Hyuntaek Oh

Amir Nayyeri

CS 325_002

23 Jan 2024

Practice Assignment 1

Problem 1. For each of the following, indicate whether f=O(g), f=Ω(g) or f=θ(g).

(a) $\mathrm{f(n)} = 12n - 5,\ \mathrm{g(n)} = 1235813\mathrm{n} + 2017.$

$\mathrm{O(g)}$: $\mathrm{c} * \mathrm{g(n)} \geq f(n)$

$c * (1235813n + 2017) \geq 12n - 5$

$c \geq \frac{12n-5}{1235813n+2017}$

$c \geq 0.xxx$

$\Omega\mathrm{(g)}$: $\mathrm{c} \leq 0 \Rightarrow$ contradiction

$\therefore \mathrm{O(g)}$

(b) $\mathrm{f(n)} = nlogn,\ \mathrm{g(n)} = 0.00000001\mathrm{n}.$

$\mathrm{O(g)}$: $\mathrm{c} * \mathrm{g(n)} \geq f(n)$

$\mathrm{c} * 0.00000001\mathrm{n} \geq nlogn$

$c \geq \frac{\cancel{n} * logn}{0.00000001 * \cancel{n}}$

$\frac{c}{10^8} \geq logn \Rightarrow$ contradiction

$\Omega\mathrm{(g)}$: $\frac{c}{10^8} \leq logn$

$\therefore \Omega\mathrm{(g)}$

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

$\mathrm{O(g)}$: $\mathrm{c} * \mathrm{g(n)} \geq f(n)$

$\mathrm{c} * 7n^{3/4} + n^{1/10} \geq n^{2/3}$

$c \geq \frac{n^{2/3}}{7n^{3/4}+n^{1/10}}$

$c \geq 0.xxx$

Ω(g): $0 < c \leq 0.xxx$

$\therefore \theta$(g)

(d) f(n) $= n^{1.0001}$, g(n) $= nlogn$.

O(g): $c * g(n) \geq f(n)$

$c * nlogn \geq n^{1.0001}$

$c \geq \frac{\cancel{n} * n^{0.0001}}{\cancel{n} * logn}$

$c \geq 0.xxx$

Ω(g): $0 < c \leq 0.xxx$

$\therefore \theta$(g)

(e) f(n) $= n6^n$, g(n) $= (3^n)^2$.

O(g): $c * g(n) \geq f(n)$

$c \geq n * \left(\frac{2}{3}\right)^n$

$c \geq 0.xxx$

Ω(g): $0 < c \leq 0.xxx$

$\therefore \theta$(g)

Problem 2. Prove that $log(n!) = \theta(nlogn)$. (Logarithms are based 2)

$$\log(n!) = \theta(nlogn) \rightarrow \begin{cases} O(nlogn) \\ \Omega(nlogn) \end{cases}$$

$$f(n) = \log(n!) = \log(1) + \log(2) + \cdots + \log(n)$$

$$c * g(n) = c * nlogn = c * [\log(n) + \log(n) + \cdots + \log(n)]$$

1) $O(nlogn)$: $f(n) \leq c * g(n)$

$$\log(1) + \log(2) + \cdots + \log(n) \leq c * [\log(n) + \log(n) + \cdots + \log(n)]$$
$$\Longrightarrow\ c * n\log(n)$$

If the value of n is infinite and c is greater than zero, every single log(n) is greater than or equal to each term in f(n). Therefore, this equation holds.

2) $\Omega(nlogn)$: $f(n) \geq c * g(n)$

$$\log(1) + \cdots + \log(n) \geq \frac{1}{2} * [\log(n) + \cdots + \log(n)]$$
$$\Longrightarrow \frac{1}{2} * nlog(n)$$

If the value of n is large enough and 0 < c < 1, the total value of right side of equation will be equal or less than left side one. Therefore, this equation also holds.

In conclusion, the equation "$\log(n!) = \theta(nlogn)$" holds.

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'.

```
def ToBinary(n):
    if n<2:
        print(int(n), end="")
    else:
        ToBinary(int(n/2))
        print(int(n)%2, end="")
```

e.g. The result of ToBinary(10) is "1010"

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 pre-order and post-order node sequences as input. (Assume all keys are distinct in the binary tree)

The answer to this question is that from the pre-order and post-order node sequences, a program reconstruct an arbitrary full binary tree in a recursive way.

For example, pre-order and post-order lists are [1, 2, 4, 8, 9, 5, 3, 6, 7], [8, 9, 4, 5, 2, 6, 7, 3, 1] respectively. While processing of generating full binary tree, we use the indices of two lists, and the program starts from the 0^{th} ("1") to the 6^{th} ("7") index of pre-order list one by one as a global value. Then, we can find a specific element that has a same value in pre-order list from post-order list by using loops and make two nodes for left and right by using the indices of these list. The code below accounts for it.

```
preIndex = 0                    # the 0th index of pre-order list
def constructTree (pre-order list, post-order list, low index, high index, size of list)
        global preIndex
        if preIndex >= size or low > high:
                return None

        node = TreeNode(pre[preIndex])
        preIndex += 1

        if l==h or preIndex >= size:
                return node

        i=l
```

```
    while i<=h:                                    # To find a same value
        if pre[preIndex] == post[i]:
            break
        i+=1

if i<=h:
        root.left = constructTree(pre, post, l, I, size)
        root.right = constructTree(pre, post, i+1, h-1, size)

return root
```

To analyze time complexity of this code, we need to focus on two things: the length of pre-order or post-order list and while loop for finding a specific element in the list. If the length of lists is large enough, generating full binary tree can be O(n) because as you can see the code, the program needs to check from start to end of the list, which means it follows the length of the list.

The while loop in the code above to find a specific, same element in the pre-order list from post-order list can be O(logN) because the order of numbers in post-order list can reduce the search area into the half of it, eliminating previous search.

Finally, these two things happen simultaneously, so we need to product these results as like O(N) * O(logN) => O(NlogN). The total time complexity of generating an arbitrary full binary tree is O(NlogN).

I got some ideas from this website (“https://www.geeksforgeeks.org/full-and-complete-binary-tree-from-given-preorder-and-postorder-traversals/”)