Solution to Homework Assignment 1 (CS 430)

Saptarshi Chatterjee
CWID: A20413922

March 23, 2018

1 Solution to Problem 2.3-3 on page 39

Q. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2, & \text{if } n = 2, \\ 2T(n/2) + n, & n = 2^k, \text{ for } k > 1 \end{cases}$$

is $T(n) = n \log(n)$

Answer -

Let $n = 2^k$ (As n is in power of 2 , given)

$$\begin{aligned} T(n) &= T(2^1) && \text{(When } k = 1 \text{)} \\ &= T(2) \\ &= 2 && \text{(From given recurrence when } n = 2\text{)} \\ &= 2 \log 2 \end{aligned}$$

Now we need to prove given $T(2^k) = 2^k \log(2^k)$ holds true , then $T(2^{k+1}) = 2^{k+1} \log(2^{k+1})$ should hold true as well.

$$\begin{aligned} T(2^{k+1}) &= 2T(2^{k+1}/2) + 2^{k+1} && \text{(From given recurrence when } k > 1\text{)} \\ &= 2T(2^k) + 2^{k+1} \\ &= 2 \times 2^k \log(2^k) + 2^{k+1} && \text{(by inductive hypothesis)} \\ &= 2^{k+1}(\log 2^k + 1) \\ &= 2^{k+1}(\log 2^k + \log 2) \\ &= 2^{k+1} \log 2^{k+1} && \ldots \text{hence proved} \end{aligned}$$

2 Solution to Problem 2.3-4 on page 39

Q. We can express insertion sort as a recursive procedure as follows. In order to sort $A[1 \ldots n]$, we recursively sort $A[1 \ldots n-1]$ and then insert $A[n]$ into the sorted array $A[1 \ldots n-1]$. Write a recurrence for the running time of this recursive version of insertion sort.

Answer - An algo for the recursive procedure will be -

```
def recursive_insertion(result, n):
    if n > 0:
        recursive_insertion(result, n - 1)
        for i in range(0, n):
            if result[i] > result[n]:
                result_n = result[n]
                del result[n]
                result.insert(i, result_n)
                break
        return result
    else:
        return result[n]


data = [12, 15, -23, -4 , 6, 10, -35, 28]
print(recursive_insertion(data, len(data) - 1))
```

From the above code the recurrence relation will be -

$$T(n) = \begin{cases} 1, & \text{if } n = 1, \\ T(n-1) + n, & n > 1 \end{cases}$$

*Solving the recurrence we get [**?**] -*

$$\begin{aligned} T(n) &= T(n-1) + n \quad \text{(From given recurrence when } n > 1) \\ &= T(n-2) + (n-1) + n \\ &= \ldots \\ &= n + (n-1) + (n-2) + \cdots + 1 \quad \text{(given, when } n = 1, T(n) = 1) \\ &= n(n+1)/2 \quad \text{(sum of n natural numbers)} \\ &= \Theta(n^2) \end{aligned}$$

3 Solution to Problem 2-3(a) on page 41

Q. In terms of Θ notation, what is the running time of code fragment for Horner's rule?

Answer -

Given the given the coefficients $a_0, a_1, \ldots a_n$ (stored as an array) an and a value for x , algo for the Horner's rule [**?**] -

```
def evaluate(x, a):
    result = 0
    for i in range(len(a)−1, −1, −1):
        result = a[i] + (x ∗ result)
    return result
```

As the number of iterations is same as the value of n (length of the array storing values of 'a') , it has an asymptotically tight bound of n . So complexity of the code fragment would be $\Theta(n)$

4 Solution to Problem 3-3(a), fourth row only, on pages $61 - 62$. Justify your answers!

Q.Rank these functions $2^{\lg n}, (\lg n)^{\lg n}, e^n, 4^{\lg n}, (n+1)!, \sqrt{\lg n}$ by order of growth; that is, find an arrangement $g_1, g_2, \ldots g_3$ of the functions satisfying $g_1 = \Omega(g_2)$, $g_2 = \Omega(g_3)$ . . .

Answer -

$g(n) = (n+1)! = \Omega(e^n) \quad \ldots$
(Analysis - $e^n = e \times e \times e ... n$ times
$(n+1)! = (n+1) \times n \times (n-1) \cdots \times 1$. As e is roughly 2.71828 , most of these terms will be greater than e , when $n > 4$. $(n+1)!$ is definitely greater than e^n for large n)

$g(n) = e^n = \Omega((\lg n)^{\lg n}) \quad \ldots$
(Analysis - If we take ln base e for both the sides we get R.H.S. as $\log n \ln_e \log n$ which is a function of logarithmic growth , where an L.H.S becomes n . Which is linear growth function.
Linear growth is faster than logarithmic growth)
$g(n) = (\lg n)^{\lg n} = \Omega(4^{\lg n})$
(Analysis - R.H.S increases only in the power of 4 , which is a constant .

textSo there is definitely an n for which $\log n > 4$)
$g(n) = 4^{\lg n} = \Omega(2^{\lg n})$
(Same as above . Both grows in same power, but LHS grows in the order of 4 and RHS is in the order of 2
$g(n) = 2^{\lg n} = \Omega(\sqrt{\lg n}) \ldots$ (LHS grows exponentially so faster , RHS have $\sqrt{}$ of logarithmic)

5 Problem 4-3(a) on page 108; solve this problem two ways: first with the master theorem on page 94, and then using secondary recurrences (page 13 in the January 10 notes)

Q. Find asymptotic tight bound for $T(n) = 4T(n/3) + n \log n$

Answer -

Using master theorem

If we write the given recurrence as $af(\frac{n}{b}) + f(n)$, then we get $f(n) = n \log n, a = 4, b = 3$. Now -

$$\begin{aligned}
\frac{a.f(\frac{n}{b})}{f(n)} &= \frac{\frac{4n}{3}\log\frac{n}{3}}{n\log n} \\
&= \frac{4}{3}\frac{(\log\frac{n}{3})}{\log n} \\
&> 1 \quad (\ldots \text{for sufficiently large n})
\end{aligned}$$

Which satisfies 2nd rule of master's theorem as per lecture notes.

$$\begin{aligned}
T(n) &= \Theta(n^{log_b a}) \\
&= \Theta(n^{log_3 4})
\end{aligned}$$

Using secondary recurrence

Given recurrence is $T(n) = 4T(n/3) + n \log n$

Lets the base case be $T(1) = 1$

Lets take a secondary sequence n_i such that $T(n_i) = 4T(n_{i-1}) + n_i \log n_i$

So n_i is the argument of T() when we are i recursion steps away from the base case $n_0 = 1$. The original recurrence gives us the following secondary recurrence for n_i :

$n_{i-1} = \frac{n_i}{3}$, ... implies $n_i = 3n_{i-1}$

The annihilator for this recurrence is $(E-3)$, so the generic solution is $n_i = \alpha 3^i$. Plugging in the base cases $n_0 = 1$ and $n_1 = 3$, we get the exact solution $n_i = 3^i$

Notice that our original function $T(n)$ is only well-defined if $n = n_i$ for some integer $i \geqslant 0$. Now to solve the original recurrence, we do a range transformation. If we set $t_i = T(n_i)$, we have the recurrence $t_i = 4t_{i-1} + 3^i \log 3^i$. The annihilator of the recurrence is $(E-4)(E-3)$, so the generic solution is $\alpha 4^i + \beta 3^i$. Plugging in the base cases$t_0 = 1, t_1 = 7$ we get $\alpha = 4, \beta = -3$ the exact solution

$$t_i = 4^{i+1} - 3^{i+1}$$

Finally, we need to substitute to get a solution for the original recurrence in terms of n, by inverting the solution of the secondary recurrence. If $n_i = 3^i$, then (after a little algebra) we have $i = log_3 n$

Substituting this into the expression for ti, we get our exact, closed-form solution.

$$\begin{aligned} T(n) &= 4^{i+1} - 3^{i+1} \\ &= 4 \times 4^{log_3 n} - 3 \times 3^{log_3 n} \\ &= 4 \times n^{log_3 4} - 3 \times n^{log_3 3} \\ &= \Theta(n^{log_3 4}) \end{aligned}$$

References

[1] Horner rule implimentation
$https://introcs.cs.princeton.edu/python/21function/horner.py.html$

[2] Insertion sort analysis
$http://crypto.stanford.edu/~dabo/courses/cs161_spring01/cs161-02.pdf$

[3] Tree Template
$http://www.texample.net/tikz/examples/merge-sort-recursion-tree/$