Problem 1478: Allocate Mailboxes

Problem Information

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

Problem Description

Given the array

houses

where

houses[i]

is the location of the

i

th

house along a street and an integer

k

, allocate

k

mailboxes in the street.

Return

the

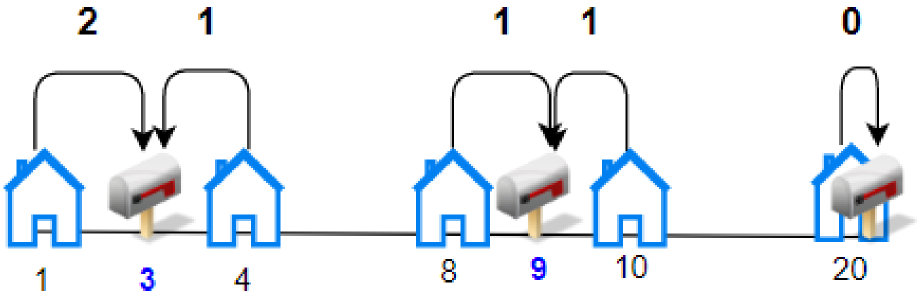
minimum

total distance between each house and its nearest mailbox

.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:



Input:

houses = [1,4,8,10,20], k = 3

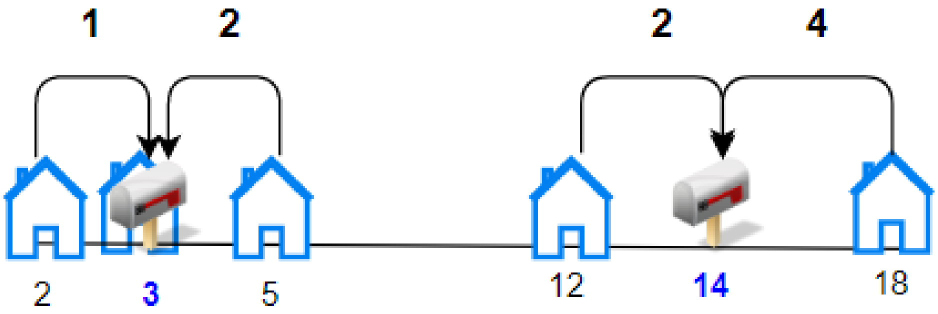
Output:

5

Explanation:

Allocate mailboxes in position 3, 9 and 20. Minimum total distance from each houses to nearest mailboxes is |3-1| + |4-3| + |9-8| + |10-9| + |20-20| = 5

Example 2:



Input:

houses = [2,3,5,12,18], k = 2

Output:

9

Explanation:

Allocate mailboxes in position 3 and 14. Minimum total distance from each houses to nearest mailboxes is |2-3| + |3-3| + |5-3| + |12-14| + |18-14| = 9.

Constraints:

1 <= k <= houses.length <= 100

1 <= houses[i] <= 10

4

All the integers of

houses

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
int minDistance(vector<int>& houses, int k) {

}
};
```

Java:

```
class Solution {
public int minDistance(int[] houses, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} houses
* @param {number} k
* @return {number}
*/
var minDistance = function(houses, k) {
};
```

TypeScript:

```
function minDistance(houses: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinDistance(int[] houses, int k) {

}
}
```

C:

```
int minDistance(int* houses, int housesSize, int k) {

}
```

Go:

```
func minDistance(houses []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minDistance(houses: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minDistance(_ houses: [Int], _ k: Int) -> Int {
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} houses
# @param {Integer} k
# @return {Integer}
def min_distance(houses, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $houses
* @param Integer $k
* @return Integer
*/
function minDistance($houses, $k) {

}
}
```

Dart:

```
class Solution {
int minDistance(List<int> houses, int k) {

}
}
```

Scala:

```
object Solution {
def minDistance(houses: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_distance(houses :: [integer], k :: integer) :: integer
def min_distance(houses, k) do

end
end
```

Erlang:

```
-spec min_distance(Houses :: [integer()], K :: integer()) -> integer().
min_distance(Houses, K) ->
.
```

Racket:

```
(define/contract (min-distance houses k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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 minDistance(vector<int>& houses, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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 minDistance(int[] houses, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Allocate Mailboxes
Difficulty: Hard
Tags: array, tree, dp, math, 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 minDistance(self, houses: List[int], k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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[]} houses
* @param {number} k
* @return {number}
*/
var minDistance = function(houses, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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 minDistance(houses: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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 MinDistance(int[] houses, int k) {

}
}
```

C Solution:

```
/*
* Problem: Allocate Mailboxes
* Difficulty: Hard
* Tags: array, tree, dp, math, 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 minDistance(int* houses, int housesSize, int k) {

}
```

Go Solution:

```
// Problem: Allocate Mailboxes
// Difficulty: Hard
// Tags: array, tree, dp, math, 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 minDistance(houses []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDistance(houses: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Allocate Mailboxes
// Difficulty: Hard
// Tags: array, tree, dp, math, 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 min_distance(houses: Vec<i32>, k: i32) -> i32 {



}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $houses
* @param Integer $k
* @return Integer
*/
function minDistance($houses, $k) {

}
}
```

Dart Solution:

```dart
class Solution {
int minDistance(List<int> houses, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_distance(houses :: [integer], k :: integer) :: integer
def min_distance(houses, k) do

end
end
```

Erlang Solution:

```
-spec min_distance(Houses :: [integer()], K :: integer()) -> integer().
min_distance(Houses, K) ->
.
```

Racket Solution:

```
(define/contract (min-distance houses k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```