Give The Asymptotic Upper And Lower Bounds For T(n) In Each Of The Following Recurrences. Assume That

Give The Asymptotic Upper And Lower Bounds For T(n) In Each Of The Following Recurrences. Assume That

Understanding the asymptotic behavior of recurrences is fundamental in analyzing the efficiency of algorithms. When dealing with recursive relations, especially those arising from divide-and-conquer algorithms, establishing tight bounds—both upper and lower—is crucial for predicting performance and optimizing implementations. This article explores how to derive asymptotic bounds for various classes of recurrence relations, providing a comprehensive framework for analysis.

Overview of Recurrence Relations and Their Significance

Recurrence relations express the running time or resource consumption of an algorithm in terms of smaller inputs. They are pivotal in analyzing algorithms like mergesort, quicksort, binary search, and many dynamic programming solutions.

For a recurrence relation T(n), the goal is to find functions that bound T(n) asymptotically, typically expressed using Big O (upper bound), Big Omega (lower bound), and Big Theta (tight bounds). Understanding these bounds allows us to classify algorithms according to their efficiency.

Common Types of Recurrences and Their Characteristics

Before delving into specific bounds, it's essential to recognize typical recurrence forms:

- **Linear Recurrences:** $T(n) = aT(n/b) + f(n)$
- **Divide-and-Conquer Recurrences:** Similar to linear but often with more complex f(n)
- **Non-Recursive Recurrences:** When the relation involves multiple recursive calls or more complex functions

The most studied are the divide-and-conquer recurrences, often expressed as:

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

where:
- $a \geq 1$ is the number of recursive calls,
- $b > 1$ is the factor by which input size decreases,
- $f(n)$ is the non-recursive work at each step.

The Master Theorem: A Tool for Asymptotic Bounds

The Master Theorem provides a straightforward way to derive asymptotic bounds for recurrences of the form $T(n) = a\,T(n/b) + f(n)$. It compares $f(n)$ with $n^{\log_b a}$, the critical exponent derived from the recursive subdivision.

Conditions and Cases:

Given the recurrence $T(n) = a\,T(n/b) + f(n)$:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then:
- $T(n) = \Theta(n^{\log_b a})$ (Case 1)

2. If $f(n) = \Theta(n^{\log_b a} (\log n)^k)$ for some $k \geq 0$, then:
- $T(n) = \Theta(n^{\log_b a} (\log n)^{k+1})$ (Case 2)

3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if $af(n/b) \leq c\,f(n)$ for some $c < 1$ and sufficiently large n, then:
- $T(n) = \Theta(f(n))$ (Case 3)

This theorem provides tight bounds but requires the regularity condition in Case 3 and is most applicable to recurrences fitting the form.

Asymptotic Upper and Lower Bounds for Specific Recurrences

While the Master Theorem covers many cases, some recurrences fall outside its scope, necessitating alternative methods like the recursion tree method, substitution, or the Akra-Bazzi theorem.

Below, we analyze common recurrence patterns, providing asymptotic bounds and their derivations.

1. Recursion of the Form $T(n) = a\,T(n/b) + c\,n^k$

Scenario:

- $a \geq 1$, $b > 1$, $c > 0$, and $k \geq 0$.
- The non-recursive work is polynomial in n.

Analysis:

- Compute $n^{\log_b a}$ and compare with n^k.
- Use the Master Theorem:

- Case 1: If $n^k = o(n^{\log_b a})$, i.e., $k < \log_b a$
- $T(n) = \Theta(n^{\log_b a})$

- Case 2: If $n^k = \Theta(n^{\log_b a})$
- $T(n) = \Theta(n^{\log_b a} \log n)$

- Case 3: If $n^k = \omega(n^{\log_b a})$, i.e., $k > \log_b a$, and regularity condition holds
- $T(n) = \Theta(n^k)$

Upper and Lower Bounds:

- The bounds are tight, i.e., $T(n) = \Theta(...)$ in each case.
- For lower bounds, the same analysis applies, as the Master Theorem provides tight bounds.

2. Recursion with $f(n) = \Theta(1)$: The Base Case

Recurrence:

$T(n) = a\,T(n/b) + \Theta(1)$

Analysis:

- Compute $n^{\log_b a}$.

- If $a = 1$: The recurrence reduces to $T(n) = T(n/b) + \Theta(1)$

- Solving yields $T(n) = \Theta(\log n)$

- If $a > 1$:

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

Asymptotic bounds:

- Upper bound: $T(n) = O(n^{\log_b a})$
- Lower bound: $T(n) = \Omega(n^{\log_b a})$

- Tight bound: $T(n) = \Theta(n^{\log_b a})$

3. Recurrence with f(n) = Θ(n^k log^p n): The Polynomial-Logarithmic Case

Recurrence:

$T(n) = a\,T(n/b) + \Theta(n^k \log^p n)$

Analysis:

- Compute $n^{\log_b a}$ and compare with $n^k \log^p n$.

- Case 1: If $n^{\log_b a}$ dominates, i.e., $k < \log_b a$

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

- Case 2: If $n^k \log^p n \approx n^{\log_b a}$

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

- Case 3: If $n^k \log^p n$ dominates, i.e., $k > \log_b a$

- $T(n) = \Theta(n^k \log^p n)$

Bounds:

- The bounds are tight, with both upper and lower bounds matching.

Beyond the Master Theorem: Other Techniques for Bounding Recurrences

While the Master Theorem is powerful, it has limitations, especially for recurrences involving multiple recursive calls with different sizes, non-standard forms, or non-polynomial f(n).

Here are alternative methods:

1. Recursion Tree Method

- Visualize the recurrence as a tree, where each node's cost is f(n) and subtrees represent recursive calls.
- Sum the costs at each level to estimate total work.
- Provides intuitive bounds and is especially useful for non-uniform recursions.

2. Substitution Method

- Guess the asymptotic bound and verify via induction.
- Useful for establishing tight bounds when the recurrence is complex.

3. Akra-Bazzi Theorem

- Generalizes the Master Theorem for recurrences of the form:

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n) + f(n)$$

- Handles multiple recursive calls with different sizes.

Examples and Applications

Example 1: Mergesort

Recurrence:

$$T(n) = 2 T(n/2) + \Theta(n)$$

- $a = 2$, $b = 2$, $f(n) = \Theta(n)$
- Compute $n^{\log_b a} = n^{\log_2 2} = n^1 = n$
- Since $f(n) = \Theta(n) = \Theta(n^{\log_b a})$, apply Case 2
- Bounds: $T(n) = \Theta(n \log n)$

Example 2: Binary Search

Recurrence:

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

- $a = 1$, $b = 2$, $f(n) = \Theta(1)$
- $n^{\log_b a} = n^0 = 1$
- $f(n) = \Theta(1) = \Theta(n^{0})$ matches, so bounds are tight
- Bounds: $T(n) = \Theta(\log n)$

Example 3: Exponential Recursion

Recurrence:

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

- Same as mergesort, bounds are $\Theta(n \log n)$

Example 4: Non-Standard Recurrence

T(n) = 3 T(n/4) + Θ(n)

- a = 3, b = 4, f(n) = Θ(n)
- n^{log_b a} = n^{log_4 3} ≈ n^{0.792}
- Since f(n) = Θ(n) dominates n^{0.792}, and regularity conditions hold, bounds are:

- Lower bound: Ω(n)
- Upper bound: O(n)

- T(n) = Θ(n

Frequently Asked Questions

What are the asymptotic upper and lower bounds for T(n) = 2T(n/2) + n?

Using the Master Theorem, the recurrence T(n) = 2T(n/2) + n has bounds of Θ(n log n). The upper and lower bounds are both asymptotically tight at Θ(n log n).

How do you determine the asymptotic bounds for T(n) = 3T(n/4) + √n?

Applying the Master Theorem, since a=3, b=4, and f(n)=√n= n^{0.5}, compare f(n) with n^{log_b a} = n^{log_4 3} ≈ n^{0.792}. Because f(n) = o(n^{0.792}), T(n) = Θ(n^{0.792}). Thus, bounds are asymptotically tight at Θ(n^{0.792}).

What are the bounds for T(n) = T(n-1) + 1?

This recurrence describes a linear recurrence with solution T(n) = Θ(n). Both upper and lower bounds are linear, so T(n) = Θ(n).

Find the asymptotic bounds for T(n) = 4T(n/2) + n^2.

Applying the Master Theorem, a=4, b=2, and f(n)=n^2. Since n^{log_b a} = n^{2}, f(n) = Θ(n^{2}), matching the critical case. Therefore, T(n) = Θ(n^{2} log n).

Given T(n) = 5T(n/3) + n, what are the asymptotic bounds?

Using the Master Theorem, a=5, b=3, and f(n)=n. Here, n^{log_3 5} ≈ n^{1.464}. Since f(n)=O(n^{1.464}), and f(n) = o(n^{log_b a}), T(n) = Θ(n^{log_3 5}).

What bounds can be established for T(n) = T(n/2) + log n?

This recurrence solves to T(n)= Θ(log^2 n). The lower and upper bounds are both tight at Θ(log^2 n).

How do you find asymptotic bounds for T(n) = 2T(n/2) + n / log n?

Applying the Master Theorem, a=2, b=2, and f(n)=n / log n. Since n^{log_b a} = n, and f(n) = o(n), T(n) = Θ(n). The bounds are tight at Θ(n).

What are the asymptotic bounds for T(n) = T(n/2) + 1?

This recurrence results in T(n) = Θ(log n). Both upper and lower bounds are logarithmic, so T(n) = Θ(log n).

Additional Resources

Recurrence Analysis: Asymptotic Upper and Lower Bounds for T(n)

Understanding the asymptotic behavior of recurrence relations is fundamental in algorithm analysis and computer science at large. When analyzing recursive algorithms, recurrence relations serve as a mathematical backbone that models the running time or space complexity relative to input size, n. The key goal is to determine tight bounds—both upper and lower—that describe how $T(n)$ scales as n grows large. These bounds help developers and researchers predict performance, optimize algorithms, and compare different approaches effectively.

In this comprehensive review, we will explore how to derive asymptotic upper and lower bounds for various common recurrence relations. We will consider the classical methods—such as the Master Theorem, recursion trees, and substitution—and apply them carefully to different types of recurrences, illustrating the process with detailed explanations and examples.

Understanding the Basics of Recurrence Relations

Before diving into specific bounds, it is essential to clarify what recurrence relations are and what we mean by asymptotic bounds.

What Is a Recurrence Relation?
A recurrence relation expresses the running time $T(n)$ of an algorithm in terms of smaller instances of the same problem, usually of size n/b, plus some additional work $f(n)$. Formally, a recurrence might look like:

$$
T(n) = a \, T\left(\frac{n}{b}\right) + f(n)
$$

where:
- $a \ge 1$ is the number of recursive calls,
- $b > 1$ is the factor by which the problem size is reduced in each recursive call,
- $f(n)$ is the non-recursive work at each step.

Why Asymptotic Bounds?
Exact solutions are often complex or impossible to derive explicitly; hence, asymptotic bounds (Big O, Big Omega, and Theta) provide a way to describe the growth rate of $T(n)$ as $n \to \infty$.

- Upper Bound (Big O): The worst-case growth rate.
- Lower Bound (Big Omega): The best-case or minimal growth rate.
- Tight Bound (Big Theta): When the upper and lower bounds match asymptotically, providing a precise growth rate.

Methodologies for Deriving Asymptotic Bounds

Several techniques are used to analyze recurrence relations:

1. The Master Theorem
A powerful tool for solving recurrences of the form:

$$
T(n) = a \, T\left(\frac{n}{b}\right) + f(n)
$$

It provides direct asymptotic bounds based on the comparison of $f(n)$ with $n^{\log_b a}$.

2. Recursion Tree Method
Visualizes the recurrence as a tree, summing the costs at each level to find the total.

3. Substitution Method
Assumes a form for $T(n)$ and proves bounds via induction.

4. Iteration Method
Expands the recurrence multiple times to observe the pattern and summation.

Each method has strengths and is suited to different types of recurrences. Here, our focus will be primarily on the Master Theorem due to its broad applicability.

Analyzing Common Recurrence Forms and Their Bounds

Let's consider several typical forms of recurrence relations encountered in algorithm analysis, discuss their asymptotic bounds, and how to derive them.

---

1. Recurrence of the Form $T(n) = a\, T(n/b) + f(n)$

This is the classic divide-and-conquer recurrence. The asymptotic bounds depend on the comparison between $f(n)$ and $n^{\log_b a}$.

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$

- Lower Bound: $T(n) = \Omega(n^{\log_b a})$
- Upper Bound: $T(n) = O(n^{\log_b a})$
- Result: $T(n) = \Theta(n^{\log_b a})$

Interpretation: The recursive part dominates; the overall growth is determined by the "divide" step.

Case 2: $f(n) = \Theta(n^{\log_b a} \log^k n)$

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

- Implication: The non-recursive work matches the recursive splitting, leading to a slightly higher complexity due to the logarithmic factor.

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ and regularity conditions hold

- Bounds:
$$
T(n) = \Theta(f(n))
$$

- Note: The non-recursive work dominates; the recurrence resolves to the form of $f(n)$.

Summary Table:

Condition on $f(n)$	Asymptotic Bound for $T(n)$	Description
$f(n) = O(n^{\log_b a - \epsilon})$	$\Theta(n^{\log_b a})$	Recursive term dominates
$f(n) = \Theta(n^{\log_b a} \log^k n)$	$\Theta(n^{\log_b a} \log^{k+1} n)$	Balanced growth
$f(n) = \Omega(n^{\log_b a + \epsilon})$ with regularity	$\Theta(f(n))$	Non-recursive work dominates

---

2. Recurrences with Polynomial Non-Recursive Work: $T(n) = 2 T(n/2) + n$

Analysis:

- Parameters: $a=2$, $b=2$, $f(n) = n$.

- Compute: $\log_b a = \log_2 2 = 1$.

- Comparison: $f(n) = \Theta(n)$, which equals $n^{\log_b a}$.

- Result: By the Master Theorem, since $f(n) = \Theta(n^{\log_b a})$, the total recurrence is:

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

Conclusion: The algorithm has an asymptotic runtime of $\Theta(n \log n)$, typical for efficient sorting algorithms like mergesort.

---

3. Recurrences with Super-Polynomial Non-Recursive Work: $T(n) = 3 T(n/2) + n^2$

Analysis:

- Parameters: $a=3$, $b=2$, $f(n)=n^2$.

- Calculate: $\log_b a = \log_2 3 \approx 1.5849$.

- Comparison: $f(n) = n^2$, which grows faster than $n^{1.5849}$.

- Implication: Since $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some ϵ, and regularity conditions are satisfied, the dominant term is $f(n)$.

- Conclusion: By the Master Theorem,

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

meaning the non-recursive work dominates overall runtime.

---

4. Recurrences with Logarithmic Non-Recursive Work: $T(n) = 2 T(n/2) + \log n$

Analysis:

- Parameters: $a=2$, $b=2$, $f(n) = \log n$.

- Compute: $\log_b a = 1$.

- Comparison: $f(n) = \Theta(\log n)$, which is smaller than n^{1}.

- Application of Master Theorem: Since $f(n) = O(n^{\log_b a - \epsilon})$, the total bounds are:

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

Result: The total runtime is linear, driven primarily by the recursive structure.

Special Cases and Non-Standard Recurrences

While the Master Theorem covers a broad class of recurrences, some relations fall outside its scope and require alternative methods.

1. Recursions with Non-Uniform Subproblem Sizes

For example:

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

which simplifies to:

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

Unfolding:

$$
T(n) = T(1) + (n-1) \implies T(n) = \Theta(n)
$$

This linear recurrence indicates a straightforward linear growth.

2. Recursions with Multiple Non-Recursive

Give The Asymptotic Upper And Lower Bounds For T N In Each Of The Following Recurrences Assume That

Give The Asymptotic Upper And Lower Bounds For T N In Each Of The Following Recurrences Assume That

Related Articles

- In International Buying, The Entity That Makes A Contract With The Buyer And Then Buys The Product In
- How Do Archaeologists Find Out About Early Peoples?*They Go To MuseumsThey Read Historical NovelsThey
- Harriet Tubman Tries To Encourage The Group Of Runaways ByA. Threatening Them With A GunB. Explaining

Back to Home