Sample Solution for Assignment 2

10.4-2
PRINT_RECURSIVE(node p)
1   if p != NIL
2      then PRINT_RECURSIVE(p→left)
3         print(p→key)
4      PRINT_RECURSIVE(p→right)

10.4-3
PRINT_NONRECUR(node p)
1 stack s ← new stack
2 while p != NIL
3   do print(p→key)
4      if p→left != NIL
5         then if p→right != NIL
6            then push(s, p→right)
7            p ← (p→left)
8            else p ← pop(s)

10.4-4
PRINT_ARBITRARY(node p)
1   if p != NIL
2      then PRINT_ARBITRARY(p→left_child)
3         print(p→key)
4      PRINT_ARBITRARY(p→right_sibling)

6.1-1
When a heap has the maximum number of elements, it forms a complete tree. The number of elements in it is $2^{h+1}-1$. The minimum number of elements of height $h$ has a complete tree of height $h-1$ plus one more node at the next level. The number of elements in it is $2^{h-1+1}+1 = 2^h$.

6.1-2
From 6.1-1 we have $2^h \leq n \leq 2^{h+1}-1$, it follows that $\lg n \geq h \geq \lg(n+1)-1 \geq \lg n - 1$. Give that $h$ is an integer, $\therefore h = \left\lfloor \lg n \right\rfloor$.

6.1-3
Let the node $i$ be the root of the sub-tree. From the max-heap property, $A[i] \geq A[left\_child(i)]$, if node $i$ has a left-child. Likewise, $A[i] \geq A[right\_child(i)]$, if node $i$ has a right-child. The argument continues for all children in the sub-tree. Hence $A[i]$, the root of the sub-tree, is larger than the value of any child in the sub-tree, so it contains the largest value in the sub-tree.

6.1-4
For max-heap (min-heap), largest (smallest) element is at root. The smallest (largest) element is at a leaf.

6.1-5
Yes the array that is sorted in increasing order is a min-heap. Similarly, an array sorted in decreasing order forms a max-heap.