1、vector

当元素个数超出当前容量时，超出的部分并不是在原有空间后追加，因为无法保证源空间之后尚有可分配的空间，而是申请更大的空间，然后将数据拷贝进新空间并释放原空间（vector会将自身容量扩大为原来的两倍），扩充空间需要经过的步骤：

* 重新配置空间
* 元素移动
* 释放得到内存空间

一旦vector空间重新分配，则指向原来vector的迭代器全部失效，因为vector的地址发生改变

2、map与set原理

map，set底层是由红黑树实现的，插入删除查找的时间复杂度为O(logN)

Q:与AVL树相比，为什么红黑树的效率更高  
A:AVL树为了维护平衡要不断调整，付出了更多的代价；而红黑树并不是一味追求平衡因子为1的平衡，而是近似平衡，因此在插入删除时会减少大量左旋右旋操作，性能更加稳定

3、list原理

底层使用双向链表实现的

4、priority\_queue

底层是用堆实现的，查找删除为O(logN)

5、unordered\_map和map

（1）map支持键值的自动排序，底层机制是红黑树，红黑树的查询和维护时间复杂度均为$O(logn)$，但是空间占用比较大，因为每个节点要保持父节点、孩子节点及颜色的信息

（2）unordered\_map是C++ 11新添加的容器，底层机制是哈希表，通过hash函数计算元素位置，其查询时间复杂度为O(1)，维护时间与bucket桶所维护的list长度有关，但是建立hash表耗时较大

从两者的底层机制和特点可以看出：map适用于有序数据的应用场景，unordered\_map适用于高效查询的应用场景

6、如何解决冲突

（1）开放定址法

它的实现是在插入一个元素的时候，先通过哈希函数进行判断，若是发生哈希冲突，就以当前地址为基准，根据再寻址的方法（探查序列），去寻找下一个地址，若发生冲突再去寻找，直至找到一个为空的地址为止。

有几种常用的探查序列的方法：

* 线性探测：di=1，2，3，…，m-1；冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表。
* 二次探测：di=12，-12，22，-22，…，k2，-k2 ( k<=m/2 )；冲突发生时，在表的左右进行跳跃式探测，比较灵活。
* 伪随机探测：

（2）拉链法

HashMap，HashSet其实都是采用的拉链法来解决哈希冲突的，就是在每个位桶实现的时候，我们采用链表（之后采用链表+红黑树）的数据结构来去存取发生哈希冲突的关键字

（3）再散列法

在使用哈希函数去散列一个输入的时候，输出是同一个位置就再次散列，直至不发生冲突位置

（4）建立公共溢出区

将哈希表分为公共表和溢出表，当溢出发生时，将所有溢出数据统一放到溢出区。