Problem 1005: Maximize Sum Of Array After K Negations

Problem Information

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

Problem Description

Given an integer array

nums

and an integer

k

, modify the array in the following way:

choose an index

i

and replace

nums[i]

with

-nums[i]

.

You should apply this process exactly

k

times. You may choose the same index

i

multiple times.

Return

the largest possible sum of the array after modifying it in this way

.

Example 1:

Input:

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

Output:

5

Explanation:

Choose index 1 and nums becomes [4,-2,3].

Example 2:

Input:

nums = [3,-1,0,2], k = 3

Output:

6

Explanation:

Choose indices (1, 2, 2) and nums becomes [3,1,0,2].

Example 3:

Input:

nums = [2,-3,-1,5,-4], k = 2

Output:

13

Explanation:

Choose indices (1, 4) and nums becomes [2,3,-1,5,4].

Constraints:

1 <= nums.length <= 10

4

-100 <= nums[i] <= 100

1 <= k <= 10

4

Code Snippets

C++:

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

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int largestSumAfterKNegations(int* nums, int numsSize, int k) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
integer
def largest_sum_after_k_negations(nums, k) do

end
end
```

Erlang:

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

Racket:

```
(define/contract (largest-sum-after-k-negations nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Sum Of Array After K Negations
* 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:
int largestSumAfterKNegations(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Sum Of Array After K Negations
* 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 int largestSumAfterKNegations(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Sum Of Array After K Negations
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 largestSumAfterKNegations(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Sum Of Array After K Negations
* 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
* @param {number} k
* @return {number}
*/
var largestSumAfterKNegations = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Sum Of Array After K Negations
* 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 largestSumAfterKNegations(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Sum Of Array After K Negations
* 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 int LargestSumAfterKNegations(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Sum Of Array After K Negations
* 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
*/

int largestSumAfterKNegations(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximize Sum Of Array After K Negations
// 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 largestSumAfterKNegations(nums []int, k int) int {
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Sum Of Array After K Negations
// 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 largest_sum_after_k_negations(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (largest-sum-after-k-negations nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```