**Number factors**

**WE'LL COVER THE FOLLOWING**

* + - [Problem Statement](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#problem-statement)
    - [Basic Solution](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#basic-solution)
    - [Top-down Dynamic Programming with Memoization](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#top-down-dynamic-programming-with-memoization)
    - [Bottom-up Dynamic Programming](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#bottom-up-dynamic-programming)
      * [Code](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#code)
    - [Fibonacci number pattern](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#fibonacci-number-pattern)

**Problem Statement**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#problem-statement)

Given a number ‘n’, implement a method to count how many possible ways there are to express ‘n’ as the sum of 1, 3, or 4.

**Example 1:**

n : 4  
Number of ways = 4  
Explanation: Following are the four ways we can exoress 'n' : {1,1,1,1}, {1,3}, {3,1}, {4}

**Example 2:**

n : 5  
Number of ways = 6  
Explanation: Following are the six ways we can express 'n' : {1,1,1,1,1}, {1,1,3}, {1,3,1}, {3,1,1},   
{1,4}, {4,1}

Let’s first start with a recursive brute-force solution.

**Basic Solution**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#basic-solution)

For every number ‘i’, we have three option: subtract either 1, 3, or 4 from ‘i’ and recursively process the remaining number. So our algorithm will look like:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

  public int CountWays(int n) {

    if( n == 0)

      return 1; // base case, we don't need to subtract any thing, so there is only one way

    if(n == 1)

      return 1; // we can subtract 1 to be left with zero, and that is the only way

    if(n == 2)

      return 1; // we can subtract 1 twice to get zero and that is the only way

    if(n == 3)

      return 2; // '3' can be expressed as {1,1,1}, {3}

    // if we subtract 1, we are left with 'n-1'

    int subtract1 = CountWays(n-1);

    // if we subtract 3, we are left with 'n-3'

    int subtract3 = CountWays(n-3);

    // if we subtract 4, we are left with 'n-4'

    int subtract4 = CountWays(n-4);

    return subtract1 + subtract3 + subtract4;

  }

  public static void main(String[] args) {

    ExpressNumber en = new ExpressNumber();

    System.out.println(en.CountWays(4));

    System.out.println(en.CountWays(5));

    System.out.println(en.CountWays(6));

  }

}





RUN

SAVERESET

The time complexity of the above algorithm is exponential O(3^n)*O*(3​*n*​​). The space complexity is O(n)*O*(*n*) which is used to store the recursion stack.

Let’s visually draw the recursion for CountWays(5) to see the overlapping subproblems:

CountWays(6)CountWays(5)CountWays(3)CountWays(2)CountWays(4)CountWays(2)CountWays(1)CountWays(3)CountWays(1)CountWays(0)

Recursion tree for calculating Fibonacci numbers

We can clearly see the overlapping subproblem pattern: CountWays(3), CountWays(2) and CountWays(1) have been called twice. We can optimize this using memoization to store the results for subproblems.

**Top-down Dynamic Programming with Memoization**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#top-down-dynamic-programming-with-memoization)

We can use an array to store the already solved subproblems. Here is the code:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

      return 1; // base case, we don't need to subtract any thing, so there is only one way

    if (n == 1)

      return 1; // we can take subtract 1 to be left with zero, and that is the only way

    if (n == 2)

      return 1; // we can subtract 1 twice to get zero and that is the only way

    if (n == 3)

      return 2; // '3' can be expressed as {1,1,1}, {3}

    if (dp[n] == 0) {

      // if we subtract 1, we are left with 'n-1'

      int subtract1 = CountWaysRecursive(dp, n - 1);

      // if we subtract 3, we are left with 'n-3'

      int subtract3 = CountWaysRecursive(dp, n - 3);

      // if we subtract 4, we are left with 'n-4'

      int subtract4 = CountWaysRecursive(dp, n - 4);

      dp[n] = subtract1 + subtract3 + subtract4;

    }

    return dp[n];

  }

  public static void main(String[] args) {

    ExpressNumber en = new ExpressNumber();

    System.out.println(en.CountWays(4));

    System.out.println(en.CountWays(5));

    System.out.println(en.CountWays(6));

  }

}





RUN

SAVERESET

**Bottom-up Dynamic Programming**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#bottom-up-dynamic-programming)

Let’s try to populate our dp[] array from the above solution, working in a bottom-up fashion. As we saw in the above code, every CountWaysRecursive(n) is the sum of the three counts. We can use this fact to populate our array.

**Code**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#code)

Here is the code for our bottom-up dynamic programming approach:

* [Java](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [JS](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [Python3](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)
* [C++](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376)

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

class ExpressNumber {

  public int CountWays(int n) {

    int dp[] = new int[n+1];

    dp[0] = 1;

    dp[1] = 1;

    dp[2] = 1;

    dp[3] = 2;

    for(int i=4; i<=n; i++)

      dp[i] = dp[i-1] + dp[i-3] + dp[i-4];

    return dp[n];

  }

  public static void main(String[] args) {

    ExpressNumber en = new ExpressNumber();

    System.out.println(en.CountWays(4));

    System.out.println(en.CountWays(5));

    System.out.println(en.CountWays(6));

  }

}





RUN

SAVERESET

The above solution has time and space complexity of O(n)*O*(*n*).

**Fibonacci number pattern**[#](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/NE52PnMY376#fibonacci-number-pattern)

We can clearly see that this problem follows the Fibonacci number pattern. However, every number in a Fibonacci series is the sum of the previous two numbers, whereas in this problem every count is a sum of previous three numbers: previous-1, previous-3, and previous-4. Here is the recursive formula for this problem:

    CountWays(n) = CountWays(n-1) + CountWays(n-3) + CountWays(n-4), for n >= 4

**MARK AS COMPLETED**

[**←    Back**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/YQy7Lx79R0K)

Staircase

[**Next    →**](https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/7nAKN0Qz67r)

Minimum jumps to reach the end

Stuck? Get help on

[**DISCUSS**](https://discuss.educative.io/c/grokking-dynamic-programming-patterns-for-coding-interviews-design-gurus/pattern-4-fibonacci-numbers-number-factors)

Send feedback

6 Recommendations