

# Problem 1403: Minimum Subsequence in Non-Increasing Order

## Problem Information

Difficulty: Easy

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given the array

`nums`

, obtain a subsequence of the array whose sum of elements is

strictly greater

than the sum of the non included elements in such subsequence.

If there are multiple solutions, return the subsequence with

minimum size

and if there still exist multiple solutions, return the subsequence with the

maximum total sum

of all its elements. A subsequence of an array can be obtained by erasing some (possibly zero) elements from the array.

Note that the solution with the given constraints is guaranteed to be

unique

. Also return the answer sorted in

non-increasing

order.

Example 1:

Input:

nums = [4,3,10,9,8]

Output:

[10,9]

Explanation:

The subsequences [10,9] and [10,8] are minimal such that the sum of their elements is strictly greater than the sum of elements not included. However, the subsequence [10,9] has the maximum total sum of its elements.

Example 2:

Input:

nums = [4,4,7,6,7]

Output:

[7,7,6]

Explanation:

The subsequence [7,7] has the sum of its elements equal to 14 which is not strictly greater than the sum of elements not included ( $14 = 4 + 4 + 6$ ). Therefore, the subsequence [7,6,7] is the minimal satisfying the conditions. Note the subsequence has to be returned in non-increasing order.

Constraints:

$1 \leq \text{nums.length} \leq 500$

$1 \leq \text{nums}[i] \leq 100$

## Code Snippets

### C++:

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

    }
};
```

### Java:

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

    }
}
```

### Python3:

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

### Python:

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

### JavaScript:

```

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

};

```

### TypeScript:

```

function minSubsequence(nums: number[]): number[] {

};

```

### C#:

```

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

    }
}

```

### C:

```

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

}

```

### Go:

```

func minSubsequence(nums []int) []int {

}

```

### Kotlin:

```

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

    }
}

```

```
}
```

### Swift:

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

### Rust:

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

### Ruby:

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

### PHP:

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

### Dart:

```

class Solution {
    List<int> minSubsequence(List<int> nums) {

    }

}

```

### Scala:

```

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

    }

}

```

### Elixir:

```

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

  end

end

```

### Erlang:

```

-spec min_subsequence(Nums :: [integer()]) -> [integer()].
min_subsequence(Nums) ->

.

```

### Racket:

```

(define/contract (min-subsequence nums)
  (-> (listof exact-integer?) (listof exact-integer?))
  )

```

## Solutions

### C++ Solution:

```

/*
 * Problem: Minimum Subsequence in Non-Increasing Order

```

```

* Difficulty: Easy
* 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> minSubsequence(vector<int>& nums) {

    }
};

```

### Java Solution:

```

/**
 * Problem: Minimum Subsequence in Non-Increasing Order
 * Difficulty: Easy
 * 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> minSubsequence(int[] nums) {

    }
}

```

### Python3 Solution:

```

"""
Problem: Minimum Subsequence in Non-Increasing Order
Difficulty: Easy
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 minSubsequence(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass

```

### Python Solution:

```

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

```

### JavaScript Solution:

```

/**
 * Problem: Minimum Subsequence in Non-Increasing Order
 * Difficulty: Easy
 * 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[]} nums
 * @return {number[]}
 */
var minSubsequence = function(nums) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Minimum Subsequence in Non-Increasing Order
 * Difficulty: Easy
 * 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 minSubsequence(nums: number[]): number[] {

};

```

### C# Solution:

```

/*
 * Problem: Minimum Subsequence in Non-Increasing Order
 * Difficulty: Easy
 * 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> MinSubsequence(int[] nums) {

    }
}

```

### C Solution:

```

/*
 * Problem: Minimum Subsequence in Non-Increasing Order
 * Difficulty: Easy
 * 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* minSubsequence(int* nums, int numsSize, int* returnSize) {

}

```

### Go Solution:

```

// Problem: Minimum Subsequence in Non-Increasing Order
// Difficulty: Easy
// 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 minSubsequence(nums []int) []int {

}

```

### Kotlin Solution:

```

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

    }
}

```

### Swift Solution:

```

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

    }
}

```

### Rust Solution:

```
// Problem: Minimum Subsequence in Non-Increasing Order
// Difficulty: Easy
// 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 min_subsequence(nums: Vec<i32>) -> Vec<i32> {

    }
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

    }

}
```

### Dart Solution:

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

    }
}
```

### Scala Solution:

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

### Elixir Solution:

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

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (min-subsequence nums)  
  (-> (listof exact-integer?) (listof exact-integer?))  
)
```