1 Prove the Bounds

This section provides rigorous mathematical proofs for upper and lower bounds of polynomial functions using Big-O and Big-Omega notation. Each proof explicitly identifies the constants c and n_0 required by the formal definitions.

1.1 Problem 1: $f(n) = 5n^4 + 3n^3 + 2n^2 + 7n + 10$

Prove that $f(n) \in \mathcal{O}(n^4)$ and $f(n) \in \Omega(n^3)$

1.1.1 Upper Bound: $f(n) \in \mathcal{O}(n^4)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $f(n) \leq c \cdot n^4$ for all $n \geq n_0$.

$$\begin{aligned} f(n) &= 5n^4 + 3n^3 + 2n^2 + 7n + 10 && (1)\\ &\leq 5n^4 + 3n^4 + 2n^4 + 7n^4 + 10n^4 \quad (\text{for } n \geq 1) && (2)\\ &= (5 + 3 + 2 + 7 + 10)n^4 && (3)\\ &= 27n^4 && (4) \end{aligned}$$

Therefore, with $\boxed{c = 27}$ and $\boxed{n_0 = 1}$, we have $f(n) \leq 27n^4$ for all $n \geq 1$.
Thus, $f(n) \in \mathcal{O}(n^4)$. □ □

1.1.2 Lower Bound: $f(n) \in \Omega(n^3)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $f(n) \geq c \cdot n^3$ for all $n \geq n_0$.
Since all coefficients are positive:

$$\begin{aligned} f(n) &= 5n^4 + 3n^3 + 2n^2 + 7n + 10 && (5)\\ &\geq 3n^3 \quad (\text{dropping all other positive terms}) && (6) \end{aligned}$$

Therefore, with $\boxed{c = 3}$ and $\boxed{n_0 = 1}$, we have $f(n) \geq 3n^3$ for all $n \geq 1$.
Thus, $f(n) \in \Omega(n^3)$. □ □

1.2 Problem 2: $f(n) = 8n^5 + 4n^4 + 6n^2 + 15$

Prove that $f(n) \in \mathcal{O}(n^5)$ and $f(n) \in \Omega(n^4)$

1.2.1 Upper Bound: $f(n) \in \mathcal{O}(n^5)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $f(n) \leq c \cdot n^5$ for all $n \geq n_0$.

$$\begin{aligned} f(n) &= 8n^5 + 4n^4 + 6n^2 + 15 && (7)\\ &\leq 8n^5 + 4n^5 + 6n^5 + 15n^5 \quad (\text{for } n \geq 1) && (8)\\ &= (8 + 4 + 6 + 15)n^5 && (9)\\ &= 33n^5 && (10) \end{aligned}$$

Therefore, with $\boxed{c = 33}$ and $\boxed{n_0 = 1}$, we have $f(n) \leq 33n^5$ for all $n \geq 1$.
Thus, $f(n) \in \mathcal{O}(n^5)$. □ □

1.2.2 Lower Bound: $f(n) \in \Omega(n^4)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $f(n) \geq c \cdot n^4$ for all $n \geq n_0$.

$$f(n) = 8n^5 + 4n^4 + 6n^2 + 15 \tag{11}$$
$$\geq 4n^4 \quad \text{(dropping all other positive terms)} \tag{12}$$

Therefore, with $\boxed{c = 4}$ and $\boxed{n_0 = 1}$, we have $f(n) \geq 4n^4$ for all $n \geq 1$.
Thus, $f(n) \in \Omega(n^4)$. □ □

1.3 Problem 3: $g(n) = 2n^6 + 9n^4 + 3n^3 + 20$

Prove that $g(n) \in \mathcal{O}(n^6)$ and $g(n) \in \Omega(n^4)$

1.3.1 Upper Bound: $g(n) \in \mathcal{O}(n^6)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $g(n) \leq c \cdot n^6$ for all $n \geq n_0$.

$$g(n) = 2n^6 + 9n^4 + 3n^3 + 20 \tag{13}$$
$$\leq 2n^6 + 9n^6 + 3n^6 + 20n^6 \quad (\text{for } n \geq 1) \tag{14}$$
$$= (2 + 9 + 3 + 20)n^6 \tag{15}$$
$$= 34n^6 \tag{16}$$

Therefore, with $\boxed{c = 34}$ and $\boxed{n_0 = 1}$, we have $g(n) \leq 34n^6$ for all $n \geq 1$.
Thus, $g(n) \in \mathcal{O}(n^6)$. □ □

1.3.2 Lower Bound: $g(n) \in \Omega(n^4)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $g(n) \geq c \cdot n^4$ for all $n \geq n_0$.

$$g(n) = 2n^6 + 9n^4 + 3n^3 + 20 \tag{17}$$
$$\geq 9n^4 \quad \text{(dropping all other positive terms)} \tag{18}$$

Therefore, with $\boxed{c = 9}$ and $\boxed{n_0 = 1}$, we have $g(n) \geq 9n^4$ for all $n \geq 1$.
Thus, $g(n) \in \Omega(n^4)$. □ □

1.4 Problem 4: $h(n) = 10n^7 + 5n^5 + 2n^4 + 1$

Prove that $h(n) \in \mathcal{O}(n^7)$ and $h(n) \in \Omega(n^5)$

1.4.1 Upper Bound: $h(n) \in \mathcal{O}(n^7)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $h(n) \leq c \cdot n^7$ for all $n \geq n_0$.

$$\begin{aligned} h(n) &= 10n^7 + 5n^5 + 2n^4 + 1 && (19)\\ &\leq 10n^7 + 5n^7 + 2n^7 + n^7 \quad (\text{for } n \geq 1) && (20)\\ &= (10 + 5 + 2 + 1)n^7 && (21)\\ &= 18n^7 && (22) \end{aligned}$$

Therefore, with $\boxed{c = 18}$ and $\boxed{n_0 = 1}$, we have $h(n) \leq 18n^7$ for all $n \geq 1$.
Thus, $h(n) \in \mathcal{O}(n^7)$. □ □

1.4.2 Lower Bound: $h(n) \in \Omega(n^5)$

Proof. We need to find constants $c > 0$ and $n_0 > 0$ such that $h(n) \geq c \cdot n^5$ for all $n \geq n_0$.

$$\begin{aligned} h(n) &= 10n^7 + 5n^5 + 2n^4 + 1 && (23)\\ &\geq 5n^5 \quad (\text{dropping all other positive terms}) && (24) \end{aligned}$$

Therefore, with $\boxed{c = 5}$ and $\boxed{n_0 = 1}$, we have $h(n) \geq 5n^5$ for all $n \geq 1$.
Thus, $h(n) \in \Omega(n^5)$. □ □

1.5 Summary of Results

Function	Upper Bound	c	Lower Bound	c
$5n^4 + 3n^3 + 2n^2 + 7n + 10$	$\mathcal{O}(n^4)$	27	$\Omega(n^3)$	3
$8n^5 + 4n^4 + 6n^2 + 15$	$\mathcal{O}(n^5)$	33	$\Omega(n^4)$	4
$2n^6 + 9n^4 + 3n^3 + 20$	$\mathcal{O}(n^6)$	34	$\Omega(n^4)$	9
$10n^7 + 5n^5 + 2n^4 + 1$	$\mathcal{O}(n^7)$	18	$\Omega(n^5)$	5

In all cases, $n_0 = 1$ suffices for both upper and lower bounds.

2 Determine the Order of Growth

This section determines the order of growth $\Theta()$ for various functions through asymptotic analysis and limit theorems.

2.1 Problem 1: $f(n) = 5n + n^4 + 400n^2 + 7000$

Task: Identify the most significant term as n becomes large and justify your answer.

2.2 Problem 2: $f(n) = 200n \log(n) + 3n$

Task: Discuss the impact of the logarithmic term in comparison to the linear term.

2.3 Problem 3: Compare $f(n) = 9n^3$ and $g(n) = n^3 + 200n + 5000$

Task: Determine if they have the same order of growth and justify your answer.

2.4 Problem 4: $f(n) = \log n$ and $g(n) = n^{1/3}$

Task: Compute $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ and determine the asymptotic relationship.

2.5 Problem 5: $f(n) = n^{\log n}$ and $g(n) = 2^n$

Task: Using the limit theorem, determine whether $f(n) \in \mathcal{O}(g(n))$ or $g(n) \in \mathcal{O}(f(n))$.

3 Analyze the Recurrence Relations

Analyze the Recurrence Relations (50 points)

This section solves recurrence relations using substitution method and the Master Theorem, providing step-by-step explanations for each solution.

Problem 3.1: $T(n) = T(n-1) + 5$, with $T(1) = 2$

Method: Backward Substitution/Telescoping

Step 1: Rewrite the relation using a different variable
First, rewrite the recurrence using x:

$$T(x) = T(x-1) + 5$$

Step 2: Use $T(x)$ to work out new expressions
Now evaluate what $T(n-1)$ equals:

$$T(n-1) = T(n-2) + 5$$

Step 3: Plug back the new expression into the original
Substituting this back into the original recurrence:

$$\begin{aligned} T(n) &= T(n-1) + 5 && (25)\\ &= [T(n-2) + 5] + 5 && (26)\\ &= T(n-2) + 10 && (27) \end{aligned}$$

Continue this pattern:

$$T(n-2) = T(n-3) + 5$$

$$\begin{aligned} T(n) &= T(n-2) + 10 && (28)\\ &= [T(n-3) + 5] + 10 && (29)\\ &= T(n-3) + 15 && (30) \end{aligned}$$

One more substitution:

$$T(n-3) = T(n-4) + 5$$

$$\begin{aligned} T(n) &= T(n-3) + 15 && (31)\\ &= [T(n-4) + 5] + 15 && (32)\\ &= T(n-4) + 20 && (33) \end{aligned}$$

Step 4: Identify the pattern after k recursions
After k recursive calls, the pattern emerges:

$$\boxed{T(n) = T(n-k) + 5k}$$

Anchor Step: Connect back to the base case
The base case is $T(1) = 2$. To reach the base case, we need:

$$\begin{aligned} n - k &= 1 && (34)\\ k &= n - 1 && (35) \end{aligned}$$

Substituting $k = n - 1$ into our pattern expression:

$$\begin{aligned} T(n) &= T(1) + 5(n-1) && (36)\\ &= 2 + 5(n-1) && (37)\\ &= 2 + 5n - 5 && (38)\\ &= 5n - 3 && (39) \end{aligned}$$

Final Answer:

$$\boxed{T(n) = 5n - 3}$$

Therefore, $T(n) = \Theta(n)$ – the time complexity is linear.
Verification:

- $T(1) = 2$ ✓(given)
- $T(2) = T(1) + 5 = 2 + 5 = 7 = 5(2) - 3 = 10 - 3 = 7$ ✓
- $T(3) = T(2) + 5 = 7 + 5 = 12 = 5(3) - 3 = 15 - 3 = 12$ ✓

Problem 3.2: $T(n) = 2T(n-1)$**, with** $T(1) = 1$

Method: Substitution

Solving $T(n) = 2T(n-1)$, $T(1) = 1$

Method: Backward Substitution

Step 1: Rewrite the relation using a different variable
First, rewrite the recurrence using x:

$$T(x) = 2T(x-1)$$

Step 2: Use $T(x)$ to work out new expressions
Now evaluate what $T(n-1)$ equals:

$$T(n-1) = 2T(n-2)$$

Step 3: Plug back the new expression into the original
Substituting this back into the original recurrence:

$$\begin{aligned} T(n) &= 2T(n-1) && (40)\\ &= 2[2T(n-2)] && (41)\\ &= 4T(n-2) && (42)\\ &= 2^2T(n-2) && (43) \end{aligned}$$

Continue this pattern:

$$T(n-2) = 2T(n-3)$$

$$\begin{aligned} T(n) &= 4T(n-2) && (44)\\ &= 4[2T(n-3)] && (45)\\ &= 8T(n-3) && (46)\\ &= 2^3T(n-3) && (47) \end{aligned}$$

One more substitution:

$$T(n-3) = 2T(n-4)$$

$$\begin{aligned} T(n) &= 8T(n-3) && (48)\\ &= 8[2T(n-4)] && (49)\\ &= 16T(n-4) && (50)\\ &= 2^4T(n-4) && (51) \end{aligned}$$

Step 4: Identify the pattern after k recursions

After k recursive calls, the pattern emerges:

$$\boxed{T(n) = 2^k T(n-k)}$$

Anchor Step: Connect back to the base case
The base case is $T(1) = 1$. To reach the base case, we need:

$$n - k = 1 \tag{52}$$
$$k = n - 1 \tag{53}$$

Substituting $k = n - 1$ into our pattern expression:

$$T(n) = 2^{n-1} T(1) \tag{54}$$
$$= 2^{n-1} \cdot 1 \tag{55}$$
$$= 2^{n-1} \tag{56}$$

Final Answer:

$$\boxed{T(n) = 2^{n-1}}$$

Therefore, $T(n) = \Theta(2^n)$ – the time complexity is exponential.
Verification:

- $T(1) = 2^{1-1} = 2^0 = 1$ ✓(given)
- $T(2) = 2T(1) = 2 \cdot 1 = 2 = 2^{2-1} = 2^1 = 2$ ✓
- $T(3) = 2T(2) = 2 \cdot 2 = 4 = 2^{3-1} = 2^2 = 4$ ✓
- $T(4) = 2T(3) = 2 \cdot 4 = 8 = 2^{4-1} = 2^3 = 8$ ✓

Problem 3.3: $T(n) = T(n-1) + n$, with $T(1) = 1$

Method: Substitution/Telescoping

Step 1: Rewrite the relation using a different variable
First, rewrite the recurrence using x:

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

Step 2: Use $T(x)$ to work out new expressions
Now evaluate what $T(n-1)$ equals:

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

Step 3: Plug back the new expression into the original
Substituting this back into the original recurrence:

$$\begin{aligned} T(n) &= T(n-1) + n && (57)\\ &= [T(n-2) + (n-1)] + n && (58)\\ &= T(n-2) + (n-1) + n && (59) \end{aligned}$$

Continue this pattern:

$$T(n-2) = T(n-3) + (n-2)$$

$$\begin{aligned} T(n) &= T(n-2) + (n-1) + n && (60)\\ &= [T(n-3) + (n-2)] + (n-1) + n && (61)\\ &= T(n-3) + (n-2) + (n-1) + n && (62) \end{aligned}$$

One more substitution:

$$T(n-3) = T(n-4) + (n-3)$$

$$\begin{aligned} T(n) &= T(n-3) + (n-2) + (n-1) + n && (63)\\ &= [T(n-4) + (n-3)] + (n-2) + (n-1) + n && (64)\\ &= T(n-4) + (n-3) + (n-2) + (n-1) + n && (65) \end{aligned}$$

Step 4: Identify the pattern after k recursions
After k recursive calls, the pattern emerges:

$$T(n) = T(n-k) + \sum_{i=0}^{k-1} (n-i)$$

This sum can be written as:

$$T(n) = T(n-k) + [n + (n-1) + (n-2) + \cdots + (n-k+1)]$$

We can simplify this sum:

$$\sum_{i=0}^{k-1}(n-i) = kn - \sum_{i=0}^{k-1} i \tag{66}$$

$$= kn - \frac{k(k-1)}{2} \tag{67}$$

$$= \frac{2kn - k(k-1)}{2} \tag{68}$$

$$= \frac{k(2n-k+1)}{2} \tag{69}$$

Therefore:

$$\boxed{T(n) = T(n-k) + \frac{k(2n-k+1)}{2}}$$

Anchor Step: Connect back to the base case
The base case is $T(1) = 1$. To reach the base case, we need:

$$n - k = 1 \tag{70}$$

$$k = n - 1 \tag{71}$$

Substituting $k = n - 1$ into our pattern expression:

$$T(n) = T(1) + \frac{(n-1)(2n-(n-1)+1)}{2} \tag{72}$$

$$= 1 + \frac{(n-1)(2n-n+1+1)}{2} \tag{73}$$

$$= 1 + \frac{(n-1)(n+2)}{2} \tag{74}$$

$$= 1 + \frac{n^2+2n-n-2}{2} \tag{75}$$

$$= 1 + \frac{n^2+n-2}{2} \tag{76}$$

$$= \frac{2+n^2+n-2}{2} \tag{77}$$

$$= \frac{n^2+n}{2} \tag{78}$$

$$= \frac{n(n+1)}{2} \tag{79}$$

Final Answer:

$$\boxed{T(n) = \frac{n(n+1)}{2}}$$

Therefore, $T(n) = \Theta(n^2)$ – the time complexity is quadratic.
Verification:

- $T(1) = \frac{1(1+1)}{2} = \frac{2}{2} = 1$ ✓(given)
- $T(2) = T(1) + 2 = 1 + 2 = 3 = \frac{2(2+1)}{2} = \frac{6}{2} = 3$ ✓
- $T(3) = T(2) + 3 = 3 + 3 = 6 = \frac{3(3+1)}{2} = \frac{12}{2} = 6$ ✓
- $T(4) = T(3) + 4 = 6 + 4 = 10 = \frac{4(4+1)}{2} = \frac{20}{2} = 10$ ✓

Note: This result is the formula for the sum of the first n positive integers: $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$.

Problem 3.4: $T(n) = 5T(n/3) + O(n^2)$

Solve the recurrence relation $T(n) = 5T(n/3) + O(n^2)$ using the Master Theorem.

Master Theorem Framework

Definition 1 (Master Theorem)**.** *The Master Theorem applies to recurrence relations of the form:*

$$T(n) = aT(n/b) + f(n) \tag{80}$$

where:

- $a \geq 1$ *(number of recursive calls)*
- $b > 1$ *(factor by which input is divided)*
- $f(n)$ *is the overhead function*

Solution

Step 1: Identify Parameters

From $T(n) = 5T(n/3) + O(n^2)$:

$$a = 5 \quad \text{(we make 5 recursive calls)} \tag{81}$$
$$b = 3 \quad \text{(input size is divided by 3)} \tag{82}$$
$$f(n) = O(n^2) \quad \text{(quadratic overhead)} \tag{83}$$

Step 2: Calculate Critical Component

The critical component is defined as:

$$c = \log_b(a) = \log_3(5) \tag{84}$$

Calculating:

$$c = \log_3(5) = \frac{\ln(5)}{\ln(3)} \approx 1.465 \tag{85}$$

Step 3: Determine Which Case Applies

Theorem 1 (Master Theorem Cases)**.** *The Master Theorem has three cases:*

Case 1: Overhead Dominates

- *Condition:* $f(n) \in \Omega(n^d)$ *where* $d > c$
- *Result:* $T(n) = \Theta(f(n))$

Case 2: Balanced

- *Condition:* $f(n) \in \Theta(n^c \cdot \log^k(n))$ *where* $k \geq 0$
- *Result:* $T(n) = \Theta(n^c \cdot \log^{k+1}(n))$

Case 3: Recursion Dominates

- *Condition:* $f(n) \in O(n^d)$ *where* $d < c$
- *Result:* $T(n) = \Theta(n^c)$

Step 4: Apply the Appropriate Case

Given $f(n) = O(n^2)$:

- We have $d = 2$
- Critical component $c \approx 1.465$
- Since $d > c$ (i.e., $2 > 1.465$), we have **Case 1: Overhead Dominates**

Step 5: Final Solution

Since this is Case 1 (overhead dominates), the time complexity is:

$$\boxed{T(n) = \Theta(f(n)) = \Theta(n^2)} \tag{86}$$

Intuitive Understanding

In this recurrence relation:

- We're making 5 recursive calls on subproblems of size $n/3$
- The overhead at each level is n^2
- Since the overhead grows as n^2, which is faster than $n^{1.465}$ (the rate at which recursion grows), the overhead dominates the overall complexity
- The recursion tree has many leaves ($5^{\log_3 n} = n^{\log_3 5} \approx n^{1.465}$), but the n^2 work at the root dominates

Recursion Tree Analysis

Level	Number of Nodes	Work per Level
0	1	n^2
1	5	$5 \times (n/3)^2 = 5n^2/9$
2	25	$25 \times (n/9)^2 = 25n^2/81$
$\vdots$	$\vdots$	$\vdots$
k	5^k	$5^k \times (n/3^k)^2 = n^2 \times (5/9)^k$

Total work:

$$\sum_{k=0}^{\log_3 n} n^2 \times \left(\frac{5}{9}\right)^k = n^2 \times \sum_{k=0}^{\log_3 n} \left(\frac{5}{9}\right)^k \tag{87}$$

Since $\frac{5}{9} < 1$, the geometric series converges:

$$n^2 \times \frac{1}{1 - 5/9} = n^2 \times \frac{9}{4} = \Theta(n^2) \tag{88}$$

This confirms that the root's n^2 work dominates the overall complexity.

Visualization

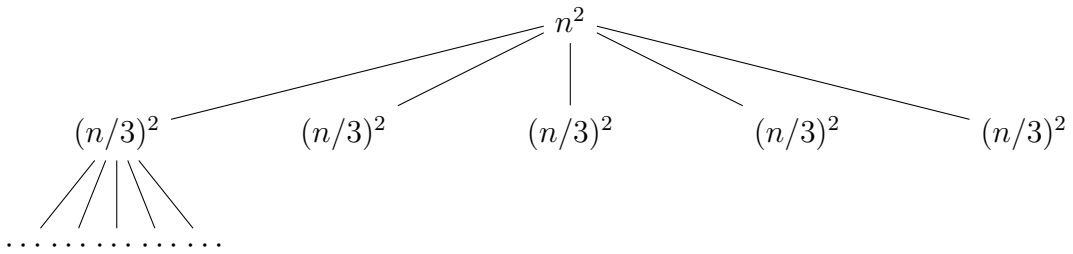

Figure 1: Recursion tree showing 5 recursive calls at each level with decreasing problem sizes

Problem 3.5: $T(n) = 2T(n/4) + O(\log n)$

Method: Master Theorem

Step 1: Identify Parameters

From the recurrence relation $T(n) = 2T(n/4) + O(\log n)$:

$$a = 2 \quad \text{(2 recursive calls)} \tag{89}$$
$$b = 4 \quad \text{(input size is divided by 4)} \tag{90}$$
$$f(n) = O(\log n) \quad \text{(logarithmic overhead)} \tag{91}$$

Step 2: Calculate Critical Component

The critical component is:

$$c = \log_b(a) = \log_4(2) \tag{92}$$

We can simplify this using the change of base formula:

$$c = \log_4(2) = \frac{\log_2(2)}{\log_2(4)} \tag{93}$$
$$= \frac{1}{2} \tag{94}$$
$$= 0.5 \tag{95}$$

Therefore, $c = 0.5$, which means $n^c = n^{0.5} = \sqrt{n}$.

Step 3: Determine Which Case Applies

We need to compare $f(n) = O(\log n)$ with $n^c = \sqrt{n}$.

Key observation: $\log n$ grows much slower than any polynomial function n^ϵ for any $\epsilon > 0$.

Since $\log n = o(n^\epsilon)$ for any $\epsilon > 0$, we have:

$$\log n = o(n^{0.5}) = o(\sqrt{n}) \tag{96}$$

This means $f(n) = O(n^d)$ where $d < c$ (in fact, we can think of d as approaching 0).

Therefore, this is **Case 3: Recursion Dominates**.

Step 4: Apply Case 3

According to the Master Theorem, when recursion dominates:

$$T(n) = \Theta(n^c) = \Theta(n^{0.5}) = \Theta(\sqrt{n}) \tag{97}$$

Step 5: Final Solution

$$\boxed{T(n) = \Theta(\sqrt{n})} \tag{98}$$

Intuitive Understanding

- We make 2 recursive calls on problems of size $n/4$
- The overhead at each level is only $O(\log n)$, which is very small
- The number of leaves in the recursion tree is $2^{\log_4 n} = n^{\log_4 2} = n^{0.5} = \sqrt{n}$
- Since the overhead ($\log n$) is negligible compared to the number of leaves ($\sqrt{n}$), the recursion dominates

Recursion Tree Analysis

Level	Number of Nodes	Work per Level
0	1	$\log n$
1	2	$2 \times \log(n/4) \approx 2\log n - 4$
2	4	$4 \times \log(n/16) \approx 4\log n - 16$
$\vdots$	$\vdots$	$\vdots$
k	2^k	$2^k \times \log(n/4^k)$

The depth of the tree is $\log_4 n$.

Number of leaves: $2^{\log_4 n} = n^{\log_4 2} = n^{0.5} = \sqrt{n}$

Since each leaf contributes $\Theta(1)$ work and there are $\sqrt{n}$ leaves, the total work from leaves is $\Theta(\sqrt{n})$, which dominates the $O(\log n)$ work at each internal node.

Verification

To verify our result, let's examine the total work:

$$\text{Total work} = \sum_{k=0}^{\log_4 n} 2^k \cdot \log\left(\frac{n}{4^k}\right) \tag{99}$$

$$= \sum_{k=0}^{\log_4 n} 2^k \cdot (\log n - k \log 4) \tag{100}$$

$$< \sum_{k=0}^{\log_4 n} 2^k \cdot \log n \tag{101}$$

$$= \log n \cdot \sum_{k=0}^{\log_4 n} 2^k \tag{102}$$

$$= \log n \cdot \frac{2^{\log_4 n+1} - 1}{2 - 1} \tag{103}$$

$$\approx 2\log n \cdot n^{\log_4 2} \tag{104}$$

$$= 2\log n \cdot \sqrt{n} \tag{105}$$

$$= o(\sqrt{n} \cdot \sqrt{n}) = o(n) \tag{106}$$

However, the leaves alone contribute $\Theta(\sqrt{n})$ work, confirming our answer.

4 Search Algorithm Implementations

This section provides pseudocode implementations for Binary Search and Ternary Search algorithms, along with detailed time complexity analysis and visual representations.

Algorithm 1: Binary Search

Pseudocode Implementation

Input: Sorted array $A[1..n]$, target value x
Output: Index of x in A, or -1 if not found

```
left ← 1;
right ← n;
while left ≤ right do
    mid ← ⌊(left + right)/2⌋;
    if A[mid] = x then
        return mid;
    end
    else if A[mid] < x then
        left ← mid + 1;
    end
    else
        right ← mid − 1;
    end
end
return −1;
```

Algorithm 1: Binary Search

Time Complexity Analysis

Recurrence Relation:

$$T(n) = T(n/2) + O(1) \tag{107}$$

Solution using Master Theorem:

- $a = 1$ (one recursive call)
- $b = 2$ (problem size halved)
- $f(n) = O(1)$ (constant time for comparison)
- Critical component: $c = \log_2(1) = 0$

Since $f(n) = O(1) = O(n^0)$, we have the balanced case (Case 2):

$$T(n) = \Theta(n^0 \log n) = \Theta(\log n) \tag{108}$$

Complexity Analysis:

- **Best Case:** $O(1)$ - Target element is at the middle position
- **Worst Case:** $O(\log n)$ - Target element is not present or at the extreme ends
- **Average Case:** $O(\log n)$ - On average, we need to search half the depth of the tree
- **Space Complexity:** $O(1)$ - Only uses a constant amount of extra space

Algorithm 2: Ternary Search

Pseudocode Implementation

Input: Sorted array $A[1..n]$, target value x
Output: Index of x in A, or -1 if not found

```
left ← 1;
right ← n;
while left ≤ right do
    mid1 ← left + ⌊(right − left)/3⌋;
    mid2 ← right − ⌊(right − left)/3⌋;
    if A[mid1] = x then
        return mid1;
    end
    if A[mid2] = x then
        return mid2;
    end
    if x < A[mid1] then
        right ← mid1 − 1;
    end
    else if x > A[mid2] then
        left ← mid2 + 1;
    end
    else
        left ← mid1 + 1;
        right ← mid2 − 1;
    end
end
return −1;
```

Algorithm 2: Ternary Search

Time Complexity Analysis

Recurrence Relation:

$$T(n) = T(n/3) + O(1) \tag{109}$$

Note: In the worst case, we eliminate only one-third of the array and continue with two-thirds.

Solution using Master Theorem:

- $a = 1$ (one recursive call on remaining portion)
- $b = 3$ (problem size reduced to one-third)
- $f(n) = O(1)$ (constant time for two comparisons)
- Critical component: $c = \log_3(1) = 0$

Since $f(n) = O(1) = O(n^0)$, we have the balanced case (Case 2):

$$T(n) = \Theta(n^0 \log n) = \Theta(\log_3 n) = \Theta(\log n) \tag{110}$$

Complexity Analysis:

- **Best Case:** $O(1)$ - Target element is at one of the two mid positions
- **Worst Case:** $O(\log_3 n) = O(\log n)$ - Target element is not present
- **Average Case:** $O(\log n)$ - Similar to binary search
- **Space Complexity:** $O(1)$ - Only uses a constant amount of extra space

Number of Comparisons:

- Binary Search: $\log_2 n$ comparisons (1 per iteration)
- Ternary Search: $2 \log_3 n$ comparisons (2 per iteration)

Using the change of base formula:

$$2 \log_3 n = 2 \cdot \frac{\log_2 n}{\log_2 3} \approx 2 \cdot \frac{\log_2 n}{1.585} \approx 1.26 \log_2 n \tag{111}$$

Therefore, ternary search actually performs **more comparisons** than binary search!

Comparison of Search Algorithms

Algorithm	**Time Complexity**	**Comparisons/Iteration**	**Total Comparisons**	**Space**
Binary Search	$O(\log_2 n)$	1	$\log_2 n$	$O(1)$
Ternary Search	$O(\log_3 n)$	2	$2 \log_3 n \approx 1.26 \log_2 n$	$O(1)$

Table 1: Complexity comparison of search algorithms

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, 4th edition. MIT Press, 2022.

[2] Donald E. Knuth. Big Omicron and Big Omega and Big Theta. *SIGACT News*, 8(2):18–24, 1976.

[3] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*, 2nd edition. Addison-Wesley, 1994.

[4] Robert Sedgewick and Kevin Wayne. *Algorithms*, 4th edition. Addison-Wesley, 2011.