Problem 1313: Decompress Run-Length Encoded List

Problem Information

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

Problem Description

We are given a list

nums

of integers representing a list compressed with run-length encoding.

Consider each adjacent pair of elements

[freq, val] = [nums[2*i], nums[2*i+1]]

(with

i >= 0

). For each such pair, there are

freq

elements with value

val

concatenated in a sublist. Concatenate all the sublists from left to right to generate the decompressed list.

Return the decompressed list.

Example 1:

Input:

nums = [1,2,3,4]

Output:

[2,4,4,4]

Explanation:

The first pair [1,2] means we have freq = 1 and val = 2 so we generate the array [2]. The second pair [3,4] means we have freq = 3 and val = 4 so we generate [4,4,4]. At the end the concatenation [2] + [4,4,4] is [2,4,4,4].

Example 2:

Input:

nums = [1,1,2,3]

Output:

[1,3,3]

Constraints:

2 <= nums.length <= 100

nums.length % 2 == 0

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (decompress-rl-elist nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Decompress Run-Length Encoded List
* 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> decompressRLElist(vector<int>& nums) {
}
};
```

Java Solution:

```
/**
* Problem: Decompress Run-Length Encoded List
* 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[] decompressRLElist(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Decompress Run-Length Encoded List
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 decompressRLElist(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Decompress Run-Length Encoded List
* 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 decompressRLElist = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Decompress Run-Length Encoded List
* 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 decompressRLElist(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Decompress Run-Length Encoded List
* 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[] DecompressRLElist(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Decompress Run-Length Encoded List
* 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* decompressRLElist(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Decompress Run-Length Encoded List
// 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 decompressRLElist(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Decompress Run-Length Encoded List
// 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 decompress_rl_elist(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (decompress-rl-elist nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```