CS325

Chih Hsuan Huang

Problem 1. & Problem 2.

Problem 1.

(a) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{12n-5}{1235813n+2019} = \frac{12}{1235813}$

$\therefore$ f(n) growing rate = g(n) growing rate, $f = \Theta(g)$

(b) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n\log n}{0.000000001n} = \infty$

$\therefore$ f(n) growing rate > g(n) growing rate, $f = \Omega(g)$

(c) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{\frac{2}{3}}}{4n^{2}+n^{\frac{1}{10}}} = 0$

$\therefore$ f(n) growing rate < g(n) growing rate, $f = O(g)$

(d) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{1.0001}}{n\log n} = \infty$

$\therefore$ f(n) growing rate > g(n) growing rate, $f = \Omega(g)$

(e) $\because \lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n6^n}{(3^n)^2} = 0$

$\therefore$ f(n) growing rate < g(n) growing rate, $f = O(g)$

Problem 2.

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

prove $\log(n!) = O(n\log n) \Rightarrow \log(n!) = \log(1\cdot 2\cdot 3 \cdot \cdots (n-1)\cdot n)$

$= \log 1 + \log 2 + \log 3 + \cdots + \log n$

$\le \log(n) + \log(n) + \cdots + \log(n) = n\log(n)$

$\Rightarrow \log(n!) \le n\log(n)$

prove $\log(n!) = \Omega(n\log n) \Rightarrow \log(n!) = \log 1 + \log 2 + \cdots + \log(n-1) + \log n$

$\ge \log(\frac{n}{2}+1) + \log(\frac{n}{2}+2) + \cdots + \log(n-1) + \log n$

$\ge \log(\frac{n}{2}) + \log(\frac{n}{2}) + \cdots + \log(\frac{n}{2}) + \log(\frac{n}{2}) \ge \frac{n}{2}\log(\frac{n}{2})$

$\therefore \log n! = \Theta(n\log n)$

Problem 3.

def binary_representation(n):

```
    if n > 1:
        binary_representation(n // 2)
    print(n % 2, end='')

number = int(input("your binary representation is: "))

binary_representation(number)
```

Problem 4.

- If either the preorder or postorder sequence is empty, return **None**
- The first node in the preorder sequence is the root of the current subtree.
- Create a node with this value.
- Identify the index of the root node in the postorder sequence. This index divides the postorder sequence into left and right subtrees.
- Recursively call the algorithm for the left and right subtrees using the appropriate portions of the preorder and postorder sequences.
- Return the root node.

Example:

```
function reconstruct(preorder, postorder):
    if not preorder or not postorder:
        return None
    root = Node(preorder[0])
    if len(preorder) > 1:
        root.left = reconstruct(preorder[1:1+postorder.index(preorder[1])+1],
postorder[:postorder.index(preorder[1])+1])
        root.right = reconstruct(preorder[postorder.index(preorder[1])+2:],
postorder[postorder.index(preorder[1])+1:-1])
    return root
```