Problem 1431: Kids With the Greatest Number of Candies

Problem Information

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

Problem Description

There are

n

kids with candies. You are given an integer array

candies

, where each

candies[i]

represents the number of candies the

i

th

kid has, and an integer

extraCandies

, denoting the number of extra candies that you have.

Return

a boolean array

result

of length

n

, where

result[i]

is

true

if, after giving the

i

th

kid all the

extraCandies

, they will have the

greatest

number of candies among all the kids

, or

false

otherwise

.

Note that

multiple

kids can have the

greatest

number of candies.

Example 1:

Input:

candies = [2,3,5,1,3], extraCandies = 3

Output:

[true,true,true,false,true]

Explanation:

If you give all extraCandies to: - Kid 1, they will have 2 + 3 = 5 candies, which is the greatest among the kids. - Kid 2, they will have 3 + 3 = 6 candies, which is the greatest among the kids. - Kid 3, they will have 5 + 3 = 8 candies, which is the greatest among the kids. - Kid 4, they will have 1 + 3 = 4 candies, which is not the greatest among the kids. - Kid 5, they will have 3 + 3 = 6 candies, which is the greatest among the kids.

Example 2:

Input:

candies = [4,2,1,1,2], extraCandies = 1

Output:

[true,false,false,false,false]

Explanation:

There is only 1 extra candy. Kid 1 will always have the greatest number of candies, even if a different kid is given the extra candy.

Example 3:

Input:

candies = [12,1,12], extraCandies = 10

Output:

[true,false,true]

Constraints:

n == candies.length

2 <= n <= 100

1 <= candies[i] <= 100

1 <= extraCandies <= 50

Code Snippets

C++:

```
class Solution {
public:
vector<bool> kidsWithCandies(vector<int>& candies, int extraCandies) {

}
};
```

Java:

```
class Solution {
public List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {

}
}
```

Python3:

```
class Solution:
def kidsWithCandies(self, candies: List[int], extraCandies: int) ->
List[bool]:
```

Python:

```
class Solution(object):
def kidsWithCandies(self, candies, extraCandies):
"""
:type candies: List[int]
:type extraCandies: int
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {number[]} candies
* @param {number} extraCandies
* @return {boolean[]}
*/
var kidsWithCandies = function(candies, extraCandies) {

};
```

TypeScript:

```
function kidsWithCandies(candies: number[], extraCandies: number): boolean[]
{

};
```

C#:

```
public class Solution {
public IList<bool> KidsWithCandies(int[] candies, int extraCandies) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* kidsWithCandies(int* candies, int candiesSize, int extraCandies, int*
returnSize) {

}
```

Go:

```
func kidsWithCandies(candies []int, extraCandies int) []bool {

}
```

Kotlin:

```
class Solution {
fun kidsWithCandies(candies: IntArray, extraCandies: Int): List<Boolean> {

}
}
```

Swift:

```
class Solution {
func kidsWithCandies(_ candies: [Int], _ extraCandies: Int) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn kids_with_candies(candies: Vec<i32>, extra_candies: i32) -> Vec<bool>
{
```

```
}
}
```

Ruby:

```
# @param {Integer[]} candies
# @param {Integer} extra_candies
# @return {Boolean[]}
def kids_with_candies(candies, extra_candies)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $extraCandies
* @return Boolean[]
*/
function kidsWithCandies($candies, $extraCandies) {

}
}
```

Dart:

```
class Solution {
List<bool> kidsWithCandies(List<int> candies, int extraCandies) {

}
}
```

Scala:

```
object Solution {
def kidsWithCandies(candies: Array[Int], extraCandies: Int): List[Boolean] =
{

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec kids_with_candies(candies :: [integer], extra_candies :: integer) ::
[boolean]
def kids_with_candies(candies, extra_candies) do

end
end
```

Erlang:

```
-spec kids_with_candies(Candies :: [integer()], ExtraCandies :: integer()) ->
[boolean()].
kids_with_candies(Candies, ExtraCandies) ->
.
```

Racket:

```
(define/contract (kids-with-candies candies extraCandies)
(-> (listof exact-integer?) exact-integer? (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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:
vector<bool> kidsWithCandies(vector<int>& candies, int extraCandies) {

}
};
```

Java Solution:

```
/**
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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 List<Boolean> kidsWithCandies(int[] candies, int extraCandies) {

}
}
```

Python3 Solution:

```
"""
Problem: Kids With the Greatest Number of Candies
Difficulty: Easy
Tags: array

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 kidsWithCandies(self, candies: List[int], extraCandies: int) ->
List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def kidsWithCandies(self, candies, extraCandies):
"""
:type candies: List[int]
:type extraCandies: int
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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[]} candies
* @param {number} extraCandies
* @return {boolean[]}
*/
var kidsWithCandies = function(candies, extraCandies) {

};
```

TypeScript Solution:

```
/**
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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 kidsWithCandies(candies: number[], extraCandies: number): boolean[]
{

};
```

C# Solution:

```
/*
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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 IList<bool> KidsWithCandies(int[] candies, int extraCandies) {

}
}
```

C Solution:

```
/*
* Problem: Kids With the Greatest Number of Candies
* Difficulty: Easy
* Tags: array
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* kidsWithCandies(int* candies, int candiesSize, int extraCandies, int*
returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Kids With the Greatest Number of Candies
// Difficulty: Easy
// Tags: array
//
// 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 kidsWithCandies(candies []int, extraCandies int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun kidsWithCandies(candies: IntArray, extraCandies: Int): List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func kidsWithCandies(_ candies: [Int], _ extraCandies: Int) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Kids With the Greatest Number of Candies
// Difficulty: Easy
// Tags: array
//
// 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 kids_with_candies(candies: Vec<i32>, extra_candies: i32) -> Vec<bool>
{

}
}
```

Ruby Solution:

```
# @param {Integer[]} candies
# @param {Integer} extra_candies
# @return {Boolean[]}
def kids_with_candies(candies, extra_candies)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $extraCandies
* @return Boolean[]
*/
function kidsWithCandies($candies, $extraCandies) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> kidsWithCandies(List<int> candies, int extraCandies) {

}
}
```

Scala Solution:

```
object Solution {
def kidsWithCandies(candies: Array[Int], extraCandies: Int): List[Boolean] =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec kids_with_candies(candies :: [integer], extra_candies :: integer) ::
[boolean]
def kids_with_candies(candies, extra_candies) do

end
end
```

Erlang Solution:

```
-spec kids_with_candies(Candies :: [integer()], ExtraCandies :: integer()) ->
[boolean()].
kids_with_candies(Candies, ExtraCandies) ->
.
```

Racket Solution:

```
(define/contract (kids-with-candies candies extraCandies)
(-> (listof exact-integer?) exact-integer? (listof boolean?))
)
```