Problem 2600: K Items With the Maximum Sum

Problem Information

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

Problem Description

There is a bag that consists of items, each item has a number

1

,

0

, or

-1

written on it.

You are given four

non-negative

integers

numOnes

,

numZeros

,

numNegOnes

, and

k

.

The bag initially contains:

numOnes

items with

1

s written on them.

numZeroes

items with

0

s written on them.

numNegOnes

items with

-1

s written on them.

We want to pick exactly

k

items among the available items. Return

the

maximum

possible sum of numbers written on the items

.

Example 1:

Input:

numOnes = 3, numZeros = 2, numNegOnes = 0, k = 2

Output:

2

Explanation:

We have a bag of items with numbers written on them {1, 1, 1, 0, 0}. We take 2 items with 1 written on them and get a sum in a total of 2. It can be proven that 2 is the maximum possible sum.

Example 2:

Input:

numOnes = 3, numZeros = 2, numNegOnes = 0, k = 4

Output:

3

Explanation:

We have a bag of items with numbers written on them {1, 1, 1, 0, 0}. We take 3 items with 1 written on them, and 1 item with 0 written on it, and get a sum in a total of 3. It can be proven that 3 is the maximum possible sum.

Constraints:

0 <= numOnes, numZeros, numNegOnes <= 50

0 <= k <= numOnes + numZeros + numNegOnes

Code Snippets

C++:

```
class Solution {
public:
int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
};
```

Java:

```
class Solution {
public int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes,
int k) {

}
}
```

Python3:

```
class Solution:
def kItemsWithMaximumSum(self, numOnes: int, numZeros: int, numNegOnes: int,
k: int) -> int:
```

Python:

```
class Solution(object):
def kItemsWithMaximumSum(self, numOnes, numZeros, numNegOnes, k):
"""
```

```
:type numOnes: int
:type numZeros: int
:type numNegOnes: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} numOnes
* @param {number} numZeros
* @param {number} numNegOnes
* @param {number} k
* @return {number}
*/
var kItemsWithMaximumSum = function(numOnes, numZeros, numNegOnes, k) {

};
```

TypeScript:

```
function kItemsWithMaximumSum(numOnes: number, numZeros: number, numNegOnes:
number, k: number): number {

};
```

C#:

```
public class Solution {
public int KItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes,
int k) {

}
}
```

C:

```
int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
```

Go:

```
func kItemsWithMaximumSum(numOnes int, numZeros int, numNegOnes int, k int)
int {

}
```

Kotlin:

```
class Solution {
fun kItemsWithMaximumSum(numOnes: Int, numZeros: Int, numNegOnes: Int, k:
Int): Int {

}
}
```

Swift:

```
class Solution {
func kItemsWithMaximumSum(_ numOnes: Int, _ numZeros: Int, _ numNegOnes: Int,
_ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn k_items_with_maximum_sum(num_ones: i32, num_zeros: i32, num_neg_ones:
i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} num_ones
# @param {Integer} num_zeros
# @param {Integer} num_neg_ones
# @param {Integer} k
# @return {Integer}
def k_items_with_maximum_sum(num_ones, num_zeros, num_neg_ones, k)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $numOnes
* @param Integer $numZeros
* @param Integer $numNegOnes
* @param Integer $k
* @return Integer
*/
function kItemsWithMaximumSum($numOnes, $numZeros, $numNegOnes, $k) {

}
}
```

Dart:

```
class Solution {
int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
}
```

Scala:

```
object Solution {
def kItemsWithMaximumSum(numOnes: Int, numZeros: Int, numNegOnes: Int, k:
Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec k_items_with_maximum_sum(num_ones :: integer, num_zeros :: integer,
num_neg_ones :: integer, k :: integer) :: integer
def k_items_with_maximum_sum(num_ones, num_zeros, num_neg_ones, k) do
```

```
end
end
```

Erlang:

```
-spec k_items_with_maximum_sum(NumOnes :: integer(), NumZeros :: integer(),
NumNegOnes :: integer(), K :: integer()) -> integer().
k_items_with_maximum_sum(NumOnes, NumZeros, NumNegOnes, K) ->
.
```

Racket:

```
(define/contract (k-items-with-maximum-sum numOnes numZeros numNegOnes k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
};
```

Java Solution:

```
/**
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes,
int k) {

}
}
```

Python3 Solution:

```
"""
Problem: K Items With the Maximum Sum
Difficulty: Easy
Tags: greedy, math

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def kItemsWithMaximumSum(self, numOnes: int, numZeros: int, numNegOnes: int,
k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def kItemsWithMaximumSum(self, numOnes, numZeros, numNegOnes, k):
"""
:type numOnes: int
:type numZeros: int
```

```
:type numNegOnes: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} numOnes
* @param {number} numZeros
* @param {number} numNegOnes
* @param {number} k
* @return {number}
*/
var kItemsWithMaximumSum = function(numOnes, numZeros, numNegOnes, k) {

};
```

TypeScript Solution:

```
/**
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function kItemsWithMaximumSum(numOnes: number, numZeros: number, numNegOnes:
```

```
number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int KItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes,
int k) {

}
}
```

C Solution:

```
/*
* Problem: K Items With the Maximum Sum
* Difficulty: Easy
* Tags: greedy, math
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
```

Go Solution:

```
// Problem: K Items With the Maximum Sum
// Difficulty: Easy
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func kItemsWithMaximumSum(numOnes int, numZeros int, numNegOnes int, k int)
int {

}
```

Kotlin Solution:

```
class Solution {
fun kItemsWithMaximumSum(numOnes: Int, numZeros: Int, numNegOnes: Int, k:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func kItemsWithMaximumSum(_ numOnes: Int, _ numZeros: Int, _ numNegOnes: Int,
_ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: K Items With the Maximum Sum
// Difficulty: Easy
// Tags: greedy, math
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
```

```
pub fn k_items_with_maximum_sum(num_ones: i32, num_zeros: i32, num_neg_ones:
i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num_ones
# @param {Integer} num_zeros
# @param {Integer} num_neg_ones
# @param {Integer} k
# @return {Integer}
def k_items_with_maximum_sum(num_ones, num_zeros, num_neg_ones, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $numOnes
* @param Integer $numZeros
* @param Integer $numNegOnes
* @param Integer $k
* @return Integer
*/
function kItemsWithMaximumSum($numOnes, $numZeros, $numNegOnes, $k) {

}
}
```

Dart Solution:

```
class Solution {
int kItemsWithMaximumSum(int numOnes, int numZeros, int numNegOnes, int k) {

}
}
```

Scala Solution:

```
object Solution {
def kItemsWithMaximumSum(numOnes: Int, numZeros: Int, numNegOnes: Int, k:
Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec k_items_with_maximum_sum(num_ones :: integer, num_zeros :: integer,
num_neg_ones :: integer, k :: integer) :: integer
def k_items_with_maximum_sum(num_ones, num_zeros, num_neg_ones, k) do

end
end
```

Erlang Solution:

```
-spec k_items_with_maximum_sum(NumOnes :: integer(), NumZeros :: integer(),
NumNegOnes :: integer(), K :: integer()) -> integer().
k_items_with_maximum_sum(NumOnes, NumZeros, NumNegOnes, K) ->
.
```

Racket Solution:

```
(define/contract (k-items-with-maximum-sum numOnes numZeros numNegOnes k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```