Practice Assignment 1

Daniel Schroeder

schrodan@oregonstate.edu

CS 325 Algorithms

Fall 2017

30 September 2017

I. Problem 1

Problem 1. For each of the following, indicate whether $f = O(g), f = \Omega(g) or f = \Theta(g)$

$(a) f(n) = 12n - 5, g(n) = 1235813n + 2017$

Upper bound proved by C=1, N=1

Lower bound proved by C=$\frac{1}{1235813}$, N=1

$\boldsymbol{f = \Theta(g)}$

$(b) f(n) = nlogn, g(n) = 0.00000001n$ $\boldsymbol{f = \Omega(g)}$

$(c) f(n) = n^{2/3}, g(n) = 7n^{3/4} + n^{1/10}$ $\boldsymbol{f = O(g)}$

$(d) f(n) = n^{1.0001}, g(n) = nlogn$ $\boldsymbol{f = \Omega(g)}$

$(e) f(n) = n6^n, g(n) = (3^n)^2$ $\boldsymbol{fO(g)}$

II. Problem 2

Prove that $log(n!) = \Theta(n * log(n))$

We can rewrite this equation as:

$$log(1) + log(2) + ... + log(n-1) + log(n) = \Theta(log(n) + log(n) + ... + log(n))$$

Which can prove the upper bound:

$$log(1) + log(2) + ... + log(n-1) + log(n) \leq log(n) + log(n) + ... + log(n)$$

This is clear because we are adding up the largest value of the left summation n times, making a significantly larger sum.

For the lower bound, we can make our constant C equal to $\frac{1}{2}$ and get the equality:

$$log(1) + log(2) + ... + log(n-1) + log(n) \geq \frac{n}{2} * log(\frac{n}{2})$$

or

$$log(1) + log(2) + ... + log(n-1) + log(n) \geq log(\frac{n}{2}) + log(\frac{n}{2}) + ... + log(\frac{n}{2})$$

We know this inequality is true because the right side is only adding up half of the amount of numbers than what are being summed on the left side of the equation, making the left side greater for all n.
So with a proven upper and lower bound for some constant c and arbitrary n >0, we prove that $log(n!) = \Theta(n * log(n))$
help found at https://stackoverflow.com/questions/2095395/is-logn-%CE%98n-logn

III. Problem 3

A Correct Solution:

```
private static void binary(int n){
   if(n > 1){
      binary(n/2);
   }
   System.out.print(``''+n%2);
}
```

This recursive function divides the input value by 2 until it is equal to 1 then enters the printing aspect of the function with prints the modulus 2 of the given value of n. This works because each time the number is devided by two, truncated because it is an integer, and forced through the same process. So each division by two is working its way up the powers of 2 ladder that makes up the binary representaion.

I needed help solving this solution because it was not initially intuitive that continuously dividing by two was creating a powers of two dynamic. I tried to create recursive functions that would pass in the current exponent of 2, then call the function again but with 2 to the power+1 like:

```
private static void binary(int power, int n){
   if(n > 0){
      n = n - Math.pow(2, power);
      System.out.print(n%2);
      binary(power+1, n);
   }
}
```

I was all over the place and could not figure out how to then work backwards and achieve the correct order for the binary digits recursively. When I searched for solutions to this problem, I cam across this solution at http://www.geeksforgeeks.org/binary-representation-of-a-given-number/ and ran it through IntelliJ and saw how it worked.

IV. Problem 4

When reading about tree traversal and reading the wiki page, I started to notice patterns in the postorder and preorder that we were to use as input for our recursive algorithm. I could tell that I would have to decriment the array based on what node was the new root and what would become the left trees and the right trees. I spent a long time trying to come up with a tricky way to create new subarrays and increment the index of the new root, but could not figure it out before turning to other resources.

I found a website that described that when you know the root of the tree, everything that comes prior to that node in the postorder array is a part of that tree. For example:

```
char[] preorder = {'F', 'B', 'A', 'D', 'G', 'H', 'I'};
char[] postorder = {'A', 'D', 'B', 'H', 'I', 'G', 'F'};
```

The "F" is the root node of the tree, and we know that the "B" then becomes the head of the left sub-tree. So, if we look at the post array, the subarray A, D, that comes before the "B" is the subtree under root "B." Similarly, we can infer that "G" is then the root of the right sub-tree, so everything before "G" and after "B" in the postorder array H, I is the right sub-tree under root "G." After a while of trying to develop a method of recursively recreating the full binary tree, I was left to search for help. I knew that the base case was when the subarray became a size of one which meant we reached a leaf, I knew I had to iterate through postorder until I found the root node I was working with and find the correct subarray of children, but I needed help developing a solution. The website http://ide.geeksforgeeks.org/pfFXou had a lot of helpful resources and code that I used to implement a working Java solution.

```
private static Node constructTree(char[] preorder, char[] postorder,
                            int start, int end){
   Node root = new Node(postorder[end]);
      loc++;
   //if there is only one element, it is a leaf so return it
   if(start == end)
      return root;

   //look for the location in the postorder array of the new root from pre
   int index;
   for (index = 0; index < postorder.length; index++) {
      if(postorder[index] == preorder[loc])
         break;
   }
```

```
    //pass in the new indeces of the subarrays for the new trees
    root.left = constructTree(preorder, postorder, start, index);
    root.right = constructTree(preorder, postorder, index+1, end-1);

    return root;
}
```