HW4

6.4

P90 6.4：2

6.4-2 试分析在使用下列循环不变量时，HEAPSORT 的正确性：

在算法的第 2～5 行 **for** 循环每次迭代开始时，子数组 $A[1..i]$是一个包含了数组$A[1..n]$中第 i 小元素的最大堆，而子数组 $A[i+1..n]$包含了数组 $A[1..n]$中已排序的$n-i$个最大元素？

```
HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i = A.length downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size - 1
5      MAX-HEAPIFY(A, 1)
```

开始时，i=A.len，A[i+1..n]无元素，循环不变式成立。

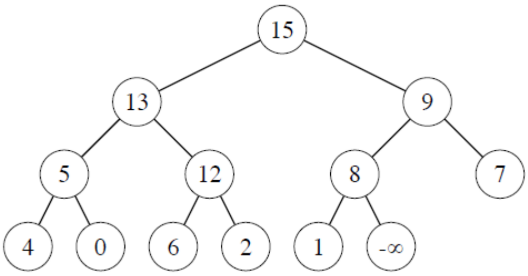
假设i=k时循环不变式成立，下面证i=k-1时仍成立（k=A.len..3）：

i=k循环开始，由假设，A[1..k]是包含全局第k小元素的max-heap，A[k+1..n]是n-k个全局最大元素。全局第k小元素是A[1..k]中最大元素，为max-heap根A[1]。i=k循环过程中，A[1]和A[k]交换，然后重新调整A[1..k-1]为max-heap。故i=k-1循环开始，A[k,n]是n-k+1个全局最大元素，A[1..k-1]是max-heap，A[1]是全局第k-1小元素，循环不变式仍成立。得证。

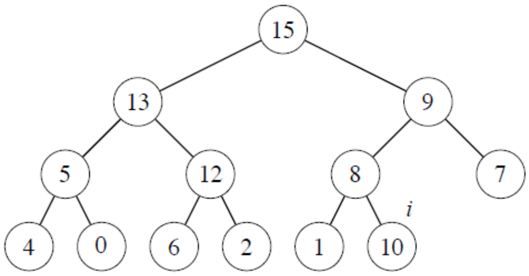
6.5

P92 6.5：2，6

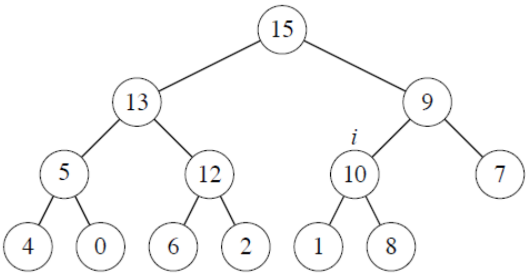
6.5-2 试说明 MAX-HEAP-INSERT(A，10)在堆 $A=\langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1\rangle$ 上的操作过程。



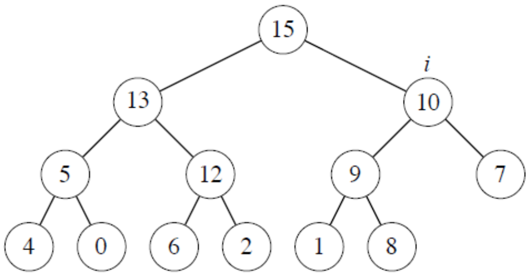
(a)



(b)



(c)



(d)

6.5-6 在 HEAP-INCREASE-KEY 的第 5 行的交换操作中，一般需要通过三次赋值来完成。想一想如何利用 INSERTION-SORT 内循环部分的思想，只用一次赋值就完成这一交换操作？

HEAP-INCREASE-KEY(A, i, key)
1 **if** $key < A[i]$
2   **error** "new key is smaller than current key"
3 $A[i] = key$
4 **while** $i > 1$ and $A[\text{PARENT}(i)] < A[i]$
5   exchange $A[i]$ with $A[\text{PARENT}(i)]$
6     $i = \text{PARENT}(i)$

```python
def HeapIncreaseKey(A,i,key):
    # increase A[i] to key

    if A[i]>=key:
        return

    parentI = Tree.Parent(i)
    while parentI>=0 and A[parentI]<=key:
        # 一次赋值
        A[i] = A[parentI]
        i = parentI
        parentI = Tree.Parent(i)
    # 最后记得放key
    A[i] = key
    return A
```

7.1

P97 7.1：4

7.1-4 如何修改 QUICKSORT，使得它能够以非递增序进行排序？

升序排的Partition()：

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p-1
3  for j = p to r-1
4        if A[j]≤x
5              i = i + 1
6              exchange A[i] with A[j]
7  exchange A[i+ 1] with A[r]
8  return i + 1
```

第4行改为if A[j]>=x即可。

7.2

P100 7.2：3，5

7.2-3 证明：当数组 A 包含的元素不同，并且是按降序排列的时候，QUICKSORT 的时间复杂度为 $\Theta(n^2)$。

A开始时已有序，按书上算法选A[r]作划分元，将导致最坏划分，n个元素划分后一组n-1，一组0元素。$T(n)=T(n-1)+T(0)+\Theta(n)$，$T(n)=\Theta(n^2)$。

7.2-5 假设快速排序的每一层所做的划分的比例都是 $1-\alpha:\alpha$，其中 $0<\alpha\leqslant 1/2$ 且是一个常数。试证明：在相应的递归树中，叶结点的最小深度大约是 $-\lg n/\lg\alpha$，最大深度大约是 $-\lg n/\lg(1-\alpha)$(无需考虑整数舍入问题)。

$0<\alpha\le\frac{1}{2}$，则$1-\alpha\ge\alpha$，α分支最早结束，对应最小深度k；$1-\alpha$分支最晚结束，对应最大深度l。

$n\alpha^k\le 1$，$k\ge-\lg n/\lg\alpha$，即最小深度。同理，最大深度$-\lg n/\lg(1-\alpha)$。

8.1

P108 8.1：3

8.1-3 证明：对 $n!$ 种长度为 n 的输入中的至少一半，不存在能达到线性运行时间的比较排序算法。如果只要求对 $1/n$ 的输入达到线性时间呢？$1/2^n$ 呢？

这种的“一半，$1/\text{n}$，$1/2^n$”都是针对n!种排序结果。输入长度为n，输出结果有$m = n!/2$，$n!/n$，$n!/2^n$时，都不存在线性时间的比较排序算法。证明：设比较排序等价的决策树高h，则算法代价$\Omega(h)$。m是最后一层节点数，故$2^h \geq m$，$h \geq \lg m = \Omega(n \lg n)$。所以对所给m，算法代价$\Omega(n \lg n)$，不存在线性时间算法。

$\lg m = \Omega(n \lg n)$是因为：

$$\begin{aligned}
\lg \frac{n!}{2} &= \lg n! - 1 \geq n \lg n - n \lg e - 1 \\
\lg \frac{n!}{n} &= \lg n! - \lg n \geq n \lg n - n \lg e - \lg n \\
\lg \frac{n!}{2^n} &= \lg n! - n \geq n \lg n - n \lg e - n
\end{aligned}$$

8.2

P110 8.2：4

8.2-4 **设计一个算法，它能够对于任何给定的介于 0 到 k 之间的 n 个整数先进行预处理，然后在 $O(1)$时间内回答输入的 n 个整数中有多少个落在区间$[a..b]$内。你设计的算法的预处理时间应为 $\Theta(n+k)$。**

预处理：对n个整数作counting sort，得前缀和数组C[0..k]，C[i] = #(n个数中小于等于i的数)。$\Theta(n+k)$时间。

#(落在[a..b]内的数) = C[b]-C[a-1]。O(1)时间。