Problem 1402: Reducing Dishes

Problem Information

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

Problem Description

A chef has collected data on the

satisfaction

level of his

n

dishes. Chef can cook any dish in 1 unit of time.

Like-time coefficient

of a dish is defined as the time taken to cook that dish including previous dishes multiplied by its satisfaction level i.e.

time[i] * satisfaction[i]

.

Return the maximum sum of

like-time coefficient

that the chef can obtain after preparing some amount of dishes.

Dishes can be prepared in

any

order and the chef can discard some dishes to get this maximum value.

Example 1:

Input:

satisfaction = [-1,-8,0,5,-9]

Output:

14

Explanation:

After Removing the second and last dish, the maximum total

like-time coefficient

will be equal to (-1*1 + 0*2 + 5*3 = 14). Each dish is prepared in one unit of time.

Example 2:

Input:

satisfaction = [4,3,2]

Output:

20

Explanation:

Dishes can be prepared in any order, (2*1 + 3*2 + 4*3 = 20)

Example 3:

Input:

satisfaction = [-1,-4,-5]

Output:

0

Explanation:

People do not like the dishes. No dish is prepared.

Constraints:

n == satisfaction.length

1 <= n <= 500

-1000 <= satisfaction[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int maxSatisfaction(vector<int>& satisfaction) {

}
};
```

Java:

```
class Solution {
public int maxSatisfaction(int[] satisfaction) {

}
}
```

Python3:

```
class Solution:
def maxSatisfaction(self, satisfaction: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxSatisfaction(self, satisfaction):
"""
:type satisfaction: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} satisfaction
* @return {number}
*/
var maxSatisfaction = function(satisfaction) {

};
```

TypeScript:

```
function maxSatisfaction(satisfaction: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxSatisfaction(int[] satisfaction) {

}
}
```

C:

```
int maxSatisfaction(int* satisfaction, int satisfactionSize) {

}
```

Go:

```
func maxSatisfaction(satisfaction []int) int {

}
```

Kotlin:

```
class Solution {
fun maxSatisfaction(satisfaction: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxSatisfaction(_ satisfaction: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_satisfaction(satisfaction: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} satisfaction
# @return {Integer}
def max_satisfaction(satisfaction)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $satisfaction
* @return Integer
*/
function maxSatisfaction($satisfaction) {

}
}
```

Dart:

```
class Solution {
int maxSatisfaction(List<int> satisfaction) {

}
}
```

Scala:

```
object Solution {
def maxSatisfaction(satisfaction: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_satisfaction(satisfaction :: [integer]) :: integer
def max_satisfaction(satisfaction) do

end
end
```

Erlang:

```
-spec max_satisfaction(Satisfaction :: [integer()]) -> integer().
max_satisfaction(Satisfaction) ->
.
```

Racket:

```
(define/contract (max-satisfaction satisfaction)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxSatisfaction(vector<int>& satisfaction) {

}
};
```

Java Solution:

```
/**
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxSatisfaction(int[] satisfaction) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Reducing Dishes
Difficulty: Hard
Tags: array, dp, greedy, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

```
class Solution(object):
def maxSatisfaction(self, satisfaction):
"""
:type satisfaction: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

```
* @param {number[]} satisfaction
* @return {number}
*/
var maxSatisfaction = function(satisfaction) {

};
```

TypeScript Solution:

```
/**
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxSatisfaction(satisfaction: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxSatisfaction(int[] satisfaction) {

}
}
```

C Solution:

```
/*
* Problem: Reducing Dishes
* Difficulty: Hard
* Tags: array, dp, greedy, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxSatisfaction(int* satisfaction, int satisfactionSize) {

}
```

Go Solution:

```
// Problem: Reducing Dishes
// Difficulty: Hard
// Tags: array, dp, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxSatisfaction(satisfaction []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSatisfaction(satisfaction: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSatisfaction(_ satisfaction: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Reducing Dishes
// Difficulty: Hard
// Tags: array, dp, greedy, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_satisfaction(satisfaction: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} satisfaction
# @return {Integer}
def max_satisfaction(satisfaction)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $satisfaction
* @return Integer
*/
function maxSatisfaction($satisfaction) {

}
}
```

Dart Solution:

```
class Solution {
int maxSatisfaction(List<int> satisfaction) {

}
}
```

Scala Solution:

```
object Solution {
def maxSatisfaction(satisfaction: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_satisfaction(satisfaction :: [integer]) :: integer
def max_satisfaction(satisfaction) do

end
end
```

Erlang Solution:

```
-spec max_satisfaction(Satisfaction :: [integer()]) -> integer().
max_satisfaction(Satisfaction) ->
.
```

Racket Solution:

```
(define/contract (max-satisfaction satisfaction)
(-> (listof exact-integer?) exact-integer?)
)
```