TIME AND SPACE COMPLEXITY ANALYSIS

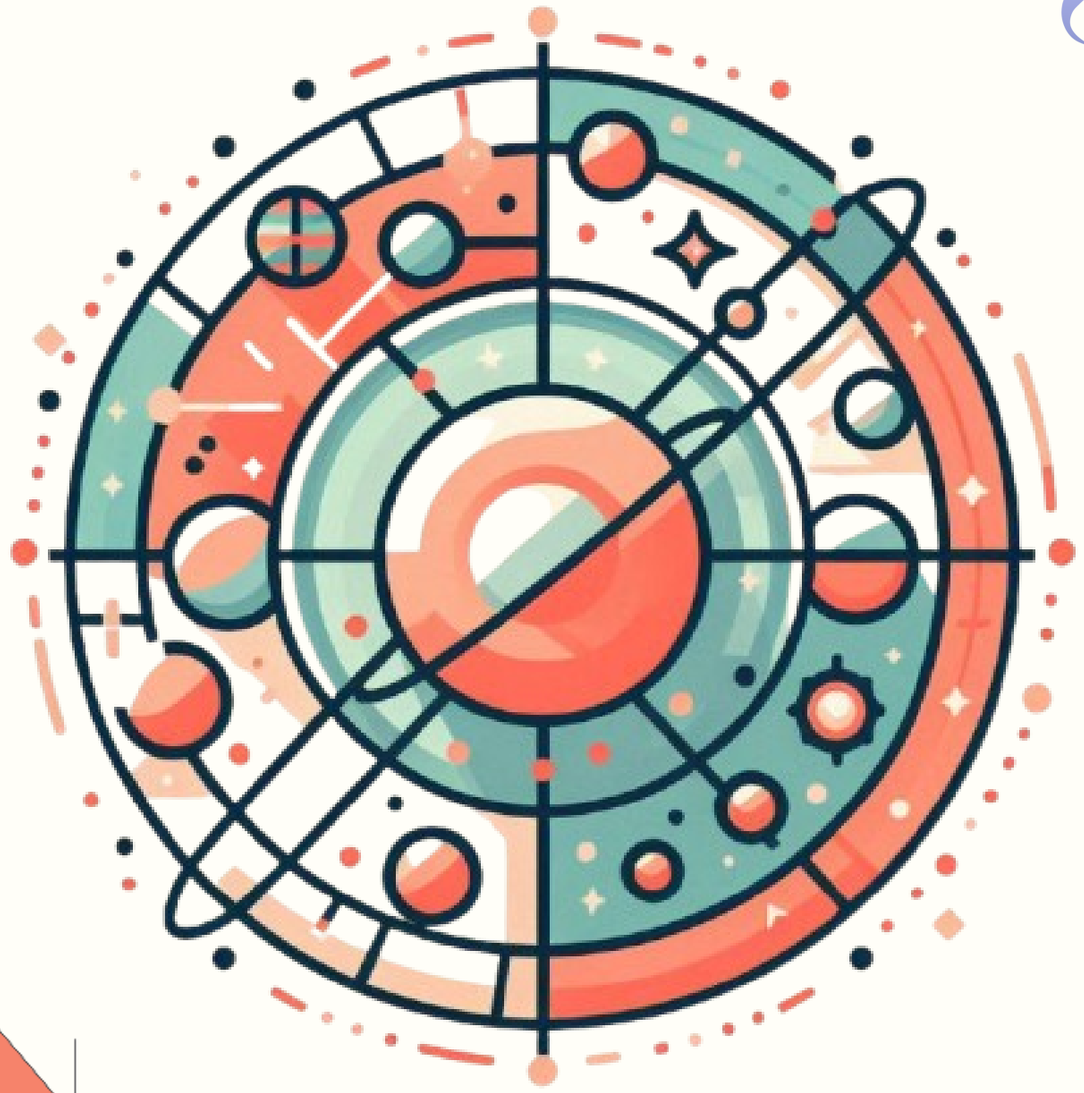


TABLE OF CONTENTS

01

WHAT IS TIME COMPLEXITY?

TIME COMPLEXITY

Measures the time an algorithm takes as input size grows.

Example Code:

```
# Linear time complexity example
for i in range(n):
    print(i)
```

Common Complexity Classes:

Constant - O(1)
Linear - O(n)
Logarithmic - O(log n)
Quadratic - O(n^2)

02

WHAT IS SPACE COMPLEXITY?

SPACE COMPLEXITY

The amount of memory an algorithm uses relative to input size

Example Code:

```
# Space-efficient function example
def sum_list(lst):
    total = 0  # O(1) space
    for num in lst:
        total += num
    return total
```

Common Complexity Classes:

Constant - O(1)
Linear - O(n)
Logarithmic - O(log n)
Quadratic - O(n^2)

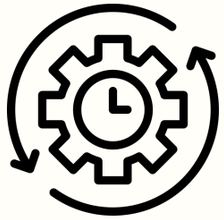
03

COMPARING TIME AND SPACE COMPLEXITIES

TRADE-OFFS:

- Faster algorithms may use more memory (e.g., caching).
- Memory-efficient algorithms may take longer (e.g., recursion).

COMPARING TIME AND SPACE COMPLEXITIES



Time-efficient

```python
def sum_cache(arr):
    return sum(arr)
```

Uses built-in caching

Space-efficient

```python
def recursive_sum(arr):
    if not arr:
        return 0
    return arr[0] + recursive_sum(arr[1:])
```

O(n) stack space

04

REAL-WORLD IMPLICATIONS

TIME COMPLEXITY

Impacts response time in real-time systems.

Examples: Sorting large datasets, database queries.

SPACE COMPLEXITY

Influences storage requirements.

Examples: Handling high-dimensional data in machine learning.

05 SUMMARY

Time complexity measures execution speed; space complexity measures memory usage.

Understanding complexity classes helps evaluate algorithm efficiency.

Balancing time and space trade-offs is key to optimal algorithm selection.

THANK YOU!