**Leetcode 509 – Fibonacci Number**

## Problem Understanding

The **Fibonacci sequence** is defined as:

* F(0) = 0
* F(1) = 1
* F(n) = F(n - 1) + F(n - 2) for n > 1

Given n, return F(n).

## Optimized Java Solution (Bottom-Up DP)

class Solution {

public int fib(int n) {

if (n <= 1) return n;

int prev = 0, curr = 1;

for (int i = 2; i <= n; i++) {

int next = prev + curr;

prev = curr;

curr = next;

}

return curr;

}

}

### Why it works:

* Avoids redundant calculations.
* Just uses two variables (prev, curr) instead of an array.

## Dry Run Using Table

Let’s dry run with n = 6:

|  |  |  |  |
| --- | --- | --- | --- |
| i | prev | curr | next = prev + curr |
| 2 | 0 | 1 | 1 |
| 3 | 1 | 1 | 2 |
| 4 | 1 | 2 | 3 |
| 5 | 2 | 3 | 5 |
| 6 | 3 | 5 | 8 |

✅ Final result: fib(6) = 8

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(n) |
| Space | O(1) |

Efficient even for large n (up to 30 or 40).

## Alternate Approaches

### 1. ****Top-Down Recursion (Naive)****

public int fib(int n) {

if (n <= 1) return n;

return fib(n - 1) + fib(n - 2);

}

❌ Time: O(2^n) — exponential  
❌ Space: O(n) (recursive call stack)

### 2. ****Top-Down + Memoization (DP)****

Map<Integer, Integer> memo = new HashMap<>();

public int fib(int n) {

if (n <= 1) return n;

if (memo.containsKey(n)) return memo.get(n);

int result = fib(n - 1) + fib(n - 2);

memo.put(n, result);

return result;

}

✅ Time: O(n)  
✅ Space: O(n) (memo + call stack)

### 3. ****Bottom-Up with Array****

public int fib(int n) {

if (n <= 1) return n;

int[] dp = new int[n + 1];

dp[0] = 0; dp[1] = 1;

for (int i = 2; i <= n; i++) {

dp[i] = dp[i - 1] + dp[i - 2];

}

return dp[n];

}

✅ Time: O(n)  
❌ Space: O(n)

### 4. ****Formula (Binet’s Formula – Not Recommended)****

public int fib(int n) {

double phi = (1 + Math.sqrt(5)) / 2;

return (int)(Math.pow(phi, n) / Math.sqrt(5) + 0.5);

}

⚠️ Fast but imprecise due to floating-point errors for large n.

## Recursive Java Solution (Without Memoization)

public class Solution {

public int fib(int n) {

if (n <= 1) return n; // base case

return fib(n - 1) + fib(n - 2); // recursive case

}

}

## Dry Run Using Table

Let’s dry run for n = 4:

fib(4)

├── fib(3)

│ ├── fib(2)

│ │ ├── fib(1) → 1

│ │ └── fib(0) → 0

│ └── fib(1) → 1

├── fib(2)

│ ├── fib(1) → 1

│ └── fib(0) → 0

|  |  |
| --- | --- |
| Call | Result |
| fib(0) | 0 |
| fib(1) | 1 |
| fib(2) | 1 |
| fib(3) | 2 |
| **fib(4)** | **3** |

Final Answer: fib(4) = 3

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | **O(2ⁿ)** |
| Space | **O(n)** |