Problem 553: Optimal Division

Problem Information

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

Problem Description

You are given an integer array

nums

. The adjacent integers in

nums

will perform the float division.

For example, for

nums = [2,3,4]

, we will evaluate the expression

"2/3/4"

.

However, you can add any number of parenthesis at any position to change the priority of operations. You want to add these parentheses such the value of the expression after the evaluation is maximum.

Return

the corresponding expression that has the maximum value in string format

.

Note:

your expression should not contain redundant parenthesis.

Example 1:

Input:

nums = [1000,100,10,2]

Output:

"1000/(100/10/2)"

Explanation:

1000/(100/10/2) = 1000/((100/10)/2) = 200 However, the bold parenthesis in "1000/(

(

100/10

)

/2)" are redundant since they do not influence the operation priority. So you should return "1000/(100/10/2)". Other cases: 1000/(100/10)/2 = 50 1000/(100/(10/2)) = 50 1000/100/10/2 = 0.5 1000/100/(10/2) = 2

Example 2:

Input:

nums = [2,3,4]

Output:

"2/(3/4)"

Explanation:

(2/(3/4)) = 8/3 = 2.667 It can be shown that after trying all possibilities, we cannot get an expression with evaluation greater than 2.667

Constraints:

1 <= nums.length <= 10

2 <= nums[i] <= 1000

There is only one optimal division for the given input.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public String optimalDivision(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function optimalDivision(nums: number[]): string {

};
```

C#:

```
public class Solution {
public string OptimalDivision(int[] nums) {

}
}
```

C:

```
char* optimalDivision(int* nums, int numsSize) {

}
```

Go:

```
func optimalDivision(nums []int) string {
}
```

Kotlin:

```
class Solution {
fun optimalDivision(nums: IntArray): String {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
String optimalDivision(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec optimal_division(nums :: [integer]) :: String.t
def optimal_division(nums) do

end
end
```

Erlang:

```
-spec optimal_division(Nums :: [integer()]) -> unicode:unicode_binary().
optimal_division(Nums) ->
.
```

Racket:

```
(define/contract (optimal-division nums)
(-> (listof exact-integer?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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:
string optimalDivision(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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 String optimalDivision(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Optimal Division
Difficulty: Medium
Tags: array, string, 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 optimalDivision(self, nums: List[int]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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 {string}
*/
var optimalDivision = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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 optimalDivision(nums: number[]): string {

};
```

C# Solution:

```
/*
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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 string OptimalDivision(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Optimal Division
* Difficulty: Medium
* Tags: array, string, 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
*/

char* optimalDivision(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Optimal Division
// Difficulty: Medium
// Tags: array, string, 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 optimalDivision(nums []int) string {

}
```

Kotlin Solution:

```
class Solution {
fun optimalDivision(nums: IntArray): String {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Optimal Division
// Difficulty: Medium
// Tags: array, string, 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 optimal_division(nums: Vec<i32>) -> String {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
String optimalDivision(List<int> nums) {
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec optimal_division(nums :: [integer]) :: String.t
def optimal_division(nums) do

end
end
```

Erlang Solution:

```
-spec optimal_division(Nums :: [integer()]) -> unicode:unicode_binary().
optimal_division(Nums) ->
.
```

Racket Solution:

```
(define/contract (optimal-division nums)
(-> (listof exact-integer?) string?)
)
```