

Problem 3678: Smallest Absent Positive Greater Than Average

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

nums

.

Return the

smallest absent positive

integer in

nums

such that it is

strictly greater

than the

average

of all elements in

nums

The

average

of an array is defined as the sum of all its elements divided by the number of elements.

Example 1:

Input:

nums = [3,5]

Output:

6

Explanation:

The average of

nums

is

$$(3 + 5) / 2 = 8 / 2 = 4$$

The smallest absent positive integer greater than 4 is 6.

Example 2:

Input:

nums = [-1,1,2]

Output:

3

Explanation:

The average of

nums

is

$$(-1 + 1 + 2) / 3 = 2 / 3 = 0.667$$

The smallest absent positive integer greater than 0.667 is 3.

Example 3:

Input:

nums = [4,-1]

Output:

2

Explanation:

The average of

nums

is

$$(4 + (-1)) / 2 = 3 / 2 = 1.50$$

The smallest absent positive integer greater than 1.50 is 2.

Constraints:

$1 \leq \text{nums.length} \leq 100$

$-100 \leq \text{nums}[i] \leq 100$

Code Snippets

C++:

```
class Solution {
public:
    int smallestAbsent(vector<int>& nums) {
        }
};
```

Java:

```
class Solution {
    public int smallestAbsent(int[] nums) {
        }
}
```

Python3:

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

Python:

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

```
"""
```

JavaScript:

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

TypeScript:

```
function smallestAbsent(nums: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int SmallestAbsent(int[] nums) {  
  
    }  
}
```

C:

```
int smallestAbsent(int* nums, int numsSize) {  
  
}
```

Go:

```
func smallestAbsent(nums []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun smallestAbsent(nums: IntArray): Int {
```

```
}
```

```
}
```

Swift:

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

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer  
     */  
    function smallestAbsent($nums) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int smallestAbsent(List<int> nums) {  
        }  
    }  
}
```

Scala:

```
object Solution {  
    def smallestAbsent(nums: Array[Int]): Int = {  
        }  
    }  
}
```

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (smallest-absent nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Smallest Absent Positive Greater Than Average  
 */
```

```

* Difficulty: Easy
* Tags: array, 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 smallestAbsent(vector<int>& nums) {

```

```

    }
};

```

Java Solution:

```

/**
 * Problem: Smallest Absent Positive Greater Than Average
 * Difficulty: Easy
 * Tags: array, 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 smallestAbsent(int[] nums) {

```

```

    }
}

```

Python3 Solution:

```

"""
Problem: Smallest Absent Positive Greater Than Average
Difficulty: Easy
Tags: array, 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 smallestAbsent(self, nums: List[int]) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Smallest Absent Positive Greater Than Average
 * Difficulty: Easy
 * Tags: array, 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[]} nums
 * @return {number}
 */
var smallestAbsent = function(nums) {

```

TypeScript Solution:

```

/**
 * Problem: Smallest Absent Positive Greater Than Average
 * Difficulty: Easy
 * Tags: array, 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 smallestAbsent(nums: number[]): number {
}

```

C# Solution:

```

/*
 * Problem: Smallest Absent Positive Greater Than Average
 * Difficulty: Easy
 * Tags: array, 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 SmallestAbsent(int[] nums) {
        }
    }

```

C Solution:

```

/*
 * Problem: Smallest Absent Positive Greater Than Average
 * Difficulty: Easy
 * Tags: array, 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

```

```
*/\n\nint smallestAbsent(int* nums, int numsSize) {\n\n}
```

Go Solution:

```
// Problem: Smallest Absent Positive Greater Than Average\n// Difficulty: Easy\n// Tags: array, hash\n//\n// Approach: Use two pointers or sliding window technique\n// Time Complexity: O(n) or O(n log n)\n// Space Complexity: O(n) for hash map\n\nfunc smallestAbsent(nums []int) int {\n\n}
```

Kotlin Solution:

```
class Solution {\n    fun smallestAbsent(nums: IntArray): Int {\n\n    }\n}
```

Swift Solution:

```
class Solution {\n    func smallestAbsent(_ nums: [Int]) -> Int {\n\n    }\n}
```

Rust Solution:

```
// Problem: Smallest Absent Positive Greater Than Average\n// Difficulty: Easy\n// Tags: array, 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 smallest_absent(nums: Vec<i32>) -> i32 {
        }

    }
}

```

Ruby Solution:

```

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

end

```

PHP Solution:

```

class Solution {

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

    }
}

```

Dart Solution:

```

class Solution {
    int smallestAbsent(List<int> nums) {
        }

    }
}

```

Scala Solution:

```
object Solution {  
    def smallestAbsent(nums: Array[Int]): Int = {  
        }  
        }  
    }
```

Elixir Solution:

```
defmodule Solution do  
  @spec smallest_absent(list(integer)) :: integer  
  def smallest_absent(nums) do  
  
  end  
  end
```

Erlang Solution:

```
-spec smallest_absent(list(integer)) -> integer().  
smallest_absent(Nums) ->  
.
```

Racket Solution:

```
(define/contract (smallest-absent nums)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```