# Problem 1300: Sum of Mutated Array Closest to Target

## Problem Information

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

## Problem Description

Given an integer array

arr

and a target value

target

, return the integer

value

such that when we change all the integers larger than

value

in the given array to be equal to

value

, the sum of the array gets as close as possible (in absolute difference) to

target

.

In case of a tie, return the minimum such integer.

Notice that the answer is not neccesarilly a number from

arr

.

Example 1:

Input:

arr = [4,9,3], target = 10

Output:

3

Explanation:

When using 3 arr converts to [3, 3, 3] which sums 9 and that's the optimal answer.

Example 2:

Input:

arr = [2,3,5], target = 10

Output:

5

Example 3:

Input:

arr = [60864,25176,27249,21296,20204], target = 56803

Output:

11361

Constraints:

1 <= arr.length <= 10

4

1 <= arr[i], target <= 10

5

## Code Snippets

**C++:**

```
class Solution {
public:
int findBestValue(vector<int>& arr, int target) {

}
};
```

**Java:**

```
class Solution {
public int findBestValue(int[] arr, int target) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function findBestValue(arr: number[], target: number): number {

};
```

**C#:**

```
public class Solution {
public int FindBestValue(int[] arr, int target) {

}
}
```

**C:**

```
int findBestValue(int* arr, int arrSize, int target) {

}
```

**Go:**

```
func findBestValue(arr []int, target int) int {
```

```
}
```

**Kotlin:**

```
class Solution {
fun findBestValue(arr: IntArray, target: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func findBestValue(_ arr: [Int], _ target: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn find_best_value(arr: Vec<i32>, target: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def find_best_value(arr, target)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $target
```

```
* @return Integer
*/
function findBestValue($arr, $target) {

}
}
```

**Dart:**

```
class Solution {
int findBestValue(List<int> arr, int target) {

}
}
```

**Scala:**

```
object Solution {
def findBestValue(arr: Array[Int], target: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec find_best_value(arr :: [integer], target :: integer) :: integer
def find_best_value(arr, target) do

end
end
```

**Erlang:**

```
-spec find_best_value(Arr :: [integer()], Target :: integer()) -> integer().
find_best_value(Arr, Target) ->
.
```

**Racket:**

```
(define/contract (find-best-value arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
```

```
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 findBestValue(vector<int>& arr, int target) {

}
};
```

### Java Solution:

```
/**
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 findBestValue(int[] arr, int target) {

}
}
```

### Python3 Solution:

```
"""
Problem: Sum of Mutated Array Closest to Target
Difficulty: Medium
Tags: array, sort, search

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 findBestValue(self, arr: List[int], target: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[]} arr
* @param {number} target
```

```
* @return {number}
*/
var findBestValue = function(arr, target) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 findBestValue(arr: number[], target: number): number {

};
```

**C# Solution:**

```
/*
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 FindBestValue(int[] arr, int target) {

}
}
```

**C Solution:**

```
/*
* Problem: Sum of Mutated Array Closest to Target
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 findBestValue(int* arr, int arrSize, int target) {

}
```

### Go Solution:

```
// Problem: Sum of Mutated Array Closest to Target
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 findBestValue(arr []int, target int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun findBestValue(arr: IntArray, target: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func findBestValue(_ arr: [Int], _ target: Int) -> Int {

}
```

```
}
```

### Rust Solution:

```
// Problem: Sum of Mutated Array Closest to Target
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 find_best_value(arr: Vec<i32>, target: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} arr
# @param {Integer} target
# @return {Integer}
def find_best_value(arr, target)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @param Integer $target
* @return Integer
*/
function findBestValue($arr, $target) {

}
}
```

### Dart Solution:

```
class Solution {
int findBestValue(List<int> arr, int target) {

}
}
```

### Scala Solution:

```
object Solution {
def findBestValue(arr: Array[Int], target: Int): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec find_best_value(arr :: [integer], target :: integer) :: integer
def find_best_value(arr, target) do

end
end
```

### Erlang Solution:

```
-spec find_best_value(Arr :: [integer()], Target :: integer()) -> integer().
find_best_value(Arr, Target) ->
.
```

### Racket Solution:

```
(define/contract (find-best-value arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```