对数据（元素/Object）要解决的问题?
对数据在哪里进行存储?
存储结构如何？该结构涉及的相关操作怎么样?

背景：
Heap 也是 ADT，Heap 是个性质（max heap, min heap）
PQ 可以用 heap 来实现 （改 priority, 按 priority 排序）
Heap 可以用树来实现，也可用指针/数组来实现

优先队列(PriorityQueue) 是一种抽象数据结构 ADT，队列内的元素都有对应的优先级，并根据其优先级出队。

Priority Queue 优先队列能解决的问题：

优先队列是一种队列，它满足优先级别最高的先出队列。它的一个重要的应用时海量数据的排序，当我们需要找出10亿数据中的Top10项，不实际的解决方案是对所有数据排序，取Top10项。使用优先队列可以在很小的辅助空间内找出Top10项。具体思路如下：

（1）建立大小为k的容器
（2）每次从大数据集中读取一个数据项
（3）如果容器没有满，则把当前数据项加入容器
（4）如果容器已经满，找出容器中最小的元素（利用优先队列特性）如果当前数据项大于最小元素，则最小元素出队，当前数据项入队
（5）重复（2）到（3）直到读取完大数据集

上述的方案可以以并行的方式运行，最后合并结果。

优先度列的五种实现方式，分别如下

1. 无序数组
2. 有序数组
3. 链表
4. 堆
5. 二叉搜索树

【优先队列的数组实现】
用数组实现优先队列：（insert 已经进行有序操作。pop 操作即为 peekMax，remove 操作）

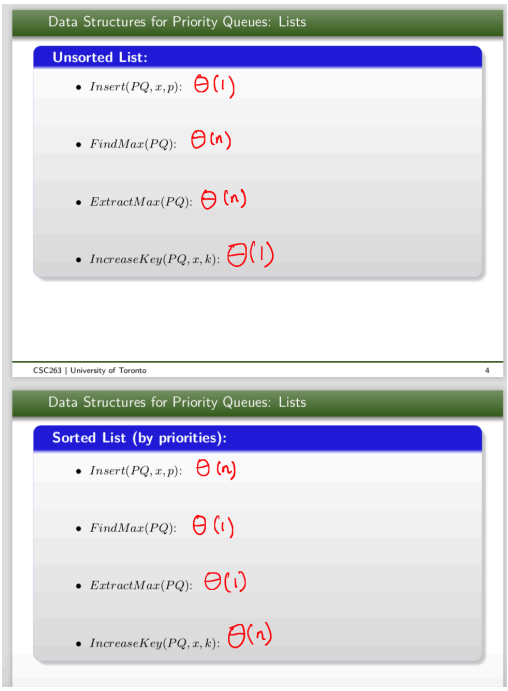
无序数组的实现

无序数组实现方式的入队操作，直接把入队元素加到数组尾部。出队需要遍历数组，找出优先级别最高的出队，空缺的位置由后面元素依次补上。因此，入队的时间复杂度为O(1)，出队为O(N)。

有序数组的实现

由于要求数组有序，因此在插入的时候需要保存有序，插入操作需要找到适合的位置，然后在该位置插入，位置后面的元素依次往后移动，时间复杂度为O(n)。而出队，由于序列是有序，可以在O(1)内出队。

相关 java code 参见笔记



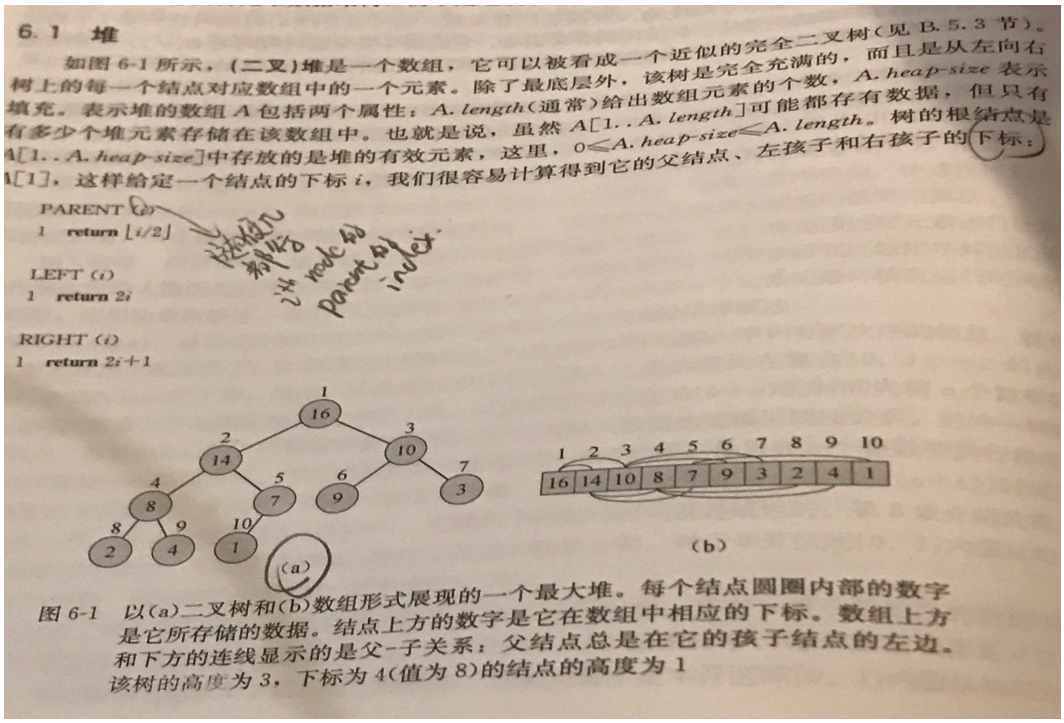
csc263 lecture2 ppt pg4-5

优先队列的堆实现

（二叉）堆是一个数组（Array），它可以看成一个近似的完全二叉树（Complete Binary Tree）。树上的每一个结点对应数组中的一个**元素**。除了最底层外，该树是完全充满的，而且是从左向右填充的。

该数组（Array）是个特殊的数组（有一定的大小比较，但并不是完全 sort），是用来表示堆的

以完全二叉树和数组形式来展现一个最大堆/最小堆（相对而言，完全二叉树表示的更为直观形象贴切）



6.1 堆

如图 6-1 所示，(二叉)堆是一个数组，它可以被看成一个近似的完全二叉树(见 B.5.3 节)。树上的每一个结点对应数组中的一个元素。除了最底层外，该树是完全充满的，而且是从左向右填充。表示堆的数组 A 包括两个属性：$A.length$(通常)给出数组元素的个数，$A.heap\text{-}size$ 表示有多少个堆元素存储在该数组中。也就是说，虽然 $A[1..A.length]$ 可能都存有数据，但只有 $A[1..A.heap\text{-}size]$ 中存放的是堆的有效元素，这里，$0 \leqslant A.heap\text{-}size \leqslant A.length$。树的根结点是 $A[1]$，这样给定一个结点的下标 i，我们很容易计算得到它的父结点、左孩子和右孩子的下标：

```
PARENT (i)
1  return ⌊i/2⌋

LEFT (i)
1  return 2i

RIGHT (i)
1  return 2i+1
```


图 6-1 以(a)二叉树和(b)数组形式展现的一个最大堆。每个结点圆圈内部的数字是它所存储的数据。结点上方的数字是它在数组中相应的下标。数组上方和下方的连线显示的是父-子关系：父结点总是在它的孩子结点的左边。该树的高度为 3，下标为 4(值为 8)的结点的高度为 1

在**排序算法**和**优先队列**中应用最大堆/最小堆

增删改查的某些相关操作会破坏最大/最小堆的性质
删除很特殊，衍生到删除最大/最小

- 堆构建（建堆） - Build Heap $(O(n))$

(先将数据源的数据元素单纯按照完全二叉树从左至右依次填入，在按照最大堆/最小堆性质进行调整-Bubble Down, Bubble Up)

3.最小堆的构建

初始数组为：9,3,7,6,5,1,10,2

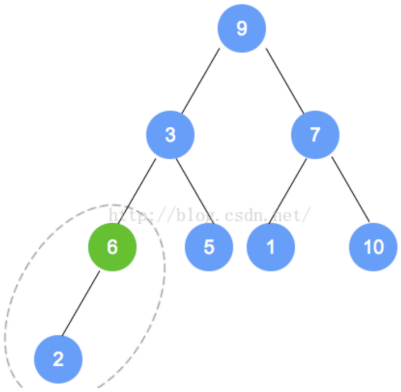
按照完全二叉树，将数字依次填入。

填入后，找到最后一个结点（本示例为数字2的节点），从它的父节点（本示例为数字6的节点）开始调整。根据性质，小的数字往上移动；至此，第1次调整完成。

注意，被调整的节点，还有子节点的情况，需要递归进行调整。

第二次调整，是数字6的节点数组下标小1的节点（比数字6的下标小1的节点是数字7的节点），用刚才的规则进行调整。以此类推，直到调整到根节点。

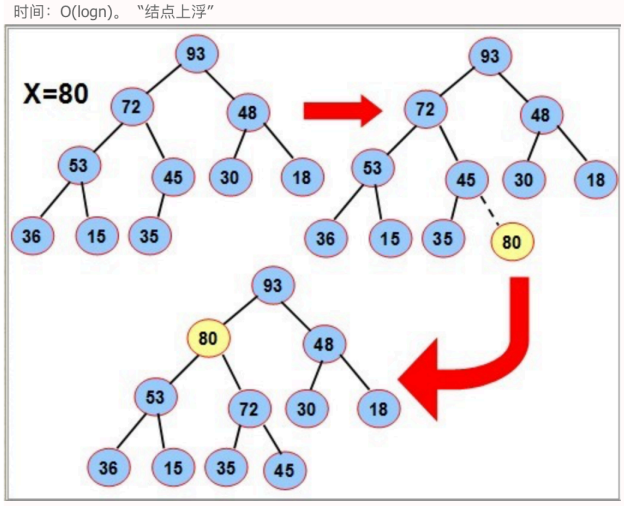
以下是本示例的图解：



- 插入- Insert$(O(\lg n))$

⑴最大堆的插入

由于需要维持完全二叉树的形态，需要先将要插入的结点x放在最底层的最右边，插入后满 足完全二叉树的特点；然后把x依次向上调整到合适位置满足堆的性质,例如下图中插入80,先将80放在最后,然后两次上浮到合适位置.
时间：O(logn)。“结点上浮”



Queue(priority+value)

Priority Queue

Complete Binary Tree

Max-Heap

Min-Heap

FindMax(PQ)：返回最大 pv 值的结点
HeapMaximum(A): 返回 root
IncreaseKey(A,i,k) ：增加堆 A 的结点 i 的 pv 值至 k（改）
HeapIncreaseKey(A,i,k)：将 Array A 第 i 个的 pv 值设置成 k（改）
Insert(PQ,x,p) ：在堆 PQ 里插入结点 x 伴随 pv 值 p
MaxHeapInsert(A,x)：在堆 A 里插入结点伴随 pv 值 x
MaxHeapify(B,i)：
ExtractMax(PQ):返回并删除最大 pv 值的结点
HeapExtractMax(H): H 是 heap 堆/array 数组，返回 root，（并删除 root），并重新修复堆/Array - H

HeapSort(A)：对数组 array A 进行 heap sort 排序
BuildMaxHeap(A)

CSC263 第三周

复习 BST Insert 和 Delete 操作