如题，二叉堆是一种基础数据结构

事实上支持的操作也是挺有限的（相对于其他数据结构而言），也就插入，查询，删除这一类

**一.堆的定义**

1.堆是一颗完全二叉树

2.堆的顶端一定是“最大”，最小”的，但是要注意一个点，这里的大和小并不是传统意义下的大和小，它是相对于优先级而言的，当然你也可以把优先级定为传统意义下的大小，但一定要牢记这一点，初学者容易把堆的“大小”直接定义为传统意义下的大小，某些题就不是按数字的大小为优先级来进行堆的操作的

（但是为了讲解方便，下文直接把堆的优先级定为传统意义下的大小）

3.堆一般有两种样子，小根堆和大根堆，分别对应第二个性质中的“堆顶最大”“堆顶最小”，对于大根堆而言，任何一个非根节点，它的优先级都小于堆顶，对于小根堆而言，任何一个非根节点，它的优先级都大于堆顶（这里的根就是堆顶）

来一张图了解一下堆（这里是小根堆）

图示

描述已自动生成

不难看出，对于堆的每个子树，它同样也是一个堆

**二.堆的操作**

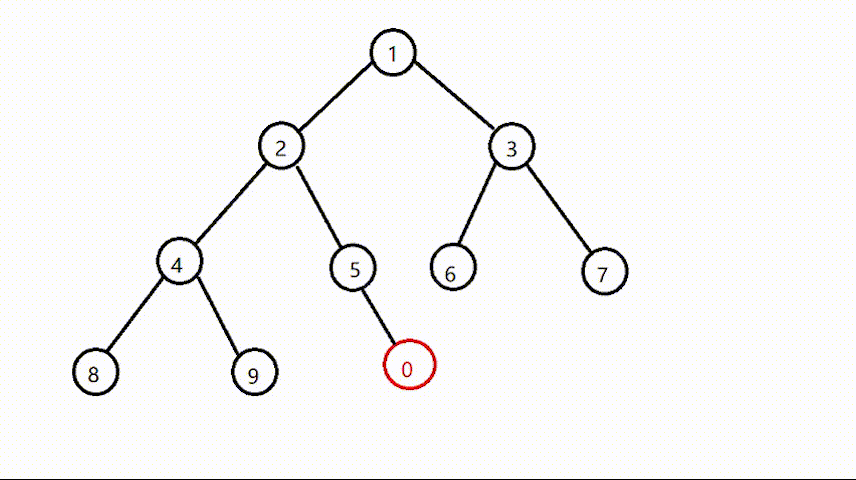
**1.插入**

假设你已经有一个堆了，比如（一）图中的那一个。

这个时候你如果想要给它加入一个节点怎么办，比如说0？

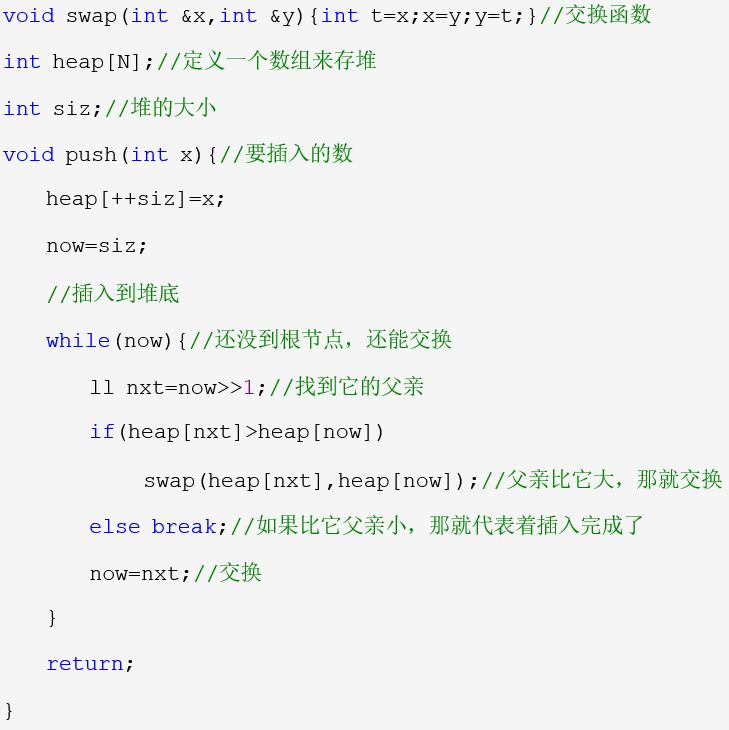
先插到堆底（严格意义上来说其实0是在5的左儿子的，图没画好放不下去，不过也不影响）

然后会发现它比它的父亲小，这样不符合小根堆的性质了，那怎么办？便需要交换一下他们的位置，交换之后还是发现不符合小根堆的性质，那么再换，还是不行，再换，这时就符合小根堆的性质了。（具体过程见动图）



事实上堆的插入就是把新的元素放到堆底，然后检查它是否符合堆的性质，如果符合就丢在那里了，如果不符合，那就和它的父亲交换一下，一直交换交换交换，直到符合堆的性质，那么就插入完成了

Code：



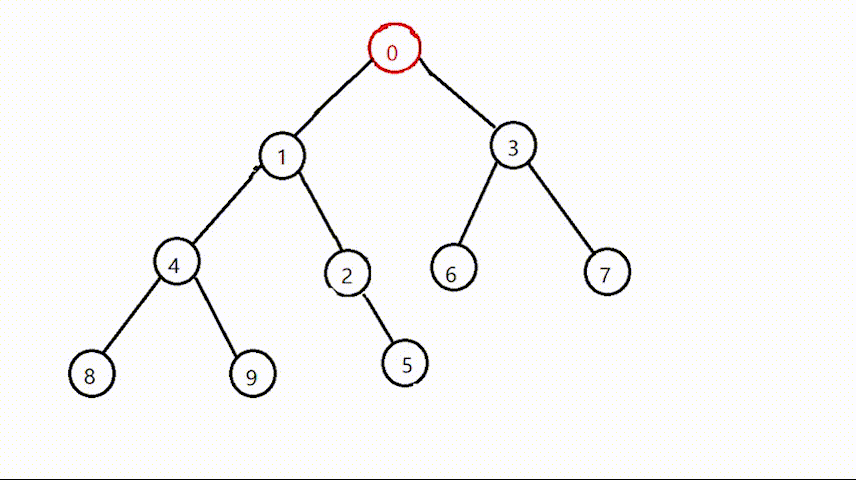
**2.删除**

假设我们要删除0这个元素。

怎么删除？在删除的过程中还是要维护小根堆的性质

如果直接删掉了，那就没有堆顶了，这个堆就直接乱了，所以我们要保证删除后这一整个堆还是个完好的小根堆。

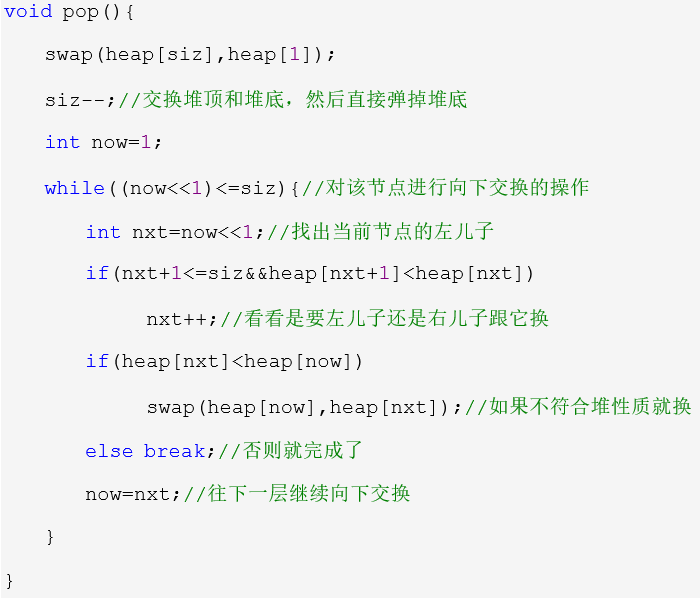
首先在它的两个儿子里面，找一个比较小的，和它交换一下，但是还是没法删除，因为下方还有节点，那就继续交换，直到这个碍眼的东西的下面没有节点了，这时候直接把它扔掉就好了。（过程如图）



这样我们就完成了删除操作，但是在实际的代码操作中，并不是这样进行删除操作的，有一定的微调，代码中是直接把堆顶和堆底交换一下，然后把交换后的堆顶不断与它的子节点交换，直到这个堆重新符合堆性质，但为了易于理解本文使用了上述解释方式。

手写堆的删除支持任意一个节点的删除，不过STL只支持堆顶删除，STL的我们后面再讲

Code：



**3.查询**

因为我们一直维护着这个堆使它满足堆性质，而堆最简单的查询就是查询优先级最低/最高的元素，对于我们维护的这个堆heap，它的优先级最低/最高的元素就是堆顶，所以查询之后输出heap[1]就好了

一般的题目里面查询操作是和删除操作捆绑的，查询完后顺便就删掉了，这个主要因题而异

**三.堆的STL实现**

在实际应用中堆一般都不通过手写实现，一是手写堆容易写错代码又多，二是STL 直接给我们提供了一个实现堆的简单方式：优先队列

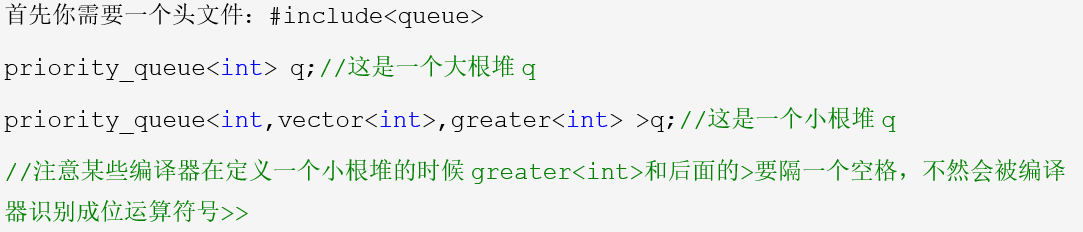
手写堆和STL的优先队列有什么区别？

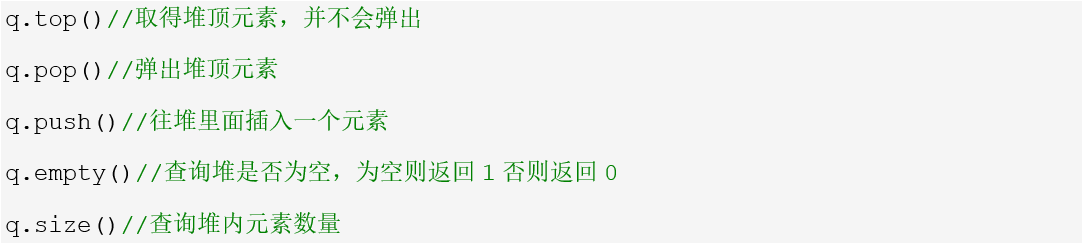
速度方面，手写堆会偏快一点，但是如果开了O2优化优先队列可能会更快；

代码实现难度方面：优先队列远比手写堆简单

这两方面综合起来，一般都是用STL的优先队列来实现堆

定义一个优先队列：

优先队列的操作：



不过有个小问题就是STL只支持删除堆顶，而不支持删除其他元素

但是问题不大，如果我们遇到需要删除其他元素的问题，只需要开一个数组del，在要删除其他元素的时候直接就标记一下del[i]=1，这里的下标是元素的值，然后在查询的时候碰到这个元素被标记了，就直接将它弹出，然后继续查询即可。

**四.堆的复杂度**

因为堆是一棵完全二叉树，所以对于一个节点数为n的堆，它的高度不会超过logn

所以对于插入，删除操作复杂度为O(logn)

查询堆顶操作的复杂度为O(1)

**五．总结**

堆是一个非常基础的数据结构，一般是我们踏入数据结构大门时接触到的第一个数据结构，是我们深入学习其他数据结构之前的“甜点”。而在算法竞赛中，堆大多都是以一个“工具”的身份出现，用于优化算法（大多时候是贪心）的时间复杂度等。