Problem 2803: Factorial Generator

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Write a generator function that takes an integer

n

as an argument and returns a generator object which yields the

factorial sequence

.

The

factorial sequence

is defined by the relation

n! = n *

(

n-1) * (n-2) * ... * 2 * 1 .

The factorial of 0 is defined as 1.

Example 1:

Input:

n = 5

Output:

[1,2,6,24,120]

Explanation:

const gen = factorial(5) gen.next().value // 1 gen.next().value // 2 gen.next().value // 6 gen.next().value // 24 gen.next().value // 120

Example 2:

Input:

n = 2

Output:

[1,2]

Explanation:

const gen = factorial(2) gen.next().value // 1 gen.next().value // 2

Example 3:

Input:

n = 0

Output:

[1]

Explanation:

const gen = factorial(0) gen.next().value // 1

Constraints:

0 <= n <= 18

Code Snippets

JavaScript:

```
/**
* @param {number} n
* @yields {number}
*/
function* factorial(n) {

};

/**
* const gen = factorial(2);
* gen.next().value; // 1
* gen.next().value; // 2
*/
```

TypeScript:

```
function* factorial(n: number): Generator<number> {

};

/**
* const gen = factorial(2);
* gen.next().value; // 1
* gen.next().value; // 2
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Factorial Generator
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @yields {number}
*/
function* factorial(n) {

};

/**
* const gen = factorial(2);
* gen.next().value; // 1
* gen.next().value; // 2
*/
```

TypeScript Solution:

```
/**
* Problem: Factorial Generator
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function* factorial(n: number): Generator<number> {

};

/**
* const gen = factorial(2);
```

```
* gen.next().value; // 1
* gen.next().value; // 2
*/
```