Problem 2558: Take Gifts From the Richest Pile

Problem Information

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

Problem Description

You are given an integer array

gifts

denoting the number of gifts in various piles. Every second, you do the following:

Choose the pile with the maximum number of gifts.

If there is more than one pile with the maximum number of gifts, choose any.

Reduce the number of gifts in the pile to the floor of the square root of the original number of gifts in the pile.

Return

the number of gifts remaining after

k

seconds.

Example 1:

Input:

gifts = [25,64,9,4,100], k = 4

Output:

29

Explanation:

The gifts are taken in the following way: - In the first second, the last pile is chosen and 10 gifts are left behind. - Then the second pile is chosen and 8 gifts are left behind. - After that the first pile is chosen and 5 gifts are left behind. - Finally, the last pile is chosen again and 3 gifts are left behind. The final remaining gifts are [5,8,9,4,3], so the total number of gifts remaining is 29.

Example 2:

Input:

gifts = [1,1,1,1], k = 4

Output:

4

Explanation:

In this case, regardless which pile you choose, you have to leave behind 1 gift in each pile. That is, you can't take any pile with you. So, the total gifts remaining are 4.

Constraints:

1 <= gifts.length <= 10

3

1 <= gifts[i] <= 10

9

1 <= k <= 10

3

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long pickGifts(int[] gifts, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function pickGifts(gifts: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long PickGifts(int[] gifts, int k) {

}
}
```

C:

```
long long pickGifts(int* gifts, int giftsSize, int k) {

}
```

Go:

```
func pickGifts(gifts []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun pickGifts(gifts: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} gifts
# @param {Integer} k
# @return {Integer}
def pick_gifts(gifts, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $gifts
* @param Integer $k
* @return Integer
*/
function pickGifts($gifts, $k) {

}
}
```

Dart:

```
class Solution {
int pickGifts(List<int> gifts, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec pick_gifts(gifts :: [integer], k :: integer) :: integer
def pick_gifts(gifts, k) do

end
end
```

Erlang:

```
-spec pick_gifts(Gifts :: [integer()], K :: integer()) -> integer().
pick_gifts(Gifts, K) ->
.
```

Racket:

```
(define/contract (pick-gifts gifts k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
```

```
* 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:
long long pickGifts(vector<int>& gifts, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
* 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 long pickGifts(int[] gifts, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Take Gifts From the Richest Pile
Difficulty: Easy
Tags: array, queue, heap

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 pickGifts(self, gifts: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
* 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[]} gifts
* @param {number} k
* @return {number}
*/
var pickGifts = function(gifts, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
* 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 pickGifts(gifts: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
* 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 long PickGifts(int[] gifts, int k) {

}
}
```

C Solution:

```
/*
* Problem: Take Gifts From the Richest Pile
* Difficulty: Easy
* Tags: array, queue, heap
*
* 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
*/
```

```
long long pickGifts(int* gifts, int giftsSize, int k) {

}
```

Go Solution:

```
// Problem: Take Gifts From the Richest Pile
// Difficulty: Easy
// Tags: array, queue, heap
//
// 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 pickGifts(gifts []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun pickGifts(gifts: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Take Gifts From the Richest Pile
// Difficulty: Easy
// Tags: array, queue, heap
//
```

```
// 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 pick_gifts(gifts: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} gifts
# @param {Integer} k
# @return {Integer}
def pick_gifts(gifts, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $gifts
* @param Integer $k
* @return Integer
*/
function pickGifts($gifts, $k) {

}
}
```

Dart Solution:

```
class Solution {
int pickGifts(List<int> gifts, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pick_gifts(gifts :: [integer], k :: integer) :: integer
def pick_gifts(gifts, k) do

end
end
```

Erlang Solution:

```
-spec pick_gifts(Gifts :: [integer()], K :: integer()) -> integer().
pick_gifts(Gifts, K) ->
.
```

Racket Solution:

```
(define/contract (pick-gifts gifts k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```