Problem 2438: Range Product Queries of Powers

Problem Information

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

Problem Description

Given a positive integer

n

, there exists a

0-indexed

array called

powers

, composed of the

minimum

number of powers of

2

that sum to

n

. The array is sorted in

non-decreasing

order, and there is

only one

way to form the array.

You are also given a

0-indexed

2D integer array

queries

, where

queries[i] = [left

i

, right

i

]

. Each

queries[i]

represents a query where you have to find the product of all

powers[j]

with

left

i

<= j <= right

i

.

Return

an array

answers

, equal in length to

queries

, where

answers[i]

is the answer to the

i

th

query

. Since the answer to the

i

th

query may be too large, each

answers[i]

should be returned

modulo

10

9

+ 7

.

Example 1:

Input:

n = 15, queries = [[0,1],[2,2],[0,3]]

Output:

[2,4,64]

Explanation:

For n = 15, powers = [1,2,4,8]. It can be shown that powers cannot be a smaller size. Answer to 1st query: powers[0] * powers[1] = 1 * 2 = 2. Answer to 2nd query: powers[2] = 4. Answer to 3rd query: powers[0] * powers[1] * powers[2] * powers[3] = 1 * 2 * 4 * 8 = 64. Each answer modulo 10

9

+ 7 yields the same answer, so [2,4,64] is returned.

Example 2:

Input:

n = 2, queries = [[0,0]]

Output:

[2]

Explanation:

For n = 2, powers = [2]. The answer to the only query is powers[0] = 2. The answer modulo 10

9

+ 7 is the same, so [2] is returned.

Constraints:

1 <= n <= 10

9

1 <= queries.length <= 10

5

0 <= start

i

<= end

i

< powers.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> productQueries(int n, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] productQueries(int n, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function productQueries(n: number, queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] ProductQueries(int n, int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func productQueries(n int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun productQueries(n: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func productQueries(_ n: Int, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn product_queries(n: i32, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[]}
def product_queries(n, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $queries
* @return Integer[]
*/
function productQueries($n, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> productQueries(int n, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def productQueries(n: Int, queries: Array[Array[Int]]): Array[Int] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec product_queries(n :: integer, queries :: [[integer]]) :: [integer]
def product_queries(n, queries) do

end
end
```

Erlang:

```
-spec product_queries(N :: integer(), Queries :: [[integer()]]) ->
[integer()].
product_queries(N, Queries) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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<int> productQueries(int n, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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[] productQueries(int n, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Range Product Queries of Powers
Difficulty: Medium
Tags: array, 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 productQueries(self, n: int, queries: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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
* @param {number[][]} queries
* @return {number[]}
*/
var productQueries = function(n, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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 productQueries(n: number, queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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[] ProductQueries(int n, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Range Product Queries of Powers
* Difficulty: Medium
* Tags: array, 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
*/

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

```
}
```

Go Solution:

```
// Problem: Range Product Queries of Powers
// Difficulty: Medium
// Tags: array, 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 productQueries(n int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun productQueries(n: Int, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func productQueries(_ n: Int, _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Range Product Queries of Powers
// Difficulty: Medium
// Tags: array, 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 product_queries(n: i32, queries: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} queries
# @return {Integer[]}
def product_queries(n, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $queries
* @return Integer[]
*/
function productQueries($n, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> productQueries(int n, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def productQueries(n: Int, queries: Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec product_queries(n :: integer, queries :: [[integer]]) :: [integer]
def product_queries(n, queries) do

end
end
```

Erlang Solution:

```
-spec product_queries(N :: integer(), Queries :: [[integer()]]) ->
[integer()].
product_queries(N, Queries) ->
.
```

Racket Solution:

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