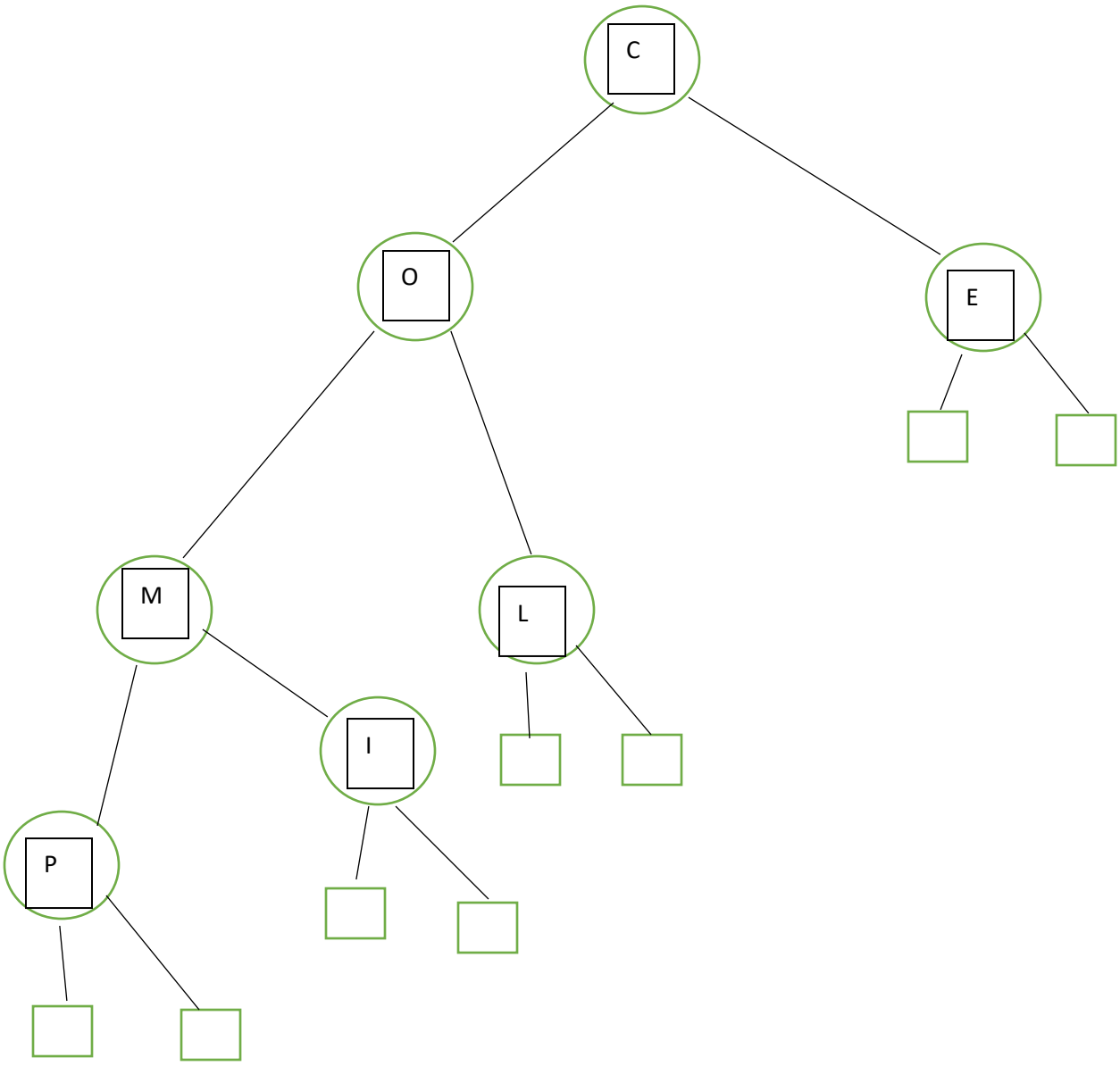
Chris Grimes
CS 46101 Algorithms
Homework# 2

1. (5 points) Draw a single binary tree T such that each of the following properties holds:

- each internal node of T stores a single character
- a preorder traversal of T yields COMPILE, and
- a inorder traversal of T yields PMIOLCE.



Chris Grimes
CS 46101 Algorithms
Homework# 2

2. (10 points) Give the pseudocode for an O(n)-time algorithm that computes the depth of each node of a tree T, where n is the number of nodes of T. Assume the existence of methods setDepth(v,d) and getDepth(v) that run in O(1)-time.

```
Algorithm Depth(B ,v):
    Input: B is a tree, v is a nod of tree B
    Output: returns the depth of each node in tree B with root v
    if B's root is v then
        setDepth(v,0)
    else
        setDepth(v, getDepth(B whose parent is v) +1)
    child<- B who is a child node of v
    while child has its own child node do
        child<- child's child
        Depth(B, child)
```

Chris Grimes
CS 46101 Algorithms
Homework# 2

3. (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.

```
Algorithm inOrderNext(v):
    Input: node v
    Output: the next node visited after node v in an inorder traversal of said tree
        if v is an internal node of said tree then
            current<- v
            while current is an internal node of said tree do
                current<- currents left child
            return current
        else
            if v is the root of said tree then
                return 0
            else
                current<-v
                x<- current's parent
                while current is the right child of x do
                    if x is the root of said tree
                        return 0
                    else
                        current<-x
                        x<-current's parent
                    return x
```

Worst case scenario for this algorithm v is the last internal node on the left side of the tree, in which case this algoritm must search down the left side of the tree to find v and then find it's external child. Under these circumstances the first while loop will run n/2 times giving this algorithm a "Big-Ohh" notation of O(n).

Chris Grimes
CS 46101 Algorithms
Homework# 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)

```
Algorithm customTraversal(T):
    Input: binary tree T
    Output: void
        create an empty queue Z and enqueue T's root
        while Z is not empty do
            current<- Z.dequeue
            visit(current)
            if current is an internal node of T then
                enqueue currents left child
                enqueue currents right child
```

Chris Grimes
CS 46101 Algorithms
Homework# 2

5. (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)

Input sequence	Priority Queue
(21, 14, 32, 10, 44, 8, 2, 11, 20, 26)	()
(14, 32, 10, 44, 8, 2, 11, 20, 26)	(21)
(32, 10, 44, 8, 2, 11, 20, 26)	(21, 14)
(10, 44, 8, 2, 11, 20, 26)	(21, 14, 32)
(44, 8, 2, 11, 20, 26)	(21, 14, 32, 10)
(8, 2, 11, 20, 26)	(21, 14, 32, 10, 44)
(2, 11, 20, 26)	(21, 14, 32, 10, 44, 8)
(11, 20, 26)	(21, 14, 32, 10, 44, 8, 2)
(20, 26)	(21, 14, 32, 10, 44, 8, 2, 11)
(26)	(21, 14, 32, 10, 44, 8, 2, 11, 20)
()	(21, 14, 32, 10, 44, 8, 2, 11, 20, 26)
(2)	(21, 14, 32, 10, 44, 8, 11, 20, 26)
(2, 8)	(21, 14, 32, 10, 44, 11, 20, 26)
(2, 8, 10)	(21, 14, 32, 44, 11, 20, 26)
(2, 8, 10, 11)	(21, 14, 32, 44, 20, 26)
(2, 8, 10, 11, 14)	(21, 32, 44, 20, 26)
(2, 8, 10, 11, 14, 20)	(21, 32, 44, 26)
(2, 8, 10, 11, 14, 20, 21)	(32, 44, 26)
(2, 8, 10, 11, 14, 20, 21, 26)	(32, 44)
(2, 8, 10, 11, 14, 20, 21, 26, 32)	(44)
(2, 8, 10, 11, 14, 20, 21, 26, 32, 44)	()

Chris Grimes
CS 46101 Algorithms
Homework# 2
(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)

Input sequence	Priority Queue
(21, 14, 32, 10, 44, 8, 2, 11, 20, 26)	()
(14, 32, 10, 44, 8, 2, 11, 20, 26)	(21)
(32, 10, 44, 8, 2, 11, 20, 26)	(14, 21)
(10, 44, 8, 2, 11, 20, 26)	(14, 21, 32)
(44, 8, 2, 11, 20, 26)	(10, 14, 21, 32)
(8, 2, 11, 20, 26)	(10, 14, 21, 32, 44)
(2, 11, 20, 26)	(8, 10, 14, 21, 32, 44)
(11, 20, 26)	(2, 8, 10, 14, 21, 32, 44)
(20, 26)	(2, 8, 10, 11, 14, 21, 32, 44)
(26)	(2, 8, 10, 11, 14, 20, 21, 32, 44)
()	(2, 8, 10, 11, 14, 20, 21, 26, 32, 44)
(2)	(8, 10, 11, 14, 20, 21, 26, 32, 44)
(2, 8)	(10, 11, 14, 20, 21, 26, 32, 44)
(2, 8, 10)	(11, 14, 20, 21, 26, 32, 44)
(2, 8, 10, 11)	(14, 20, 21, 26, 32, 44)
(2, 8, 10, 11, 14)	(20, 21, 26, 32, 44)
(2, 8, 10, 11, 14, 20)	(21, 26, 32, 44)
(2, 8, 10, 11, 14, 20, 21)	(26, 32, 44)
(2, 8, 10, 11, 14, 20, 21, 26)	(32, 44)
(2, 8, 10, 11, 14, 20, 21, 26, 32)	(44)
(2, 8, 10, 11, 14, 20, 21, 26, 32, 44)	()

Chris Grimes
CS 46101 Algorithms
Homework# 2

6. 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.

```
Algorithm countSort(S):
    Input: sequence S with n elements
    Output: a new sequence with S's n elements sorted in ascending order
        create an array X of length n to store the new sequence
        while i>=0 && I <= n do  //where I is the accessed element of array X ex: X[i]
            current<-X[i]
            count<- subFunc(S, current's key)
            X[count]<- current
        create a sequence rtnVal
        while i>=0 && I <= n do
            rtnVal.pushBack(X[i])
        return rtnVal

Algorithm subFunc(S, b):
    Input: a sequence S with n elements, a key b
    Output: the number of elements in sequence S whose keys < b
        count<-0
        while i>=0 && I <= n do
            if S[i]'s key < b
                count++
        return count
```

Chris Grimes
CS 46101 Algorithms
Homework# 2
(b) (3 points) Determine the number of comparisons made by count-sort. What is its running time?

Because countSort contains the function subFunc which is called once for each element in sequence S to make comparisions for each of the n elements in sequence S the “Big-Ohh” notation for countSort is O(n^2).

Chris Grimes
CS 46101 Algorithms
Homework# 2
(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.

One way to deal with countSort not working correctly if two or more keys have the same value is to modify subFunc to increment count if S[i]'s key <= b. This would increment count for each key that is less than or equal to S[i]'s key