Problem 3075: Maximize Happiness of Selected Children

Problem Information

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

Problem Description

You are given an array

happiness

of length

n

, and a

positive

integer

k

.

There are

n

children standing in a queue, where the

i

th

child has

happiness value

happiness[i]

. You want to select

k

children from these

n

children in

k

turns.

In each turn, when you select a child, the

happiness value

of all the children that have

not

been selected till now decreases by

1

. Note that the happiness value

cannot

become negative and gets decremented

only

if it is positive.

Return

the

maximum

sum of the happiness values of the selected children you can achieve by selecting

k

children

.

Example 1:

Input:

happiness = [1,2,3], k = 2

Output:

4

Explanation:

We can pick 2 children in the following way: - Pick the child with the happiness value == 3. The happiness value of the remaining children becomes [0,1]. - Pick the child with the happiness value == 1. The happiness value of the remaining child becomes [0]. Note that the happiness value cannot become less than 0. The sum of the happiness values of the selected children is 3 + 1 = 4.

Example 2:

Input:

happiness = [1,1,1,1], k = 2

Output:

1

Explanation:

We can pick 2 children in the following way: - Pick any child with the happiness value == 1. The happiness value of the remaining children becomes [0,0,0]. - Pick the child with the happiness value == 0. The happiness value of the remaining child becomes [0,0]. The sum of the happiness values of the selected children is 1 + 0 = 1.

Example 3:

Input:

happiness = [2,3,4,5], k = 1

Output:

5

Explanation:

We can pick 1 child in the following way: - Pick the child with the happiness value == 5. The happiness value of the remaining children becomes [1,2,3]. The sum of the happiness values of the selected children is 5.

Constraints:

1 <= n == happiness.length <= 2 * 10

5

1 <= happiness[i] <= 10

8

1 <= k <= n

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maximumHappinessSum(int[] happiness, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumHappinessSum(happiness: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MaximumHappinessSum(int[] happiness, int k) {

}
}
```

C:

```
long long maximumHappinessSum(int* happiness, int happinessSize, int k) {

}
```

Go:

```
func maximumHappinessSum(happiness []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumHappinessSum(happiness: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} happiness
# @param {Integer} k
# @return {Integer}
def maximum_happiness_sum(happiness, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $happiness
* @param Integer $k
* @return Integer
*/
function maximumHappinessSum($happiness, $k) {

}
}
```

Dart:

```
class Solution {
int maximumHappinessSum(List<int> happiness, int k) {



}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_happiness_sum(happiness :: [integer], k :: integer) :: integer
def maximum_happiness_sum(happiness, k) do

end
end
```

Erlang:

```
-spec maximum_happiness_sum(Happiness :: [integer()], K :: integer()) ->
integer().
maximum_happiness_sum(Happiness, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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 maximumHappinessSum(vector<int>& happiness, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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 maximumHappinessSum(int[] happiness, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Happiness of Selected Children
Difficulty: Medium
Tags: array, greedy, sort, queue

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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[]} happiness
* @param {number} k
* @return {number}
*/
var maximumHappinessSum = function(happiness, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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 maximumHappinessSum(happiness: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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 MaximumHappinessSum(int[] happiness, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Happiness of Selected Children
* Difficulty: Medium
* Tags: array, greedy, sort, queue
*
* 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 maximumHappinessSum(int* happiness, int happinessSize, int k) {

}
```

Go Solution:

```go
// Problem: Maximize Happiness of Selected Children
// Difficulty: Medium
// Tags: array, greedy, sort, queue
//
// 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 maximumHappinessSum(happiness []int, k int) int64 {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maximumHappinessSum(happiness: IntArray, k: Int): Long {

}
}
```

Swift Solution:

```swift
class Solution {
func maximumHappinessSum(_ happiness: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```rust
// Problem: Maximize Happiness of Selected Children
// Difficulty: Medium
// Tags: array, greedy, sort, queue
```

```
//
// 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_happiness_sum(happiness: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} happiness
# @param {Integer} k
# @return {Integer}
def maximum_happiness_sum(happiness, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $happiness
* @param Integer $k
* @return Integer
*/
function maximumHappinessSum($happiness, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maximumHappinessSum(List<int> happiness, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_happiness_sum(happiness :: [integer], k :: integer) :: integer
def maximum_happiness_sum(happiness, k) do

end
end
```

Erlang Solution:

```
-spec maximum_happiness_sum(Happiness :: [integer()], K :: integer()) ->
integer().
maximum_happiness_sum(Happiness, K) ->
.
```

Racket Solution:

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