

Problem 3002: Maximum Size of a Set After Removals

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

of even length

n

.

You must remove

$n / 2$

elements from

nums1

and

$n / 2$

elements from

nums2

. After the removals, you insert the remaining elements of

nums1

and

nums2

into a set

s

.

Return

the

maximum

possible size of the set

s

.

Example 1:

Input:

nums1 = [1,2,1,2], nums2 = [1,1,1,1]

Output:

2

Explanation:

We remove two occurrences of 1 from nums1 and nums2. After the removals, the arrays become equal to nums1 = [2,2] and nums2 = [1,1]. Therefore, $s = \{1,2\}$. It can be shown that 2 is the maximum possible size of the set s after the removals.

Example 2:

Input:

nums1 = [1,2,3,4,5,6], nums2 = [2,3,2,3,2,3]

Output:

5

Explanation:

We remove 2, 3, and 6 from nums1, as well as 2 and two occurrences of 3 from nums2. After the removals, the arrays become equal to nums1 = [1,4,5] and nums2 = [2,3,2]. Therefore, $s = \{1,2,3,4,5\}$. It can be shown that 5 is the maximum possible size of the set s after the removals.

Example 3:

Input:

nums1 = [1,1,2,2,3,3], nums2 = [4,4,5,5,6,6]

Output:

6

Explanation:

We remove 1, 2, and 3 from nums1, as well as 4, 5, and 6 from nums2. After the removals, the arrays become equal to nums1 = [1,2,3] and nums2 = [4,5,6]. Therefore, $s = \{1,2,3,4,5,6\}$. It can be shown that 6 is the maximum possible size of the set s after the removals.

Constraints:

$n == \text{nums1.length} == \text{nums2.length}$

$1 \leq n \leq 2 * 10$

4

n

is even.

$1 \leq \text{nums1}[i], \text{nums2}[i] \leq 10$

9

Code Snippets

C++:

```
class Solution {
public:
    int maximumSetSize(vector<int>& nums1, vector<int>& nums2) {

    }
};
```

Java:

```
class Solution {
    public int maximumSetSize(int[] nums1, int[] nums2) {

    }
}
```

```
}
```

Python3:

```
class Solution:
    def maximumSetSize(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
    def maximumSetSize(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: int
        """
```

JavaScript:

```
/**
 * @param {number[]} nums1
 * @param {number[]} nums2
 * @return {number}
 */
var maximumSetSize = function(nums1, nums2) {

};
```

TypeScript:

```
function maximumSetSize(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
    public int MaximumSetSize(int[] nums1, int[] nums2) {

    }
}
```

C:

```
int maximumSetSize(int* nums1, int nums1Size, int* nums2, int nums2Size) {  
  
}
```

Go:

```
func maximumSetSize(nums1 []int, nums2 []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maximumSetSize(nums1: IntArray, nums2: IntArray): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maximumSetSize(_ nums1: [Int], _ nums2: [Int]) -> Int {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn maximum_set_size(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} nums1  
# @param {Integer[]} nums2  
# @return {Integer}  
def maximum_set_size(nums1, nums2)
```

```
end
```

PHP:

```
class Solution {

    /**
     * @param Integer[] $nums1
     * @param Integer[] $nums2
     * @return Integer
     */
    function maximumSetSize($nums1, $nums2) {

    }

}
```

Dart:

```
class Solution {
  int maximumSetSize(List<int> nums1, List<int> nums2) {

  }

}
```

Scala:

```
object Solution {
  def maximumSetSize(nums1: Array[Int], nums2: Array[Int]): Int = {

  }

}
```

Elixir:

```
defmodule Solution do
  @spec maximum_set_size(nums1 :: [integer], nums2 :: [integer]) :: integer
  def maximum_set_size(nums1, nums2) do

  end

end
```

Erlang:

```
-spec maximum_set_size(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
maximum_set_size(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (maximum-set-size nums1 nums2)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Maximum Size of a Set After Removals
 * Difficulty: Medium
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    int maximumSetSize(vector<int>& nums1, vector<int>& nums2) {

    }
};
```

Java Solution:

```
/**
 * Problem: Maximum Size of a Set After Removals
 * Difficulty: Medium
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 */
```



```

* Space Complexity: O(n) for hash map
*/

class Solution {
public int maximumSetSize(int[] nums1, int[] nums2) {

}
}

```

Python3 Solution:

```

"""
Problem: Maximum Size of a Set After Removals
Difficulty: Medium
Tags: array, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def maximumSetSize(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

class Solution(object):
def maximumSetSize(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""

```

JavaScript Solution:

```

/**
* Problem: Maximum Size of a Set After Removals
* Difficulty: Medium

```

```

* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var maximumSetSize = function(nums1, nums2) {

};

```

TypeScript Solution:

```

/**
* Problem: Maximum Size of a Set After Removals
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maximumSetSize(nums1: number[], nums2: number[]): number {

};

```

C# Solution:

```

/*
* Problem: Maximum Size of a Set After Removals
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)

```

```

* Space Complexity: O(n) for hash map
*/

public class Solution {
    public int MaximumSetSize(int[] nums1, int[] nums2) {

    }
}

```

C Solution:

```

/*
* Problem: Maximum Size of a Set After Removals
* Difficulty: Medium
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int maximumSetSize(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}

```

Go Solution:

```

// Problem: Maximum Size of a Set After Removals
// Difficulty: Medium
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maximumSetSize(nums1 []int, nums2 []int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun maximumSetSize(nums1: IntArray, nums2: IntArray): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func maximumSetSize(_ nums1: [Int], _ nums2: [Int]) -> Int {

    }
}

```

Rust Solution:

```

// Problem: Maximum Size of a Set After Removals
// Difficulty: Medium
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
    pub fn maximum_set_size(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

    }
}

```

Ruby Solution:

```

# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def maximum_set_size(nums1, nums2)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $nums1
     * @param Integer[] $nums2
     * @return Integer
     */
    function maximumSetSize($nums1, $nums2) {

    }

}

```

Dart Solution:

```

class Solution {
  int maximumSetSize(List<int> nums1, List<int> nums2) {

  }

}

```

Scala Solution:

```

object Solution {
  def maximumSetSize(nums1: Array[Int], nums2: Array[Int]): Int = {

  }

}

```

Elixir Solution:

```

defmodule Solution do
  @spec maximum_set_size(nums1 :: [integer], nums2 :: [integer]) :: integer
  def maximum_set_size(nums1, nums2) do

  end

end

```

Erlang Solution:

```

-spec maximum_set_size(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
maximum_set_size(Nums1, Nums2) ->

```

.

Racket Solution:

```
(define/contract (maximum-set-size nums1 nums2)
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
  )
```