Problem 2226: Maximum Candies Allocated to K Children

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

candies

. Each element in the array denotes a pile of candies of size

candies[i]

. You can divide each pile into any number of

sub piles

, but you

cannot

merge two piles together.

You are also given an integer

k

. You should allocate piles of candies to

k

children such that each child gets the

same

number of candies. Each child can be allocated candies from

only one

pile of candies and some piles of candies may go unused.

Return

the

maximum number of candies

each child can get.

Example 1:

Input:

candies = [5,8,6], k = 3

Output:

5

Explanation:

We can divide candies[1] into 2 piles of size 5 and 3, and candies[2] into 2 piles of size 5 and 1. We now have five piles of candies of sizes 5, 5, 3, 5, and 1. We can allocate the 3 piles of size 5 to 3 children. It can be proven that each child cannot receive more than 5 candies.

Example 2:

Input:

candies = [2,5], k = 11

Output:

0

Explanation:

There are 11 children but only 7 candies in total, so it is impossible to ensure each child receives at least one candy. Thus, each child gets no candy and the answer is 0.

Constraints:

1 <= candies.length <= 10

5

1 <= candies[i] <= 10

7

1 <= k <= 10

12

Code Snippets

C++:

```
class Solution {
public:
int maximumCandies(vector<int>& candies, long long k) {

}
};
```

Java:

```
class Solution {
public int maximumCandies(int[] candies, long k) {

}
}
```

Python3:

```
class Solution:
def maximumCandies(self, candies: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def maximumCandies(self, candies, k):
"""
:type candies: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} candies
* @param {number} k
* @return {number}
*/
var maximumCandies = function(candies, k) {

};
```

TypeScript:

```
function maximumCandies(candies: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximumCandies(int[] candies, long k) {

}
}
```

C:

```
int maximumCandies(int* candies, int candiesSize, long long k) {

}
```

Go:

```
func maximumCandies(candies []int, k int64) int {

}
```

Kotlin:

```
class Solution {
fun maximumCandies(candies: IntArray, k: Long): Int {

}
}
```

Swift:

```
class Solution {
func maximumCandies(_ candies: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_candies(candies: Vec<i32>, k: i64) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} candies
# @param {Integer} k
# @return {Integer}
def maximum_candies(candies, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $k
* @return Integer
*/
function maximumCandies($candies, $k) {

}
}
```

Dart:

```
class Solution {
int maximumCandies(List<int> candies, int k) {

}
}
```

Scala:

```
object Solution {
def maximumCandies(candies: Array[Int], k: Long): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_candies(candies :: [integer], k :: integer) :: integer
def maximum_candies(candies, k) do
```

```
end
end
```

Erlang:

```
-spec maximum_candies(Candies :: [integer()], K :: integer()) -> integer().
maximum_candies(Candies, K) ->
.
```

Racket:

```
(define/contract (maximum-candies candies k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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 maximumCandies(vector<int>& candies, long long k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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 maximumCandies(int[] candies, long k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Candies Allocated to K Children
Difficulty: Medium
Tags: array, 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 maximumCandies(self, candies: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumCandies(self, candies, k):
"""
:type candies: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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[]} candies
* @param {number} k
* @return {number}
*/
var maximumCandies = function(candies, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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 maximumCandies(candies: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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 MaximumCandies(int[] candies, long k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Candies Allocated to K Children
* Difficulty: Medium
* Tags: array, 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 maximumCandies(int* candies, int candiesSize, long long k) {

}
```

Go Solution:

```
// Problem: Maximum Candies Allocated to K Children
// Difficulty: Medium
// Tags: array, 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 maximumCandies(candies []int, k int64) int {


}
```

Kotlin Solution:

```
class Solution {
fun maximumCandies(candies: IntArray, k: Long): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumCandies(_ candies: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Candies Allocated to K Children
// Difficulty: Medium
// Tags: array, 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 maximum_candies(candies: Vec<i32>, k: i64) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} candies
# @param {Integer} k
# @return {Integer}
def maximum_candies(candies, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $candies
* @param Integer $k
* @return Integer
*/
function maximumCandies($candies, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumCandies(List<int> candies, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maximumCandies(candies: Array[Int], k: Long): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_candies(candies :: [integer], k :: integer) :: integer
def maximum_candies(candies, k) do

end
end
```

Erlang Solution:

```
-spec maximum_candies(Candies :: [integer()], K :: integer()) -> integer().
maximum_candies(Candies, K) ->
.
```

Racket Solution:

```
(define/contract (maximum-candies candies k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```