**1. Time Complexity**

|  |  |  |
| --- | --- | --- |
| **Method** | **Time Complexity** | **Space Complexity** |
| Recursive | **O(n)** | O(n) stack frames |
| Optimized | **O(1)** | O(1) |

**2. Optimization Tips**

* **Avoid deep recursion** for large n to prevent **stack overflow**.
* Use **Math.Pow()** (as shown) for efficient exponential calculation.
* If recursion is needed with overlapping subproblems, use **memoization** or **dynamic programming**.

**3. Sample Output**

=== Financial Forecasting Tool ===

Initial Amount: ₹10000

Growth Rate: 8%

Years: 10

[Recursive] Forecast after 10 years: ₹21589.25

[Optimized] Forecast after 10 years: ₹21589.25

**4. Summary**

* **Recursion** helps solve problems by breaking them into subproblems.
* Use **recursive logic** for understanding, but **optimize** for performance in real apps.
* This exercise models **compound growth**, like in investments, savings, or inflation.