# Problem 3139: Minimum Cost to Equalize Array

## Problem Information

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

## Problem Description

You are given an integer array

nums

and two integers

cost1

and

cost2

. You are allowed to perform

either

of the following operations

any

number of times:

Choose an index

i

from

nums

and

increase

nums[i]

by

1

for a cost of

cost1

.

Choose two

different

indices

i

,

j

, from

nums

and

increase

nums[i]

and

nums[j]

by

1

for a cost of

cost2

.

Return the

minimum

cost

required to make all elements in the array

equal

.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [4,1], cost1 = 5, cost2 = 2

Output:

15

Explanation:

The following operations can be performed to make the values equal:

Increase

nums[1]

by 1 for a cost of 5.

nums

becomes

[4,2]

.

Increase

nums[1]

by 1 for a cost of 5.

nums

becomes

[4,3]

.

Increase

nums[1]

by 1 for a cost of 5.

nums

becomes

[4,4]

.

The total cost is 15.

Example 2:

Input:

nums = [2,3,3,3,5], cost1 = 2, cost2 = 1

Output:

6

Explanation:

The following operations can be performed to make the values equal:

Increase

nums[0]

and

nums[1]

by 1 for a cost of 1.

nums

becomes

[3,4,3,3,5]

.

Increase

nums[0]

and

nums[2]

by 1 for a cost of 1.

nums

becomes

[4,4,4,3,5]

.

Increase

nums[0]

and

nums[3]

by 1 for a cost of 1.

nums

becomes

[5,4,4,4,5]

.

Increase

nums[1]

and

nums[2]

by 1 for a cost of 1.

nums

becomes

[5,5,5,4,5]

.

Increase

nums[3]

by 1 for a cost of 2.

nums

becomes

[5,5,5,5,5]

.

The total cost is 6.

Example 3:

Input:

nums = [3,5,3], cost1 = 1, cost2 = 3

Output:

4

Explanation:

The following operations can be performed to make the values equal:

Increase

nums[0]

by 1 for a cost of 1.

nums

becomes

[4,5,3]

.

Increase

nums[0]

by 1 for a cost of 1.

nums

becomes

[5,5,3]

.

Increase

nums[2]

by 1 for a cost of 1.

nums

becomes

[5,5,4]

.

Increase

nums[2]

by 1 for a cost of 1.

nums

becomes

[5,5,5]

.

The total cost is 4.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

1 <= cost1 <= 10

6

1 <= cost2 <= 10

6

## Code Snippets

**C++:**

```
class Solution {
public:
int minCostToEqualizeArray(vector<int>& nums, int cost1, int cost2) {

}
};
```

**Java:**

```
class Solution {
public int minCostToEqualizeArray(int[] nums, int cost1, int cost2) {

}
}
```

**Python3:**

```
class Solution:
def minCostToEqualizeArray(self, nums: List[int], cost1: int, cost2: int) ->
int:
```

**Python:**

```
class Solution(object):
def minCostToEqualizeArray(self, nums, cost1, cost2):
"""
:type nums: List[int]
:type cost1: int
:type cost2: int
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} nums
* @param {number} cost1
* @param {number} cost2
* @return {number}
*/
var minCostToEqualizeArray = function(nums, cost1, cost2) {

};
```

**TypeScript:**

```
function minCostToEqualizeArray(nums: number[], cost1: number, cost2:
number): number {

};
```

**C#:**

```
public class Solution {
public int MinCostToEqualizeArray(int[] nums, int cost1, int cost2) {

}
}
```

**C:**

```
int minCostToEqualizeArray(int* nums, int numsSize, int cost1, int cost2) {

}
```

**Go:**

```
func minCostToEqualizeArray(nums []int, cost1 int, cost2 int) int {

}
```

**Kotlin:**

```
class Solution {
fun minCostToEqualizeArray(nums: IntArray, cost1: Int, cost2: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func minCostToEqualizeArray(_ nums: [Int], _ cost1: Int, _ cost2: Int) -> Int
{

}
}
```

**Rust:**

```
impl Solution {
pub fn min_cost_to_equalize_array(nums: Vec<i32>, cost1: i32, cost2: i32) ->
i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer} cost1
# @param {Integer} cost2
# @return {Integer}
def min_cost_to_equalize_array(nums, cost1, cost2)
```

```
end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $cost1
* @param Integer $cost2
* @return Integer
*/
function minCostToEqualizeArray($nums, $cost1, $cost2) {

}
}
```

**Dart:**

```
class Solution {
int minCostToEqualizeArray(List<int> nums, int cost1, int cost2) {

}
}
```

**Scala:**

```
object Solution {
def minCostToEqualizeArray(nums: Array[Int], cost1: Int, cost2: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec min_cost_to_equalize_array(nums :: [integer], cost1 :: integer, cost2
:: integer) :: integer
def min_cost_to_equalize_array(nums, cost1, cost2) do

end
```

```
end
```

**Erlang:**

```
-spec min_cost_to_equalize_array(Nums :: [integer()], Cost1 :: integer(),
Cost2 :: integer()) -> integer().
min_cost_to_equalize_array(Nums, Cost1, Cost2) ->
.
```

**Racket:**

```
(define/contract (min-cost-to-equalize-array nums cost1 cost2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Cost to Equalize Array
* Difficulty: Hard
* Tags: array, greedy
*
* 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 minCostToEqualizeArray(vector<int>& nums, int cost1, int cost2) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Cost to Equalize Array
```

```
* Difficulty: Hard
* Tags: array, greedy
*
* 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 minCostToEqualizeArray(int[] nums, int cost1, int cost2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Cost to Equalize Array
Difficulty: Hard
Tags: array, greedy

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 minCostToEqualizeArray(self, nums: List[int], cost1: int, cost2: int) ->
int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def minCostToEqualizeArray(self, nums, cost1, cost2):
"""
:type nums: List[int]
:type cost1: int
:type cost2: int
:rtype: int
```

```
"""
```

### JavaScript Solution:

```
/**
* Problem: Minimum Cost to Equalize Array
* Difficulty: Hard
* Tags: array, greedy
*
* 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} cost1
* @param {number} cost2
* @return {number}
*/
var minCostToEqualizeArray = function(nums, cost1, cost2) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Cost to Equalize Array
* Difficulty: Hard
* Tags: array, greedy
*
* 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 minCostToEqualizeArray(nums: number[], cost1: number, cost2:
number): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Cost to Equalize Array
* Difficulty: Hard
* Tags: array, greedy
*
* 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 MinCostToEqualizeArray(int[] nums, int cost1, int cost2) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Cost to Equalize Array
* Difficulty: Hard
* Tags: array, greedy
*
* 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 minCostToEqualizeArray(int* nums, int numsSize, int cost1, int cost2) {

}
```

### Go Solution:

```
// Problem: Minimum Cost to Equalize Array
// Difficulty: Hard
// Tags: array, greedy
//
// 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 minCostToEqualizeArray(nums []int, cost1 int, cost2 int) int {

}
```

**Kotlin Solution:**

```
class Solution {
fun minCostToEqualizeArray(nums: IntArray, cost1: Int, cost2: Int): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func minCostToEqualizeArray(_ nums: [Int], _ cost1: Int, _ cost2: Int) -> Int
{

}
}
```

**Rust Solution:**

```
// Problem: Minimum Cost to Equalize Array
// Difficulty: Hard
// Tags: array, greedy
//
// 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_cost_to_equalize_array(nums: Vec<i32>, cost1: i32, cost2: i32) ->
i32 {

}
}
```

**Ruby Solution:**

```
# @param {Integer[]} nums
# @param {Integer} cost1
# @param {Integer} cost2
# @return {Integer}
def min_cost_to_equalize_array(nums, cost1, cost2)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $cost1
* @param Integer $cost2
* @return Integer
*/
function minCostToEqualizeArray($nums, $cost1, $cost2) {

}
}
```

**Dart Solution:**

```
class Solution {
int minCostToEqualizeArray(List<int> nums, int cost1, int cost2) {

}
}
```

**Scala Solution:**

```
object Solution {
def minCostToEqualizeArray(nums: Array[Int], cost1: Int, cost2: Int): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec min_cost_to_equalize_array(nums :: [integer], cost1 :: integer, cost2
:: integer) :: integer
def min_cost_to_equalize_array(nums, cost1, cost2) do

end
end
```

### Erlang Solution:

```
-spec min_cost_to_equalize_array(Nums :: [integer()], Cost1 :: integer(),
Cost2 :: integer()) -> integer().
min_cost_to_equalize_array(Nums, Cost1, Cost2) ->
.
```

### Racket Solution:

```
(define/contract (min-cost-to-equalize-array nums cost1 cost2)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```