

# **Analysis of Recursion**

Recursive algorithms are functions that call themselves to solve smaller instances of the same problem. Analyzing their time complexity involves understanding recurrence relations, which describe how the problem size reduces with each recursive call. Let's break it down with examples.

#### What is a Recursive Function?

A recursive function is a function that calls itself during its execution. For example:

```
6
              console.log("GFG");
              fun(n / 2); // Recursive call
              fun(n / 2); // Recursive call
```

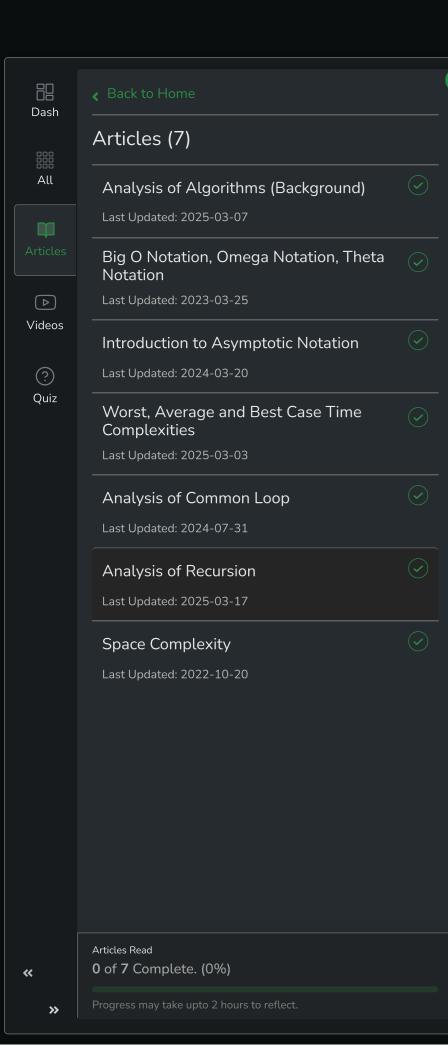
#### **Recurrence Relation**

To analyze recursive algorithms, we use **recurrence relations**, which express the time complexity in terms of smaller inputs.

### **Example 1: Simple Recursion**

```
©
              console.log("GFG");
              fun(n / 2); // Recursive call
              fun(n / 2); // Recursive call
```

- ullet Recurrence Relation:  $T(n)=2T(n/2)+\Theta(1)$ 
  - $\circ$  2T(n/2): Two recursive calls with half the input size.
  - $\circ$   $\Theta(1)$ : Constant work done in each call.



- Base Case:  $T(0)=\Theta(1)$ .
- Time Complexity: Θ(n).

# **Example 2: Recursion with a Loop**

```
JavaScript

1  function fun(n) {
2    if (n == 0) return; // Base case
3    for (let i = 0; i < n; i++) { // Θ(n)}
4         console.log("GFG");
5    }
6    fun(n / 2); // T(n/2)
7    fun(n / 3); // T(n/3)
8 }</pre>
```

- ullet Recurrence Relation:  $T(n) = T(n/2) + T(n/3) + \Theta(n)$
- Base Case:  $T(0)=\Theta(1)$
- Time Complexity: Θ(n)

# **Example 3: Linear Recursion**

```
JavaScript

1  function fun(n) {
2    if (n == 1) return; // Base case
3    console.log("GFG"); // 0(1)
4    fun(n - 1); // T(n-1)
5 }
```

- Recurrence Relation: T(n)=T(n−1)+⊕(1)
- Base Case:  $T(1)=\Theta(1)$
- Time Complexity: Θ(n).

#### **Conclusion**

In this article, we studied how to analyze recursive algorithms using recurrence relations. We explored examples like simple recursion, recursion with loops, and linear recursion, and learned how to express their time complexity. By understanding recurrence relations and base cases, you can determine the efficiency of recursive functions and improve their performance.

