**The substitution method**

The substitution method is one of the techniques used to solve recurrence relations. In this method, we guess a solution to the recurrence relation and then use mathematical induction to prove that it is correct. Let's consider the following recurrence relation:

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

where T(1) = 1.

To solve this recurrence relation using the substitution method, we first guess a solution of the form T(n) = O(nlogn). Now we need to prove that this guess is correct.

We will use mathematical induction to prove that T(n) ≤ cnlogn for some constant c > 0.

**Base case:** T(1) = 1 ≤ c. This is true for any value of c ≥ 1.

Inductive hypothesis: Assume that T(k) ≤ cklogk for all k < n.

**Inductive step:** We want to show that T(n) ≤ cnlogn. Using the recurrence relation, we have:

T(n) = 2T(n/2) + n ≤ 2c(n/2)log(n/2) + n = cnlogn - cn + n

We need to show that cnlogn - cn + n ≤ cnlogn for some constant c > 0.

Taking c = 1/2, we have:

cnlogn - cn + n = (1/2)nlogn - (1/2)n + n = (1/2)nlogn + (1/2)n ≤ (1/2)nlogn for n ≥ 2.

Therefore, T(n) = O(nlogn) is a valid solution to the recurrence relation.

**Recursive Tree Method**

The recursive tree method is another approach for solving recurrence relations. It involves representing the entire recursion tree as a series of levels, where each level represents one recursive call. This method is particularly useful when the recurrence relation has multiple levels of recursion.

To illustrate this method, let's consider the following recurrence relation:

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

We can start by drawing the recursion tree for the first few levels:

T(n)

/ \

T(n/2) T(n/2)

/ \ / \

T(n/4) T(n/4) T(n/4) T(n/4)

... ... ...

Top of Form

Bottom of Form

Each level in the tree represents one recursive call, and the number of nodes at each level is equal to the number of subproblems generated by that call. In this case, each level has twice as many nodes as the previous level, since each subproblem is half the size of the previous one.

We can compute the total cost of the algorithm by adding up the costs of all the nodes in the tree. The cost of each node is simply the size of the corresponding subproblem, which is given by the recurrence relation. For example, the cost of the root node is T(n) = 2T(n/2) + n.

To simplify the analysis, we can make a few observations about the tree. First, the number of nodes at each level is equal to 2^k, where k is the level number. Second, the size of each subproblem at level k is n/2^k. Finally, the total cost of all the nodes at level k is equal to the product of the number of nodes and the size of each subproblem:

cost(k) = 2^k \* (n / 2^k) = n

Therefore, the total cost of the algorithm is given by the sum of the costs of all the levels:

T(n) = n + 2n + 4n + ... + n\*2^log(n) = n \* (1 + 2 + 4 + ... + 2^log(n)) = n \* (2^log(n+1) - 1) = nlog(n)

Therefore, the solution to the recurrence relation is T(n) = O(nlog(n)).

The recursive tree method involves representing the entire recursion tree as a series of levels, and computing the total cost of the algorithm by adding up the costs of all the nodes in the tree. This method is particularly useful when the recurrence relation has multiple levels of recursion, and can be used to derive a closed-form solution to the recurrence relation using mathematical induction.

**Master Theorem Method**

The master theorem is a tool used to analyze the time complexity of recursive algorithms. It provides a formula to compute the running time of a recursive algorithm in terms of its recursive calls. The master theorem is usually stated in terms of a recurrence relation of the form:

T(n) = aT(n/b) + f(n)

where:

* T(n) is the running time of the algorithm on an input of size n
* a is the number of recursive calls made by the algorithm
* n/b is the size of each subproblem (assuming that all subproblems have the same size)
* f(n) is the time taken by the algorithm to solve the subproblems and combine the results.

The master theorem has three cases, depending on the relative magnitudes of f(n) and n^(log\_b a):

**Case 1:** If f(n) = O(n^(log\_b a - ε)) for some ε > 0, then T(n) = Θ(n^(log\_b a)).

This case occurs when the work done outside the recursive calls is relatively small compared to the work done inside the recursive calls. In this case, the running time of the algorithm is dominated by the work done inside the recursive calls, which is proportional to the number of recursive calls aT(n/b). The running time is therefore Θ(n^(log\_b a)).

**Example:** Merge sort algorithm, where a = 2, b = 2, and f(n) = Θ(n). Here, f(n) = Θ(n) = O(n^(log\_2 2 - ε)) for any ε > 0, so we can apply case 1. Thus, T(n) = Θ(n log n).

**Case 2:** If f(n) = Θ(n^(log\_b a)), then T(n) = Θ(n^(log\_b a) log n).

This case occurs when the work done outside the recursive calls is of the same order of magnitude as the work done inside the recursive calls. In this case, the running time of the algorithm is dominated by both the work done inside the recursive calls and the work done outside the recursive calls. The running time is therefore Θ(n^(log\_b a) log n).

**Example:** Quicksort algorithm, where a = 2, b = 2, and f(n) = Θ(n). Here, f(n) = Θ(n) = Θ(n^(log\_2 2)), so we can apply case 2. Thus, T(n) = Θ(n log n).

**Case 3:** If f(n) = Ω(n^(log\_b a + ε)) for some ε > 0, and if af(n/b) ≤ cf(n) for some constant c < 1 and all sufficiently large n, then T(n) = Θ(f(n)).

This case occurs when the work done outside the recursive calls is relatively large compared to the work done inside the recursive calls. In this case, the running time of the algorithm is dominated by the work done outside the recursive calls. The running time is therefore Θ(f(n)).

**Example:** Binary search tree construction algorithm, where a = 2, b = 2, and f(n) = Θ(n). Here, f(n) = Θ(n) = Ω(n^(log\_2 2 + ε)) for any ε > 0, so we can apply case 3. Also, af(n/b) = 2f(n/2) ≤ f(n) for all n ≥ 2, so we can also apply the condition af(n/b) ≤ cf(n) for some constant c < 1. Thus, T(n) =

Top of Form

Bottom of Form

Top of Form

Bottom of Form