Problem 3731: Find Missing Elements

Problem Information

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

Problem Description

You are given an integer array

nums

consisting of

unique

integers.

Originally,

nums

contained

every integer

within a certain range. However, some integers might have gone

missing

from the array.

The

smallest

and

largest

integers of the original range are still present in

nums

.

Return a

sorted

list of all the missing integers in this range. If no integers are missing, return an

empty

list.

Example 1:

Input:

nums = [1,4,2,5]

Output:

[3]

Explanation:

The smallest integer is 1 and the largest is 5, so the full range should be

[1,2,3,4,5]

. Among these, only 3 is missing.

Example 2:

Input:

nums = [7,8,6,9]

Output:

[]

Explanation:

The smallest integer is 6 and the largest is 9, so the full range is

[6,7,8,9]

. All integers are already present, so no integer is missing.

Example 3:

Input:

nums = [5,1]

Output:

[2,3,4]

Explanation:

The smallest integer is 1 and the largest is 5, so the full range should be

[1,2,3,4,5]

. The missing integers are 2, 3, and 4.

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

```
class Solution {
public:
vector<int> findMissingElements(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public List<Integer> findMissingElements(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
* @return {number[]}
*/
var findMissingElements = function(nums) {

};
```

TypeScript:

```
function findMissingElements(nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> FindMissingElements(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findMissingElements(int* nums, int numsSize, int* returnSize) {

}
```

Go:

```
func findMissingElements(nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun findMissingElements(nums: IntArray): List<Int> {

}
}
```

Swift:

```
class Solution {
func findMissingElements(_ nums: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_missing_elements(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def find_missing_elements(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function findMissingElements($nums) {

}
}
```

Dart:

```
class Solution {
List<int> findMissingElements(List<int> nums) {

}
```

```
}
```

Scala:

```
object Solution {
def findMissingElements(nums: Array[Int]): List[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_missing_elements(nums :: [integer]) :: [integer]
def find_missing_elements(nums) do

end
end
```

Erlang:

```
-spec find_missing_elements(Nums :: [integer()]) -> [integer()].
find_missing_elements(Nums) ->
.
```

Racket:

```
(define/contract (find-missing-elements nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
*
```

```
* 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:
vector<int> findMissingElements(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 List<Integer> findMissingElements(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Missing Elements
Difficulty: Easy
Tags: array, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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[]} nums
* @return {number[]}
*/
var findMissingElements = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
```

```
*
* 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 findMissingElements(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
*
* 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 IList<int> FindMissingElements(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find Missing Elements
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findMissingElements(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Missing Elements
// Difficulty: Easy
// Tags: array, hash, sort
//
// 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 findMissingElements(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun findMissingElements(nums: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func findMissingElements(_ nums: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Missing Elements
// Difficulty: Easy
// Tags: array, hash, sort
```

```
//
// 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 find_missing_elements(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def find_missing_elements(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function findMissingElements($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findMissingElements(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def findMissingElements(nums: Array[Int]): List[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_missing_elements(nums :: [integer]) :: [integer]
def find_missing_elements(nums) do

end
end
```

Erlang Solution:

```
-spec find_missing_elements(Nums :: [integer()]) -> [integer()].
find_missing_elements(Nums) ->
.
```

Racket Solution:

```
(define/contract (find-missing-elements nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```