系统和数据库系统。

## 3. 图

## 4. Java 的数据结构

List(ArrayList、LinkedList)、Set（HashSet、TreeSet）、Map（HashMap，TreeMap、ConcurrentHashMap）、Tree、Stack。

# Note:

元素加时考虑会不会溢出：整型的的范围：-2147483648（-231） 至2147483647（231-1）

有关数组操作，小心下标越界，**i++--**

**循环时造成死循环：**

**private int** binarySearchForLastBE(**int**[] nums, **int** num){

**int** len = nums.length;

**if** (len == 0)

**return** 0;

**int** i = 1;

**int** j = len;

**while** (i<=j){

**int** mid = (i+j)/2;

**if** (nums[mid-1]>=num){

j = mid; //*因为这里不是* *j = mid-1,所以在i=j时如果没有下面的if判断，就会造成死循环。*

**if** (i==j)

**break**;

}**else**{

i = mid + 1;

}

}

**return** i-1;

}

# 二、 算法

## 1. 排序算法

### 比较类排序：

### 交换排序

### 插入排序

### 选择排序

### 归并排序

### 非比较类排序：

### 计数排序：

### 问题描述：int[] nums 有n=50000个元素，每个元素的取值范围

### k [-50000,50000]=>[0,100000],对他们进行排序。

### 方法：使用一个数组res = int [100000]，将 nums 的元素作为下标映射到 res 中， 这样映射的时候就排好了序。

### 空间复杂度：O(k)

### 计数排序中的关键问题是res的数组大小为nums中最大的数。所以空间复 杂度跟nums 的取值范围相关的，即与K相关。

### 时间复杂度：O(n+k)

### 遍历res中的所有元素(O(k)) + 实际有用的的个数(n) 。

### 总结：

### 速度比任何的比较排序都快。

### 适合整数排序。

### k的取值范围和n接近 ，或者nums 元素取值比较集中的情况，或者nums 取值比 较均匀。

### 桶排序：

### 使用k个桶，桶之间是有序的，然后对桶中的元素排序，把桶中的元素串联起来。

## 2. 查找算法

查找指定的元素：

有序：

* 折半查找
* 插值查找(分布很均匀)：

对折半查找的一种升级，折半查找总是查mid=(low+high)/2=low+1/2\*(high-low)，

而插值查找，则根据key与mid的关系适当得上下波动一下：

mid = low + (key-a[low])/(a[high]-a[low]) \* (high-low);

* 斐波那契查找：？？

无序：

顺序查找O(n)

对元素进行排序：列表、查找树、平衡查找树（红黑树、B、B+）

Hash

其它：

查找

## 3. 动态规划( Dynamic Programming)

## 动态规划、深度遍历、广度遍历、贪婪算法、回溯、联合表、分治、Bit Manipulation、滑动窗口、Trie、迭代、Segment Tree、Ordered Map、Binary Indexed Tree、拓扑排序、Random、BrainTeaser、Geometry、Line Sweep、二叉搜索树、Rejection Sampling、Reservoir Sampling、Memorization、Rolling Hash、Suffix Array.

# 题目：

## 字符串与数组