Problem 2221: Find Triangular Sum of an Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

, where

nums[i]

is a digit between

0

and

9

(

inclusive

).

The

triangular sum

of

nums

is the value of the only element present in

nums

after the following process terminates:

Let

nums

comprise of

n

elements. If

n == 1

,

end

the process. Otherwise,

create

a new

0-indexed

integer array

newNums

of length

n - 1

.

For each index

i

, where

0 <= i < n - 1

,

assign

the value of

newNums[i]

as

(nums[i] + nums[i+1]) % 10

, where

%

denotes modulo operator.

Replace

the array

nums

with

newNums

.

Repeat

the entire process starting from step 1.

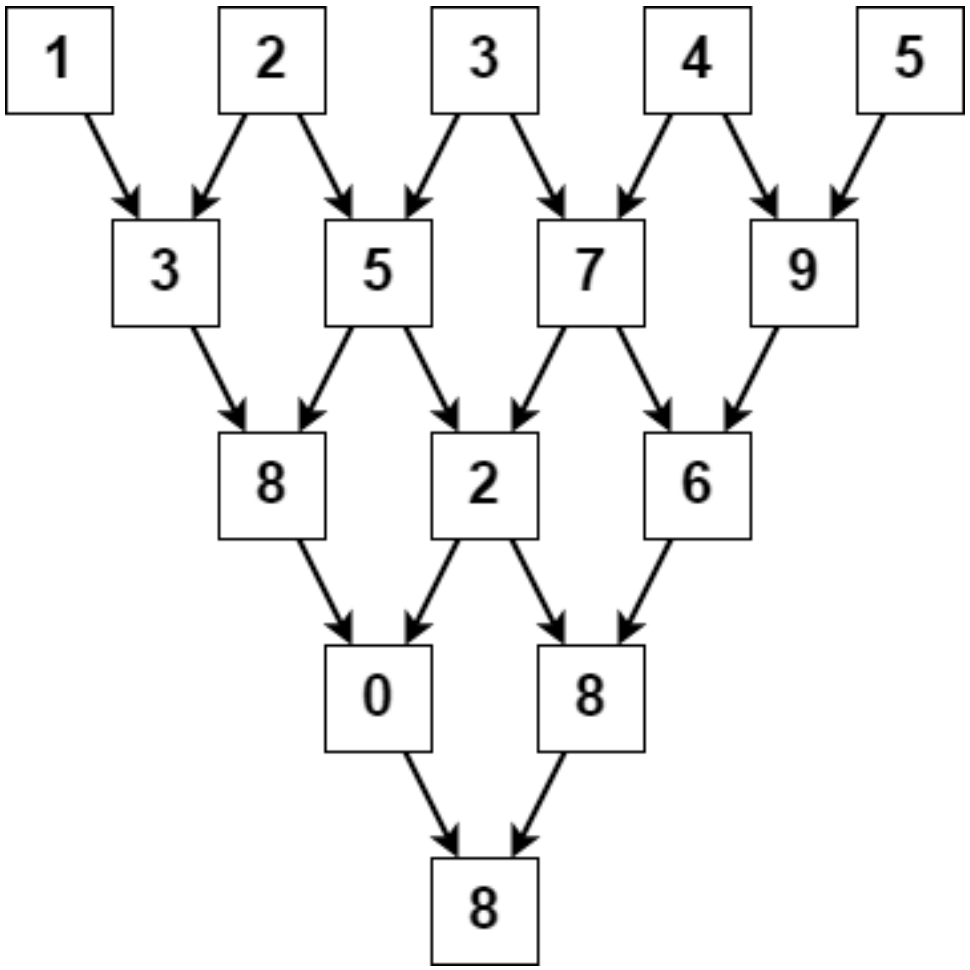
Return

the triangular sum of

nums

.

Example 1:



Input:

nums = [1,2,3,4,5]

Output:

8

Explanation:

The above diagram depicts the process from which we obtain the triangular sum of the array.

Example 2:

Input:

nums = [5]

Output:

5

Explanation:

Since there is only one element in nums, the triangular sum is the value of that element itself.

Constraints:

1 <= nums.length <= 1000

0 <= nums[i] <= 9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (triangular-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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 triangularSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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 triangularSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Triangular Sum of an Array
Difficulty: Medium
Tags: array, math
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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
* @return {number}
*/
var triangularSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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 triangularSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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 TriangularSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find Triangular Sum of an Array
* Difficulty: Medium
* Tags: array, math
*
* 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 triangularSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find Triangular Sum of an Array
// Difficulty: Medium
// Tags: array, math
//
// 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 triangularSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Triangular Sum of an Array
// Difficulty: Medium
// Tags: array, math
```

```
//
// 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 triangular_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (triangular-sum nums)
(-> (listof exact-integer?) exact-integer?)
)
```