

Problem 2431: Maximize Total Tastiness of Purchased Fruits

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given two non-negative integer arrays

`price`

and

`tastiness`

, both arrays have the same length

`n`

. You are also given two non-negative integers

`maxAmount`

and

`maxCoupons`

.

For every integer

`i`

in range

$[0, n - 1]$

:

`price[i]`

describes the price of

i

th

fruit.

`tastiness[i]`

describes the tastiness of

i

th

fruit.

You want to purchase some fruits such that total tastiness is maximized and the total price does not exceed

`maxAmount`

.

Additionally, you can use a coupon to purchase fruit for

half of its price

(rounded down to the closest integer). You can use at most

maxCoupons

of such coupons.

Return

the maximum total tastiness that can be purchased

.

Note that:

You can purchase each fruit at most once.

You can use coupons on some fruit at most once.

Example 1:

Input:

price = [10,20,20], tastiness = [5,8,8], maxAmount = 20, maxCoupons = 1

Output:

13

Explanation:

It is possible to make total tastiness 13 in following way: - Buy first fruit without coupon, so that total price = 0 + 10 and total tastiness = 0 + 5. - Buy second fruit with coupon, so that total price = 10 + 10 and total tastiness = 5 + 8. - Do not buy third fruit, so that total price = 20 and total tastiness = 13. It can be proven that 13 is the maximum total tastiness that can be obtained.

Example 2:

Input:

price = [10,15,7], tastiness = [5,8,20], maxAmount = 10, maxCoupons = 2

Output:

28

Explanation:

It is possible to make total tastiness 20 in following way: - Do not buy first fruit, so that total price = 0 and total tastiness = 0. - Buy second fruit with coupon, so that total price = 0 + 7 and total tastiness = 0 + 8. - Buy third fruit with coupon, so that total price = 7 + 3 and total tastiness = 8 + 20. It can be proven that 28 is the maximum total tastiness that can be obtained.

Constraints:

$n == \text{price.length} == \text{tastiness.length}$

$1 \leq n \leq 100$

$0 \leq \text{price}[i], \text{tastiness}[i], \text{maxAmount} \leq 1000$

$0 \leq \text{maxCoupons} \leq 5$

Code Snippets

C++:

```
class Solution {
public:
    int maxTastiness(vector<int>& price, vector<int>& tastiness, int maxAmount,
                    int maxCoupons) {

    }
};
```

Java:

```
class Solution {
    public int maxTastiness(int[] price, int[] tastiness, int maxAmount, int
    maxCoupons) {
```

```
}  
}
```

Python3:

```
class Solution:  
    def maxTastiness(self, price: List[int], tastiness: List[int], maxAmount:  
int, maxCoupons: int) -> int:
```

Python:

```
class Solution(object):  
    def maxTastiness(self, price, tastiness, maxAmount, maxCoupons):  
        """  
        :type price: List[int]  
        :type tastiness: List[int]  
        :type maxAmount: int  
        :type maxCoupons: int  
        :rtype: int  
        """
```

JavaScript:

```
/**  
 * @param {number[]} price  
 * @param {number[]} tastiness  
 * @param {number} maxAmount  
 * @param {number} maxCoupons  
 * @return {number}  
 */  
var maxTastiness = function(price, tastiness, maxAmount, maxCoupons) {  
  
};
```

TypeScript:

```
function maxTastiness(price: number[], tastiness: number[], maxAmount:  
number, maxCoupons: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int MaxTastiness(int[] price, int[] tastiness, int maxAmount, int  
    maxCoupons) {  
  
    }  
}
```

C:

```
int maxTastiness(int* price, int priceSize, int* tastiness, int  
tastinessSize, int maxAmount, int maxCoupons) {  
  
}
```

Go:

```
func maxTastiness(price []int, tastiness []int, maxAmount int, maxCoupons  
int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxTastiness(price: IntArray, tastiness: IntArray, maxAmount: Int,  
    maxCoupons: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxTastiness(_ price: [Int], _ tastiness: [Int], _ maxAmount: Int, _  
    maxCoupons: Int) -> Int {  
  
    }  
}
```

Rust:

```

impl Solution {
    pub fn max_tastiness(price: Vec<i32>, tastiness: Vec<i32>, max_amount: i32,
max_coupons: i32) -> i32 {

    }

}

```

Ruby:

```

# @param {Integer[]} price
# @param {Integer[]} tastiness
# @param {Integer} max_amount
# @param {Integer} max_coupons
# @return {Integer}
def max_tastiness(price, tastiness, max_amount, max_coupons)

end

```

PHP:

```

class Solution {

    /**
     * @param Integer[] $price
     * @param Integer[] $tastiness
     * @param Integer $maxAmount
     * @param Integer $maxCoupons
     * @return Integer
     */
    function maxTastiness($price, $tastiness, $maxAmount, $maxCoupons) {

    }

}

```

Dart:

```

class Solution {
    int maxTastiness(List<int> price, List<int> tastiness, int maxAmount, int
maxCoupons) {

    }

}

```

Scala:

```
object Solution {  
  def maxTastiness(price: Array[Int], tastiness: Array[Int], maxAmount: Int,  
    maxCoupons: Int): Int = {  
  
  }  
}
```

Elixir:

```
defmodule Solution do  
  @spec max_tastiness(price :: [integer], tastiness :: [integer], max_amount ::  
    integer, max_coupons :: integer) :: integer  
  def max_tastiness(price, tastiness, max_amount, max_coupons) do  
  
  end  
end
```

Erlang:

```
-spec max_tastiness(Price :: [integer()], Tastiness :: [integer()], MaxAmount  
  :: integer(), MaxCoupons :: integer()) -> integer().  
max_tastiness(Price, Tastiness, MaxAmount, MaxCoupons) ->  
  .
```

Racket:

```
(define/contract (max-tastiness price tastiness maxAmount maxCoupons)  
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?  
    exact-integer? exact-integer?)  
  )
```

Solutions

C++ Solution:

```
/*  
 * Problem: Maximize Total Tastiness of Purchased Fruits  
 * Difficulty: Medium  
 * Tags: array, dp  
 */
```

```

* 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 maxTastiness(vector<int>& price, vector<int>& tastiness, int maxAmount,
    int maxCoupons) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximize Total Tastiness of Purchased Fruits
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTastiness(int[] price, int[] tastiness, int maxAmount, int
    maxCoupons) {

    }
}

```

Python3 Solution:

```

"""
Problem: Maximize Total Tastiness of Purchased Fruits
Difficulty: Medium
Tags: array, dp

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 maxTastiness(self, price: List[int], tastiness: List[int], maxAmount:
int, maxCoupons: int) -> int:
```

```
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
```

```
def maxTastiness(self, price, tastiness, maxAmount, maxCoupons):
```

```
"""
```

```
:type price: List[int]
```

```
:type tastiness: List[int]
```

```
:type maxAmount: int
```

```
:type maxCoupons: int
```

```
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
```

```
* Problem: Maximize Total Tastiness of Purchased Fruits
```

```
* Difficulty: Medium
```

```
* Tags: array, dp
```

```
*
```

```
* 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[]} price
```

```
* @param {number[]} tastiness
```

```
* @param {number} maxAmount
```

```
* @param {number} maxCoupons
```

```
* @return {number}
```

```
*/
```

```
var maxTastiness = function(price, tastiness, maxAmount, maxCoupons) {
```

```
};
```

TypeScript Solution:

```
/**
 * Problem: Maximize Total Tastiness of Purchased Fruits
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTastiness(price: number[], tastiness: number[], maxAmount:
number, maxCoupons: number): number {

};
```

C# Solution:

```
/*
 * Problem: Maximize Total Tastiness of Purchased Fruits
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 MaxTastiness(int[] price, int[] tastiness, int maxAmount, int
maxCoupons) {

    }
}
```

C Solution:

```

/*
 * Problem: Maximize Total Tastiness of Purchased Fruits
 * Difficulty: Medium
 * Tags: array, dp
 *
 * 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 maxTastiness(int* price, int priceSize, int* tastiness, int
tastinessSize, int maxAmount, int maxCoupons) {

}

```

Go Solution:

```

// Problem: Maximize Total Tastiness of Purchased Fruits
// Difficulty: Medium
// Tags: array, dp
//
// 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 maxTastiness(price []int, tastiness []int, maxAmount int, maxCoupons
int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun maxTastiness(price: IntArray, tastiness: IntArray, maxAmount: Int,
maxCoupons: Int): Int {

    }
}

```

Swift Solution:

```

class Solution {
    func maxTastiness(_ price: [Int], _ tastiness: [Int], _ maxAmount: Int, _
maxCoupons: Int) -> Int {

    }

}

```

Rust Solution:

```

// Problem: Maximize Total Tastiness of Purchased Fruits
// Difficulty: Medium
// Tags: array, dp
//
// 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_tastiness(price: Vec<i32>, tastiness: Vec<i32>, max_amount: i32,
max_coupons: i32) -> i32 {

    }

}

```

Ruby Solution:

```

# @param {Integer[]} price
# @param {Integer[]} tastiness
# @param {Integer} max_amount
# @param {Integer} max_coupons
# @return {Integer}
def max_tastiness(price, tastiness, max_amount, max_coupons)

end

```

PHP Solution:

```

class Solution {

    /**
     * @param Integer[] $price
     * @param Integer[] $tastiness

```

```

* @param Integer $maxAmount
* @param Integer $maxCoupons
* @return Integer
*/
function maxTastiness($price, $tastiness, $maxAmount, $maxCoupons) {

}
}

```

Dart Solution:

```

class Solution {
  int maxTastiness(List<int> price, List<int> tastiness, int maxAmount, int
maxCoupons) {

  }
}

```

Scala Solution:

```

object Solution {
  def maxTastiness(price: Array[Int], tastiness: Array[Int], maxAmount: Int,
maxCoupons: Int): Int = {

  }
}

```

Elixir Solution:

```

defmodule Solution do
  @spec max_tastiness(price :: [integer], tastiness :: [integer], max_amount ::
integer, max_coupons :: integer) :: integer
  def max_tastiness(price, tastiness, max_amount, max_coupons) do

  end
end

```

Erlang Solution:

```

-spec max_tastiness(Price :: [integer()], Tastiness :: [integer()], MaxAmount
:: integer(), MaxCoupons :: integer()) -> integer().

```

```
max_tastiness(Price, Tastiness, MaxAmount, MaxCoupons) ->  
.
```

Racket Solution:

```
(define/contract (max-tastiness price tastiness maxAmount maxCoupons)  
  (-> (listof exact-integer?) (listof exact-integer?) exact-integer?  
    exact-integer? exact-integer?)  
  )
```