(1) Assuming two function $f(x)$ and $g(x)$ s.t.

$$c_1 g(x) \le f(x) \le c_2 g(x)$$

Name: Krishna Biswakarma
Roll: 24CS60R71
Date: 19/08/2024

If $f = \theta(g)$ then we need to prove $g = \theta(f(\theta))$

So, let $f(x) = \theta(g(x))$

then, $c_1 g(x) \le f(x) \le c_2 g(x)$ — (i).

$\Rightarrow c_1' f(x) \le g(x) \le c_2' f(x)$ [$g(x) = \theta(f(x))$].

using (i)

$\frac{c_1'}{c_2} f(x) \le g(x) \le \frac{c_2'}{c_1} f(x)$

$\frac{c_1'}{c_2}$ → Assume: m ; $\frac{c_2'}{c_1}$ → n

We can rewrite it as: $mf(x) \le g(x) \le nf(x)$

$\therefore g(x) = \theta(f(x))$

Hence, symmetric is proved for "θ".

•) Let $f(x)$, $g(x)$ and $h(x)$ be three functions.

if $f(x) = \theta(g(x))$, $g(x) = \theta(h(x)) \Rightarrow f(x) = \theta(h(x))$

Now,

$c_1 g(x) \le f(x) \le c_2 g(x)$ — (i)

$c_3 h(x) \le g(x) \le c_4 h(x)$ — (ii)

Put (ii) in (i)

$c_1 c_3 h(x) \le f(x) \le c_2 c_4 h(x)$

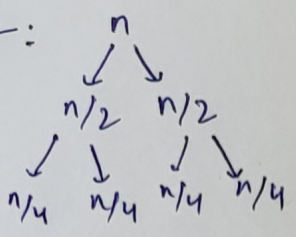
$c_1 c_3$ → m ; $c_2 c_4$ → n

$\therefore mh(x) \le f(x) \le nh(x) \Rightarrow f(x) = \theta(h(x))$

$\therefore \theta$ is transitive.

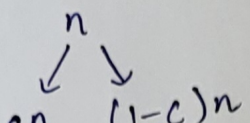
(3) •> For Merge sort we know, $T(n) = 2T(n/2) + n$

For Half-Half Split:



Total depth = $\log_2^n$

For Other splits



[NOTE: $c \neq \frac{1}{2}$]

Total depth > $\log_2^n$

because split is unbalanced.

∵ less recursion call will be needed in half-half split compared to any other split and hence it is optimal.

•> Proving: other split also takes $O(n\log n)$

Assume, $T(n) = T(cn) + T((1-c)n) + n \quad [c \neq \frac{1}{2}]$

$0 < c < 1$

Using Masters Theorem: $T(n) = aT(n/b) + f(n)$.

Here, $a, b \neq 2$.

Now, ∵ $0 < c < 1$ so, none of the parts can be exponentially larger and hence still satisfying the criteria $O(n\log n)$.

(4) Yes pruning can be used in LCS using DP to improve efficiency by skipping some parts of the problem space.

Pseudo Code:

NOTE- str1 = [$a_1, a_2, \ldots, a_m$]
str2 = [$b_1, b_2, \ldots, b_n$]

```
LCS_with_pruning(str1, str2):
# Length of str1 = m
# Length of str2 = n
done[m+1][n+1], value[m+1][n+1];
max_length = 0
function LCS(i, j): # Function starts here.
    if (i==0 or j==0) return 0;
    if (done[i][j] == True) return value[i][j];
    # Pruning condition
    if(abs((m-i)-(n-j)) > max_length)):
        done[i][j] = True
        value[i][j] = 0
        return 0
    if (str1[i-1] == str2[j-1]):
        value[i][j] = LCS(i-1, j-1) + 1
    else
        value[i][j] = max(LCS(i-1, j), LCS(i, j-1))
    done[i][j] = True
    max_length = max(max_length, value[i][j])
    return value[i][j]
```