1. Fibonacci: Iterative

|  |
| --- |
| Fibonacci(x)   1. Arr [x+1] 2. Arr [0] = 0 3. **for** i = 1 **to** x 4. **if** i ≤ 2 5. Arr[i] = 1 6. **else** 7. Arr[i] = Arr[i-1] + Arr[i-2] 8. **return** Arr[x] |

1. Fibonacci: Recursive

|  |
| --- |
| Fibonacci-Rec(n)   1. **if** n = 1 **or** n = 2 2. **return** 1 3. **else** 4. **return** Fibonacci-Rec(n-1) + Fibonacci-Rec(n-2) |

1. Fibonacci: DP
   1. Top-Down

|  |
| --- |
|  |

* 1. Bottom-Up

|  |
| --- |
|  |

1. 2-power-of-n

|  |
| --- |
| two\_power(n)   1. if n = 0 2. return 1 3. else 4. return two\_power (n-1) + two\_power (n-1) |

Make a **memoization** and **look up** strategy to speed up the process

|  |
| --- |
| two\_power(n)   1. **if** n = 0 2. // memoization 3. power\_memo[n] = 1 4. **return** power\_memo[n] 5. // look up 6. **if** power\_memo[n] != 0 7. **return** power\_memo[n] 8. **else** 9. // memoization 10. power\_memo[n] = two\_power (n-1) + two\_power (n-1) 11. **return** power\_memo[n] |

DP Bottom-Up

1. Memo
2. Base solution
3. Overlapping solutions

Memo

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| memo | 1 | 2 | 4 | 8 | 16 | 32 | 64 |  |  |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | … |

Base solution

n = 0 🡪 memo[n] = 1

Overlapping solution

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

n = 2 🡪 memo[n] = memo[n-1] \* 2 = 2 \* 2 = 4

n = 3 🡪 memo[n] = memo[n-1] \* 2 = 4 \* 2 = 8

…

n = y 🡪 memo[n] = memo[n-1] \* 2 = … \* 2 = …

Pseudocode

|  |
| --- |
| two\_power\_dp\_bu(n)   1. **if** n = 0 2. power\_mem[n] = 1 3. **return** power\_mem[n] 4. **else** 5. power\_mem[0] = 1 6. **for** i = 1 **to** n 7. power\_mem[i] = power\_mem[i-1] \* 2 8. **return** power\_mem[n] |

|  |
| --- |
| two\_power\_dp\_bu2(n)   1. **if** n = 0 2. memo[n] = 1 3. **return** memo[n] 4. **else** if memo[n] != 0 5. **return** memo[n] 6. **else** 7. **memo[n] = memo[n-1] \* 2** 8. **return** memo[n] |
| two\_power\_dp\_bu3(n)   1. **for** i = 0 **to** n 2. **if** i = 0 3. power[i] = 1 // memoization 4. **else** 5. power[i] = 2 \* power[i-1] // memoization & lookup 6. **return** power[n] |