Problem 3237: Alt and Tab Simulation

Problem Information

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

Problem Description

There are

n

windows open numbered from

1

to

n

, we want to simulate using alt + tab to navigate between the windows.

You are given an array

windows

which contains the initial order of the windows (the first element is at the top and the last one is at the bottom).

You are also given an array

queries

where for each query, the window

queries[i]

is brought to the top.

Return the final state of the array

windows

.

Example 1:

Input:

windows = [1,2,3], queries = [3,3,2]

Output:

[2,3,1]

Explanation:

Here is the window array after each query:

Initial order:

[1,2,3]

After the first query:

[

3

,1,2]

After the second query:

[

3

,1,2]

After the last query:

[

2

,3,1]

Example 2:

Input:

windows = [1,4,2,3], queries = [4,1,3]

Output:

[3,1,4,2]

Explanation:

Here is the window array after each query:

Initial order:

[1,4,2,3]

After the first query:

[

4

,1,2,3]

After the second query:

[

1

,4,2,3]

After the last query:

[

3

,1,4,2]

Constraints:

1 <= n == windows.length <= 10

5

windows

is a permutation of

[1, n]

.

1 <= queries.length <= 10

5

1 <= queries[i] <= n

Code Snippets

C++:

```
class Solution {
public:
vector<int> simulationResult(vector<int>& windows, vector<int>& queries) {

}
};
```

Java:

```
class Solution {
public int[] simulationResult(int[] windows, int[] queries) {

}
}
```

Python3:

```
class Solution:
def simulationResult(self, windows: List[int], queries: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
def simulationResult(self, windows, queries):
"""
:type windows: List[int]
:type queries: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} windows
* @param {number[]} queries
* @return {number[]}
*/
var simulationResult = function(windows, queries) {

};
```

TypeScript:

```
function simulationResult(windows: number[], queries: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] SimulationResult(int[] windows, int[] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* simulationResult(int* windows, int windowsSize, int* queries, int
queriesSize, int* returnSize) {

}
```

Go:

```
func simulationResult(windows []int, queries []int) []int {

}
```

Kotlin:

```
class Solution {
fun simulationResult(windows: IntArray, queries: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func simulationResult(_ windows: [Int], _ queries: [Int]) -> [Int] {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn simulation_result(windows: Vec<i32>, queries: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} windows
# @param {Integer[]} queries
# @return {Integer[]}
def simulation_result(windows, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $windows
* @param Integer[] $queries
* @return Integer[]
*/
function simulationResult($windows, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> simulationResult(List<int> windows, List<int> queries) {

}
}
```

Scala:

```
object Solution {
def simulationResult(windows: Array[Int], queries: Array[Int]): Array[Int] =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec simulation_result(windows :: [integer], queries :: [integer]) ::
[integer]
def simulation_result(windows, queries) do

end
end
```

Erlang:

```
-spec simulation_result(Windows :: [integer()], Queries :: [integer()]) ->
[integer()].
simulation_result(Windows, Queries) ->
.
```

Racket:

```
(define/contract (simulation-result windows queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Alt and Tab Simulation
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> simulationResult(vector<int>& windows, vector<int>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Alt and Tab Simulation
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] simulationResult(int[] windows, int[] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Alt and Tab Simulation
Difficulty: Medium
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""
```

```
class Solution:
def simulationResult(self, windows: List[int], queries: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def simulationResult(self, windows, queries):
"""
:type windows: List[int]
:type queries: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Alt and Tab Simulation
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} windows
* @param {number[]} queries
* @return {number[]}
*/
var simulationResult = function(windows, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Alt and Tab Simulation
```

```
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function simulationResult(windows: number[], queries: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Alt and Tab Simulation
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] SimulationResult(int[] windows, int[] queries) {

}
}
```

C Solution:

```
/*
* Problem: Alt and Tab Simulation
* Difficulty: Medium
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* simulationResult(int* windows, int windowsSize, int* queries, int
queriesSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Alt and Tab Simulation
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func simulationResult(windows []int, queries []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun simulationResult(windows: IntArray, queries: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func simulationResult(_ windows: [Int], _ queries: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Alt and Tab Simulation
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn simulation_result(windows: Vec<i32>, queries: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} windows
# @param {Integer[]} queries
# @return {Integer[]}
def simulation_result(windows, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $windows
* @param Integer[] $queries
* @return Integer[]
*/
function simulationResult($windows, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> simulationResult(List<int> windows, List<int> queries) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def simulationResult(windows: Array[Int], queries: Array[Int]): Array[Int] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec simulation_result(windows :: [integer], queries :: [integer]) ::
[integer]
def simulation_result(windows, queries) do

end
end
```

Erlang Solution:

```
-spec simulation_result(Windows :: [integer()], Queries :: [integer()]) ->
[integer()].
simulation_result(Windows, Queries) ->
.
```

Racket Solution:

```
(define/contract (simulation-result windows queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```