Problem 3149: Find the Minimum Cost Array Permutation

Problem Information

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

Problem Description

You are given an array

nums

which is a

permutation

of

[0, 1, 2, ..., n - 1]

. The

score

of any permutation of

[0, 1, 2, ..., n - 1]

named

perm

is defined as:

score(perm) = |perm[0] - nums[perm[1]]| + |perm[1] - nums[perm[2]]| + ... + |perm[n - 1] - nums[perm[0]]|

Return the permutation

perm

which has the

minimum

possible score. If

multiple

permutations exist with this score, return the one that is

lexicographically smallest

among them.

Example 1:

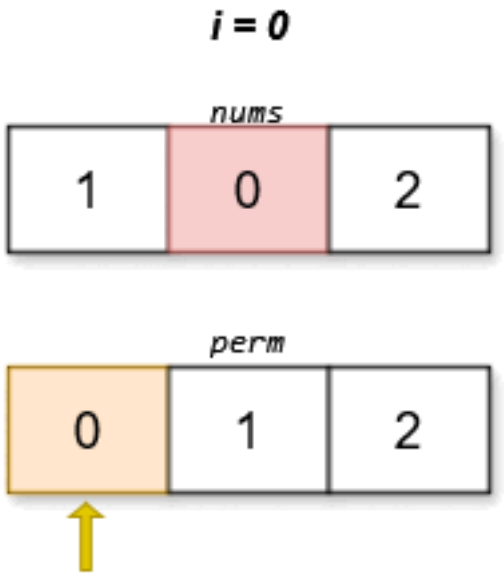
Input:

nums = [1,0,2]

Output:

[0,1,2]

Explanation:



The lexicographically smallest permutation with minimum cost is

[0,1,2]

. The cost of this permutation is

|0 - 0| + |1 - 2| + |2 - 1| = 2

.

Example 2:

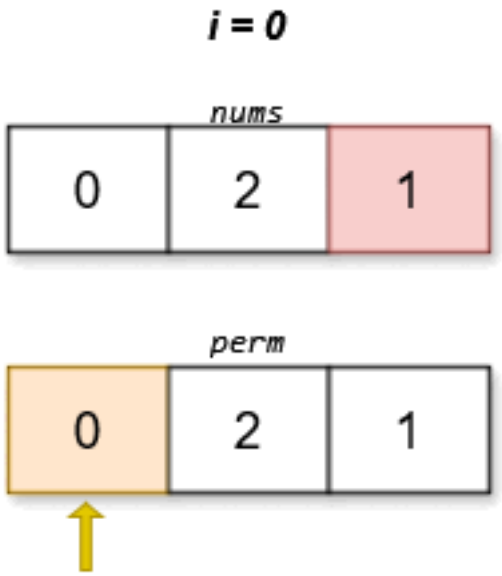
Input:

nums = [0,2,1]

Output:

[0,2,1]

Explanation:



The lexicographically smallest permutation with minimum cost is

[0,2,1]

. The cost of this permutation is

|0 - 1| + |2 - 2| + |1 - 0| = 2

.

Constraints:

2 <= n == nums.length <= 14

nums

is a permutation of

[0, 1, 2, ..., n - 1]

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-permutation nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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:
vector<int> findPermutation(vector<int>& nums) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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[] findPermutation(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Minimum Cost Array Permutation
Difficulty: Hard
Tags: array, graph, dp

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 findPermutation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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 findPermutation(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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[] FindPermutation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Minimum Cost Array Permutation
* Difficulty: Hard
* Tags: array, graph, dp
*
* 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
*/

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

}
```

Go Solution:

```
// Problem: Find the Minimum Cost Array Permutation
// Difficulty: Hard
// Tags: array, graph, dp
```

```
//
// 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 findPermutation(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Minimum Cost Array Permutation
// Difficulty: Hard
// Tags: array, graph, dp
//
// 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 find_permutation(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-permutation nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```