算法基础：

- 把输入称为 n，一定是在有限时间内结束计算的
- O（n）时间复杂度大O计法，O（1）就表示一个元素的一个操作，O（n）表示需要遍历一次所有元素，进行一次操作
- O（log2^n）对数阶 --log2^n表示遍历多少个就能达到n
  - **O(1) < O(logn) < O(n) < O(nlogn) < O(n2) < O(n3) < O(2n) < O(n!) < O(nn)**
- **列表形式**
  - **一般Index() --O(1),而pop(i)---O(n) 因为第一步是查找，然后是把后面重新补上去**
- **字典**
  - **copy ---O(n) 需要一个个复制过去，查找删除都是 O（1）因为有键的存在**

顺序表：

- 一个整型或者内存地址就是4个字节，一个字节就是8位二进制(对于32位系统来讲)
- 获取一个列表就是一个执向着连续整型的内存空间地址
  - 有一体式和分离式，推荐表头(存着地址)和数据空间分离
  - 数据空间有一体和元素外置，元素外置可以存不同类型数据，因为里面都是地址
- Python列表和tuple

- 使用顺序表，队尾插入，时间复杂度O(1)；队头插入，O(n)
- 采用元素外置，所以列表可以存任何类型数据
- 采用分离式存储，扩容增加后list的id值不变
- 存储策略：
  - 达到阈值时候，变成一倍一倍，是为了减少空间浪费，前面是为了减少时间耗费

链表：与顺序表称为线性表

- 单向链表
  - 需要有个头节点引入第一个节点位置，每个节点存着元素elem和下一个节点地址next，尾节点next为空
- 单向循环链表
  - 尾节点next指向头节点位置
- 双向链表
  - 节点还多了个pro--前节点地址
- 与顺序表对比
  - **删除或者插入尾节点都是需要遍历到最后的，因此O（n）**
  - **访问元素也是需要每个节点都去遍历，while 递归寻找、**
  - **链接可以节省空间，空间灵活，但是操作时间复杂度都较大，效率较低**

栈：

- 先进后出

队列：

- 先进先出
- 双端队列，两头都可进可出

排序算法：

- 冒泡算法：
  - 从第一个开始，比较相邻元素，比它大就交换
  - 对每一对相邻元素作同样工作，直接结尾

- 选择排序：
  - 取第一个开始，比较后面每个，如果是最小就交换位置--通过指针（索引）
  - 然后对列表每个进行
- 插入排序：
  - 把列表第一个作为有序开始，列表后面无序的往里面插入
  - 外层循环陆续选择每一个无序插入，内层负责无序在有序中的排序
- 希尔排序
  - 按下表的一定增量分组，对每组采用直接插入排序算法
- 快速排序：
  - 选出中间基准元素
  - 分区操作，比基准小的放前区，大的放后区
  - 递归把两个子数列排序
  - 递归退出条件：起始位置大于=末位置
- 归并排序
  - 先递归分解数组
  - 数组分解最小之后，合并两个有序数组--使用左右指针，谁小先把谁放进结果里，一层一层返回

排序方法	平均情况	最好情况	最坏情况	辅助空间	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	O(1)	稳定
希尔排序	$O(n \log n)$~ $O(n^2)$	$O(n^{1.3})$	$O(n^2)$	O(1)	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	O(1)	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$~ $O(n)$	不稳定

二分查找：

- 思路：
  - 递归调用，通过索引//2，判断大小，走左边还是右边
  - 注意找不到元素的条件
  - 时间复杂度
    - 最优时间复杂度：O(1)
    - 最坏时间复杂度：O(logn)