# Problem 1673: Find the Most Competitive Subsequence

## Problem Information

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

## Problem Description

Given an integer array

nums

and a positive integer

k

, return

the most

competitive

subsequence of

nums

of size

k

.

An array's subsequence is a resulting sequence obtained by erasing some (possibly zero) elements from the array.

We define that a subsequence

a

is more

competitive

than a subsequence

b

(of the same length) if in the first position where

a

and

b

differ, subsequence

a

has a number

less

than the corresponding number in

b

. For example,

[1,3,4]

is more competitive than

[1,3,5]

because the first position they differ is at the final number, and

4

is less than

5

.

Example 1:

Input:

nums = [3,5,2,6], k = 2

Output:

[2,6]

Explanation:

Among the set of every possible subsequence: {[3,5], [3,2], [3,6], [5,2], [5,6], [2,6]}, [2,6] is the most competitive.

Example 2:

Input:

nums = [2,4,3,3,5,4,9,6], k = 4

Output:

[2,3,3,4]

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

1 <= k <= nums.length

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> mostCompetitive(vector<int>& nums, int k) {

}
};
```

**Java:**

```
class Solution {
public int[] mostCompetitive(int[] nums, int k) {

}
}
```

**Python3:**

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

**Python:**

```
class Solution(object):
def mostCompetitive(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: List[int]
"""
```

### JavaScript:

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

};
```

### TypeScript:

```
function mostCompetitive(nums: number[], k: number): number[] {

};
```

### C#:

```
public class Solution {
public int[] MostCompetitive(int[] nums, int k) {

}
}
```

### C:

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

}
```

### Go:

```
func mostCompetitive(nums []int, k int) []int {

}
```

### Kotlin:

```
class Solution {
fun mostCompetitive(nums: IntArray, k: Int): IntArray {

}
}
```

### Swift:

```
class Solution {
func mostCompetitive(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

### Rust:

```
impl Solution {
pub fn most_competitive(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

### Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def most_competitive(nums, k)

end
```

### PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function mostCompetitive($nums, $k) {

}
}
```

### Dart:

```
class Solution {
List<int> mostCompetitive(List<int> nums, int k) {

}
}
```

### Scala:

```
object Solution {
def mostCompetitive(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec most_competitive(nums :: [integer], k :: integer) :: [integer]
def most_competitive(nums, k) do

end
end
```

### Erlang:

```
-spec most_competitive(Nums :: [integer()], K :: integer()) -> [integer()].
most_competitive(Nums, K) ->
.
```

### Racket:

```
(define/contract (most-competitive nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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> mostCompetitive(vector<int>& nums, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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[] mostCompetitive(int[] nums, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find the Most Competitive Subsequence
Difficulty: Medium
Tags: array, greedy, stack

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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} k
* @return {number[]}
*/
var mostCompetitive = function(nums, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 mostCompetitive(nums: number[], k: number): number[] {

};
```

### C# Solution:

```
/*
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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[] MostCompetitive(int[] nums, int k) {

}

}
```

### C Solution:

```
/*
* Problem: Find the Most Competitive Subsequence
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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* mostCompetitive(int* nums, int numsSize, int k, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Find the Most Competitive Subsequence
// Difficulty: Medium
// Tags: array, greedy, stack
//
// 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 mostCompetitive(nums []int, k int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun mostCompetitive(nums: IntArray, k: Int): IntArray {

}
}
```

### Swift Solution:

```
class Solution {
func mostCompetitive(_ nums: [Int], _ k: Int) -> [Int] {

}
}
```

### Rust Solution:

```
// Problem: Find the Most Competitive Subsequence
// Difficulty: Medium
// Tags: array, greedy, stack
//
// 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 most_competitive(nums: Vec<i32>, k: i32) -> Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer[]}
def most_competitive(nums, k)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer[]
*/
function mostCompetitive($nums, $k) {

}
}
```

### Dart Solution:

```
class Solution {
List<int> mostCompetitive(List<int> nums, int k) {

}
}
```

### Scala Solution:

```
object Solution {
def mostCompetitive(nums: Array[Int], k: Int): Array[Int] = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec most_competitive(nums :: [integer], k :: integer) :: [integer]
def most_competitive(nums, k) do

end
end
```

### Erlang Solution:

```
-spec most_competitive(Nums :: [integer()], K :: integer()) -> [integer()].
most_competitive(Nums, K) ->
.
```

### Racket Solution:

```
(define/contract (most-competitive nums k)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```