Problem 1515: Best Position for a Service Centre

Problem Information

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

Problem Description

A delivery company wants to build a new service center in a new city. The company knows the positions of all the customers in this city on a 2D-Map and wants to build the new center in a position such that

the sum of the euclidean distances to all customers is minimum

.

Given an array

positions

where

positions[i] = [x

i

, y

i

]

is the position of the

ith

customer on the map, return

the minimum sum of the euclidean distances

to all customers.

In other words, you need to choose the position of the service center

[x

centre

, y

centre

]

such that the following formula is minimized:

$$\sum_{i=0}^{n-1} \sqrt{(x_{centre} - x_i)^2 + (y_{centre} - y_i)^2}$$

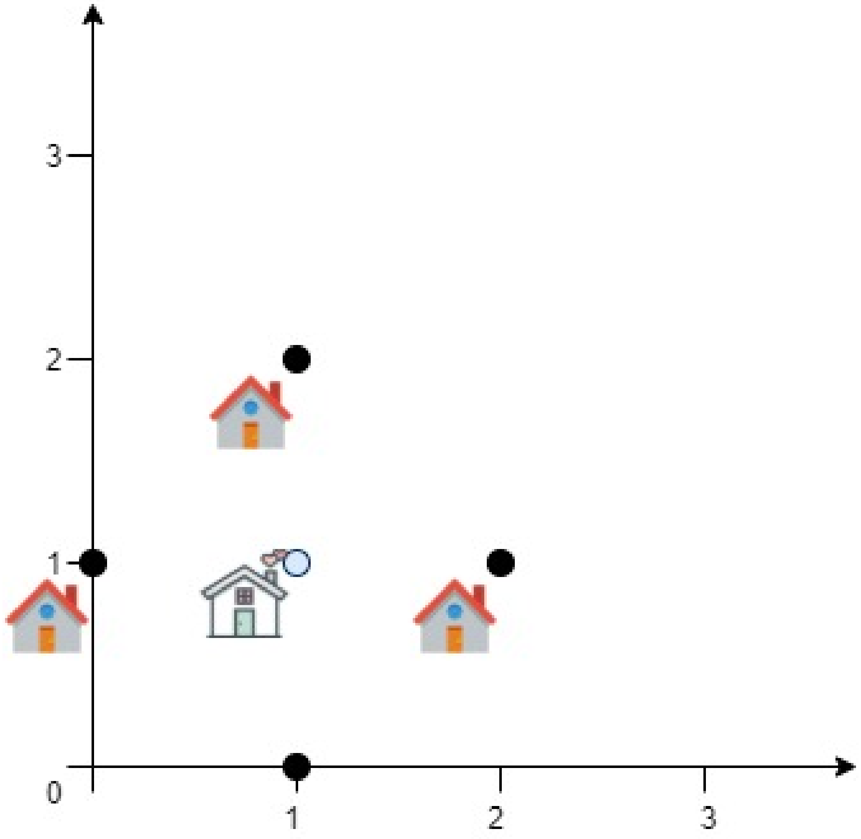
Answers within

10

-5

of the actual value will be accepted.

Example 1:



Input:

positions = [[0,1],[1,0],[1,2],[2,1]]

Output:

4.00000

Explanation:

As shown, you can see that choosing [x

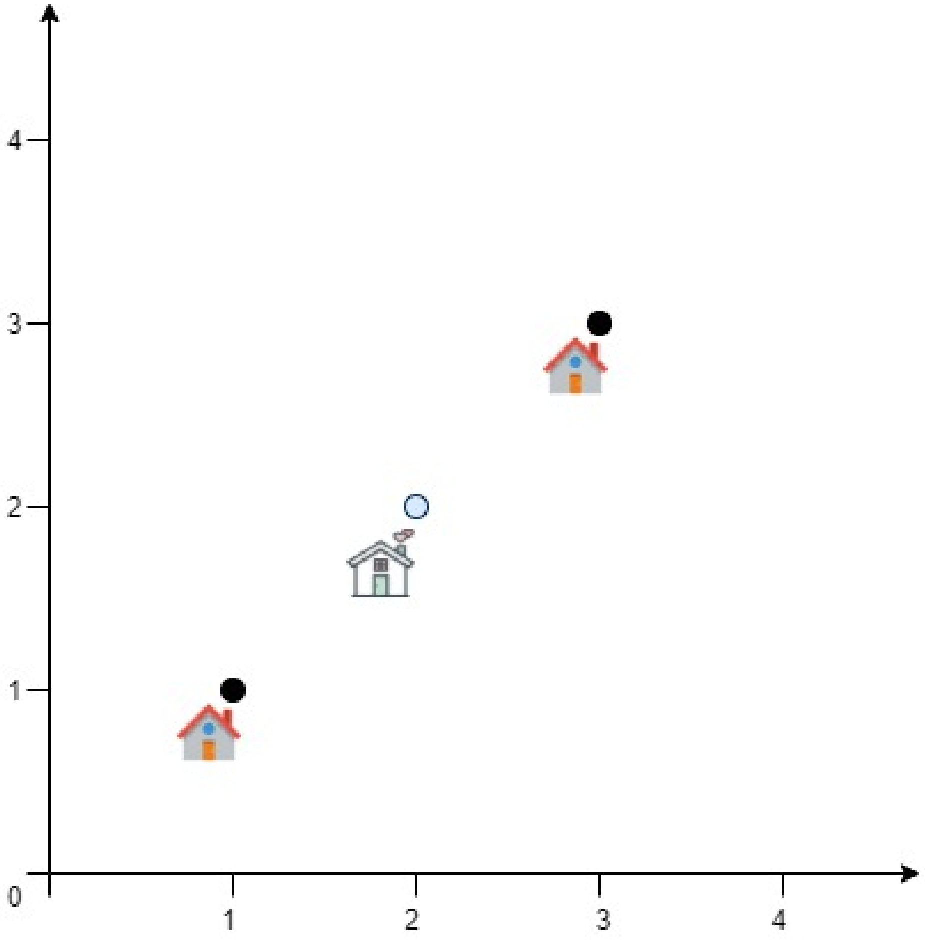
centre

, y

centre

] = [1, 1] will make the distance to each customer = 1, the sum of all distances is 4 which is the minimum possible we can achieve.

Example 2:



Input:

positions = [[1,1],[3,3]]

Output:

2.82843

Explanation:

The minimum possible sum of distances = sqrt(2) + sqrt(2) = 2.82843

Constraints:

1 <= positions.length <= 50

positions[i].length == 2

0 <= x

i

, y

i

<= 100

Code Snippets

C++:

```
class Solution {
public:
double getMinDistSum(vector<vector<int>>& positions) {

}
};
```

Java:

```
class Solution {
public double getMinDistSum(int[][] positions) {

}
}
```

Python3:

```
class Solution:
def getMinDistSum(self, positions: List[List[int]]) -> float:
```

Python:

```
class Solution(object):
def getMinDistSum(self, positions):
"""
:type positions: List[List[int]]
:rtype: float
"""
```

JavaScript:

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

};
```

TypeScript:

```
function getMinDistSum(positions: number[][]): number {

};
```

C#:

```
public class Solution {
public double GetMinDistSum(int[][] positions) {

}
}
```

C:

```
double getMinDistSum(int** positions, int positionsSize, int*
positionsColSize) {

}
```

Go:

```
func getMinDistSum(positions [][]int) float64 {

}
```

Kotlin:

```
class Solution {
fun getMinDistSum(positions: Array<IntArray>): Double {

}
}
```

Swift:

```
class Solution {
func getMinDistSum(_ positions: [[Int]]) -> Double {

}
}
```

Rust:

```
impl Solution {
pub fn get_min_dist_sum(positions: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby:

```
# @param {Integer[][]} positions
# @return {Float}
def get_min_dist_sum(positions)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $positions
* @return Float
*/
function getMinDistSum($positions) {

}

}
```

Dart:

```
class Solution {
double getMinDistSum(List<List<int>> positions) {

}
}
```

Scala:

```
object Solution {
def getMinDistSum(positions: Array[Array[Int]]): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_min_dist_sum(positions :: [[integer]]) :: float
def get_min_dist_sum(positions) do

end
end
```

Erlang:

```
-spec get_min_dist_sum(Positions :: [[integer()]]) -> float().
get_min_dist_sum(Positions) ->
.
```

Racket:

```
(define/contract (get-min-dist-sum positions)
(-> (listof (listof exact-integer?)) flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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 getMinDistSum(vector<vector<int>>& positions) {

}
};
```

Java Solution:

```
/**
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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 getMinDistSum(int[][] positions) {

}
}
```

Python3 Solution:

```
"""
Problem: Best Position for a Service Centre
Difficulty: Hard
Tags: array, math
```

```
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 getMinDistSum(self, positions: List[List[int]]) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getMinDistSum(self, positions):
"""
:type positions: List[List[int]]
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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[][]} positions
* @return {number}
*/
var getMinDistSum = function(positions) {

};
```

TypeScript Solution:

```
/**
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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 getMinDistSum(positions: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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 GetMinDistSum(int[][] positions) {

}
}
```

C Solution:

```
/*
* Problem: Best Position for a Service Centre
* Difficulty: Hard
* Tags: array, math
*
* 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 getMinDistSum(int** positions, int positionsSize, int*
positionsColSize) {

}
```

Go Solution:

```
// Problem: Best Position for a Service Centre
// Difficulty: Hard
// Tags: array, math
//
// 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 getMinDistSum(positions [][]int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun getMinDistSum(positions: Array<IntArray>): Double {

}
}
```

Swift Solution:

```
class Solution {
func getMinDistSum(_ positions: [[Int]]) -> Double {

}
}
```

Rust Solution:

```
// Problem: Best Position for a Service Centre
// Difficulty: Hard

// Tags: array, math
//
// 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 get_min_dist_sum(positions: Vec<Vec<i32>>) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} positions
# @return {Float}
def get_min_dist_sum(positions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $positions
* @return Float
*/
function getMinDistSum($positions) {

}
}
```

Dart Solution:

```
class Solution {
double getMinDistSum(List<List<int>> positions) {

}
}
```

Scala Solution:

```
object Solution {
def getMinDistSum(positions: Array[Array[Int]]): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_min_dist_sum(positions :: [[integer]]) :: float
def get_min_dist_sum(positions) do

end
end
```

Erlang Solution:

```
-spec get_min_dist_sum(Positions :: [[integer()]]) -> float().
get_min_dist_sum(Positions) ->
.
```

Racket Solution:

```
(define/contract (get-min-dist-sum positions)
(-> (listof (listof exact-integer?)) flonum?)
)
```