Homework 2 (70pts)

1. (10 points) Design an algorithm, inorderNext(v), which returns the node visited after node v in an inorder traversal of binary tree T of size n. Analyze its worst-case running time. Your algorithm should avoid performing traversals of the entire tree.

2. (10 points) Let T be a binary tree with n nodes. It is realized with an implementation of the Binary Tree ADT that has $O(1)$ running time for all methods except *positions*() and *elements*(), which have $O(n)$ running time. Give the **pseudocode** for a $O(n)$ time algorithm that uses the methods of the Binary Tree interface to visit the nodes of T by increasing values of the level numbering function p given in Section 2.3.4. This traversal is known as the **level order traversal**. Assume the existence of an $O(1)$ time visit(v) method (it should get called once on each vertex of T during the execution of your algorithm)

3. (a) (5 points) Illustrate the execution of the selection-sort algorithm on the following input sequence: (21, 14, 32, 10, 44, 8, 2, 11, 20, 26)
   (b) (5 points) Illustrate the execution of the insertion-sort algorithm on the following input sequence: (21, 14, 32, 10, 44, 8, 2, 11, 20, 26)

4. Let S be a sequence containing pairs (k, e) where e is an element and k is its key. There is a simple algorithm called count-sort that will construct a new sorted sequence from S provided that all the keys in S are different from each other. For each key k, count-sort scans S to count how many keys are less than k. If c is the count for k then (k, e) should have rank c in the sorted sequence.
   (a) (5 points) Give the **pseudocode** for count-sort as it is described above.
   (b) (3 points) Determine the number of comparisons made by count-sort. What is its running time?
   (c) (2 points) As written, count-sort only works if all of the keys have different values. Explain how to modify count-sort to work if multiple keys have the same value.

5. (a) (5 points) Illustrate the execution of the heap-sort algorithm on the following sequence: (2, 5, 16, 4, 10, 23, 39, 18, 26, 15). Show the contents of the heap and the sequence at each step of the algorithm. Indicate upheap or downheap bubbling where appropriate.
   (b) (5 points) Illustrate the execution of the bottom-up construction of a heap (like in Figure 2.49) on the following sequence: (2, 5, 16, 4, 10, 23, 39, 18, 26, 15, 7, 9, 30, 31, 40).

6. (10 points) Let T be a heap storing n keys. Give the **pseudocode** for an efficient algorithm for printing all the keys in T that are smaller than or equal to a given query key x (which is not necessarily in T). You can assume the existence of a $O(1)$-time *print(key)* function. For example, given the heap of Figure 2.41 and query key $x = 7$, the algorithm should report 4,5,6,7. Note that the keys do not need to be reported in sorted order. Your algorithm should run in $O(k)$ time, where k is the number of keys reported.

7. Use the table below to convert a character key to an integer for the following questions.

Letter	A	B	C	D	E	F	G	H	I	J	K	L	M
Key	0	1	2	3	4	5	6	7	8	9	10	11	12
Letter	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Key	13	14	15	16	17	18	19	20	21	22	23	24	25

(a) (5 points) Give the contents of the hash table that results when the following keys are inserted in that order into an initially empty 13-item hash table: $(E_1, A, S_1, Y, Q, U, E_2, S_2, T, I, O, N)$. Use $h(k) = k \mod 13$ for the hash function for the k-th letter of the alphabet (see above table for converting letter keys to integer values). Use linear probing.

(b) (5 points) Give the contents of the hash table that results when the same keys are inserted in that order into an initially empty 13-item hash table. Use $h(k) = k \mod 13$ for the hash function for the k-th letter of the alphabet (see above table for converting letter keys to integer values). Use double hashing and let $h'(k) = 1 + (k \mod 11)$ be the secondary hash function.