# Problem 713: Subarray Product Less Than K

## Problem Information

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

## Problem Description

Given an array of integers

nums

and an integer

k

, return

the number of contiguous subarrays where the product of all the elements in the subarray is strictly less than

k

.

Example 1:

Input:

nums = [10,5,2,6], k = 100

Output:

8

Explanation:

The 8 subarrays that have product less than 100 are: [10], [5], [2], [6], [10, 5], [5, 2], [2, 6], [5, 2, 6] Note that [10, 5, 2] is not included as the product of 100 is not strictly less than k.

Example 2:

Input:

nums = [1,2,3], k = 0

Output:

0

Constraints:

1 <= nums.length <= 3 * 10

4

1 <= nums[i] <= 1000

0 <= k <= 10

6

## Code Snippets

**C++:**

```
class Solution {
public:
int numSubarrayProductLessThanK(vector<int>& nums, int k) {

}
};
```

**Java:**

```
class Solution {
public int numSubarrayProductLessThanK(int[] nums, int k) {

}
}
```

**Python3:**

```
class Solution:
def numSubarrayProductLessThanK(self, nums: List[int], k: int) -> int:
```

**Python:**

```
class Solution(object):
def numSubarrayProductLessThanK(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} nums
* @param {number} k
* @return {number}
*/
var numSubarrayProductLessThanK = function(nums, k) {

};
```

**TypeScript:**

```
function numSubarrayProductLessThanK(nums: number[], k: number): number {

};
```

**C#:**

```
public class Solution {
public int NumSubarrayProductLessThanK(int[] nums, int k) {

}
}
```

**C:**

```
int numSubarrayProductLessThanK(int* nums, int numsSize, int k) {

}
```

**Go:**

```
func numSubarrayProductLessThanK(nums []int, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun numSubarrayProductLessThanK(nums: IntArray, k: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func numSubarrayProductLessThanK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn num_subarray_product_less_than_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def num_subarray_product_less_than_k(nums, k)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function numSubarrayProductLessThanK($nums, $k) {

}
}
```

**Dart:**

```
class Solution {
int numSubarrayProductLessThanK(List<int> nums, int k) {

}
}
```

**Scala:**

```
object Solution {
def numSubarrayProductLessThanK(nums: Array[Int], k: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec num_subarray_product_less_than_k(nums :: [integer], k :: integer) ::
integer
def num_subarray_product_less_than_k(nums, k) do
```

```
end
end
```

**Erlang:**

```
-spec num_subarray_product_less_than_k(Nums :: [integer()], K :: integer())
-> integer().
num_subarray_product_less_than_k(Nums, K) ->
.
```

**Racket:**

```
(define/contract (num-subarray-product-less-than-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Subarray Product Less Than K
* Difficulty: Medium
* Tags: array, search
*
* 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 numSubarrayProductLessThanK(vector<int>& nums, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Subarray Product Less Than K
* Difficulty: Medium
* Tags: array, search
*
* 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 numSubarrayProductLessThanK(int[] nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Subarray Product Less Than K
Difficulty: Medium
Tags: array, search

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 numSubarrayProductLessThanK(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def numSubarrayProductLessThanK(self, nums, k):
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Subarray Product Less Than K
* Difficulty: Medium
* Tags: array, search
*
* 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[]} nums
* @param {number} k
* @return {number}
*/
var numSubarrayProductLessThanK = function(nums, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Subarray Product Less Than K
* Difficulty: Medium
* Tags: array, search
*
* 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 numSubarrayProductLessThanK(nums: number[], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Subarray Product Less Than K
* Difficulty: Medium
```

```
* Tags: array, search
*
* 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 NumSubarrayProductLessThanK(int[] nums, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Subarray Product Less Than K
* Difficulty: Medium
* Tags: array, search
*
* 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
*/

int numSubarrayProductLessThanK(int* nums, int numsSize, int k) {

}
```

### Go Solution:

```
// Problem: Subarray Product Less Than K
// Difficulty: Medium
// Tags: array, search
//
// 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 numSubarrayProductLessThanK(nums []int, k int) int {
```

```
}
```

### Kotlin Solution:

```
class Solution {
fun numSubarrayProductLessThanK(nums: IntArray, k: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func numSubarrayProductLessThanK(_ nums: [Int], _ k: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Subarray Product Less Than K
// Difficulty: Medium
// Tags: array, search
//
// 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 num_subarray_product_less_than_k(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def num_subarray_product_less_than_k(nums, k)
```

```
end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function numSubarrayProductLessThanK($nums, $k) {

}
}
```

### Dart Solution:

```
class Solution {
int numSubarrayProductLessThanK(List<int> nums, int k) {

}
}
```

### Scala Solution:

```
object Solution {
def numSubarrayProductLessThanK(nums: Array[Int], k: Int): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec num_subarray_product_less_than_k(nums :: [integer], k :: integer) ::
integer
def num_subarray_product_less_than_k(nums, k) do

end
end
```

### Erlang Solution:

```
-spec num_subarray_product_less_than_k(Nums :: [integer()], K :: integer())
-> integer().
num_subarray_product_less_than_k(Nums, K) ->
.
```

### Racket Solution:

```
(define/contract (num-subarray-product-less-than-k nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```