Anand S Kothari

10/03/2017

Practice Assignment 1:

Problem 1: Indicate whether f = O(g), f = Ω(g) or f = Θ(g)

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

(b) $f(n) = n \log n$, $g(n) = 0.00000001n$ ----->$f(n)= O(g(n))$

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

(d) $f(n) = n^{1.0001}$, $g(n) = n \log n$ -----> $f(n)= \Omega (g(n))$

(e) $f(n) = n6^n$, $g(n) = (3^n)^2$ ----->$f(n)=\Theta(g(n))$

Problem 2. Prove that log(n!) = Θ (n log n)

Ans:

Proving log (n!)=Ω (n log n)

log n!=log(1*2*3…n)
=log1+log2+log3+…+log n {Powers are equal such as $\log_a(xy) = \log_a x + \log_a y$}
=log1+…+log n/2+…+log n
≥log n/2+log((n/2)+1)+…+log n
≥log(n/2)+log(n/2)+ log(n/2)+…+log(n/2) {Adding n/2 }
≥log(n/2* n/2*n/2… n/2) {Multiplying n/2 }
≥log(n/2 ^(n/2))
≥n/2 log(n/2) {Exponential Rule}

Therefore we can say that , log(n!)≥ n/2 log(n/2)
log(n!)≥ n/2* log(n/2)

So, hence proved that log(n!)=Ω (n log n)

Referred: C. (2017). *Confused about proof that log(n!) = Θ (n log n)*. *Cs.stackexchange.com*. Retrieved 3 October 2017, from https://cs.stackexchange.com/questions/47561/confused-about-proof-that-logn-thetan-log-n

Problem 3. Write a recursive algorithm to print the binary representation of a non-negative integer. Try to make your algorithm as simple as possible. Your input is a non-negative integer n. Your output would be the binary representation of n. For example, on input 5, your program would print '101'.

Ans:

```
#include <stdio.h>
 int main()
{
   int binary,number;  //Initialize
   printf("Please enter a positive number: ");  //User Entry
   scanf("%d", &number);
   binary = convertToB(number);  //Calling the function convertToB
   printf("The binary equivalent of %d is %d\n", number, binary);
}
 int convertToB(int number)
{
  if(number < 0)    //If user entered a negative number
  {
  printf("Enter a positive number");
  }
  else
  {
    if (number == 0)
    {
       return 0;
    }
 else
    {
        return (number % 2) + 10 * convertToB(number / 2);
  }}}
```

Problem 4.

(a) Read tree traversal from wikipedia: https://en.wikipedia.org/wiki/Tree_traversal , the first section, Types.

(b) Recall that a binary tree is full if every non-leaf node has exactly two children. Describe and analyze a recursive algorithm to reconstruct an arbitrary full binary tree, given its preorder and postorder node sequences as input. (Assume all keys are distinct in the binary tree)

Ans:

As said in the question and we know that a full binary tree will have exactly two children other than the leaves. We cannot create a single binary tree with only one of the data from preorder or postorder given but since we are given both the list of preorder and postorder we can create a binary tree by diving the number of values into half and placing them either left or right into the tree accordingly being the root node which is the first element of preorder list is the same for both. Considering it as a left to right traversal.

For example:

Preorder : A B C D E

Postorder: C D B E A

One being the root of both of them and diving remaining of both of the list into half we get

	Root in preorder	Left Nodes of the tree	Right Nodes of the tree	Root in postorder
Preorder:	A	B E D	C	
Postorder:		E D B	C	A

So using a recursive algorithm we can create the tree by placing the sides of the tree as mentioned above.