## 6.1-1

高度为h的(二叉)堆中，元素最多为，最少为。

## 6.1-2

因为堆是个完全二叉树，根据上一题中高度为h的堆，最少结点为，结点最多为。取对数可知当结点为n时，堆的高度为 。

注意：在算法导论中树的高度是从0开始算的。

## 6.1-3

证：在最大堆中，根节点的值总是比孩子的结点的值大。所以，最大元素在子树的根结点的位置上。

## 6.1-4

最小元素在最大堆最后一层的最右边的叶子结点。

## 6.1-5

不一定。如在《算法导论》第三版，p84中的例子。

## 6.1-6

不是。

## 6.2-3

若A[i]比孩子的值都大，则MAX-HEAPIFY不会对元素进行变化。

## 6.2-4

若i>A.heap-size/2,则l，r要大于A.heap-size，故MAX-HEAPIFY第3~6行将不会满足，故整个递归也没什么意义。

## 6.3-2

循环不变量从A.heapsize/2到1，是因为在堆数组中，下标从n/2+1到n都为叶子结点。从A.heapsize/2执行MAX-HEAPIF可以从最少的结点建堆。若是从1开始，则不保证从根结点开始建立有效堆，如建立大根堆时，根结点为4，左右孩子分别为1,3。则此时MAX-HEAPIFY过程将不会工作。

## 6.4-3

若是升序数组，堆排序时，执行BuildHeap时执行复杂度为O(nlgn)，for循环调用n-1次MAX-HeapIFY，MAX-HeapIFY时间复杂度为O(lgn),故整个过程的时间复杂度为O(nlgn).

若为降序数组，执行BuildHeap时执行复杂度为O(n), for循环调用n-1次MAX-HeapIFY，MAX-HeapIFY时间复杂度为O(lgn),故整个过程的时间复杂度为O(nlgn).

## 6.5-3

|  |
| --- |
| HEAP-MINIMU(A, i){  return A[i];  }  HEAP-EXTRACT-MIN(A){  If (A.heapsize < 1)  error”heap underflow”  min = A[1]  A[1] = A[A.heapsize]  A.heapsize -= 1  MINHEAPIFY(A, 1)  return min  }  HEAP-DECRESE-KEY(A, i, key){  If key > A[i]  error “new key is greater than current key”  A[i] = key  while i > 1 and A[PARENT(i)] > A[i]  swap(A[i], A[PARENT(i)]  i = PARENT(i)  } |

## 6.5-4

MAX-HEAP-INSERT先把关键字设为,即在堆中有一个比任何元素都小的元素，从而保证了插入的元素能顺利插入，若是没有这一步，则有可能会出现插入的元素比堆中的元素都小，这样HEAP-INCREASE-KEY就执行不了，从而插入失败。

## 6.5-6

从第三行开始：

|  |
| --- |
| while i > 1 and A[PARENT(i)] > A[i]  A[i] = A[PARENT(i)]  i = PARENT(i)  A[i] = key |

## 6.5-7

使用优先级序列实现一个先进先出队列的思路：

为一个进入队列的元素进行标号，然后在按照标号进行最小优先级队列的建立，要出队时，则调用HEAP-EXTRACT-MIN，返回出队的元素的小标，并用小标返回元素。

栈同理，只不过时用最大优先级队列来实现。

## 6.5-8

|  |
| --- |
| HEAP-DELETE(A, i){  A[i] = A[A.heapsize]  A.heapsize --  MAX-HEAPIFY(A, 1)  } |