# Problem 1648: Sell Diminishing-Valued Colored Balls

## Problem Information

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

## Problem Description

You have an

inventory

of different colored balls, and there is a customer that wants

orders

balls of

any

color.

The customer weirdly values the colored balls. Each colored ball's value is the number of balls

of that color

you currently have in your

inventory

. For example, if you own

6

yellow balls, the customer would pay

6

for the first yellow ball. After the transaction, there are only

5

yellow balls left, so the next yellow ball is then valued at

5

(i.e., the value of the balls decreases as you sell more to the customer).

You are given an integer array,

inventory

, where

inventory[i]

represents the number of balls of the

i

th

color that you initially own. You are also given an integer

orders

, which represents the total number of balls that the customer wants. You can sell the balls

in any order

.

Return

the

maximum

total value that you can attain after selling

orders

colored balls

. As the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

inventory = [2,5], orders = 4

Output:

14

Explanation:

Sell the 1st color 1 time (2) and the 2nd color 3 times (5 + 4 + 3). The maximum total value is 2 + 5 + 4 + 3 = 14.

Example 2:

Input:

inventory = [3,5], orders = 6

Output:

19

Explanation:

Sell the 1st color 2 times (3 + 2) and the 2nd color 4 times (5 + 4 + 3 + 2). The maximum total value is 3 + 2 + 5 + 4 + 3 + 2 = 19.

Constraints:

1 <= inventory.length <= 10

5

1 <= inventory[i] <= 10

9

1 <= orders <= min(sum(inventory[i]), 10

9

)

## Code Snippets

**C++:**

```
class Solution {
public:
int maxProfit(vector<int>& inventory, int orders) {

}
};
```

**Java:**

```
class Solution {
public int maxProfit(int[] inventory, int orders) {

}
}
```

**Python3:**

```
class Solution:
def maxProfit(self, inventory: List[int], orders: int) -> int:
```

**Python:**

```
class Solution(object):
def maxProfit(self, inventory, orders):
"""
:type inventory: List[int]
:type orders: int
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} inventory
* @param {number} orders
* @return {number}
*/
var maxProfit = function(inventory, orders) {

};
```

**TypeScript:**

```
function maxProfit(inventory: number[], orders: number): number {

};
```

**C#:**

```
public class Solution {
public int MaxProfit(int[] inventory, int orders) {

}
}
```

**C:**

```
int maxProfit(int* inventory, int inventorySize, int orders) {

}
```

**Go:**

```
func maxProfit(inventory []int, orders int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxProfit(inventory: IntArray, orders: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func maxProfit(_ inventory: [Int], _ orders: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_profit(inventory: Vec<i32>, orders: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} inventory
# @param {Integer} orders
# @return {Integer}
def max_profit(inventory, orders)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $inventory
* @param Integer $orders
* @return Integer
*/
function maxProfit($inventory, $orders) {

}
}
```

**Dart:**

```
class Solution {
int maxProfit(List<int> inventory, int orders) {
```

```
}
}
```

**Scala:**

```
object Solution {
def maxProfit(inventory: Array[Int], orders: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_profit(inventory :: [integer], orders :: integer) :: integer
def max_profit(inventory, orders) do

end
end
```

**Erlang:**

```
-spec max_profit(Inventory :: [integer()], Orders :: integer()) -> integer().
max_profit(Inventory, Orders) ->
.
```

**Racket:**

```
(define/contract (max-profit inventory orders)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
```

```
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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 maxProfit(vector<int>& inventory, int orders) {

}
};
```

### Java Solution:

```
/**
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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 maxProfit(int[] inventory, int orders) {

}
}
```

### Python3 Solution:

```
"""
Problem: Sell Diminishing-Valued Colored Balls
Difficulty: Medium
Tags: array, greedy, math, sort, search, queue, heap

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 maxProfit(self, inventory: List[int], orders: int) -> int:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def maxProfit(self, inventory, orders):
"""
:type inventory: List[int]
:type orders: int
:rtype: int
"""
```

**JavaScript Solution:**

```
/**
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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[]} inventory
* @param {number} orders
* @return {number}
*/
var maxProfit = function(inventory, orders) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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 maxProfit(inventory: number[], orders: number): number {

};
```

### C# Solution:

```
/*
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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 MaxProfit(int[] inventory, int orders) {

}
}
```

### C Solution:

```
/*
* Problem: Sell Diminishing-Valued Colored Balls
* Difficulty: Medium
* Tags: array, greedy, math, sort, search, queue, heap
*
* 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 maxProfit(int* inventory, int inventorySize, int orders) {

}
```

### Go Solution:

```
// Problem: Sell Diminishing-Valued Colored Balls
// Difficulty: Medium
// Tags: array, greedy, math, sort, search, queue, heap
//
// 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 maxProfit(inventory []int, orders int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxProfit(inventory: IntArray, orders: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func maxProfit(_ inventory: [Int], _ orders: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Sell Diminishing-Valued Colored Balls
// Difficulty: Medium
// Tags: array, greedy, math, sort, search, queue, heap
```

```
//
// 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 max_profit(inventory: Vec<i32>, orders: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} inventory
# @param {Integer} orders
# @return {Integer}
def max_profit(inventory, orders)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $inventory
* @param Integer $orders
* @return Integer
*/
function maxProfit($inventory, $orders) {

}
}
```

### Dart Solution:

```
class Solution {
int maxProfit(List<int> inventory, int orders) {

}
}
```

### Scala Solution:

```
object Solution {
def maxProfit(inventory: Array[Int], orders: Int): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec max_profit(inventory :: [integer], orders :: integer) :: integer
def max_profit(inventory, orders) do

end
end
```

### Erlang Solution:

```
-spec max_profit(Inventory :: [integer()], Orders :: integer()) -> integer().
max_profit(Inventory, Orders) ->
.
```

### Racket Solution:

```
(define/contract (max-profit inventory orders)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```