

# Problem 2389: Longest Subsequence With Limited Sum

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given an integer array

`nums`

of length

`n`

, and an integer array

`queries`

of length

`m`

.

Return

an array

answer

of length

m

where

answer[i]

is the

maximum

size of a

subsequence

that you can take from

nums

such that the

sum

of its elements is less than or equal to

queries[i]

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [4,5,2,1], queries = [3,10,21]

Output:

[2,3,4]

Explanation:

We answer the queries as follows: - The subsequence [2,1] has a sum less than or equal to 3. It can be proven that 2 is the maximum size of such a subsequence, so  $\text{answer}[0] = 2$ . - The subsequence [4,5,1] has a sum less than or equal to 10. It can be proven that 3 is the maximum size of such a subsequence, so  $\text{answer}[1] = 3$ . - The subsequence [4,5,2,1] has a sum less than or equal to 21. It can be proven that 4 is the maximum size of such a subsequence, so  $\text{answer}[2] = 4$ .

Example 2:

Input:

nums = [2,3,4,5], queries = [1]

Output:

[0]

Explanation:

The empty subsequence is the only subsequence that has a sum less than or equal to 1, so  $\text{answer}[0] = 0$ .

Constraints:

$n == \text{nums.length}$

$m == \text{queries.length}$

$1 \leq n, m \leq 1000$

$1 \leq \text{nums}[i], \text{queries}[i] \leq 10$

## Code Snippets

### C++:

```
class Solution {  
public:  
    vector<int> answerQueries(vector<int>& nums, vector<int>& queries) {  
  
    }  
};
```

### Java:

```
class Solution {  
    public int[] answerQueries(int[] nums, int[] queries) {  
  
    }  
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

```

* @return {number[]}
*/
var answerQueries = function(nums, queries) {

};

```

### TypeScript:

```

function answerQueries(nums: number[], queries: number[]): number[] {

};

```

### C#:

```

public class Solution {
    public int[] AnswerQueries(int[] nums, int[] queries) {

    }
}

```

### C:

```

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

}

```

### Go:

```

func answerQueries(nums []int, queries []int) []int {

}

```

### Kotlin:

```

class Solution {
    fun answerQueries(nums: IntArray, queries: IntArray): IntArray {

    }
}

```

```
}
```

### Swift:

```
class Solution {  
    func answerQueries(_ nums: [Int], _ queries: [Int]) -> [Int] {  
  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn answer_queries(nums: Vec<i32>, queries: Vec<i32>) -> Vec<i32> {  
  
    }  
}
```

### Ruby:

```
# @param {Integer[]} nums  
# @param {Integer[]} queries  
# @return {Integer[]}  
def answer_queries(nums, queries)  
  
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer[] $queries  
     * @return Integer[]  
     */  
    function answerQueries($nums, $queries) {  
  
    }  
}
```

### Dart:

```

class Solution {
    List<int> answerQueries(List<int> nums, List<int> queries) {

    }

}

```

### Scala:

```

object Solution {
    def answerQueries(nums: Array[Int], queries: Array[Int]): Array[Int] = {

    }

}

```

### Elixir:

```

defmodule Solution do
  @spec answer_queries(nums :: [integer], queries :: [integer]) :: [integer]
  def answer_queries(nums, queries) do

  end

end

```

### Erlang:

```

-spec answer_queries(Nums :: [integer()], Queries :: [integer()]) ->
    [integer()].
answer_queries(Nums, Queries) ->
    .

```

### Racket:

```

(define/contract (answer-queries nums queries)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
  )

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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> answerQueries(vector<int>& nums, vector<int>& queries) {

    }
};

```

### Java Solution:

```

/**
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 int[] answerQueries(int[] nums, int[] queries) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Longest Subsequence With Limited Sum
Difficulty: Easy
Tags: array, greedy, sort, search

```

```

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

```

### Python Solution:

```

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

```

### JavaScript Solution:

```

/**
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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[]} nums
 * @param {number[]} queries
 * @return {number[]}
 */
var answerQueries = function(nums, queries) {

};

```

### TypeScript Solution:

```
/**
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 answerQueries(nums: number[], queries: number[]): number[] {

};
```

### C# Solution:

```
/*
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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 int[] AnswerQueries(int[] nums, int[] queries) {

    }
}
```

### C Solution:

```
/*
 * Problem: Longest Subsequence With Limited Sum
 * Difficulty: Easy
 * Tags: array, greedy, sort, search
 *
 * 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* answerQueries(int* nums, int numsSize, int* queries, int queriesSize,
int* returnSize) {

}

```

### Go Solution:

```

// Problem: Longest Subsequence With Limited Sum
// Difficulty: Easy
// Tags: array, greedy, sort, search
//
// 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 answerQueries(nums []int, queries []int) []int {

}

```

### Kotlin Solution:

```

class Solution {
    fun answerQueries(nums: IntArray, queries: IntArray): IntArray {

    }
}

```

### Swift Solution:

```

class Solution {
    func answerQueries(_ nums: [Int], _ queries: [Int]) -> [Int] {

    }
}

```

### Rust Solution:

```
// Problem: Longest Subsequence With Limited Sum
// Difficulty: Easy
// Tags: array, greedy, sort, search
//
// 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 answer_queries(nums: Vec<i32>, queries: Vec<i32>) -> Vec<i32> {

    }
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} queries
# @return {Integer[]}
def answer_queries(nums, queries)

end
```

### PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer[] $queries
     * @return Integer[]
     */
    function answerQueries($nums, $queries) {

    }

}
```

### Dart Solution:

```
class Solution {
  List<int> answerQueries(List<int> nums, List<int> queries) {

  }
}
```

### Scala Solution:

```
object Solution {
  def answerQueries(nums: Array[Int], queries: Array[Int]): Array[Int] = {

  }
}
```

### Elixir Solution:

```
defmodule Solution do
  @spec answer_queries(nums :: [integer], queries :: [integer]) :: [integer]
  def answer_queries(nums, queries) do

  end
end
```

### Erlang Solution:

```
-spec answer_queries(Nums :: [integer()], Queries :: [integer()]) ->
[integer()].
answer_queries(Nums, Queries) ->
.
```

### Racket Solution:

```
(define/contract (answer-queries nums queries)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```