1. Fibonacci Iteratif

|  |
| --- |
| Fibonacci(n)   1. f1 = 0, f2 = 1, f 2. **while** n > 1 **do** 3. f = f1 + f2 4. f1 = f2 5. f2 = f 6. n = n -1 7. **return** f |

|  |
| --- |
| Fibonacci(A)   1. **if** A = 1 **or** A = 2 2. **return** 1 3. **else** 4. hasil = 0 5. temp1 = 1 6. temp2 = 1 7. **for** i = 3 **to** A 8. hasil = temp1 + temp2 9. temp1 = temp2 10. temp2 = hasil 11. **return** hasil |

1. Fibonacci Rekursif

|  |
| --- |
| Fibonacci(n)   1. if n = 0 2. return 0 3. else if n = 1 4. return 1 5. else 6. return Fibonacci(n-1) + Fibonacci(n-2) |

|  |
| --- |
| Fibo(n)   1. if n = 0 or n = 1 2. return n 3. else 4. return Fibo(n-1) + Fibo(n-2) |

1. Fibonacci DP

|  |
| --- |
| fibo\_memo: array of fibonacci memo, all elements initialized with 0  Fibonacci(n)   1. **if** n = 0 2. **// memoization** 3. fibo\_memo[n] = 0 4. **return** fibo\_memo[n] 5. **else if** n = 1 6. **// memoization** 7. fibo\_memo[n] = 1 8. **return** fibo\_memo[n] 9. **else if** fibo\_memo[n] != 0 10. **// look up** 11. **return** fibo\_memo[n] 12. **else** 13. **// memoization** 14. fibo\_memo[n] = Fibonacci(n-1) + Fibonacci(n-2) 15. **return** fibo\_memo[n] |

1. Two Power of n: Recursive

|  |
| --- |
| Calc(A, n)   1. if n = 0 2. return 1 3. if n = 1 4. return A 5. else 6. return Calc(A, n-1) + Calc(A, n-1) |

|  |
| --- |
| // 2 power of n  Calc(n)   1. if n = 0 2. return 1 3. else 4. return Calc(n-1) + Calc(n-1) |

1. Two Power of n: Dynamic Programming

|  |
| --- |
| power\_memo: array of power memo, all elements initialized by 0  Calc(A, n)   1. if n = 0 2. power\_memo[n] = 1 3. return power\_memo[n] 4. if n = 1 5. power\_memo[n] = A 6. return power\_memo[n] 7. if power\_memo[n] != 0 8. return power\_memo[n] 9. else 10. power\_memo[n] = Calc(A, n-1) + Calc(A, n-1) 11. return power\_memo[n] |

|  |
| --- |
| // 2 power of n  // memo 🡪 array 1-dimensional with size n  Calc(n)   1. **if** n = 0 2. // memoization 3. memo[n] = 1 4. **return** memo[n] 5. // Lookup 6. **if** memo[n] != 0 7. **return** memo[n] 8. **else** 9. // memoization 10. memo[n] = Calc(n-1) + Calc(n-1) 11. **return** memo[n] |

1. Two power of n menggunakan DP Bottom-up

Pola

memo[n] = memo[n-1] \* 2

|  |
| --- |
| TwoPowerOfN\_DPBU1(n)   1. memo[0] = 1 2. for i = 0 to n 3. if memo[n] != 0 4. return memo[n] 5. else 6. memo[i] = memo[i-1] \* 2 7. return memo[n] |

|  |
| --- |
| // power\_memo[0] = 1  // power\_memo[1] = 2  TwoPowerOfN\_DPBU2(n)   1. if n = 0 2. return power\_memo[n] 3. if n = 1 4. return power\_memo[n] 5. else 6. for i = 2 to n 7. power\_memo[i] = power\_memo[i-1] \* 2 8. return power\_memo[n] |

|  |
| --- |
| TwoPowerOfN\_DPBU3(n)   1. **for** i = 0 to n 2. **if** i = 0 3. memo[i] = 1 4. **else** 5. memo[i] = memo[i-1] \* 2 6. **return** memo[n] |