90% Refund Courses
Placement
IBM Data Science
GATE
Practice

Dash
All
Articles
Videos
Quiz

Back to Home

Articles (7)

Analysis of Algorithms (Background) ✓
Last Updated: 2025-03-07

Big O Notation, Omega Notation, Theta Notation ✓
Last Updated: 2023-03-25

Introduction to Asymptotic Notation ✓
Last Updated: 2024-03-20

Worst, Average and Best Case Time Complexities ✓
Last Updated: 2025-03-03

Analysis of Common Loop ✓
Last Updated: 2024-07-31

Analysis of Recursion ✓
Last Updated: 2025-03-17

Space Complexity ✓
Last Updated: 2022-10-20

Articles Read
0 of 7 Complete. (0%)
Progress may take upto 2 hours to reflect.

« Prev
Next »

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:

JavaScript

```javascript
function fun(n) {
    if (n <= 0) return;
    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

JavaScript

```javascript
function fun(n) {
    if (n <= 0) return;
    console.log("GFG");
    fun(n / 2); // Recursive call
    fun(n / 2); // Recursive call
}
```

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

Dash

All

Articles

Videos

Quiz

‹ Back to Home

Articles (7)

Analysis of Algorithms (Background) ✓

Last Updated: 2025-03-07

Big O Notation, Omega Notation, Theta Notation ✓

Last Updated: 2023-03-25

Introduction to Asymptotic Notation ✓

Last Updated: 2024-03-20

Worst, Average and Best Case Time Complexities ✓

Last Updated: 2025-03-03

Analysis of Common Loop ✓

Last Updated: 2024-07-31

Analysis of Recursion ✓

Last Updated: 2025-03-17

Space Complexity ✓

Last Updated: 2022-10-20

Articles Read

0 of 7 Complete. (0%)

Progress may take upto 2 hours to reflect.

«

»

- **Base Case**: T(0)=Θ(1).
- **Time Complexity**: **Θ(n)**.

Example 2: Recursion with a Loop

JavaScript

```javascript
function fun(n) {
    if (n == 0) return; // Base case
    for (let i = 0; i < n; i++) { // Θ(n)
        console.log("GFG");
    }
    fun(n / 2); // T(n/2)
    fun(n / 3); // T(n/3)
}
```

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

Example 3: Linear Recursion

JavaScript

```javascript
function fun(n) {
    if (n == 1) return; // Base case
    console.log("GFG"); // Θ(1)
    fun(n - 1); // T(n-1)
}
```

- **Recurrence Relation**: T(n)=T(n−1)+Θ(1)
- **Base Case**: T(1)=Θ(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.

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.

Dash

All

Articles

Videos

Quiz

‹ Back to Home

Articles (7)

Analysis of Algorithms (Background) ✓

Last Updated: 2025-03-07

Big O Notation, Omega Notation, Theta Notation ✓

Last Updated: 2023-03-25

Introduction to Asymptotic Notation ✓

Last Updated: 2024-03-20

Worst, Average and Best Case Time Complexities ✓

Last Updated: 2025-03-03

Analysis of Common Loop ✓

Last Updated: 2024-07-31

Analysis of Recursion ✓

Last Updated: 2025-03-17

Space Complexity ✓

Last Updated: 2022-10-20

Articles Read

0 of 7 Complete. (0%)

Progress may take upto 2 hours to reflect.

«

»