Problem 3584: Maximum Product of First and Last Elements of a Subsequence

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

m

.

Return the

maximum

product of the first and last elements of any

subsequence

of

nums

of size

m

.

Example 1:

Input:

nums = [-1,-9,2,3,-2,-3,1], m = 1

Output:

81

Explanation:

The subsequence

[-9]

has the largest product of the first and last elements:

-9 * -9 = 81

. Therefore, the answer is 81.

Example 2:

Input:

nums = [1,3,-5,5,6,-4], m = 3

Output:

20

Explanation:

The subsequence

[-5, 6, -4]

has the largest product of the first and last elements.

Example 3:

Input:

nums = [2,-1,2,-6,5,2,-5,7], m = 2

Output:

35

Explanation:

The subsequence

[5, 7]

has the largest product of the first and last elements.

Constraints:

1 <= nums.length <= 10

5

-10

5

<= nums[i] <= 10

5

1 <= m <= nums.length

Code Snippets

C++:

```
class Solution {
public:
long long maximumProduct(vector<int>& nums, int m) {

}
};
```

Java:

```
class Solution {
public long maximumProduct(int[] nums, int m) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function maximumProduct(nums: number[], m: number): number {

};
```

C#:

```
public class Solution {
public long MaximumProduct(int[] nums, int m) {

}
}
```

C:

```
long long maximumProduct(int* nums, int numsSize, int m) {

}
```

Go:

```
func maximumProduct(nums []int, m int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumProduct(nums: IntArray, m: Int): Long {

}
}
```

Swift:

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

}
```

```
}
```

Rust:

```
impl Solution {
pub fn maximum_product(nums: Vec<i32>, m: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} m
# @return {Integer}
def maximum_product(nums, m)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $m
* @return Integer
*/
function maximumProduct($nums, $m) {

}
}
```

Dart:

```
class Solution {
int maximumProduct(List<int> nums, int m) {

}
}
```

Scala:

```
object Solution {
def maximumProduct(nums: Array[Int], m: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_product(nums :: [integer], m :: integer) :: integer
def maximum_product(nums, m) do

end
end
```

Erlang:

```
-spec maximum_product(Nums :: [integer()], M :: integer()) -> integer().
maximum_product(Nums, M) ->
.
```

Racket:

```
(define/contract (maximum-product nums m)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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:
long long maximumProduct(vector<int>& nums, int m) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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 long maximumProduct(int[] nums, int m) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of First and Last Elements of a Subsequence
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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} m
* @return {number}
*/
var maximumProduct = function(nums, m) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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 maximumProduct(nums: number[], m: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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 long MaximumProduct(int[] nums, int m) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product of First and Last Elements of a Subsequence
* Difficulty: Medium
* Tags: array
*
* 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
*/

long long maximumProduct(int* nums, int numsSize, int m) {

}
```

Go Solution:

```
// Problem: Maximum Product of First and Last Elements of a Subsequence
// Difficulty: Medium
// Tags: array
//
// 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 maximumProduct(nums []int, m int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumProduct(nums: IntArray, m: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of First and Last Elements of a Subsequence
// Difficulty: Medium
// Tags: array
//
// 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 maximum_product(nums: Vec<i32>, m: i32) -> i64 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} m
# @return {Integer}
def maximum_product(nums, m)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $m
* @return Integer
*/
function maximumProduct($nums, $m) {

}
}
```

Dart Solution:

```
class Solution {
int maximumProduct(List<int> nums, int m) {

}
}
```

Scala Solution:

```
object Solution {
def maximumProduct(nums: Array[Int], m: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_product(nums :: [integer], m :: integer) :: integer
def maximum_product(nums, m) do

end
end
```

Erlang Solution:

```
-spec maximum_product(Nums :: [integer()], M :: integer()) -> integer().
maximum_product(Nums, M) ->
.
```

Racket Solution:

```
(define/contract (maximum-product nums m)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```