Problem 3437: Permutations III

Problem Information

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

Problem Description

Given an integer

n

, an

alternating permutation

is a permutation of the first

n

positive integers such that no

two

adjacent elements are

both

odd or

both

even.

Return

all such

alternating permutations

sorted in lexicographical order.

Example 1:

Input:

n = 4

Output:

[[1,2,3,4],[1,4,3,2],[2,1,4,3],[2,3,4,1],[3,2,1,4],[3,4,1,2],[4,1,2,3],[4,3,2,1]]

Example 2:

Input:

n = 2

Output:

[[1,2],[2,1]]

Example 3:

Input:

n = 3

Output:

[[1,2,3],[3,2,1]]

Constraints:

1 <= n <= 10

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> permute(int n) {

}
};
```

Java:

```
class Solution {
public int[][] permute(int n) {

}
}
```

Python3:

```
class Solution:
def permute(self, n: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def permute(self, n):
"""
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number[][]}
*/
```

```
var permute = function(n) {

};
```

TypeScript:

```
function permute(n: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] Permute(int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** permute(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func permute(n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun permute(n: Int): Array<IntArray> {

}
```

```
}
```

Swift:

```
class Solution {
func permute(_ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn permute(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[][]}
def permute(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function permute($n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> permute(int n) {

}
}
```

Scala:

```
object Solution {
def permute(n: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec permute(n :: integer) :: [[integer]]
def permute(n) do

end
end
```

Erlang:

```
-spec permute(N :: integer()) -> [[integer()]].
permute(N) ->
.
```

Racket:

```
(define/contract (permute n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Permutations III
```

```
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<vector<int>> permute(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Permutations III
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] permute(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Permutations III
Difficulty: Medium
Tags: array, graph, sort

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def permute(self, n: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def permute(self, n):
"""
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Permutations III
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {number[][]}
*/
var permute = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Permutations III
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function permute(n: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Permutations III
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[][] Permute(int n) {

}
}
```

C Solution:

```
/*
* Problem: Permutations III
* Difficulty: Medium
* Tags: array, graph, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** permute(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Permutations III
// Difficulty: Medium
// Tags: array, graph, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func permute(n int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun permute(n: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func permute(_ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Permutations III
// Difficulty: Medium
// Tags: array, graph, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn permute(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[][]}
def permute(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function permute($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> permute(int n) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def permute(n: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec permute(n :: integer) :: [[integer]]
def permute(n) do

end
end
```

Erlang Solution:

```
-spec permute(N :: integer()) -> [[integer()]].
permute(N) ->
.
```

Racket Solution:

```
(define/contract (permute n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```