**Recursive Relations**

**Assignment Questions and Solutions**

**Substitution Method**

1. Solve the recurrence relation T(n) = 2T(n/2) + n log n with T(1) = 1.

**Solution:**

We first guess a solution of the form T(n) = O(n log^2 n), and we will prove it using induction. We need to show that T(n) <= cn log^2 n for some constant c > 0.

**Base case:** T(1) = 1 <= c, so the base case holds.

Inductive hypothesis: Assume that T(k) <= ck log^2 k for all k < n.

**Inductive step:** We have T(n) = 2T(n/2) + n log n <= 2c(n/2) log^2 (n/2) + n log n = cn log^2 n - cn log n + n log n = cn log^2 n - (c - 1)n log n <= cn log^2 n for c >= 1.

Thus, by induction, T(n) = O(n log^2 n).

1. Solve the recurrence relation T(n) = T(n/2) + n^2 with T(1) = 1.

**Solution:**

We first guess a solution of the form T(n) = O(n^2), and we will prove it using induction. We need to show that T(n) <= cn^2 for some constant c > 0.

Base case: T(1) = 1 <= c, so the base case holds.

Inductive hypothesis: Assume that T(k) <= ck^2 for all k < n.

Inductive step: We have T(n) = T(n/2) + n^2 <= c(n/2)^2 + n^2 = cn^2/4 + n^2 = cn^2 - (c - 4n^2)/4 <= cn^2 for c >= 4.

Thus, by induction, T(n) = O(n^2).

1. Solve the recurrence relation T(n) = 2T(n/4) + n^2 with T(1) = 1.

**Solution:**

We first guess a solution of the form T(n) = O(n^2), and we will prove it using induction. We need to show that T(n) <= cn^2 for some constant c > 0.

Base case: T(1) = 1 <= c, so the base case holds.

Inductive hypothesis: Assume that T(k) <= ck^2 for all k < n.

Inductive step: We have T(n) = 2T(n/4) + n^2 <= 2c(n/4)^2 + n^2 = cn^2/8 + n^2 = cn^2 - (7c/8)n^2 <= cn^2 for c >= 8/7.

Thus, by induction, T(n) = O(n^2).

**Recursive Tree Method**

1. Solve the recurrence relation T(n) = 2T(n/2) + nlogn using the recursion tree method

**Solution:**

nlogn

/ \

n/2log(n/2) n/2log(n/2)

/ \ / \

n/4log(n/4) n/4log(n/4) n/4log(n/4) n/4log(n/4)

... ... ... ...

The tree has logn levels, and at each level i (0 <= i <= logn), there are 2^i nodes, each with size n/(2^i). The total cost at each level i is 2^i \* (n/(2^i)) \* log(n/(2^i)) = nlog(n/(2^i)). Therefore, the total cost of all levels is:

T(n) = nlogn + nlog(n/2) + nlog(n/4) + ... + nlog(1)

= nlogn + nlogn/2 + nlogn/4 + ... + nlog1

= nlogn + nlogn

= O(nlogn)

1. Solve the recurrence relation T(n) = T(n/3) + T(2n/3) + n using the recursion tree method.

**Solution:**

The recursion tree for the given recurrence relation looks like:

n

/ \

n/3 2n/3

/ \ / \

n/9 2n/9 4n/9 8n/9

... ... ... ...

The height of the tree is log(3/2)n. At each level i, there are 2^i nodes, and the total cost at each level is n. Therefore, the total cost of all levels is:

T(n) = n log(3/2)n

= n logn - n log(3/2)

= O(n logn)

1. Solve the recurrence relation T(n) = 4T(n/2) + n^2/logn using the recursion tree method.

**Solution:**

The recursion tree for the given recurrence relation looks like:

n^2/logn

/ \

(n/2)^2/log(n/2) (n/2)^2/log(n/2)

/ \ / \

(n/4)^2/log(n/4) (n/4)^2/log(n/4) (n/4)^2/log(n/4) (n/4)^2/log(n/4)

... ... ... ...

The tree has logn levels, and at each level i (0 <= i <= logn), there are 4^i nodes, each with size n/(2^i)^2. The total cost at each level i is 4^i \* (n/(2^i)^2) \* (log(n/(2^i))/log2)^2 = n^2 \* (log(n/(2^i))/log2)^2. Therefore, the total cost of all levels is:

T(n) = n^2/logn + 4n^2/(log(n/2))^2 + 16n^2/(log(n/4))^2

**Master Theorem**

1. What is the time complexity of the following recursive algorithm using the Master Theorem?

T(n) = 3T(n/3) + n/log n

**Solution:**

In this case, we have **a = 3**, **b = 3**, and **f(n) = n/log n**. Therefore, we have **log\_b(a) = log\_3(3) = 1**. Since **f(n) = Θ(n^(log\_b(a)))**, we can use case 2 of the Master Theorem, which gives us a time complexity of **Θ(n log n)**.

1. What is the time complexity of the following recursive algorithm using the Master Theorem?

T(n) = 2T(n/4) + n^2

**Solution:**

In this case, we have **a = 2**, **b = 4**, and **f(n) = n^2**. Therefore, we have **log\_b(a) = log\_4(2) = 1/2**. Since **f(n) = Θ(n^(log\_b(a)))**, we can use case 3 of the Master Theorem, which gives us a time complexity of **Θ(n^(log\_4(2))) ≈ Θ(n^0.5)**.

1. What is the time complexity of the following recursive algorithm using the Master Theorem?

T(n) = 5T(n/2) + n^3 log n

**Solution:**

In this case, we have **a = 5**, **b = 2**, and **f(n) = n^3 log n**. Therefore, we have **log\_b(a) = log\_2(5) ≈ 2.32**. Since **f(n) = Θ(n^(log\_b(a)))**, we can use case 3 of the Master Theorem, which gives us a time complexity of **Θ(n^(log\_2(5))) ≈ Θ(n^2.32)**.