Practice Analyzing Code

Strategies for Determining Complexity

Prof. Antonio Khalil Moretti

SCIS 313: Data Structures and Algorithm Analysis

Spelman College

Our Analysis Framework

For each code snippet, answer:

1. **How many times does each loop execute?**
2. **What values do the loop variables take?**
3. **What is the total operation count** $T(n)$?
4. **What is the Big-O complexity?**

Our Analysis Framework

For each code snippet, answer:

1. **How many times does each loop execute?**
2. **What values do the loop variables take?**
3. **What is the total operation count** $T(n)$?
4. **What is the Big-O complexity?**

Key Principle

Build up from the innermost operations outward, counting systematically.

Single Loop - Doubling

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Question: How many times does this loop execute?

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1
2	2	Yes	Yes	2

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1
2	2	Yes	Yes	2
3	4	Yes	Yes	3

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1
2	2	Yes	Yes	2
3	4	Yes	Yes	3
4	8	Yes	Yes	4

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1
2	2	Yes	Yes	2
3	4	Yes	Yes	3
4	8	Yes	Yes	4
5	16	No	Stop	4

Trace the Loop Variable

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Let's trace with $n = 16$:

Iteration	**Before:** i	$i < 16$?	**Execute?**	**After: count**
1	1	Yes	Yes	1
2	2	Yes	Yes	2
3	4	Yes	Yes	3
4	8	Yes	Yes	4
5	16	No	Stop	4

Look for Patterns

Values of i: $1, 2, 4, 8, 16, \ldots$

Look for Patterns

Values of i: $1, 2, 4, 8, 16, \ldots$

This is: $2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

Look for Patterns

Values of i: $1, 2, 4, 8, 16, \ldots$

This is: $2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

We stop when $i \geq n$, so when $2^k \geq n$

Look for Patterns

Values of i: $1, 2, 4, 8, 16, \ldots$

This is: $2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

We stop when $i \geq n$, so when $2^k \geq n$

Taking $\log_2$ of both sides:

$$k \geq \log_2 n$$

Look for Patterns

Values of i: $1, 2, 4, 8, 16, \ldots$

This is: $2^0, 2^1, 2^2, 2^3, 2^4, \ldots$

We stop when $i \geq n$, so when $2^k \geq n$

Taking $\log_2$ of both sides:

$$k \geq \log_2 n$$

Key Insight

When a loop variable **multiplies by a constant** (doubles, triples, etc.), the number of iterations is **logarithmic**.

Final Answer

Total operations:

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

$$\boxed{O(\log n)}$$

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

$$\boxed{O(\log n)}$$

In Big-O notation, the base of the logarithm doesn't matter!

$$O(\log_2 n) = O(\log_{10} n) = O(\log n)$$

Single Loop - Halving

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Question: How many times does this loop execute?

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16
2	16	Yes	Yes	8

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16
2	16	Yes	Yes	8
3	8	Yes	Yes	4

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16
2	16	Yes	Yes	8
3	8	Yes	Yes	4
4	4	Yes	Yes	2

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	Before: k	$k > 1$?	Execute?	After: k
1	32	Yes	Yes	16
2	16	Yes	Yes	8
3	8	Yes	Yes	4
4	4	Yes	Yes	2
5	2	Yes	Yes	1

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16
2	16	Yes	Yes	8
3	8	Yes	Yes	4
4	4	Yes	Yes	2
5	2	Yes	Yes	1
6	1	No	Stop	1

Trace the Loop Variable

```
int k = n;
while (k > 1) {
    k = k / 2;
}
```

Let's trace with $n = 32$:

Iteration	**Before:** k	$k > 1$?	**Execute?**	**After:** k
1	32	Yes	Yes	16
2	16	Yes	Yes	8
3	8	Yes	Yes	4
4	4	Yes	Yes	2
5	2	Yes	Yes	1
6	1	No	Stop	1

Loop body executes 5 times when $n = 32$

Analysis

After each iteration, k is halved:

Analysis

After each iteration, k is halved:

$$\begin{aligned}
\text{Start:} \quad & k = n \\
\text{After 1 iteration:} \quad & k = \frac{n}{2} \\
\text{After 2 iterations:} \quad & k = \frac{n}{4} = \frac{n}{2^2} \\
\text{After 3 iterations:} \quad & k = \frac{n}{8} = \frac{n}{2^3} \\
\text{After } m \text{ iterations:} \quad & k = \frac{n}{2^m}
\end{aligned}$$

Analysis

After each iteration, k is halved:

$$\begin{aligned}
\text{Start:} \quad & k = n \\
\text{After 1 iteration:} \quad & k = \frac{n}{2} \\
\text{After 2 iterations:} \quad & k = \frac{n}{4} = \frac{n}{2^2} \\
\text{After 3 iterations:} \quad & k = \frac{n}{8} = \frac{n}{2^3} \\
\text{After } m \text{ iterations:} \quad & k = \frac{n}{2^m}
\end{aligned}$$

When do we stop?

Analysis

After each iteration, k is halved:

$$\begin{aligned}
\text{Start:} \quad & k = n \\
\text{After 1 iteration:} \quad & k = \frac{n}{2} \\
\text{After 2 iterations:} \quad & k = \frac{n}{4} = \frac{n}{2^2} \\
\text{After 3 iterations:} \quad & k = \frac{n}{8} = \frac{n}{2^3} \\
\text{After } m \text{ iterations:} \quad & k = \frac{n}{2^m}
\end{aligned}$$

When do we stop?

We stop when $k \leq 1$:

Analysis

After each iteration, k is halved:

$$\begin{aligned}
\text{Start:} \quad & k = n \\
\text{After 1 iteration:} \quad & k = \frac{n}{2} \\
\text{After 2 iterations:} \quad & k = \frac{n}{4} = \frac{n}{2^2} \\
\text{After 3 iterations:} \quad & k = \frac{n}{8} = \frac{n}{2^3} \\
\text{After } m \text{ iterations:} \quad & k = \frac{n}{2^m}
\end{aligned}$$

When do we stop?

We stop when $k \leq 1$:

$$\frac{n}{2^m} \leq 1 \quad \Rightarrow \quad n \leq 2^m \quad \Rightarrow \quad \log_2 n \leq m$$

Final Answer

Total operations:

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

$$\boxed{O(\log n)}$$

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

$$\boxed{O(\log n)}$$

Key Insight

When a loop variable **divides by a constant** (halving, dividing by 3, etc.), the number of iterations is **logarithmic**.

Final Answer

Total operations:

$$T(n) = \log_2 n$$

Big-O complexity:

$$\boxed{O(\log n)}$$

Key Insight

When a loop variable **divides by a constant** (halving, dividing by 3, etc.), the number of iterations is **logarithmic**.

Common Pattern

Both doubling (Problem 1) and halving (Problem 2) are $O(\log n)$

Problem 3: Triangular Pattern - Growing

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        count++;
    }
}
```

Question: How many times does the innermost statement execute?

Problem 3: Triangular Pattern - Growing

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        count++;
    }
}
```

Question: How many times does the innermost statement execute?

Strategy: Analyze the inner loop for different values of the outer variable.

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i**?**

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3
3	0, 1, 2, 3	4

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3
3	0, 1, 2, 3	4
⋮	⋮	⋮

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3
3	0, 1, 2, 3	4
$\vdots$	$\vdots$	$\vdots$
$n-1$	$0, 1, \ldots, n-1$	n

Problem 3: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {  // Inner loop
        count++;
    }
}
```

How many times does the inner loop run for each i?

Outer i	Inner j values	Inner iterations
0	0	1
1	0, 1	2
2	0, 1, 2	3
3	0, 1, 2, 3	4
$\vdots$	$\vdots$	$\vdots$
$n-1$	$0, 1, \ldots, n-1$	n

Pattern: When $i = k$, inner loop runs $k + 1$ times

Problem 3: Summing the Iterations

Total executions of `count++`:

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

This is the sum of first n positive integers!

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

This is the sum of first n positive integers!

Formula:

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

This is the sum of first n positive integers!

Formula:

$$T(n) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

This is the sum of first n positive integers!

Formula:

$$T(n) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Expanding:

Problem 3: Summing the Iterations

Total executions of `count++`:

$$T(n) = 1 + 2 + 3 + 4 + \cdots + n$$

This is the sum of first n positive integers!

Formula:

$$T(n) = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Expanding:

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

Problem 3: Deriving Big-O

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

Problem 3: Deriving Big-O

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

For Big-O, we:

Problem 3: Deriving Big-O

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

For Big-O, we:

1. Keep only the **highest-order term**: $\cancel{\frac{1}{2}n^2} + \cancel{\frac{1}{2}n} \quad \Rightarrow \quad n^2$

Problem 3: Deriving Big-O

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

For Big-O, we:

1. Keep only the **highest-order term**: $\cancel{\frac{1}{2}n^2} + \cancel{\frac{1}{2}n} \quad \Rightarrow \quad n^2$
2. Drop the **constant coefficient**: $\cancel{\frac{1}{2}}n^2 \quad \Rightarrow \quad n^2$

Problem 3: Deriving Big-O

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

For Big-O, we:

1. Keep only the **highest-order term**: $\frac{1}{2}n^2 + \cancel{\frac{1}{2}n} \quad \Rightarrow \quad n^2$
2. Drop the **constant coefficient**: $\frac{1}{2}n^2 \quad \Rightarrow \quad n^2$

Big-O complexity:

Problem 3: Deriving Big-O

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

For Big-O, we:

1. Keep only the **highest-order term**: $\cancel{\frac{1}{2}n^2} + \cancel{\frac{1}{2}n} \quad \Rightarrow \quad n^2$
2. Drop the **constant coefficient**: $\cancel{\frac{1}{2}}n^2 \quad \Rightarrow \quad n^2$

Big-O complexity:

$$\boxed{O(n^2)}$$

Problem 3: Deriving Big-O

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

For Big-O, we:

1. Keep only the **highest-order term**: $\cancel{\frac{1}{2}n^2} + \cancel{\frac{1}{2}n} \quad \Rightarrow \quad n^2$
2. Drop the **constant coefficient**: $\cancel{\frac{1}{2}}n^2 \quad \Rightarrow \quad n^2$

Big-O complexity:

$$\boxed{O(n^2)}$$

Triangular Pattern Recognition

Whenever you see the pattern $1 + 2 + 3 + \cdots + n$, it's $\frac{n(n+1)}{2} = O(n^2)$

Problem 4: Nested with Doubling

```
int total = 0;
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j = j * 2) {
        total++;
    }
}
```

Question: What's the complexity when we nest different loop types?

Problem 4: Nested with Doubling

```
int total = 0;
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j = j * 2) {
        total++;
    }
}
```

Question: What's the complexity when we nest different loop types?

Strategy: Analyze each loop separately, then multiply.

Problem 4: Outer Loop Analysis

```
for (int i = 0; i < n; i++) {  // Outer loop
    for (int j = 1; j < n; j = j * 2) {
        total++;
    }
}
```

Outer loop:

Problem 4: Outer Loop Analysis

```
for (int i = 0; i < n; i++) {  // Outer loop
    for (int j = 1; j < n; j = j * 2) {
        total++;
    }
}
```

Outer loop:

- Starts at $i = 0$
- Continues while $i < n$
- Increments by 1
- **Executes n times**

Problem 4: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j = j * 2) {  // Inner loop
        total++;
    }
}
```

Inner loop (for each value of i):

Problem 4: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j = j * 2) {  // Inner loop
        total++;
    }
}
```

Inner loop (for each value of i):

- Starts at $j = 1$
- Continues while $j < n$
- **Doubles** each iteration: $j = 1, 2, 4, 8, \ldots$
- From Problem 3, we know this is **logarithmic**
- **Executes** $\log_2 n$ **times**

Problem 4: Inner Loop Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = 1; j < n; j = j * 2) {  // Inner loop
        total++;
    }
}
```

Inner loop (for each value of i):

- Starts at $j = 1$
- Continues while $j < n$
- **Doubles** each iteration: $j = 1, 2, 4, 8, \ldots$
- From Problem 3, we know this is **logarithmic**
- **Executes** $\log_2 n$ **times**

Key: The inner loop complexity is the **same** for every outer iteration!

Problem 4: Combining the Loops

Iteration table:

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$
2	$\log_2 n$	$3 \log_2 n$

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$
2	$\log_2 n$	$3 \log_2 n$
$\vdots$	$\vdots$	$\vdots$

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$
2	$\log_2 n$	$3 \log_2 n$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\log_2 n$	$n \log_2 n$

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$
2	$\log_2 n$	$3 \log_2 n$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\log_2 n$	$n \log_2 n$

Total operations:

Problem 4: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative total
0	$\log_2 n$	$\log_2 n$
1	$\log_2 n$	$2 \log_2 n$
2	$\log_2 n$	$3 \log_2 n$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\log_2 n$	$n \log_2 n$

Total operations:

$$T(n) = n \times \log_2 n = n \log n$$

Problem 4: Final Answer

Total operations:

$$T(n) = n \log n$$

Big-O complexity:

$$\boxed{O(n \log n)}$$

Nested Loop Rule

When loops are nested, **multiply** their complexities:

- Outer: $O(n)$
- Inner: $O(\log n)$
- Combined: $O(n) \times O(\log n) = O(n \log n)$

Problem 4: Final Answer

Total operations:

$$T(n) = n \log n$$

Big-O complexity:

$$\boxed{O(n \log n)}$$

Nested Loop Rule

When loops are nested, **multiply** their complexities:

- Outer: $O(n)$
- Inner: $O(\log n)$
- Combined: $O(n) \times O(\log n) = O(n \log n)$

Problem 5: Challenge - Doubling Outer, Linear Inner

```
int total = 0;
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Question: What happens when the inner loop bound depends on the outer variable?

Problem 5: Challenge - Doubling Outer, Linear Inner

```
int total = 0;
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Question: What happens when the inner loop bound depends on the outer variable?

Challenge: The inner loop size **changes** with each outer iteration!

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$
3	4	4	$3+4=7$

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$
3	4	4	$3+4=7$
4	8	8	$7+8=15$

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$
3	4	4	$3+4=7$
4	8	8	$7+8=15$
5	16	16	$15+16=31$

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$
3	4	4	$3+4=7$
4	8	8	$7+8=15$
5	16	16	$15+16=31$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Problem 5: Detailed Iteration Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 0; j < i; j++) {
        total++;
    }
}
```

Let's trace carefully:

Iteration	Outer i	Inner runs	Subtotal
1	1	1	1
2	2	2	$1+2=3$
3	4	4	$3+4=7$
4	8	8	$7+8=15$
5	16	16	$15+16=31$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\log_2 n$	$n/2$	$n/2$	?

Problem 5: Recognizing the Pattern

Total operations:

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

This is a **geometric series** with ratio 2!

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

This is a **geometric series** with ratio 2!

Geometric series formula:

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

This is a **geometric series** with ratio 2!

Geometric series formula:

$$\sum_{k=0}^{m} 2^k = 2^{m+1} - 1$$

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

This is a **geometric series** with ratio 2!

Geometric series formula:

$$\sum_{k=0}^{m} 2^k = 2^{m+1} - 1$$

The largest term is $\frac{n}{2} = 2^{\log_2 n - 1}$, so:

Problem 5: Recognizing the Pattern

Total operations:

$$T(n) = 1 + 2 + 4 + 8 + 16 + \cdots + \frac{n}{2}$$

This is a **geometric series** with ratio 2!

Geometric series formula:

$$\sum_{k=0}^{m} 2^k = 2^{m+1} - 1$$

The largest term is $\frac{n}{2} = 2^{\log_2 n - 1}$, so:

$$T(n) = 2^{\log_2 n} - 1 = n - 1$$

Problem 5: The Surprising Result

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

Problem 5: The Surprising Result

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

Big-O complexity:

Problem 5: The Surprising Result

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

Big-O complexity:

$$\boxed{O(n)}$$

Problem 5: The Surprising Result

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

Big-O complexity:

$$\boxed{O(n)}$$

Counterintuitive!

Even though the outer loop is $O(\log n)$, the overall complexity is $O(n)$ because:

Problem 5: The Surprising Result

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

Big-O complexity:

$$\boxed{O(n)}$$

Counterintuitive!

Even though the outer loop is $O(\log n)$, the overall complexity is $O(n)$ because:

- The inner loop grows **exponentially**

Problem 5: The Surprising Result

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

Big-O complexity:

$$\boxed{O(n)}$$

Counterintuitive!

Even though the outer loop is $O(\log n)$, the overall complexity is $O(n)$ because:

- The inner loop grows **exponentially**
- The geometric series $1 + 2 + 4 + \cdots + \frac{n}{2}$ sums to approximately n

Problem 5: The Surprising Result

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

Big-O complexity:

$$\boxed{O(n)}$$

Counterintuitive!

Even though the outer loop is $O(\log n)$, the overall complexity is $O(n)$ because:

- The inner loop grows **exponentially**
- The geometric series $1 + 2 + 4 + \cdots + \frac{n}{2}$ sums to approximately n
- The last few iterations dominate the total work!

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

- Sum $= 31$

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

- Sum $= 31$
- Largest term (16) is more than half the total!

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

- Sum $= 31$
- Largest term (16) is more than half the total!
- General rule: $1 + 2 + 4 + \cdots + k \approx 2k$

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

- Sum $= 31$
- Largest term (16) is more than half the total!
- General rule: $1 + 2 + 4 + \cdots + k \approx 2k$

Important Formula

$$\sum_{i=0}^{\log_2 n - 1} 2^i = 2^{\log_2 n} - 1 = n - 1 = O(n)$$

Problem 5: Geometric Series Insight

Key pattern: In a geometric series, the sum is dominated by the largest term.

For $1 + 2 + 4 + 8 + 16$:

- Sum $= 31$
- Largest term (16) is more than half the total!
- General rule: $1 + 2 + 4 + \cdots + k \approx 2k$

Important Formula

$$\sum_{i=0}^{\log_2 n - 1} 2^i = 2^{\log_2 n} - 1 = n - 1 = O(n)$$

Don't be fooled: $O(\log n)$ outer loop $\times$ exponentially growing inner $\neq O(n \log n)$!

Summary: Key Patterns

Loop Pattern	Complexity

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$
Two nested loops, both n	$O(n^2)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$
Two nested loops, both n	$O(n^2)$
Triangular: $1 + 2 + \cdots + n$	$O(n^2)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$
Two nested loops, both n	$O(n^2)$
Triangular: $1 + 2 + \cdots + n$	$O(n^2)$
Outer n, inner $\log n$	$O(n \log n)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$
Two nested loops, both n	$O(n^2)$
Triangular: $1 + 2 + \cdots + n$	$O(n^2)$
Outer n, inner $\log n$	$O(n \log n)$
Geometric: $1 + 2 + 4 + \cdots + \frac{n}{2}$	$O(n)$

Summary: Key Patterns

Loop Pattern	Complexity
Single loop, increment by 1	$O(n)$
Single loop, multiply by constant	$O(\log n)$
Single loop, divide by constant	$O(\log n)$
Two nested loops, both n	$O(n^2)$
Triangular: $1 + 2 + \cdots + n$	$O(n^2)$
Outer n, inner $\log n$	$O(n \log n)$
Geometric: $1 + 2 + 4 + \cdots + \frac{n}{2}$	$O(n)$

The Methodology

1. Identify loop bounds and increments
2. Trace a small example
3. Recognize the pattern
4. Apply the appropriate formula
5. Simplify to Big-O

Important Formulas to Remember

Arithmetic series (linear):

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Geometric series (powers of 2):

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Geometric series (powers of 2):

$$\sum_{k=0}^{m} 2^k = 1 + 2 + 4 + \cdots + 2^m = 2^{m+1} - 1 = O(2^m)$$

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Geometric series (powers of 2):

$$\sum_{k=0}^{m} 2^k = 1 + 2 + 4 + \cdots + 2^m = 2^{m+1} - 1 = O(2^m)$$

Logarithm properties:

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Geometric series (powers of 2):

$$\sum_{k=0}^{m} 2^k = 1 + 2 + 4 + \cdots + 2^m = 2^{m+1} - 1 = O(2^m)$$

Logarithm properties:

$$\log_a n = \frac{\log_b n}{\log_b a} = O(\log n) \quad \text{(base doesn't matter)}$$

$$\log(n \cdot m) = \log n + \log m$$

$$\log(n^k) = k \log n$$

Important Formulas to Remember

Arithmetic series (linear):

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} = O(n^2)$$

Geometric series (powers of 2):

$$\sum_{k=0}^{m} 2^k = 1 + 2 + 4 + \cdots + 2^m = 2^{m+1} - 1 = O(2^m)$$

Logarithm properties:

$$\log_a n = \frac{\log_b n}{\log_b a} = O(\log n) \quad \text{(base doesn't matter)}$$

$$\log(n \cdot m) = \log n + \log m$$

$$\log(n^k) = k \log n$$

Problem 16: Sequential Sections - Dominant Term

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}

// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Question: When code has multiple sequential sections, how do we combine their complexities?

Problem 6: Analyze Each Section

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Section 1 Analysis:

Problem 6: Analyze Each Section

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Section 1 Analysis:

- Loop variable doubles: $i = 1, 2, 4, 8, \ldots$

Problem 6: Analyze Each Section

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Section 1 Analysis:

- Loop variable doubles: $i = 1, 2, 4, 8, \ldots$
- This is logarithmic (from Problem 3)

Problem 6: Analyze Each Section

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Section 1 Analysis:

- Loop variable doubles: $i = 1, 2, 4, 8, \ldots$
- This is logarithmic (from Problem 3)
- $T_1(n) = \log_2 n$

Problem 6: Analyze Each Section

```
// Section 1
for (int i = 1; i < n; i = i * 2) {
    count++;
}
```

Section 1 Analysis:

- Loop variable doubles: $i = 1, 2, 4, 8, \ldots$
- This is logarithmic (from Problem 3)
- $T_1(n) = \log_2 n$
- Complexity: $O(\log n)$

Problem 6: Analyze Section 2

```
// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Section 2 Analysis:

Problem 6: Analyze Section 2

```
// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Section 2 Analysis:

- Outer loop: n iterations

Problem 6: Analyze Section 2

```
// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Section 2 Analysis:

- Outer loop: n iterations
- Inner loop: n iterations for each outer

Problem 6: Analyze Section 2

```
// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Section 2 Analysis:

- Outer loop: n iterations
- Inner loop: n iterations for each outer
- $T_2(n) = n \times n = n^2$

Problem 6: Analyze Section 2

```
// Section 2
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        count++;
    }
}
```

Section 2 Analysis:

- Outer loop: n iterations
- Inner loop: n iterations for each outer
- $T_2(n) = n \times n = n^2$
- Complexity: $O(n^2)$

Problem 16: Combining Sequential Code

Total operations:

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2
10	3.3	100
100	6.6	10,000
1000	10	1,000,000

The n^2 term completely dominates!

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2
10	3.3	100

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2
10	3.3	100
100	6.6	10,000

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2
10	3.3	100
100	6.6	10,000
1000	10	1,000,000

Problem 16: Combining Sequential Code

Total operations:

$$\begin{aligned} T(n) &= T_1(n) + T_2(n) \\ &= \log n + n^2 \end{aligned}$$

For Big-O, keep only the dominant (fastest-growing) term:

n	$\log_2 n$	n^2
10	3.3	100
100	6.6	10,000
1000	10	1,000,000

The n^2 term completely dominates!

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

- Add the complexities: $O(f(n)) + O(g(n))$

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

- Add the complexities: $O(f(n)) + O(g(n))$
- Keep only the dominant term

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

- Add the complexities: $O(f(n)) + O(g(n))$
- Keep only the dominant term
- $O(\log n) + O(n^2) = O(n^2)$

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

- Add the complexities: $O(f(n)) + O(g(n))$
- Keep only the dominant term
- $O(\log n) + O(n^2) = O(n^2)$
- $O(n) + O(n \log n) = O(n \log n)$

Problem 6: Final Answer

$$T(n) = \log n + n^2$$

Big-O complexity:

$$\boxed{O(n^2)}$$

Sequential Code Rule

When combining sequential sections:

- Add the complexities: $O(f(n)) + O(g(n))$
- Keep only the dominant term
- $O(\log n) + O(n^2) = O(n^2)$
- $O(n) + O(n \log n) = O(n \log n)$

Problem 17: Shrinking Triangular Pattern

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Question: How does this compare to Problem 8 (growing triangular)?

Problem 17: Shrinking Triangular Pattern

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Question: How does this compare to Problem 8 (growing triangular)?

Key difference: Inner loop starts at i instead of 0!

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$
2	2 to $n-1$	$n-2$	$3n-3$

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$
2	2 to $n-1$	$n-2$	$3n-3$
3	3 to $n-1$	$n-3$	$4n-6$

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$
2	2 to $n-1$	$n-2$	$3n-3$
3	3 to $n-1$	$n-3$	$4n-6$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$
2	2 to $n-1$	$n-2$	$3n-3$
3	3 to $n-1$	$n-3$	$4n-6$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Problem 17: Detailed Iteration Analysis

```
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        count++;
    }
}
```

Trace the inner loop:

Outer i	Inner j range	Inner iterations	Cumulative
0	0 to $n-1$	n	n
1	1 to $n-1$	$n-1$	$2n-1$
2	2 to $n-1$	$n-2$	$3n-3$
3	3 to $n-1$	$n-3$	$4n-6$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Problem 17: Summing the Pattern

Total operations:

Problem 17: Summing the Pattern

Total operations:

$$T(n) = n + (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

Problem 17: Summing the Pattern

Total operations:

$$T(n) = n + (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

This is the same sum as Problem 8, just written backwards!

Problem 17: Summing the Pattern

Total operations:

$$T(n) = n + (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

This is the same sum as Problem 8, just written backwards!

$$\begin{aligned} T(n) &= \sum_{k=1}^{n} k \\ &= \frac{n(n+1)}{2} \\ &= \frac{n^2 + n}{2} \end{aligned}$$

Problem 17: Summing the Pattern

Total operations:

$$T(n) = n + (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

This is the same sum as Problem 8, just written backwards!

$$T(n) = \sum_{k=1}^{n} k$$

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

Problem 17: Summing the Pattern

Total operations:

$$T(n) = n + (n-1) + (n-2) + (n-3) + \cdots + 2 + 1$$

This is the same sum as Problem 8, just written backwards!

$$\begin{aligned} T(n) &= \sum_{k=1}^{n} k \\ &= \frac{n(n+1)}{2} \\ &= \frac{n^2 + n}{2} \end{aligned}$$

Problem 17: Final Answer

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

Problem 17: Final Answer

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

Big-O complexity:

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Key Insight: Direction Doesn't Matter

Both triangular patterns give $O(n^2)$:

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Key Insight: Direction Doesn't Matter

Both triangular patterns give $O(n^2)$:

- Growing: $1 + 2 + 3 + \cdots + n = O(n^2)$

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Key Insight: Direction Doesn't Matter

Both triangular patterns give $O(n^2)$:

- Growing: $1 + 2 + 3 + \cdots + n = O(n^2)$
- Shrinking: $n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Key Insight: Direction Doesn't Matter

Both triangular patterns give $O(n^2)$:

- Growing: $1 + 2 + 3 + \cdots + n = O(n^2)$
- Shrinking: $n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$
- The sum is the same!

Problem 17: Final Answer

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

Big-O complexity:

$$\boxed{O(n^2)}$$

Key Insight: Direction Doesn't Matter

Both triangular patterns give $O(n^2)$:

- Growing: $1 + 2 + 3 + \cdots + n = O(n^2)$
- Shrinking: $n + (n-1) + (n-2) + \cdots + 1 = O(n^2)$
- The sum is the same!

Problem 18: Nested Logarithmic Loops

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Question: What happens when we nest two logarithmic loops with different bases?

Problem 18: Nested Logarithmic Loops

```
int count = 0;
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Question: What happens when we nest two logarithmic loops with different bases?

Challenge: Does $\log \times \log = \log$?

Problem 18: Outer Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {  // Outer
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Outer loop (doubling):

Problem 18: Outer Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {  // Outer
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Outer loop (doubling):

- i takes values: $1, 2, 4, 8, 16, \ldots$

Problem 18: Outer Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {  // Outer
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Outer loop (doubling):

- i takes values: $1, 2, 4, 8, 16, \ldots$
- Stops when $i \geq n$, so when $2^k \geq n$

Problem 18: Outer Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {  // Outer
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Outer loop (doubling):

- i takes values: $1, 2, 4, 8, 16, \ldots$
- Stops when $i \geq n$, so when $2^k \geq n$
- Number of iterations: $\log_2 n$

Problem 18: Outer Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {  // Outer
    for (int j = 1; j < n; j = j * 3) {
        count++;
    }
}
```

Outer loop (doubling):

- i takes values: $1, 2, 4, 8, 16, \ldots$
- Stops when $i \geq n$, so when $2^k \geq n$
- Number of iterations: $\log_2 n$

Note: The outer loop is $O(\log n)$ regardless of base!

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

- j takes values: $1, 3, 9, 27, 81, \ldots$

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

- j takes values: $1, 3, 9, 27, 81, \ldots$
- This is: $3^0, 3^1, 3^2, 3^3, 3^4, \ldots$

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

- j takes values: $1, 3, 9, 27, 81, \ldots$
- This is: $3^0, 3^1, 3^2, 3^3, 3^4, \ldots$
- Stops when $j \geq n$, so when $3^m \geq n$

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

- j takes values: $1, 3, 9, 27, 81, \ldots$
- This is: $3^0, 3^1, 3^2, 3^3, 3^4, \ldots$
- Stops when $j \geq n$, so when $3^m \geq n$
- Number of iterations: $\log_3 n$

Problem 18: Inner Loop Analysis

```
for (int i = 1; i < n; i = i * 2) {
    for (int j = 1; j < n; j = j * 3) {  // Inner
        count++;
    }
}
```

Inner loop (tripling):

- j takes values: $1, 3, 9, 27, 81, \ldots$
- This is: $3^0, 3^1, 3^2, 3^3, 3^4, \ldots$
- Stops when $j \geq n$, so when $3^m \geq n$
- Number of iterations: $\log_3 n$

Key: Inner loop runs $\log_3 n$ times for EACH outer iteration!

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \frac{\log_2 n}{1.585} \approx 0.631 \cdot \log_2 n$$

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \frac{\log_2 n}{1.585} \approx 0.631 \cdot \log_2 n$$

The $\frac{1}{\log_2 3}$ is just a constant!

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \frac{\log_2 n}{1.585} \approx 0.631 \cdot \log_2 n$$

The $\frac{1}{\log_2 3}$ is just a constant!

So in Big-O notation:

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \frac{\log_2 n}{1.585} \approx 0.631 \cdot \log_2 n$$

The $\frac{1}{\log_2 3}$ is just a constant!

So in Big-O notation:

$$\log_3 n = O(\log n)$$

Problem 18: Different Bases, Same Big-O

Converting logarithm bases:

$$\log_3 n = \frac{\log_2 n}{\log_2 3} = \frac{\log_2 n}{1.585} \approx 0.631 \cdot \log_2 n$$

The $\frac{1}{\log_2 3}$ is just a constant!

So in Big-O notation:

$$\log_3 n = O(\log n)$$

Critical Point

The base of the logarithm doesn't matter in Big-O!

$$\log_2 n = O(\log n) = \log_3 n = \log_{10} n = \log_e n$$

Problem 18: Combining the Loops

Iteration breakdown:

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$
3	$\log_3 n$	$3 \log_3 n$

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$
3	$\log_3 n$	$3 \log_3 n$
$\vdots$	$\vdots$	$\vdots$

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$
3	$\log_3 n$	$3 \log_3 n$
$\vdots$	$\vdots$	$\vdots$
$\log_2 n$	$\log_3 n$	$(\log_2 n) \times (\log_3 n)$

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$
3	$\log_3 n$	$3 \log_3 n$
$\vdots$	$\vdots$	$\vdots$
$\log_2 n$	$\log_3 n$	$(\log_2 n) \times (\log_3 n)$

Total operations:

Problem 18: Combining the Loops

Iteration breakdown:

Outer iteration	Inner executions	Cumulative
1	$\log_3 n$	$\log_3 n$
2	$\log_3 n$	$2 \log_3 n$
3	$\log_3 n$	$3 \log_3 n$
$\vdots$	$\vdots$	$\vdots$
$\log_2 n$	$\log_3 n$	$(\log_2 n) \times (\log_3 n)$

Total operations:

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$T(n) = O(\log n) \times O(\log n)$$

$$= O(\log^2 n)$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 18: Final Answer

$$T(n) = (\log_2 n) \times (\log_3 n)$$

Since both logarithms are $O(\log n)$:

$$\begin{aligned} T(n) &= O(\log n) \times O(\log n) \\ &= O(\log n \cdot \log n) \\ &= O(\log^2 n) \end{aligned}$$

Big-O complexity:

$$\boxed{O(\log^2 n)}$$

Important: Logarithms Multiply!

Problem 19: Square Root Loop (Challenge)

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j * j < n; j++) {
        count++;
    }
}
```

Challenge: The inner loop condition is unusual! How many times does it run?

Problem 19: Square Root Loop (Challenge)

```
int count = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j * j < n; j++) {
        count++;
    }
}
```

Challenge: The inner loop condition is unusual! How many times does it run?

Key question: For what values of j is $j \times j < n$ true?

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Understanding the Inner Bound

```
for (int j = 0; j * j < n; j++) {
    count++;
}
```

The condition $j \times j < n$ is equivalent to $j^2 < n$

Taking the square root of both sides:

$$j < \sqrt{n}$$

Example with $n = 16$:

j	j^2	$j^2 < 16$?	Continue?
0	0		Yes

Problem 19: Outer Loop Analysis

Outer loop:

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n - 1$

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Inner loop:

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Inner loop:

- Runs while $j^2 < n$, i.e., while $j < \sqrt{n}$

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Inner loop:

- Runs while $j^2 < n$, i.e., while $j < \sqrt{n}$
- Runs approximately $\sqrt{n}$ times

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Inner loop:

- Runs while $j^2 < n$, i.e., while $j < \sqrt{n}$
- Runs approximately $\sqrt{n}$ times
- Importantly: this is the SAME for every outer iteration!

Problem 19: Outer Loop Analysis

Outer loop:

- Standard loop: $i = 0, 1, 2, \ldots, n-1$
- Runs n times

Inner loop:

- Runs while $j^2 < n$, i.e., while $j < \sqrt{n}$
- Runs approximately $\sqrt{n}$ times
- Importantly: this is the SAME for every outer iteration!

Key Insight

The inner loop bound depends on n, not on i. So every outer iteration does the same amount of work.

Problem 19: Combining the Loops

Iteration table:

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$
2	$\sqrt{n}$	$3\sqrt{n}$

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$
2	$\sqrt{n}$	$3\sqrt{n}$
$\vdots$	$\vdots$	$\vdots$

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$
2	$\sqrt{n}$	$3\sqrt{n}$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\sqrt{n}$	$n \cdot \sqrt{n}$

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$
2	$\sqrt{n}$	$3\sqrt{n}$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\sqrt{n}$	$n \cdot \sqrt{n}$

Total operations:

Problem 19: Combining the Loops

Iteration table:

Outer i	Inner executions	Cumulative
0	$\sqrt{n}$	$\sqrt{n}$
1	$\sqrt{n}$	$2\sqrt{n}$
2	$\sqrt{n}$	$3\sqrt{n}$
$\vdots$	$\vdots$	$\vdots$
$n-1$	$\sqrt{n}$	$n \cdot \sqrt{n}$

Total operations:

$$T(n) = n \times \sqrt{n} = n^1 \times n^{1/2} = n^{3/2}$$

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$
- For $n = 1000$: $n = 1000$, $n^{3/2} \approx 31{,}623$, $n^2 = 1{,}000{,}000$

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$
- For $n = 1000$: $n = 1000$, $n^{3/2} \approx 31{,}623$, $n^2 = 1{,}000{,}000$

Where You See This

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$
- For $n = 1000$: $n = 1000$, $n^{3/2} \approx 31{,}623$, $n^2 = 1{,}000{,}000$

Where You See This

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$
- For $n = 1000$: $n = 1000$, $n^{3/2} \approx 31{,}623$, $n^2 = 1{,}000{,}000$

Where You See This

Problem 19: Final Answer

$$T(n) = n \times \sqrt{n} = n^{3/2}$$

Big-O complexity:

$$\boxed{O(n^{3/2})} \quad \text{or equivalently} \quad \boxed{O(n\sqrt{n})}$$

Understanding the Complexity

- $n^{3/2}$ is between n and n^2
- For $n = 100$: $n = 100$, $n^{3/2} = 1000$, $n^2 = 10{,}000$
- For $n = 1000$: $n = 1000$, $n^{3/2} \approx 31{,}623$, $n^2 = 1{,}000{,}000$

Where You See This

Practice Strategy

When analyzing code:

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Common Mistakes to Avoid

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Common Mistakes to Avoid

- Assuming nested loops always multiply naively

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Common Mistakes to Avoid

- Assuming nested loops always multiply naively
- Forgetting that logarithm base doesn't matter in Big-O

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Common Mistakes to Avoid

- Assuming nested loops always multiply naively
- Forgetting that logarithm base doesn't matter in Big-O
- Not checking loop bounds carefully (especially with while loops)

Practice Strategy

When analyzing code:

1. **Don't guess** – trace small examples
2. **Build a table** – see the pattern emerge
3. **Recognize the pattern** – arithmetic? geometric? logarithmic?
4. **Apply the formula** – calculate $T(n)$
5. **Simplify to Big-O** – drop constants and lower-order terms

Common Mistakes to Avoid

- Assuming nested loops always multiply naively
- Forgetting that logarithm base doesn't matter in Big-O
- Not checking loop bounds carefully (especially with while loops)
- Confusing arithmetic series (linear sum) with geometric series (exponential sum)

Questions?

Ready to practice more problems?

Remember: **Trace, recognize, calculate, simplify**