Problem 969: Pancake Sorting

Problem Information

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

Problem Description

Given an array of integers

arr

, sort the array by performing a series of

pancake flips

.

In one pancake flip we do the following steps:

Choose an integer

k

where

1 <= k <= arr.length

.

Reverse the sub-array

arr[0...k-1]

(

0-indexed

).

For example, if

arr = [3,2,1,4]

and we performed a pancake flip choosing

k = 3

, we reverse the sub-array

[3,2,1]

, so

arr = [

1

,

2

,

3

,4]

after the pancake flip at

k = 3

.

Return

an array of the

k

-values corresponding to a sequence of pancake flips that sort

arr

. Any valid answer that sorts the array within

10 * arr.length

flips will be judged as correct.

Example 1:

Input:

arr = [3,2,4,1]

Output:

[4,2,4,3]

Explanation:

We perform 4 pancake flips, with k values 4, 2, 4, and 3. Starting state: arr = [3, 2, 4, 1] After 1st flip (k = 4): arr = [

1

,

4

,

2

,

3

] After 2nd flip (k = 2): arr = [

4

,

1

, 2, 3] After 3rd flip (k = 4): arr = [

3

,

2

,

1

,

4

] After 4th flip (k = 3): arr = [

1

,

2

,

3

, 4], which is sorted.

Example 2:

Input:

arr = [1,2,3]

Output:

[]

Explanation:

The input is already sorted, so there is no need to flip anything. Note that other answers, such as [3, 3], would also be accepted.

Constraints:

1 <= arr.length <= 100

1 <= arr[i] <= arr.length

All integers in

arr

are unique (i.e.

arr

is a permutation of the integers from

1

to

arr.length

).

Code Snippets

C++:

```
class Solution {
public:
vector<int> pancakeSort(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public List<Integer> pancakeSort(int[] arr) {

}
}
```

Python3:

```
class Solution:
def pancakeSort(self, arr: List[int]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function pancakeSort(arr: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> PancakeSort(int[] arr) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* pancakeSort(int* arr, int arrSize, int* returnSize) {

}
```

Go:

```
func pancakeSort(arr []int) []int {

}
```

Kotlin:

```
class Solution {
fun pancakeSort(arr: IntArray): List<Int> {

}

}
```

Swift:

```
class Solution {
func pancakeSort(_ arr: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn pancake_sort(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer[]}
def pancake_sort(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function pancakeSort($arr) {

}
}
```

Dart:

```
class Solution {
List<int> pancakeSort(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def pancakeSort(arr: Array[Int]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec pancake_sort(arr :: [integer]) :: [integer]
def pancake_sort(arr) do

end
end
```

Erlang:

```
-spec pancake_sort(Arr :: [integer()]) -> [integer()].
pancake_sort(Arr) ->
.
```

Racket:

```
(define/contract (pancake-sort arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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> pancakeSort(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 List<Integer> pancakeSort(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Pancake Sorting
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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[]} arr
* @return {number[]}
*/
var pancakeSort = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 pancakeSort(arr: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 IList<int> PancakeSort(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Pancake Sorting
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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* pancakeSort(int* arr, int arrSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Pancake Sorting
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 pancakeSort(arr []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun pancakeSort(arr: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func pancakeSort(_ arr: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Pancake Sorting
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 pancake_sort(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer[]}
def pancake_sort(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function pancakeSort($arr) {

}
}
```

Dart Solution:

```
class Solution {
List<int> pancakeSort(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def pancakeSort(arr: Array[Int]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pancake_sort(arr :: [integer]) :: [integer]
def pancake_sort(arr) do

end
end
```

Erlang Solution:

```
-spec pancake_sort(Arr :: [integer()]) -> [integer()].
pancake_sort(Arr) ->
.
```

Racket Solution:

```
(define/contract (pancake-sort arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```