Problem 491: Non-decreasing Subsequences

Problem Information

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

Problem Description

Given an integer array

nums

, return

all the different possible non-decreasing subsequences of the given array with at least two elements

. You may return the answer in

any order

.

Example 1:

Input:

nums = [4,6,7,7]

Output:

[[4,6],[4,6,7],[4,6,7,7],[4,7],[4,7,7],[6,7],[6,7,7],[7,7]]

Example 2:

Input:

nums = [4,4,3,2,1]

Output:

[[4,4]]

Constraints:

1 <= nums.length <= 15

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> findSubsequences(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> FindSubsequences(int[] nums) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findSubsequences(int* nums, int numsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun findSubsequences(nums: IntArray): List<List<Int>> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Non-decreasing Subsequences
* 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<vector<int>> findSubsequences(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Non-decreasing Subsequences
* 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 List<List<Integer>> findSubsequences(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Non-decreasing Subsequences
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 findSubsequences(self, nums: List[int]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Non-decreasing Subsequences
* 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[]} nums
* @return {number[][]}
*/
var findSubsequences = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Non-decreasing Subsequences
* 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 findSubsequences(nums: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Non-decreasing Subsequences
* 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 IList<IList<int>> FindSubsequences(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Non-decreasing Subsequences
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** findSubsequences(int* nums, int numsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Non-decreasing Subsequences
// 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 findSubsequences(nums []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun findSubsequences(nums: IntArray): List<List<Int>> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Non-decreasing Subsequences
// 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 find_subsequences(nums: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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