Problem 368: Largest Divisible Subset

Problem Information

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

Problem Description

Given a set of

distinct

positive integers

nums

, return the largest subset

answer

such that every pair

(answer[i], answer[j])

of elements in this subset satisfies:

answer[i] % answer[j] == 0

, or

answer[j] % answer[i] == 0

If there are multiple solutions, return any of them.

Example 1:

Input:

nums = [1,2,3]

Output:

[1,2]

Explanation:

[1,3] is also accepted.

Example 2:

Input:

nums = [1,2,4,8]

Output:

[1,2,4,8]

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 2 * 10

9

All the integers in

nums

are

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
```

```
var largestDivisibleSubset = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (largest-divisible-subset nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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> largestDivisibleSubset(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 List<Integer> largestDivisibleSubset(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Divisible Subset
Difficulty: Medium
Tags: array, dp, math, sort

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 largestDivisibleSubset = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 largestDivisibleSubset(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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 IList<int> LargestDivisibleSubset(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Largest Divisible Subset
* Difficulty: Medium
* Tags: array, dp, math, sort
*
* 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* largestDivisibleSubset(int* nums, int numsSize, int* returnSize) {
}
```

Go Solution:

```
// Problem: Largest Divisible Subset
// Difficulty: Medium
// Tags: array, dp, math, sort
//
// 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 largestDivisibleSubset(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Divisible Subset
// Difficulty: Medium
// Tags: array, dp, math, sort
//
// 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 largest_divisible_subset(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (largest-divisible-subset nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```