Problem 396: Rotate Function

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

Assume

arr

k

to be an array obtained by rotating

nums

by

k

positions clock-wise. We define the

rotation function

F

on

nums

as follow:

F(k) = 0 * arr

k

[0] + 1 * arr

k

[1] + ... + (n - 1) * arr

k

[n - 1].

Return

the maximum value of

F(0), F(1), ..., F(n-1)

.

The test cases are generated so that the answer fits in a

32-bit

integer.

Example 1:

Input:

nums = [4,3,2,6]

Output:

26

Explanation:

F(0) = (0 * 4) + (1 * 3) + (2 * 2) + (3 * 6) = 0 + 3 + 4 + 18 = 25 F(1) = (0 * 6) + (1 * 4) + (2 * 3) + (3 * 2) = 0 + 4 + 6 + 6 = 16 F(2) = (0 * 2) + (1 * 6) + (2 * 4) + (3 * 3) = 0 + 6 + 8 + 9 = 23 F(3) = (0 * 3) + (1 * 2) + (2 * 6) + (3 * 4) = 0 + 2 + 12 + 12 = 26 So the maximum value of F(0), F(1), F(2), F(3) is F(3) = 26.

Example 2:

Input:

nums = [100]

Output:

0

Constraints:

n == nums.length

1 <= n <= 10

5

-100 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxRotateFunction(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int maxRotateFunction(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxRotateFunction(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxRotateFunction(int[] nums) {

}
}
```

C:

```
int maxRotateFunction(int* nums, int numsSize) {

}
```

Go:

```
func maxRotateFunction(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun maxRotateFunction(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxRotateFunction(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_rotate_function(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def max_rotate_function(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxRotateFunction($nums) {

}
}
```

Dart:

```
class Solution {
int maxRotateFunction(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def maxRotateFunction(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_rotate_function(nums :: [integer]) :: integer
def max_rotate_function(nums) do

end
end
```

Erlang:

```
-spec max_rotate_function(Nums :: [integer()]) -> integer().
max_rotate_function(Nums) ->
.
```

Racket:

```
(define/contract (max-rotate-function nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxRotateFunction(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxRotateFunction(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Rotate Function
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @return {number}
*/
var maxRotateFunction = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxRotateFunction(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxRotateFunction(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Rotate Function
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxRotateFunction(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Rotate Function
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func maxRotateFunction(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxRotateFunction(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxRotateFunction(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Rotate Function
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_rotate_function(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def max_rotate_function(nums)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function maxRotateFunction($nums) {

}
}
```

Dart Solution:

```
class Solution {
int maxRotateFunction(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def maxRotateFunction(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_rotate_function(nums :: [integer]) :: integer
def max_rotate_function(nums) do

end
end
```

Erlang Solution:

```
-spec max_rotate_function(Nums :: [integer()]) -> integer().
max_rotate_function(Nums) ->
.
```

Racket Solution:

```
(define/contract (max-rotate-function nums)
(-> (listof exact-integer?) exact-integer?)
)
```