CS2210 Assignment #3:

1. (2 marks) Consider a hash table of size M = 7 where we are going to store integer key values. The hash function is h(k) = k mod 7. Draw the table that results after inserting, in the given order, the following key values: 18, 11, 12, 47, 22. Assume that collisions are handled by separate chaining.

0		
1	22	
2		
3		
4	18	11
5	12	47
6		

2. (2 marks) Show the result of the previous exercise, assuming collisions are handled by linear probing.

0	47
1	22
2	
3	
4	18
5	11
6	12

3. (2 marks) Repeat exercise (1) assuming collisions are handled by double hashing, using secondary hash function $h'(k) = 5 - (k \bmod 5)$.

k	**h(k) = k mod 7**	**d(k) = 5 – (k mod 5)**	**Probes**		
18	4	2	4		
11	4	4	4	8	0
12	5	3	5		
47	5	3	5	8	1
22	1	3	3		

0	11
1	47
2	
3	22
4	18
5	12
6	

4. (3.5 marks) Consider the following algorithm. Algorithm foo(n) if n = 0 then return 1 else { x ← 0 for i ← 1 to n do x ← x + x/i x ← x + foo(n − 1) return x } The time complexity of this algorithm is given by the following recurrence equation: f(0) = c1 f(n) = f(n − 1) + c2n + c3, for n > 0 where c1, c2, and c3 are constants. Solve the recurrence equation and give the value of f(n) and its order using big-Oh notation. You must explain how you solved the recurrence equation.

f(0) = c1

f(n) = f(n-1) + c2n + c3, for > 0

f(n-1) = f((n-1)-1) + c2(n-1) + c3

f(n-1) = f(n- 2) + c2(n-1) + c3

f(n-2) = f(n- 3) + c2(n-2) + c3

f(n-3) = f(n- 4) + c2(n-3) + c3

f(n) = [f(n- 2) + c2(n-1) + c3] + [c2n + c3]

f(n) = [f(n- 3) + c2(n-2) + c3] + [c2(n-1) + c3] + [c2n + c3]

f(n) = [f(n- 4) + c2(n-3) + c3] + [c2(n-2) + c3] + [c2(n-1) + c3] + [c2n + c3]

f(n) = f(n-4) + c2(n-3) + c2(n-2) + c2(n-1) + c2n + c3 + c3 + c3 + c3

f(n) = f(n-i) + c2(n-i+1) + c2(n-i+2) + c2(n-i+3) + c2(n-i+i) + ic3

Let (n-i) = 0, n = i in order to get to the base case.

f(n) = f(i-i) + c2(i-i+1) + c2(i-i+2) + c2(i-i+3) + c2(i-i+i) + ic3

f(n) = f(0) + c2(1) + c2(2) + c2(3) + c2(n) + nc3

f(n) = c1 + 6c2 + c2n + nc3

Therefore the big-Oh notation is O(n).

5.(i) (7 marks) Write in pseudocode an algorithm maxValue(r) that receives as input the root r of a tree (not necessarily binary) in which every node stores an integer value and it outputs the largest value stored in the nodes of the tree.

For a node v use v.value to denote the value stored in v; v.isLeaf has value true if node v is a leaf and it has value false otherwise. To access the children of a node v use the following pseudocode: for each child c of v do

```
int maxValue(node r) {
    // Initalize variables:
    node currentNode = r;
    int leftLargest, rightLargest;
    // Check's to see if the root is empty.
    if (root == null) return null;
    // Base case if there's only the root.
    if (currentNode.getLeft() == null && currentNode.getRight() == null)
        return currentNode.value();
    // Return the largest out of the left side.
    if (currentNode.getLeft() != null) {
        if (!currentNode.getLeft().isLeaf)
            leftLargest = maxValue(currentNode.getLeft());
        else {
            // Once the leaf parent is reached, it checks each child.
            for each child c of currentNode do {
                if (currentNode.value() > leftLargest)
                    leftLargest = currentNode.value();
            }
        }
    }
```

```
        // Return the largest out of the right side.
        if (currentNode.getRight() !=null)
            if (!currentNode.getRight().isLeaf)
                rightLargest = maxValue(currentNode.getRight());
            else {
                // Once the leaf parent is reached, it checks each child.
                for each child c of currentNode do {
                    if (currentNode.value() > rightLargest)
                        rightLargest = currentNode.value();
                }
            }
        }
        //return the rightLargest if it's the largest value.
        if (rightLargest > leftSide && rightLargest > currentNode.value())
            return rightLargest;
        //return the leftLargest if it's the largest value.
        if (leftLargest > rightLargest && leftLargest > currentNode.value())
            return leftLargest;
        //Otherwise the currentNode is the largest node by default.
        return currentNode.value();
}
```

5.(ii) (3.5 marks) Compute the worst case time complexity of your algorithm as a function of the total number n of nodes in the tree. You must:

• explain how you computed the time complexity:

The time complexity would equal O(n) since there are the 2 initial constants being the initialized variables, and the base case. The algorithm recursively searches in a linear fashion through the left side of the tree till it searches once for each child on a leaf parent, and then it repeats this process for the right side of the tree. Afterwards it performs 2 final comparisons between the left and right largest to finish.

So the formula would be O(n) = c1 + c2 + c3 + n + c4 + c5

In which: c1 is the initialized variables, c2 is the base case that's always checked, c3 is the case of the tree ended at a root value, n is the amount of nodes in the tree, c4 is the first comparison and c5 is the second comparison.

• give the order of the time complexity of the algorithm:

O(n)