Problem 932: Beautiful Array

Problem Information

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

Problem Description

An array

nums

of length

n

is

beautiful

if:

nums

is a permutation of the integers in the range

[1, n]

.

For every

0 <= i < j < n

, there is no index

k

with

i < k < j

where

2 * nums[k] == nums[i] + nums[j]

.

Given the integer

n

, return

any

beautiful

array

nums

of length

n

. There will be at least one valid answer for the given

n

.

Example 1:

Input:

n = 4

Output:

[2,1,4,3]

Example 2:

Input:

n = 5

Output:

[3,1,2,5,4]

Constraints:

1 <= n <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* beautifulArray(int n, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun beautifulArray(n: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
* 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<int> beautifulArray(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
* 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[] beautifulArray(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Beautiful Array
Difficulty: Medium
Tags: array, math

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 beautifulArray(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
```

```
* 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 beautifulArray = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
* 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 beautifulArray(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
* 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[] BeautifulArray(int n) {

}
}
```

C Solution:

```
/*
* Problem: Beautiful Array
* Difficulty: Medium
* Tags: array, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* beautifulArray(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Beautiful Array
// Difficulty: Medium
// Tags: array, math
//
// 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 beautifulArray(n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun beautifulArray(n: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Beautiful Array
// Difficulty: Medium
// Tags: array, math
//
// 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 beautiful_array(n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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