Java中的HashSet/HashMap，C++中的unordered\_map/unordered\_set，Python中的dict/set都是常用的哈希数据结构

哈希表的时间复杂度O(size of key)。

哈希函数：将key计算成一个固定且无规律的整数。

重哈希(Rehash)：当存入的元素个数与哈希表的容量到达一定的比例时，就需要重新分配哈希表的容量，以避免较多的冲突。这个比例不太固定，可以是10%，30%或者50%。

二叉堆是一颗完全二叉树（每一层都从左到右填满，最底层例外），它的树高就是，主要分为小顶堆和大顶堆。

小顶堆：父结点的值总是不大于它的孩子结点的值。

大顶堆：父结点的值总是不小于它的孩子结点的值。

如果从根结点开始编号，根结点编号为0，则编号为i的结点，其左孩子的结点编号为2\*i+1，右孩子的结点编号为2\*i+2。

二叉堆插入元素时，首先把元素插入到最后的位置，然后执行上滤操作，由于上滤操作每次只和它的父亲交换，所以调整次数就是树的高度，时间复杂度为。

二叉堆删除堆顶的元素时，首先将堆顶元素和堆中最后一个元素进行交换，然后将其删除掉，之后要将交换到堆顶的元素执行下滤操作，因为其不一定满足堆的性质，调整的次数也是树高，时间复杂度为。

二叉堆删除其它位置的元素时，时间复杂度一般为。因为它需要遍历整个堆来查找到元素的位置。如果希望删除操作都是，就需要有一个辅助的hash map，它的key对应了堆中的一个结点的位置，并且要求key不能重复，否则无法确定结点的位置，这样可以快速定位到要删除的结点，然后和删除堆顶元素类似，用最后的元素覆盖要删除的结点，然后进行上滤或下滤。

堆排序的时间复杂为。首先将数组构建成一个堆，升序排列为大顶堆，降序排列为小顶堆，然后将堆顶元素，也就是A[0]与最后一个位置的元素进行交换，这样最大值就交换到了数组最后，在对交换后的A[0]执行下滤操作，范围是堆中刨除掉最后已排序好的元素，这样会执行n次下滤，所以时间复杂为。

数据结构时间复杂度的衡量方法：数据结构通常会提供多个“接口”，需要对每个接口都进行时间复杂度的评价。

例：设计一个Set数据结构，提供lowerBound和add两个接口，lowerBound要返回比某个值大的最小值。

设计1：使用数组存储，add的时间复杂度就是O(1)，lowerBound则采用打擂台算法，时间复杂度为O(n)

设计2：使用红黑树存储，add的时间复杂度为O(logn)，lowerBound也是O(logn)

这两个设计的好坏如何比较？

如果lowerBound很少被调用，add非常频繁，则对n个数据而言，设计1的时间复杂度就是O(n)，而设计2的时间复杂度就是O(nlogn)，这时设计1就好一些。如果lowerBound和add调用的频率相同，则设计2要更好些，

哈希表增删查(修改只能先删后添加)的时间复杂度为O(1)，前提是通过key计算的hash值的时间是O(1)，所以实际的时间复杂度为O(size of key)。

lintcode 134中Cache是缓存，Buffer是缓冲。

Buffer：例如从文件中读1个字节和读10个字节的效率是相同，如果程序每次都处理1个字节，但读取时不能每次都读1个字节，这样太耗费时间，可以设置一个Buffer，一次读10个字节，程序每次都从Buffer中读取，这样效率更高。本质上Buffer是消费者和生产者之间的速率上有差别导致的。

Cache：如果从硬盘上某个地方读取数据时，假如频繁读取，每次直接从硬盘上读取，会耗时(大约ms级)，如果把数据从硬盘放入到Memory中，读取就会更快(ns级)，这就是Cache，本质上是不同的介质访问效率的不同。

Buffer中的数据结构本质上是一个队列，先进先出。Cache是一个哈希表，访问频率高的存进去，访问频率低的就删除掉，以降低容量。

python中的优先队列：heapq。

优先队列与堆的区别：heap是数据结构，优先队列是heap的实现。heap和优先队列的主要区别是，heap的remove操作时间复杂度可以为O(logn)，利用哈希表可以实现，但优先队列的remove操作是O(n)。

优先队列和堆都是无序的。

在线算法和离线算法：

在线算法：数据结构设计类问题=数据类问题=数据不可二次访问=多次输入和输出。

离线算法：一次输入输出=数据是一开始给定的=数据可以多次访问。

