Problem 2517: Maximum Tastiness of Candy Basket

Problem Information

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

Problem Description

You are given an array of positive integers

price

where

price[i]

denotes the price of the

i

th

candy and a positive integer

k

.

The store sells baskets of

k

distinct

candies. The

tastiness

of a candy basket is the smallest absolute difference of the

prices

of any two candies in the basket.

Return

the

maximum

tastiness of a candy basket.

Example 1:

Input:

price = [13,5,1,8,21,2], k = 3

Output:

8

Explanation:

Choose the candies with the prices [13,5,21]. The tastiness of the candy basket is: min(|13 - 5|, |13 - 21|, |5 - 21|) = min(8, 8, 16) = 8. It can be proven that 8 is the maximum tastiness that can be achieved.

Example 2:

Input:

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

Output:

2

Explanation:

Choose the candies with the prices [1,3]. The tastiness of the candy basket is: min(|1 - 3|) = min(2) = 2. It can be proven that 2 is the maximum tastiness that can be achieved.

Example 3:

Input:

price = [7,7,7,7], k = 2

Output:

0

Explanation:

Choosing any two distinct candies from the candies we have will result in a tastiness of 0.

Constraints:

2 <= k <= price.length <= 10

5

1 <= price[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maximumTastiness(vector<int>& price, int k) {

}
};
```

Java:

```
class Solution {
public int maximumTastiness(int[] price, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumTastiness(price: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaximumTastiness(int[] price, int k) {

}
}
```

C:

```
int maximumTastiness(int* price, int priceSize, int k) {

}
```

Go:

```
func maximumTastiness(price []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maximumTastiness(price: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn maximum_tastiness(price: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} price
# @param {Integer} k
# @return {Integer}
def maximum_tastiness(price, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $price
* @param Integer $k
* @return Integer
*/
function maximumTastiness($price, $k) {

}
}
```

Dart:

```
class Solution {
int maximumTastiness(List<int> price, int k) {

}
}
```

Scala:

```
object Solution {
def maximumTastiness(price: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_tastiness(price :: [integer], k :: integer) :: integer
def maximum_tastiness(price, k) do

end
end
```

Erlang:

```
-spec maximum_tastiness(Price :: [integer()], K :: integer()) -> integer().
maximum_tastiness(Price, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maximumTastiness(vector<int>& price, int k) {
}
};
```

Java Solution:

```
/**
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maximumTastiness(int[] price, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Tastiness of Candy Basket
Difficulty: Medium
Tags: array, greedy, sort, 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 maximumTastiness(self, price: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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[]} price
* @param {number} k
* @return {number}
*/
var maximumTastiness = function(price, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maximumTastiness(price: number[], k: number): number {
};
```

C# Solution:

```
/*
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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 MaximumTastiness(int[] price, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Tastiness of Candy Basket
* Difficulty: Medium
* Tags: array, greedy, sort, 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 maximumTastiness(int* price, int priceSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Tastiness of Candy Basket
// Difficulty: Medium
// Tags: array, greedy, sort, 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 maximumTastiness(price []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumTastiness(price: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Tastiness of Candy Basket
// Difficulty: Medium
// Tags: array, greedy, sort, 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_tastiness(price: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} price
# @param {Integer} k
# @return {Integer}
def maximum_tastiness(price, k)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $price
* @param Integer $k
* @return Integer
*/
function maximumTastiness($price, $k) {

}
}
```

Dart Solution:

```dart
class Solution {
int maximumTastiness(List<int> price, int k) {

}
}
```

Scala Solution:

```scala
object Solution {
def maximumTastiness(price: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_tastiness(price :: [integer], k :: integer) :: integer
def maximum_tastiness(price, k) do

end
end
```

Erlang Solution:

```
-spec maximum_tastiness(Price :: [integer()], K :: integer()) -> integer().
maximum_tastiness(Price, K) ->
.
```

Racket Solution:

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