📓 **Beautiful Arrangement - Documentation**

1. Problem Statement

Given an integer n, return the number of beautiful arrangements that can be constructed using the integers from 1 to n.

A permutation perm of length n is called a beautiful arrangement if for every index i (1-indexed), either:

- perm[i] % i == 0 or
- i % perm[i] == 0

2. Intuition

We are generating permutations of numbers from 1 to n under specific divisibility rules. Instead of generating all n! permutations, we can use backtracking to only explore valid candidates at each position based on the condition, which greatly reduces computation.

3. Key Observations

- The constraint perm[i] % i == 0 or i % perm[i] == 0 acts as a filter during permutation generation.
- We can prune invalid paths early during recursion.
- Using a boolean visited array avoids placing the same number more than once.

4. Approach

- Start from position 1 to n.
- For each position i, try all numbers 1 to n.
- Only place a number at index i if:
  - num hasn't been used (not visited[num]) and
  - num % i == 0 or i % num == 0.

- Recurse to the next position.
- When pos > n, increment the count of valid arrangements.

5. Edge Cases

- n = 1: Only one number → valid by default.
- n = 15: Maximum value, performance still acceptable due to pruning.

6. Complexity Analysis

 Time Complexity

- Worst case: O(n!) without pruning.
- With pruning: Much less than n!, exact depends on divisibility conditions.

 Space Complexity

- O(n) for visited array and recursion stack.

7. Alternative Approaches

- Dynamic Programming with Bitmasking: Store state as a bitmask and use memoization to avoid recomputation.
    - More efficient for large n.
    - More complex to implement.

8. Test Cases

Input	Output	Explanation
n = 1	1	Only one permutation: [1] ✅
n = 2	2	[1,2] and [2,1] both valid ✅
n = 3	3	Valid permutations: [1,2,3], [3,2,1], [2,1,3] ✅

9. Final Thoughts

This problem is a classic example of using backtracking with constraints to avoid unnecessary computation. The core idea is not generating permutations blindly, but making informed choices using divisibility rules. While the brute-force approach is infeasible for n = 15, the optimized backtracking handles it efficiently.