Problem 1480: Running Sum of 1d Array

Problem Information

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

Problem Description

Given an array

nums

. We define a running sum of an array as

runningSum[i] = sum(nums[0]…nums[i])

.

Return the running sum of

nums

.

Example 1:

Input:

nums = [1,2,3,4]

Output:

[1,3,6,10]

Explanation:

Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].

Example 2:

Input:

nums = [1,1,1,1,1]

Output:

[1,2,3,4,5]

Explanation:

Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1].

Example 3:

Input:

nums = [3,1,2,10,1]

Output:

[3,4,6,16,17]

Constraints:

1 <= nums.length <= 1000

-10^6 <= nums[i] <= 10^6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* runningSum(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun runningSum(nums: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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:
vector<int> runningSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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 int[] runningSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Running Sum of 1d Array
Difficulty: Easy
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 runningSum(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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
* @return {number[]}
*/
var runningSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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 runningSum(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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 int[] RunningSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Running Sum of 1d Array
* Difficulty: Easy
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* runningSum(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Running Sum of 1d Array
// Difficulty: Easy
// 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 runningSum(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun runningSum(nums: IntArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Running Sum of 1d Array
// Difficulty: Easy
// 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 running_sum(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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