**Recursion**

Recursion is a programming technique where a function calls itself to solve a smaller piece of the original problem, until it reaches a simple base case that can be solved directly.  
  
It **breaks complex problems** into smaller, manageable ones.

It mirrors how we often think logically

Especially useful for **tree**, **graph**, **divide & conquer**, and **combinatorial** problems.  
  
**ANALYSIS  
  
Time Complexity (Recursive Approach):**

Time: O(n)   
Space: O(n)

**Drawback:**  
For large values of n (e.g., 10,000 years), this approach can lead to a stack overflow because of too many recursive calls.

**Optimization Options:**

* **Tail Recursion**: Helps reduce stack usage if the language/compiler supports tail call optimization. Note that Java doesn’t support this natively.
* **Iteration**: A loop-based solution is generally more efficient and avoids stack overflow for large inputs.
* **Memoization**: Not needed in this specific case since there are no overlapping subproblems, but it can greatly improve performance in other recursive problems like Fibonacci or dynamic programming scenarios.