Problem 774: Minimize Max Distance to Gas Station

Problem Information

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

Problem Description

You are given an integer array

stations

that represents the positions of the gas stations on the

x-axis

. You are also given an integer

k

.

You should add

k

new gas stations. You can add the stations anywhere on the

x-axis

, and not necessarily on an integer position.

Let

penalty()

be the maximum distance between

adjacent

gas stations after adding the

k

new stations.

Return

the smallest possible value of

penalty()

. Answers within

10

-6

of the actual answer will be accepted.

Example 1:

Input:

stations = [1,2,3,4,5,6,7,8,9,10], k = 9

Output:

0.50000

Example 2:

Input:

stations = [23,24,36,39,46,56,57,65,84,98], k = 1

Output:

14.00000

Constraints:

10 <= stations.length <= 2000

0 <= stations[i] <= 10

8

stations

is sorted in a

strictly increasing

order.

1 <= k <= 10

6

Code Snippets

C++:

```
class Solution {
public:
double minmaxGasDist(vector<int>& stations, int k) {

}
};
```

Java:

```
class Solution {
public double minmaxGasDist(int[] stations, int k) {

}
}
```

Python3:

```
class Solution:
def minmaxGasDist(self, stations: List[int], k: int) -> float:
```

Python:

```
class Solution(object):
def minmaxGasDist(self, stations, k):
"""
:type stations: List[int]
:type k: int
:rtype: float
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minmaxGasDist(stations: number[], k: number): number {

};
```

C#:

```
public class Solution {
public double MinmaxGasDist(int[] stations, int k) {

}
}
```

C:

```
double minmaxGasDist(int* stations, int stationsSize, int k) {

}
```

Go:

```
func minmaxGasDist(stations []int, k int) float64 {

}
```

Kotlin:

```
class Solution {
fun minmaxGasDist(stations: IntArray, k: Int): Double {

}
}
```

Swift:

```
class Solution {
func minmaxGasDist(_ stations: [Int], _ k: Int) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn minmax_gas_dist(stations: Vec<i32>, k: i32) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[]} stations
# @param {Integer} k
# @return {Float}
def minmax_gas_dist(stations, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $stations
* @param Integer $k
* @return Float
*/
function minmaxGasDist($stations, $k) {

}
}
```

Dart:

```
class Solution {
double minmaxGasDist(List<int> stations, int k) {

}
}
```

Scala:

```
object Solution {
def minmaxGasDist(stations: Array[Int], k: Int): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minmax_gas_dist(stations :: [integer], k :: integer) :: float
def minmax_gas_dist(stations, k) do
```

```
end
end
```

Erlang:

```
-spec minmax_gas_dist(Stations :: [integer()], K :: integer()) -> float().
minmax_gas_dist(Stations, K) ->
.
```

Racket:

```
(define/contract (minmax-gas-dist stations k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimize Max Distance to Gas Station
* Difficulty: Hard
* Tags: array, 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:
double minmaxGasDist(vector<int>& stations, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize Max Distance to Gas Station
```

```
* Difficulty: Hard
* Tags: array, 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 double minmaxGasDist(int[] stations, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize Max Distance to Gas Station
Difficulty: Hard
Tags: array, 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 minmaxGasDist(self, stations: List[int], k: int) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minmaxGasDist(self, stations, k):
"""
:type stations: List[int]
:type k: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Minimize Max Distance to Gas Station
* Difficulty: Hard
* Tags: array, 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[]} stations
* @param {number} k
* @return {number}
*/
var minmaxGasDist = function(stations, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize Max Distance to Gas Station
* Difficulty: Hard
* Tags: array, 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 minmaxGasDist(stations: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimize Max Distance to Gas Station
* Difficulty: Hard
* Tags: array, 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 double MinmaxGasDist(int[] stations, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimize Max Distance to Gas Station
* Difficulty: Hard
* Tags: array, 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
*/

double minmaxGasDist(int* stations, int stationsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimize Max Distance to Gas Station
// Difficulty: Hard
// Tags: array, 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 minmaxGasDist(stations []int, k int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun minmaxGasDist(stations: IntArray, k: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func minmaxGasDist(_ stations: [Int], _ k: Int) -> Double {

}
}
```

Rust Solution:

```
// Problem: Minimize Max Distance to Gas Station
// Difficulty: Hard
// Tags: array, 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 minmax_gas_dist(stations: Vec<i32>, k: i32) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} stations
# @param {Integer} k
# @return {Float}
def minmax_gas_dist(stations, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $stations
* @param Integer $k
* @return Float
*/
function minmaxGasDist($stations, $k) {

}
}
```

Dart Solution:

```
class Solution {
double minmaxGasDist(List<int> stations, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minmaxGasDist(stations: Array[Int], k: Int): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minmax_gas_dist(stations :: [integer], k :: integer) :: float
def minmax_gas_dist(stations, k) do

end
end
```

Erlang Solution:

```
-spec minmax_gas_dist(Stations :: [integer()], K :: integer()) -> float().
minmax_gas_dist(Stations, K) ->
.
```

Racket Solution:

```
(define/contract (minmax-gas-dist stations k)
(-> (listof exact-integer?) exact-integer? flonum?)
)
```