一、基础概念

1. 堆的作用

(1) **优先级队列**，一般队列实现类LinkedList是按添加顺序排队的，但现实中，经常需要按优先级来，每次都应该处理当前队列中优先级最高的，高优先级的，即使来得晚，也应该被优先处理。

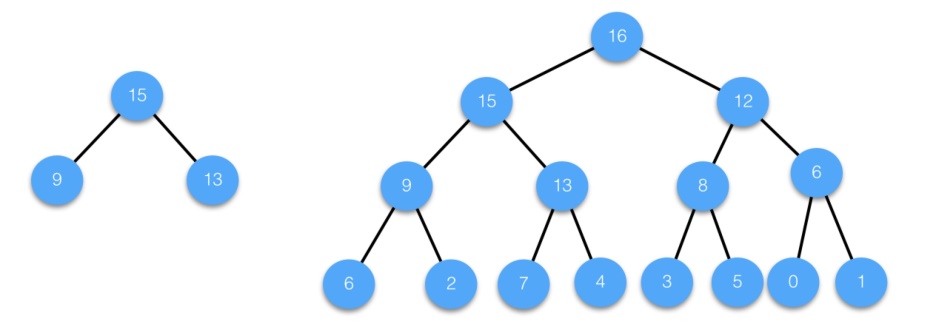
(2) **求前K个最大的元素**，元素个数不确定，数据量可能很大，甚至源源不断到来，但需要知道到目前为止的最大的前K个元素。这个问题的变体有：求前K个最小的元素，求第K个最大的，求第K个最小的。

(3) **求中值元素**，中值不是平均值，而是排序后中间那个元素的值，同样，数据量可能很大，甚至源源不断到来。

2. 堆的概念 - 完全二叉树

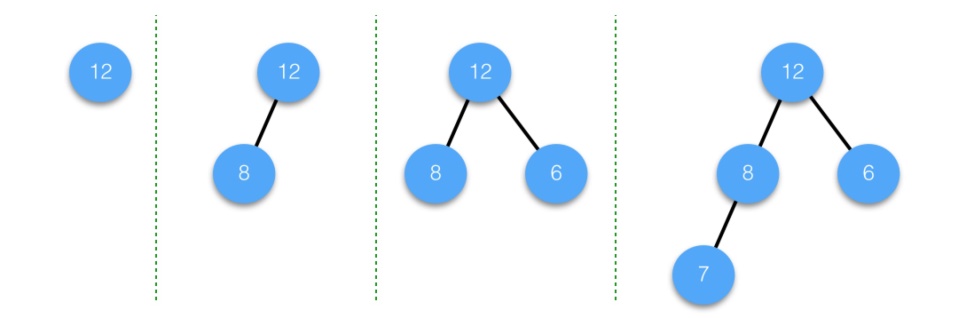
(1) 满二叉树

除最后一层外，每一层上的所有结点都有两个子结点。在满二叉树中，每一层上的结点数都达到最大值，即在满二叉树的第k层上有2k-1个结点，且深度为m的满二叉树有2m－1个结点。



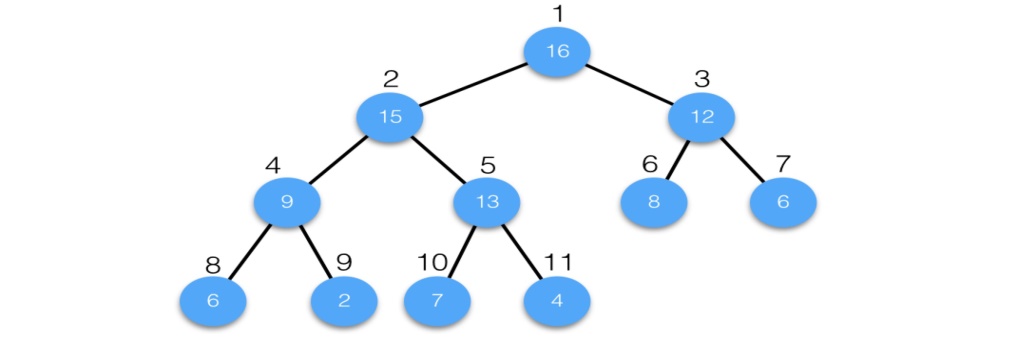
(2) 完全二叉树

除最后一层（带有节点的最后一层）外，每一层上的结点数均达到最大值；在最后一层上只缺少右边的若干结点。（如果有节点，必定先填满左边的节点位置，再去填右边的节点位置）



3. 完全二叉树的特点

(1) 在完全二叉树中，给定任意一个节点，可以根据其编号直接快速计算出其父节点和孩子节点编号。



① 如果编号为i，则父节点编号即为i/2，左孩子编号即为2\*i,右孩子编号即为2\*i+1。

② 如：节点5，父节点为5/2，左子结点为2\*5，右子节点为2\*5+1

(2) 最大堆和最小堆

① 最大堆是指，**每个节点都不大于其父节点**。这样，对每个父节点，一定不小于其所有孩子节点，而根节点就是所有节点中最大的，对每个子树，子树的根也是子树所有节点中最大的。

② 最小堆与最大堆正好相反，**每个节点都不小于其父节点**。这样，对每个父节点，一定不大于其所有孩子节点，而根节点就是所有节点中最小的，对每个子树，子树的根也是子树所有节点中最小的。

二、堆的具体算法 （以最小堆为例）

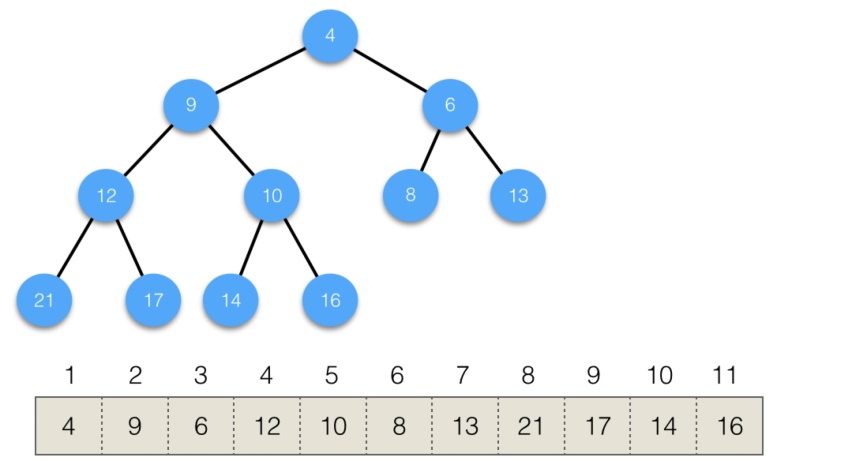
1. 添加元素

(1) 基本步骤

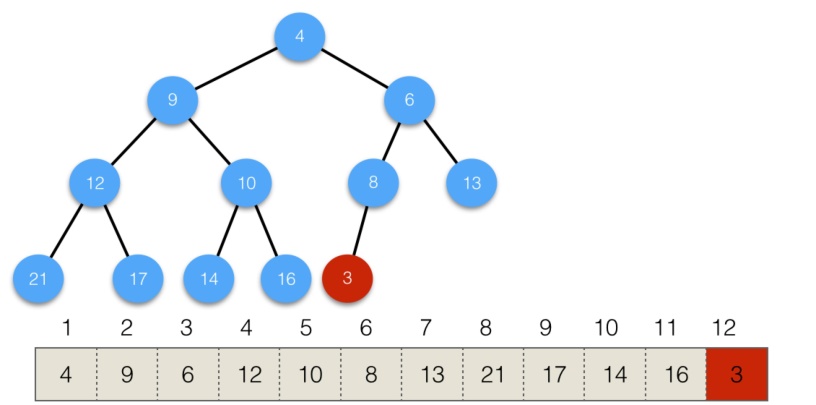
① 添加元素到最后位置。（直接添加到数组的最后一个位置）

② 与父节点比较，如果大于等于父节点，则满足堆的性质，结束，否则与父节点进行交换，然后再与父节点比较和交换，直到父节点为空或者大于等于父节点。

(2) 具体例子



添加元素3



先和父节点8交换，再和父节点6交换，再和父节点4交换，最后到起始结点。

2. 从头部删除元素

(1) 基本步骤

① 用最后一个元素替换头部元素，并删掉最后一个元素。

② **将新的头部与两个孩子节点中较小的比较**，如果不大于该孩子节点，则满足堆的性质，结束，否则与较小的孩子进行交换，交换后，再与较小的孩子比较和交换，一直到没有孩子，或者不大于两个孩子节点。这个过程我们般称为siftdown。

3. 从中间删除元素

① 与从头部删除一样，都是先用最后一个元素替换待删元素。

② 不过替换后，有两种情况，如果该元素大于某孩子节点，则需向下调整(siftdown)，否则，如果小于父节点，则需向上调整(siftup)。

3. 将无序数组构建成一个最小堆

(1) 基本步骤：

从最后一个非叶子节点开始，一直往前直到根，对每个节点，执行向下调整siftdown。换句话说，是自底向上，先使每个最小子树为堆，然后每对左右子树和其父节点合并，调整为更大的堆，因为每个子树已经为堆，所以调整就是对父节点执行siftdown，就这样一直合并调整直到根。将普通无序数组变为堆的过程我们称之为heapify。

void heapify() {

for (int i=size/2; i >= 1; i--)

siftdown(i);

} //size表示节点个数, 节点编号从1开始，size/2表示第一个非叶节点的编号。

4. 堆的特点

① 在添加和删除元素时，有两个关键的过程以保持堆的性质，一个是向上调整(siftup)，另一个是向下调整(siftdown)，它们的效率都为O(log2(N))。由无序数组构建堆的过程heapify是一个自底向上循环的过程，效率为O(N)。

② 查找和遍历就是对数组的查找和遍历，效率为O(N)。