Cs 2210a - Assignment 3

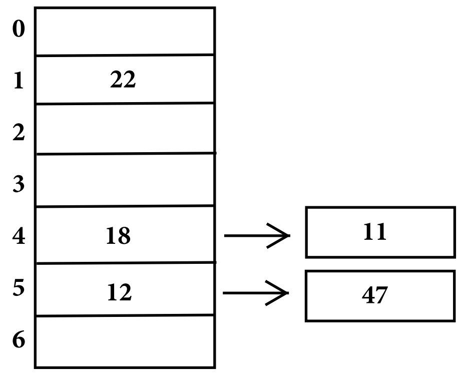
Student Name: MingCong, Zhou

Student Number: 250945414

Student Account: mzhou272

CS210a Data Structures and Algorithms
Assignment 3
Due date: October 24 at 11:59 pm
Total of 20 Marks

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.



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 mod 5).

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

$h'(11) = 5 - (11 \bmod 5) = 4$

$h'(47) = 5 - (47 \bmod 5) = 3$

$h'(22) = 5 - (22 \bmod 5) = 3$

4. (3.5 marks) Consider the following algorithm.

Algorithm foo(n)

```
c1 { if n = 0 then return 1
     else {
        $x \leftarrow 0$
        for $i \leftarrow 1$ to $n$ do $x \leftarrow x + x/i$  } c2n
  c2 {  $x \leftarrow x + foo(n-1)$
        return $x$ } c3
     }
```

The time complexity of this algorithm is given by the following recurrence equation:

$f(0) = c_1$
$f(n) = f(n-1) + c_2 n + c_3$, for $n > 0$

where c_1, c_2, and c_3 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 n > 0

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

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

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

.

.

f (n) = f (0) + c2 * 1 + c2 * 2 + c2 * 3 + ... + c2(n-3) + c2(n-2) + c2(n-1) + c2n + c3

= c1 + c2(1+2+3+... + n) + nc3

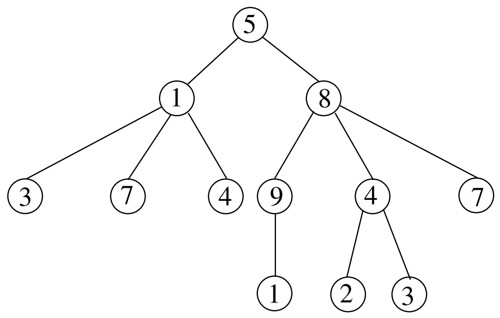
= c1 + c2(n+1)n/2 + nc3

= c1 + 1/2 * (n^2 + n) c2 + nc3

therefore O(f(n)) is n^2 which is n squre.

Each one of the recurrence equation is calling the next one. The time complexity is the sum of all the equations

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 example, for the following tree the algorithm must output the value 9.



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**

```
    Algorithm maxValue(Node r) {
    max ← r.value;
    if !u.isLeaf  {
       for each child u of r do {
          if  maxValue(u) > max {
             max ← maxValue(u);
          }
       }
    }
    return max;
 }
```

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
- give the order of the time complexity of the algorithm

```
        Algorithm maxValue(Node r) {
        max ← r.value;
        {if !u.isLeaf  {
           for each child u of r do {
C1            if  maxValue(u) > max {  } 2C2   ] 2C2 × degree
                 max ← maxValue(u);
              }
           }
        }
        { return max;
     }
```

First of all, no matter what the algorithm looks like. We have to traverse all the node and compare them with the parent to find the one that contains the largest integer. Outside of the for loop there is a constant value of C1 that is produced. Inside the for loop each recursion run only once upon the if statement and once on the max value assignment. Therefore a constant of 2C2 is performed. The for loop traverse all the node of the tree which is denoted as n(degree). Finally, the time complexity of the for loop is 2C2 * n
The entire algorithm produce a time complexity of f(n) = 2C2 * n + c1.
The O(f(n)) is equal to n.