

Critical Section of Algorithm

- **Heart of algorithm**
- **Dominates overall execution time**
- **Characteristics**
 - **Operation central to functioning of program**
 - **Contained inside deeply nested loops**
 - **Executed as often as any other part of algorithm**
- **Sources**
 - **Loops**
 - **Recursion**

Critical Section Example 1

- **Code (for input size n)**

1. **A**
2. **for (int i = 0; i < n ; i++)**
3. **B**
4. **C**

- **Code execution**

- **A** \Rightarrow
- **B** \Rightarrow
- **C** \Rightarrow

- **Time** \Rightarrow

Critical Section Example 1

■ Code (for input size n)

1. A
2. for (int i = 0; i < n ; i++)
3. B
4. C

critical
section



■ Code execution

- A \Rightarrow once
- B \Rightarrow n times
- C \Rightarrow once

■ Time $\Rightarrow 1 + n + 1 = O(n)$

Critical Section Example 2

■ Code (for input size n)

1. A
2. for (int i = 0; i < n ; i++)
3. B
4. for (int j = 0; j < n ; j++)
5. C
6. D

■ Code execution

- | | |
|-------------------|-------------------|
| ■ A \Rightarrow | ■ C \Rightarrow |
| ■ B \Rightarrow | ■ D \Rightarrow |

■ Time \Rightarrow

Critical Section Example 2

■ Code (for input size n)

1. A

2. for (int i = 0; i < n ; i++)

3. B

4. for (int j = 0; j < n ; j++)

5. C

6. D

critical
section



■ Code execution

■ A \Rightarrow once

■ C $\Rightarrow n^2$ times

■ B $\Rightarrow n$ times

■ D \Rightarrow once

■ Time $\Rightarrow 1 + n + n^2 + 1 = O(n^2)$

Critical Section Example 3

■ Code (for input size n)

1. A
2. for (int $i = 0$; $i < n$; $i++$)
3. for (int $j = i+1$; $j < n$; $j++$)
4. B

■ Code execution

- A \Rightarrow
- B \Rightarrow

■ Time \Rightarrow

Critical Section Example 3

■ Code (for input size n)

1. A
2. for (int $i = 0$; $i < n$; $i++$)
3. for (int $j = i+1$; $j < n$; $j++$)
4. B

critical
section



■ Code execution

- A \Rightarrow once
- B $\Rightarrow \frac{1}{2} n (n-1)$ times

■ Time $\Rightarrow 1 + \frac{1}{2} n^2 = O(n^2)$

Critical Section Example 4

■ Code (for input size n)

1. A
2. for (int i = 0; i < n ; i++)
3. for (int j = 0; j < 10000; j++)
4. B

■ Code execution

- A \Rightarrow
- B \Rightarrow

■ Time \Rightarrow

Critical Section Example 4

■ Code (for input size n)

1. A
2. for (int i = 0; i < n ; i++)
3. for (int j = 0; j < 10000; j++)
4. B

critical
section



■ Code execution

- A \Rightarrow once
- B \Rightarrow 10000 n times

■ Time $\Rightarrow 1 + 10000 n = O(n)$

Critical Section Example 5

■ Code (for input size n)

1. `for (int i = 0; i < n; i++)`
2. `for (int j = 0; j < n; j++)`
3. A
4. `for (int i = 0; i < n; i++)`
5. `for (int j = 0; j < n; j++)`
6. B

■ Code execution

■ A \Rightarrow

■ B \Rightarrow

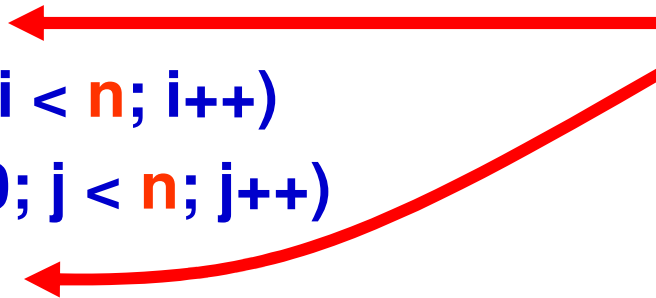
■ Time \Rightarrow

Critical Section Example 5

■ Code (for input size n)

1. for (int i = 0; i < n ; i++)
2. for (int j = 0; j < n ; j++)
3. A
4. for (int i = 0; i < n ; i++)
5. for (int j = 0; j < n ; j++)
6. B

critical
sections



■ Code execution

- A $\Rightarrow n^2$ times
- B $\Rightarrow n^2$ times

■ Time $\Rightarrow n^2 + n^2 = O(n^2)$

Critical Section Example 6

■ Code (for input size n)

1. $i = 1$
2. **while** ($i < n$)
3. **A**
4. $i = 2 \times i$
5. **B**

■ Code execution

■ **A** \Rightarrow

■ **B** \Rightarrow

■ Time \Rightarrow

Critical Section Example 6

■ Code (for input size n)

1. $i = 1$

2. while ($i < n$)

3. A

4. $i = 2 \times i$

5. B



**critical
section**

■ Code execution

■ A $\Rightarrow \log(n)$ times

■ B $\Rightarrow 1$ times

■ Time $\Rightarrow \log(n) + 1 = O(\log(n))$

Critical Section Example 7

■ Code (for input size n)

1. **DoWork** (int n)
2. **if** ($n == 1$)
3. **A**
4. **else**
5. **DoWork**($n/2$)
6. **DoWork**($n/2$)

■ Code execution

- **A** \Rightarrow
- **DoWork**($n/2$) \Rightarrow

■ **Time**(1) \Rightarrow **Time**(n) =

Critical Section Example 7

■ Code (for input size n)

1. **DoWork** (int n)

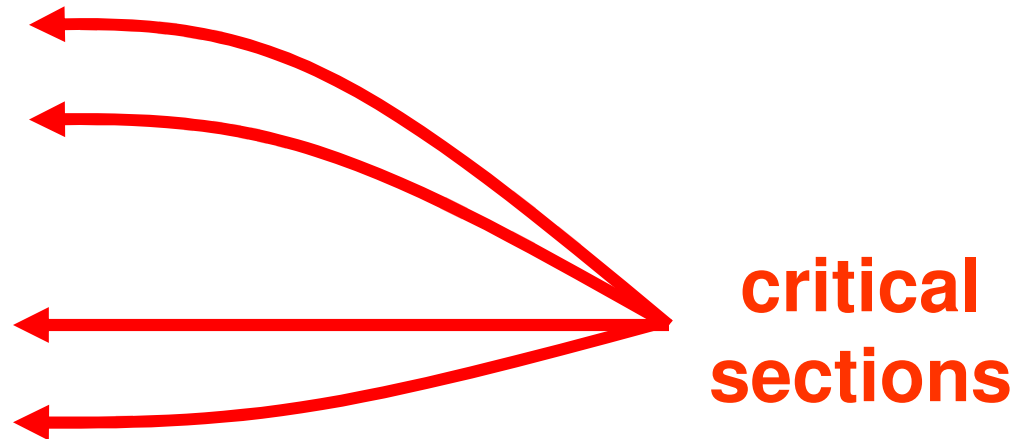
2. **if** ($n == 1$)

3. **A**

4. **else**

5. **DoWork**($n/2$)

6. **DoWork**($n/2$)



■ Code execution

■ **A** \Rightarrow **1** times

■ **DoWork**($n/2$) \Rightarrow **2** times

■ **Time**(1) \Rightarrow **1** **Time**(n) = $2 \times$ **Time**($n/2$) + 1

Asymptotic Complexity Categories

Complexity	Name	Example
■ $O(1)$	Constant	Array access
■ $O(\log(n))$	Logarithmic	Binary search
■ $O(n)$	Linear	Largest element
■ $O(n \log(n))$	N log N	Optimal sort
■ $O(n^2)$	Quadratic	2D Matrix addition
■ $O(n^3)$	Cubic	2D Matrix multiply
■ $O(n^k)$	Polynomial	Linear programming
■ $O(k^n)$	Exponential	Integer programming

From smallest to largest

For size n , constant $k > 1$

Comparing Complexity

- Compare two algorithms

 - $f(n)$, $g(n)$

- Determine which increases at faster rate

 - As problem size n increases

- Can compare ratio

 - If ∞ , $f()$ is larger

 - If 0, $g()$ is larger

 - If constant, then same complexity

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$$

Complexity Comparison Examples

■ $\log(n)$ vs. $n^{1/2}$

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \quad \rightarrow \quad \lim_{n \rightarrow \infty} \frac{\log(n)}{n^{1/2}} \quad \rightarrow \quad 0$$

■ 1.001^n vs. n^{1000}

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \quad \rightarrow \quad \lim_{n \rightarrow \infty} \frac{1.001^n}{n^{1000}} \quad \rightarrow \quad ??$$

Not clear, use
L'Hopital's Rule

L'Hopital's Rule

- If ratio is indeterminate

- 0 / 0 or ∞ / ∞

- Ratio of **derivatives** computes same value

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

- Can simplify ratio by repeatedly taking derivatives of numerator & denominator

Using L'Hopital's Rule

■ 1.001^n vs. n^{1000}

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} &\longrightarrow \lim_{n \rightarrow \infty} \frac{1.001^n}{n^{1000}} \\ &\longrightarrow \lim_{n \rightarrow \infty} \frac{1.001^n \ln(1.001)}{1000 n^{999}} \\ &\longrightarrow \lim_{n \rightarrow \infty} \frac{1.001^n \ln(1.001) \ln(1.001)}{1000 \times 999 n^{998}} \\ &\longrightarrow \dots \longrightarrow \infty \end{aligned}$$

Additional Complexity Measures

■ Upper bound

■ Big-O $\Rightarrow O(\dots)$

■ Represents upper bound on # steps

■ Lower bound

■ Big-Omega $\Rightarrow \Omega(\dots)$

■ Represents lower bound on # steps

■ Combined bound

■ Big-Theta $\Rightarrow \Theta(\dots)$

■ Represents combined upper/lower bound on # steps

■ Best possible asymptotic solution

2D Matrix Multiplication Example

■ Problem

■ $C = A * B$



■ Lower bound

■ $\Omega(n^2)$

Required to examine 2D matrix

■ Upper bounds

■ $O(n^3)$

Basic algorithm

■ $O(n^{2.807})$

Strassen's algorithm (1969)

■ $O(n^{2.376})$

Coppersmith & Winograd (1987)

■ Improvements still possible (open problem)

■ Since upper & lower bounds do not match