

Problem 1619: Mean of Array After Removing Some Elements

Problem Information

Difficulty: **Easy**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given an integer array

arr

, return

the mean of the remaining integers after removing the smallest

5%

and the largest

5%

of the elements.

Answers within

10

-5

of the

actual answer

will be considered accepted.

Example 1:

Input:

arr = [1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,3]

Output:

2.00000

Explanation:

After erasing the minimum and the maximum values of this array, all elements are equal to 2, so the mean is 2.

Example 2:

Input:

arr = [6,2,7,5,1,2,0,3,10,2,5,0,5,5,0,8,7,6,8,0]

Output:

4.00000

Example 3:

Input:

arr = [6,0,7,0,7,5,7,8,3,4,0,7,8,1,6,8,1,1,2,4,8,1,9,5,4,3,8,5,10,8,6,6,1,0,6,10,8,2,3,4]

Output:

4.77778

Constraints:

$20 \leq \text{arr.length} \leq 1000$

`arr.length`

is a multiple

of

`20`

.

$0 \leq \text{arr}[i] \leq 10$

`5`

Code Snippets

C++:

```
class Solution {  
public:  
    double trimMean(vector<int>& arr) {  
        }  
    };
```

Java:

```
class Solution {  
public double trimMean(int[] arr) {  
    }  
}
```

Python3:

```
class Solution:  
    def trimMean(self, arr: List[int]) -> float:
```

Python:

```
class Solution(object):
    def trimMean(self, arr):
        """
        :type arr: List[int]
        :rtype: float
        """

```

JavaScript:

```
/**
 * @param {number[]} arr
 * @return {number}
 */
var trimMean = function(arr) {

};


```

TypeScript:

```
function trimMean(arr: number[]): number {
}

;
```

C#:

```
public class Solution {
    public double TrimMean(int[] arr) {
    }
}
```

C:

```
double trimMean(int* arr, int arrSize) {
}

;
```

Go:

```
func trimMean(arr []int) float64 {
```

```
}
```

Kotlin:

```
class Solution {  
    fun trimMean(arr: IntArray): Double {  
  
    }  
}
```

Swift:

```
class Solution {  
    func trimMean(_ arr: [Int]) -> Double {  
  
    }  
}
```

Rust:

```
impl Solution {  
    pub fn trim_mean(arr: Vec<i32>) -> f64 {  
  
    }  
}
```

Ruby:

```
# @param {Integer[]} arr  
# @return {Float}  
def trim_mean(arr)  
  
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $arr  
     * @return Float  
     */
```

```
function trimMean($arr) {  
}  
}  
}
```

Dart:

```
class Solution {  
double trimMean(List<int> arr) {  
  
}  
}  
}
```

Scala:

```
object Solution {  
def trimMean(arr: Array[Int]): Double = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec trim_mean(arr :: [integer]) :: float  
def trim_mean(arr) do  
  
end  
end
```

Erlang:

```
-spec trim_mean(Arr :: [integer()]) -> float().  
trim_mean(Arr) ->  
.
```

Racket:

```
(define/contract (trim-mean arr)  
(-> (listof exact-integer?) flonum?)  
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Mean of Array After Removing Some Elements
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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:
    double trimMean(vector<int>& arr) {

    }
};
```

Java Solution:

```
/**
 * Problem: Mean of Array After Removing Some Elements
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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 double trimMean(int[] arr) {

    }
}
```

Python3 Solution:

```

"""
Problem: Mean of Array After Removing Some Elements
Difficulty: Easy
Tags: array, sort

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 trimMean(self, arr: List[int]) -> float:
    # TODO: Implement optimized solution
    pass

```

Python Solution:

```

class Solution(object):

def trimMean(self, arr):
    """
:type arr: List[int]
:rtype: float
"""

```

JavaScript Solution:

```

/**
 * Problem: Mean of Array After Removing Some Elements
 * Difficulty: Easy
 * Tags: array, sort
 *
 * 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
 */

var trimMean = function(arr) {

```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Mean of Array After Removing Some Elements  
 * Difficulty: Easy  
 * Tags: array, sort  
 *  
 * 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 trimMean(arr: number[]): number {  
  
};
```

C# Solution:

```
/*  
 * Problem: Mean of Array After Removing Some Elements  
 * Difficulty: Easy  
 * Tags: array, sort  
 *  
 * 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 double TrimMean(int[] arr) {  
  
    }  
}
```

C Solution:

```
/*  
 * Problem: Mean of Array After Removing Some Elements  
 * Difficulty: Easy
```

```

* Tags: array, sort
*
* 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
*/
double trimMean(int* arr, int arrSize) {
}

```

Go Solution:

```

// Problem: Mean of Array After Removing Some Elements
// Difficulty: Easy
// Tags: array, sort
//
// 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 trimMean(arr []int) float64 {
}

```

Kotlin Solution:

```

class Solution {
    fun trimMean(arr: IntArray): Double {
        }
    }
}
```

Swift Solution:

```

class Solution {
    func trimMean(_ arr: [Int]) -> Double {
        }
    }
}
```

Rust Solution:

```
// Problem: Mean of Array After Removing Some Elements
// Difficulty: Easy
// Tags: array, sort
//
// 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 trim_mean(arr: Vec<i32>) -> f64 {
        }

    }
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Float}
def trim_mean(arr)

end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $arr
     * @return Float
     */
    function trimMean($arr) {

    }
}
```

Dart Solution:

```
class Solution {
    double trimMean(List<int> arr) {
```

```
}
```

```
}
```

Scala Solution:

```
object Solution {  
    def trimMean(arr: Array[Int]): Double = {  
  
    }  
    }  
}
```

Elixir Solution:

```
defmodule Solution do  
  @spec trim_mean(list) :: float  
  def trim_mean(arr) do  
  
  end  
end
```

Erlang Solution:

```
-spec trim_mean(list) -> float().  
trim_mean(List) ->  
.
```

Racket Solution:

```
(define/contract (trim-mean arr)  
  (-> (listof exact-integer?) flonum?)  
)
```