12. (16 Points) Consider The Recurrence Relation T(n) = 9T(n/3) + F(n), T(1) = (1). What Is The Order

Understanding the Recurrence Relation: T(n) = 9T(n/3) + F(n), T(1) = 1

12. (16 Points) Consider The Recurrence Relation T(n) = 9T(n/3) + F(n), T(1) = 1. What Is The Order is a fundamental question in the analysis of algorithms, particularly in the context of divide-and-conquer strategies. This recurrence relation models the time complexity of certain recursive algorithms, where the problem size is reduced by a factor (here, dividing by 3), and additional work (represented by F(n)) is performed at each step. Determining the order of T(n) helps in understanding how the algorithm scales with input size, which is crucial for efficiency analysis and optimization.

Breaking Down the Recurrence Relation

Components of the Recurrence

- **Recursive Part:** 9T(n/3)
- **Non-recursive Work:** F(n)
- **Base Case:** T(1) = 1

This recurrence suggests that the problem of size n is broken into 9 subproblems of size n/3, with some additional work F(n) done outside the recursive calls. To analyze the order, we need to understand the nature of F(n) and apply suitable methods like the Master Theorem or recursion tree analysis.

Analyzing the Recursive Structure

Applying the Master Theorem

The Master Theorem provides a straightforward way to determine the asymptotic behavior of recurrences of the form:

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

where:

- **a** is the number of subproblems in the recursion (here, 9)
- **b** is the factor by which the problem size is reduced (here, 3)
- **F(n)** is the non-recursive work at each step

In our recurrence:

a = 9, b = 3, T(n) = 9T(n/3) + F(n)

Calculating log_b a

Compute:

log_b a = log_3 9 = log_3 (3^2) = 2

Understanding F(n)

The nature of F(n) determines the dominant term. Common cases include:

- F(n) = 0(n^c), for some constant c
- F(n) = 0(log^k n)
- F(n) = 0(1)

For the most comprehensive analysis, we will consider various forms of F(n).

The order of T(n) depends heavily on the growth rate of F(n).

Case 1: F(n) = 0(n^c) – Polynomial Work

Comparing F(n) with n^{log_b a} = n^{2}

- If F(n) = 0(n^{c}), then:

- Case 1a: c < 2

- The recursive term dominates; T(n) = 0(n^{2})

- Case 1b: c = 2

- The work is balanced; T(n) = 0(n^{2} log n)

- Case 1c: c > 2

- The non-recursive work dominates; T(n) = 0(F(n)) = 0(n^{c})

Summary for Polynomial F(n):

c-value	Dominant Term	T(n) Order
c < 2	Recursive part	0(n^{2})
c = 2	Balanced	0(n^{2} log n)
c > 2	F(n) dominates	0(n^{c})

Case 2: F(n) = 0(log^k n) – Logarithmic Work

- Since polynomial growth (n^{2}) dominates logarithmic growth, the recursive term dominates.

- Therefore, T(n) = 0(n^{2})

Case 3: F(n) = 0(1) – Constant Work

- The recursive term dominates for large n, so T(n) = 0(n^{2})

Summary of the Asymptotic Behavior

- When F(n) grows slower than n^{2}, the recurrence behaves like 0(n^{2}).
- When F(n) matches n^{2} (e.g., F(n) = 0(n^{2})), the solution involves an extra logarithmic factor.
- When F(n) grows faster than n^{2}, the non-recursive work dominates, and T(n) is 0(F(n)).

Using the Recursion Tree Method for Intuitive Understanding

Constructing the Recursion Tree

- At the top level, work is F(n).
- At each subsequent level, the total work is 9^{k} F(n/3^{k}), where k is the depth level.
- Total work sums over all levels until the base case is reached.

Analyzing the Depth of the Tree

- The depth d is approximately log_b n = log_3 n.
- Total work T(n) ≈ Σ_{k=0}^{d} 9^{k} F(n/3^{k})

Estimating Total Work

- For F(n) = 0(n^{c}), the total work sum behaves like a geometric series, leading to the same conclusions as the Master Theorem.

Final Conclusion: What Is The Order of T(n)?

- The order of the recurrence T(n) = 9T(n/3) + F(n) depends primarily on the form of F(n).
- If F(n) = 0(n^{c}):

- For c < 2, T(n) = 0(n^{2})
- For c = 2, T(n) = 0(n^{2} log n)
- For c > 2, T(n) = 0(F(n)) = 0(n^{c})

- If F(n) grows slower than polynomial, such as logarithmic or constant, T(n)

= 0(n^{2}).

- If F(n) is polynomial with c > 2, the recurrence is dominated by F(n).

In summary, the order of T(n) is primarily driven by the larger of the recursive component's cost (which is 0(n^{log_b a}) = 0(n^{2})) and the work done outside the recursion F(n). Therefore, the overall order is:

T(n) = 0(n^{2}) + F(n)

and the dominant term depends on the specific growth rate of F(n).

Implications for Algorithm Design and Analysis

Understanding the Significance of Recursion Order

- Knowing the order of T(n) allows developers and researchers to predict how algorithms scale.
- This understanding helps in identifying bottlenecks and optimizing code.

Practical Use Cases

- Divide-and-conquer algorithms such as mergesort, quicksort, and matrix multiplication often follow recurrence relations similar to the one discussed.
- Analyzing their order ensures efficient implementations.

Summary

- The recurrence relation T(n) = 9T(n/3) + F(n) models recursive algorithms with multiple subproblems.
- Applying the Master Theorem provides a quick way to determine asymptotic behavior.
- The key is to compare F(n) to n^{log_b a} (here, n^{2}).
- The overall order depends on this comparison, with T(n) typically being 0(n^{2}) when F(n) is polynomial with degree less than 2.
- For more complex F(n), the dominant term guides the asymptotic analysis.

Final Thoughts

Understanding recurrence relations like T(n) = 9T(n/3) + F(n) is fundamental for computer scientists and software engineers aiming to optimize algorithms. By systematically analyzing each component and utilizing tools like the Master Theorem, practitioners can accurately predict algorithm performance, leading to more efficient and scalable software solutions.

Frequently Asked Questions

What is the recurrence relation given in the problem?

The recurrence relation is T(n) = 9T(n/3) + F(n), with T(1) = 1.

What is the main goal when analyzing this recurrence relation?

The main goal is to determine the asymptotic order or the Big-O classification of T(n).

Which method can be used to solve this recurrence relation?

The Master Theorem is a common method used to analyze recurrences of this form.

In the recurrence T(n) = 9T(n/3) + F(n), what are the values of 'a' and 'b'?

a = 9 and b = 3.

How do we apply the Master Theorem to this recurrence relation?

We compare F(n) to n^{log_b a} which is n^{log_3 9} = n^{2} to determine the asymptotic order.

If F(n) = O(n^{2}), what is the order of T(n)?

T(n) is 0(n^{2}) since F(n) matches the critical exponent n^{log_b a}.

What is the overall order of T(n) considering the recurrence relation and F(n)?

The order of T(n) depends on F(n), but if F(n) = 0(n^{2}), then T(n) =

$0(n^{2})$.

Additional Resources

Understanding the Order of the Recurrence Relation $T(n) = 9T(n/3) + F(n)$

Analyzing the order of a recurrence relation is fundamental in algorithm analysis, especially when assessing the efficiency of divide-and-conquer algorithms. The recurrence relation provided:

$$ T(n) = 9T(n/3) + F(n), \quad T(1) = 1 $$

is a classic example often encountered in the study of recursive algorithms, particularly those that divide the problem size by a factor (here, 3) at each step. In this comprehensive review, we will explore the process of determining the order of this recurrence, delve into the underlying principles, and examine how the function $F(n)$ influences the overall complexity.

Overview of Recursion and Its Significance

Before analyzing the specific recurrence, it's crucial to understand why recurrence relations matter:

- Modeling Recursive Algorithms: Recurrences provide a mathematical way to describe the runtime or space complexity of algorithms that break down problems into smaller subproblems.
- Predicting Algorithm Behavior: By solving recurrences, we can predict how the algorithm scales with input size n.
- Comparing Algorithm Efficiency: Different recurrence solutions help compare the efficiency of algorithms, guiding optimal algorithm design.

Fundamentals of Recurrence Relations

A recurrence relation expresses the value of a function in terms of its values at smaller inputs. The general form for divide-and-conquer algorithms often resembles:

$$ T(n) = a\,T(n/b) + F(n) $$

where:

- a = number of subproblems at each step
- b = factor by which the problem size is reduced
- $F(n)$ = cost of dividing the problem and combining solutions

In our specific problem:

- $a = 9$
- $b = 3$
- $F(n)$ = additional cost function (to be specified or analyzed)

Applying the Master Theorem

The Master Theorem provides a quick way to determine the asymptotic behavior of recurrences of the form:

$$T(n) = a\,T(n/b) + F(n)$$

The theorem compares $F(n)$ with $n^{\log_b a}$:

1. Compute the critical exponent:

$$\boxed{\log_b a}$$

2. Compare $F(n)$ with $n^{\log_b a}$:

- If $F(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then:

$$T(n) = \Theta(n^{\log_b a})$$

- If $F(n) = \Theta(n^{\log_b a} \log^k n)$ for some $k \ge 0$, then:

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

- If $F(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and regularity conditions hold, then:

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

Calculating $\log_b a$ for Our Recurrence

Let's compute:

$$ \log_b a = \log_3 9 $$

Since:

$$ 9 = 3^2 $$

then:

$$ \log_3 9 = 2 $$

Thus, the critical exponent:

$$ n^{\log_b a} = n^2 $$

This means the behavior of $T(n)$ depends heavily on the form of $F(n)$ relative to n^2.

Analyzing Different Forms of $F(n)$

The nature of $F(n)$ determines the asymptotic order of $T(n)$. Let's consider several common scenarios:

Case 1: $F(n) = O(n^{2 - \varepsilon})$ where $\varepsilon > 0$

- Here, $F(n)$ grows slower than n^2.
- According to the Master Theorem, since $F(n)$ is polynomially smaller than $n^{\log_b a}$, the solution is dominated by the recurrence tree's work at the top levels:

$$ T(n) = \Theta(n^{2}) $$

This case indicates the recursive calls contribute the most to complexity, with $F(n)$ being asymptotically insignificant.

Case 2: $F(n) = \Theta(n^{2} \log^k n)$, for some $k \ge 0$

- $F(n)$ matches the critical exponent up to a polylogarithmic factor.
- The Master Theorem suggests:

$$ T(n) = \Theta(n^{2} \log^{k+1} n) $$

This is a classic case where the additional work at each level adds a logarithmic factor.

Case 3: $F(n) = \Omega(n^{2 + \varepsilon})$, with regularity conditions satisfied

- $F(n)$ grows faster than n^2.
- The solution then is dominated by $F(n)$:

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

- For example, if $F(n) = n^{3}$, then:

$$T(n) = \Theta(n^{3})$$

Special Cases and the Use of the Recursion Tree Method

While the Master Theorem offers quick solutions, analyzing the recurrence through the recursion tree method provides deeper insight, especially when $F(n)$ doesn't fit neatly into the theorem's cases.

Constructing the Recursion Tree

1. Root Level: The total work at the top level is $F(n)$.
2. Subproblem Levels: Each node spawns a subproblems, each of size n/b.
3. Work at each level i:

$$a^i \times F(n / b^i)$$

4. Total work: Summed over all levels until the base case:

$$T(n) = \sum_{i=0}^{\log_b n} a^i \times F(n / b^i)$$

Applying this to our specific recurrence:

- $a=9$, $b=3$,
- The number of levels:

$$\log_3 n$$

- Work at level i:

$$9^i \times F(n/3^i)$$

Case Study: When $F(n) = O(1)$

Suppose $F(n)$ is constant, $F(n) = c$. Then:

$$T(n) = 9T(n/3) + c$$

Applying the Master Theorem:

- $\log_3 9 = 2$,
- $F(n) = O(1) = O(n^{0})$,
- Since $0 < 2$, the recurrence falls into the first case:

$$T(n) = \Theta(n^{2})$$

Interpretation: The recursive division results in a quadratic total runtime, dominated by the depth of recursion and the number of subproblems.

Case Study: When $F(n) = n^2$

Here, $F(n)$ matches the critical exponent $n^{\log_b a} = n^{2}$. The Master Theorem suggests:

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

This scenario indicates that the cost at each level scales proportionally to the size of the problem, and the cumulative work over all levels results in a polylogarithmic factor.

Implications for Algorithm Design

Understanding the order of the recurrence relation has direct consequences:

- Efficiency: If $F(n)$ grows faster than n^{2}, the overall complexity is dominated by $F(n)$.

- Optimization: Knowing the dominant term guides optimization strategies—either reducing $F(n)$ or restructuring the algorithm to minimize the recursive depth.
- Parallelization: The recurrence structure indicates how well the problem can be parallelized; for example, high values of a and $F(n)$ suggest more parallel work.

Limitations and Considerations

While the Master Theorem is powerful, it has constraints:

- It applies primarily to recurrences of the form $T(n) = a T(n/b) + F(n)$ with regular $F(n)$.
- For irregular or more complex $F(n)$, other techniques like the recursion tree method, substitution method, or Akra-Bazzi theorem may be necessary.
- The initial condition $T(1) = 1$ helps in base case analysis but doesn't affect asymptotic behavior.

Summary and Key Takeaways

- The recurrence relation:

$$T(n) = 9T(n/3) + F(n), \quad T(1) =$$

12 16 Points Consider The Recurrence Relation T N 9t N 3 F N T 1 1 What Is The Order

12 16 Points Consider The Recurrence Relation T N 9t N 3 F N T 1 1 What Is The Order

Related Articles

- 1. Suppose Daily Production Of A CONWIP Line Is Nearly Normally Distributed With A Mean Of 250 Pieces
- 1) A Politician Is About To Give A Campaign Speech And Is Holding A 'stack Of Ten Cue Cards, Of Which
- A Basketball Player Makes A Jump Shot. The 0.6 Kg Ball Is Released At A Height Of 2 M Above The Floor

Back to Home