CS204: Design and Analysis of Algorithms

220001010, 220001011, 220001012

January 18, 2024

1 Solving T(n)=T(n-1)+n

The given recurrence relation is T(n) = T(n-1) + n. To solve this recurrence relation, we can use iteration and then find a pattern. Let's expand the relation:

$$\begin{aligned}T(n) &= T(n-1)+n\\ &= T(n-2)+(n-1)+n\\ &= T(n-3)+(n-2)+(n-1)+n\\ &= T(n-4)+(n-3)+(n-2)+(n-1)+n\\ .\\ .\\ &= T(1)+2+3+4+.....+(n-2)+(n-1)+n\end{aligned}$$

Now, observe that the sum n + (n-1) + (n-2) + ... + 2 + 1 is the sum of the first n natural numbers, which is given by the formula Sum = n(n+1)/2.

So,

$$T(n) = n(n+1)/2$$

Now, in terms of big O notation, we focus on the dominant term as (n) becomes large. The dominant term in n(n+1)/2 is n^2, so we can express the time complexity as:

$$T(n) = O(n^2)$$

This means that the time complexity of the algorithm described by the recurrence relation is quadratic.

2 Solving T(n)=T(n/2)+1

The given recurrence relation is T(n)=T(n/2)+1. To solve it, we can use the master theorem.

The master theorem has the form T(n) = aT(n/b) + f(n), where a >= 1, b > 1, and f(n) is an asymptotically positive function.

In this case, we have a = 1, b = 2, and f(n) = 1.

Now, let's compare f(n) with $n^{log_b a}$:

$n^{log_b a} = n^{log_2 1} = n^0 = 1$.

Compare f(n) and $\theta(n^{log_b a})$:

Since f(n) is a constant (1), and it is equal to $\theta(n^{log_b a})$, we are in case 2 of the master theorem.

Case 2 states that if $f(n) = \theta(n^{log_b a})$, then the solution to the recurrence relation is $T(n) = \theta((n^{log_b a})) * logn)$

In our case, $f(n) = \theta(n^{log_b a})$, so the solution is:

$$T(n) = \theta(n^{log_b a}) * logn) = \theta(logn)$$

Therefore, the time complexity of the algorithm described by the recurrence relation $T(n) = T(n/2) + 1$is logarithmic, specifically $\theta(logn)$.

3 Solving $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$

The given recurrence relation is $T(n) = 9 \cdot T\left(\frac{n}{3}\right) + n$. To solve this recurrence relation, we can use the master theorem.

In the master theorem, the recurrence relation has the form $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, where $a \geq 1$, $b > 1$, and $f(n)$ is an asymptotically positive function.

In this case, $a = 9$, $b = 3$, and $f(n) = n$.

Now, let's compare $f(n)$ with $n^{\log_b a}$:

- $n^{\log_b a} = n^{\log_3 9} = n^2$.

Compare $f(n)$ and $n^{\log_b a}$:

- Since $f(n)$ is n and n is $n^{\log_b a - \epsilon}$ for $\epsilon = 1$, we are in case 1 of the master theorem.

Case 1 states that if $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then the solution to the recurrence relation is $T(n) = \Theta(n^{\log_b a})$.

In our case, $f(n) = O(n^{\log_3 9 - 1}) = O(n^{2-1}) = O(n)$.

Therefore, the solution is:

$$\begin{aligned} T(n) &= \Theta(n^{\log_3 9}) \\ &= \Theta(n^2) \end{aligned}$$

4 Solving $T(n) = T\left(\frac{2n}{3}\right) + 1$

The given recurrence relation is $T(n) = T\left(\frac{2n}{3}\right) + 1$. To solve this recurrence relation, we can use the master theorem.

In the master theorem, the recurrence relation has the form $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, where $a = 1$, $b = \frac{3}{2}$, and $f(n) = 1$.

Now, let's compare $f(n)$ with $n^{\log_b a}$:

- $n^{\log_b a} = n^{\log_{\frac{3}{2}} 1} = n^0 = 1$.

Compare $f(n)$ and $n^{\log_b a}$:

- Since $f(n)$ is a constant (1), and it is equal to $\theta(n^{log_b a})$, we are in case 2 of the master theorem.

Case 2 states that if $f(n) = \Theta(n^{\log_b a})$, then the solution to the recurrence relation is $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.

In our case, $f(n) = \Theta(1)$, and $\log n$ is the term for case 2.

Therefore, the solution is:

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

5 Solving $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$

The given recurrence relation is $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$. To solve this recurrence relation, we can use the master theorem.

In the master theorem, the recurrence relation has the form $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$, where $a = 3$, $b = 4$, and $f(n) = n \log n$.

Now, let's compare $f(n)$ with $n^{\log_b a}$:

- $n^{\log_b a} = n^{\log_4 3}$.

Compare $f(n)$ and $n^{\log_b a}$:

- Since $f(n)$ is $n \log n$ and it is larger than $n^{\log_b a}$, we are in case 3 of the master theorem.

Case 3 states that if $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some $\epsilon > 0$, and $a \cdot f\left(\frac{n}{b}\right) \leq k \cdot f(n)$ for some $k < 1$ and sufficiently large n, then the solution to the recurrence relation is $T(n) = \Theta(f(n))$.

In our case, $f(n) = \Omega(n^{\log_4 3+\epsilon})$ and $3 \cdot \frac{n}{4} \log\left(\frac{n}{4}\right) \leq k \cdot n \log n$ for some $k < 1$.

Therefore, the solution is:

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

6 Solving the Recurrence Relation

Given recurrence relation:

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Using the provided method:

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \log n \\ &= 4T\left(\frac{n}{4}\right) + 2n \log n - n \log 2 \\ &= n \log n + n \log\left(\frac{n}{2}\right) + 2^2 T\left(\frac{n}{2^2}\right) \\ &= \ldots \end{aligned}$$

Continuing this pattern, we get:

$$T(n) = n\left(\log n + \log\left(\frac{n}{2}\right) + \ldots + \log\left(\frac{n}{2^{\log_2 n-1}}\right)\right) + nT(1)$$

Supposing $T(1) = 0$:

$$T(n) = n\left(\frac{\log n(\log n + 1)}{2}\right)$$

So,

$$T(n) = \Theta(n \log^2 n)$$

7 Divide and conquer method

"Divide and conquer" is a problem-solving strategy that involves breaking down a complex problem into simpler, more manageable sub-problems. The idea is to tackle each sub-problem individually, solve them, and then combine their solutions to address the overall problem.

8 Analysis of merge sort

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller sub-arrays, sorting each sub-array, and then merging the sorted sub-arrays back together to form the final sorted array.

ALGORITHM:-

```
procedure Merge(A, m, B, n, C)
    i = 0, j = 0, k = 0
    while i ≤ m and j ≤ n do
        if A[i] ≤ B[j] then
            C[k] = A[i]
            i++
        else
            C[k] = B[j]
            j++
        end if
        k++
    end while
    while i ≤ m do
        C[k] = A[i]
        i++, k++
    end while
    while j ≤ n do
        C[k] = B[j]
        j++, k++
    end while
end procedure
```