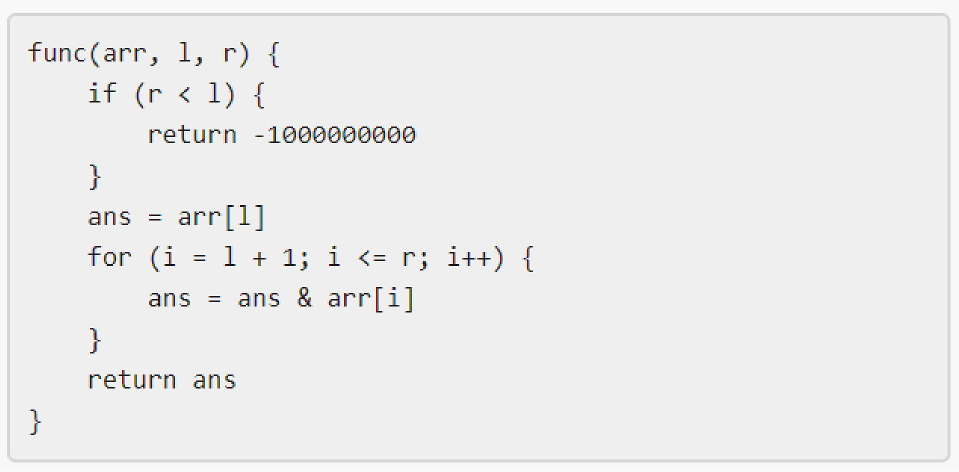
Problem 1521: Find a Value of a Mysterious Function Closest to Target

Problem Information

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

Problem Description



```
func(arr, l, r) {
    if (r < l) {
        return -1000000000
    }
    ans = arr[l]
    for (i = l + 1; i <= r; i++) {
        ans = ans & arr[i]
    }
    return ans
}
```

Winston was given the above mysterious function

func

. He has an integer array

arr

and an integer

target

and he wants to find the values

l

and

r

that make the value

|func(arr, l, r) - target|

minimum possible.

Return

the minimum possible value

of

|func(arr, l, r) - target|

.

Notice that

func

should be called with the values

l

and

r

where

0 <= l, r < arr.length

.

Example 1:

Input:

arr = [9,12,3,7,15], target = 5

Output:

2

Explanation:

Calling func with all the pairs of [l,r] = [[0,0],[1,1],[2,2],[3,3],[4,4],[0,1],[1,2],[2,3],[3,4],[0,2],[1,3],[2,4],[0,3],[1,4],[0,4]], Winston got the following results [9,12,3,7,15,8,0,3,7,0,0,3,0,0,0]. The value closest to 5 is 7 and 3, thus the minimum difference is 2.

Example 2:

Input:

arr = [1000000,1000000,1000000], target = 1

Output:

999999

Explanation:

Winston called the func with all possible values of [l,r] and he always got 1000000, thus the min difference is 999999.

Example 3:

Input:

arr = [1,2,4,8,16], target = 0

Output:

0

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i] <= 10

6

0 <= target <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (closest-to-target arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Find a Value of a Mysterious Function Closest to Target
Difficulty: Hard
Tags: array, tree, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function closestToTarget(arr: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

}

}
```

C Solution:

```
/*
* Problem: Find a Value of a Mysterious Function Closest to Target
* Difficulty: Hard
* Tags: array, tree, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int closestToTarget(int* arr, int arrSize, int target) {

}
```

Go Solution:

```
// Problem: Find a Value of a Mysterious Function Closest to Target
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func closestToTarget(arr []int, target int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find a Value of a Mysterious Function Closest to Target
// Difficulty: Hard
// Tags: array, tree, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (closest-to-target arr target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```