

# Problem 296: Best Meeting Point

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given an

$m \times n$

binary grid

grid

where each

1

marks the home of one friend, return

the minimal

total travel distance

.

The

total travel distance

is the sum of the distances between the houses of the friends and the meeting point.

The distance is calculated using

Manhattan Distance

, where

$$\text{distance}(p1, p2) = |p2.x - p1.x| + |p2.y - p1.y|$$

.

Example 1:

<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>

Input:

grid = [[1,0,0,0,1],[0,0,0,0,0],[0,0,1,0,0]]

Output:

6

Explanation:

Given three friends living at (0,0), (0,4), and (2,2). The point (0,2) is an ideal meeting point, as the total travel distance of  $2 + 2 + 2 = 6$  is minimal. So return 6.

Example 2:

Input:

`grid = [[1,1]]`

Output:

1

Constraints:

`m == grid.length`

`n == grid[i].length`

`1 <= m, n <= 200`

`grid[i][j]`

is either

0

or

1

.

There will be

at least two

friends in the

grid

.

## Code Snippets

### C++:

```
class Solution {
public:
    int minTotalDistance(vector<vector<int>>& grid) {

    }
};
```

### Java:

```
class Solution {
    public int minTotalDistance(int[][] grid) {

    }
}
```

### Python3:

```
class Solution:
    def minTotalDistance(self, grid: List[List[int]]) -> int:
```

### Python:

```
class Solution(object):
    def minTotalDistance(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
```

### JavaScript:

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

};
```

### TypeScript:

```
function minTotalDistance(grid: number[][]): number {  
  
};
```

### C#:

```
public class Solution {  
    public int MinTotalDistance(int[][] grid) {  
  
    }  
}
```

### C:

```
int minTotalDistance(int** grid, int gridSize, int* gridColSize) {  
  
}
```

### Go:

```
func minTotalDistance(grid [][]int) int {  
  
}
```

### Kotlin:

```
class Solution {  
    fun minTotalDistance(grid: Array<IntArray>): Int {  
  
    }  
}
```

### Swift:

```
class Solution {  
    func minTotalDistance(_ grid: [[Int]]) -> Int {  
  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn min_total_distance(grid: Vec<Vec<i32>>) -> i32 {  
  
    }  
}
```

### Ruby:

```
# @param {Integer[][]} grid  
# @return {Integer}  
def min_total_distance(grid)  
  
end
```

### PHP:

```
class Solution {  
  
    /**  
     * @param Integer[][] $grid  
     * @return Integer  
     */  
    function minTotalDistance($grid) {  
  
    }  
}
```

### Dart:

```
class Solution {  
    int minTotalDistance(List<List<int>> grid) {  
  
    }  
}
```

### Scala:

```
object Solution {  
    def minTotalDistance(grid: Array[Array[Int]]): Int = {  
  
    }  
}
```

```
}
```

### Elixir:

```
defmodule Solution do
  @spec min_total_distance(grid :: [[integer]]) :: integer
  def min_total_distance(grid) do

  end
end
```

### Erlang:

```
-spec min_total_distance(Grid :: [[integer()]]) -> integer().
min_total_distance(Grid) ->
.
```

### Racket:

```
(define/contract (min-total-distance grid)
  (-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Best Meeting Point
 * Difficulty: Hard
 * Tags: array, math, sort
 *
 * 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 minTotalDistance(vector<vector<int>>& grid) {
```

```
}  
};
```

### Java Solution:

```
/**  
 * Problem: Best Meeting Point  
 * Difficulty: Hard  
 * Tags: array, math, sort  
 *  
 * 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 minTotalDistance(int[][] grid) {  
  
    }  
}
```

### Python3 Solution:

```
"""  
Problem: Best Meeting Point  
Difficulty: Hard  
Tags: array, math, sort  
  
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 minTotalDistance(self, grid: List[List[int]]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

```

class Solution(object):
def minTotalDistance(self, grid):
    """
    :type grid: List[List[int]]
    :rtype: int
    """

```

## JavaScript Solution:

```

/**
 * Problem: Best Meeting Point
 * Difficulty: Hard
 * Tags: array, math, sort
 *
 * 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[][]} grid
 * @return {number}
 */
var minTotalDistance = function(grid) {

};

```

## TypeScript Solution:

```

/**
 * Problem: Best Meeting Point
 * Difficulty: Hard
 * Tags: array, math, sort
 *
 * 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 minTotalDistance(grid: number[][]): number {

};

```

## C# Solution:

```
/*
 * Problem: Best Meeting Point
 * Difficulty: Hard
 * Tags: array, math, sort
 *
 * 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 MinTotalDistance(int[][] grid) {

    }
}
```

## C Solution:

```
/*
 * Problem: Best Meeting Point
 * Difficulty: Hard
 * Tags: array, math, sort
 *
 * 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 minTotalDistance(int** grid, int gridSize, int* gridColSize) {

}
```

## Go Solution:

```
// Problem: Best Meeting Point
// Difficulty: Hard
// Tags: array, math, sort
//
// 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 minTotalDistance(grid [][]int) int {

}
```

### Kotlin Solution:

```
class Solution {
    fun minTotalDistance(grid: Array<IntArray>): Int {

    }
}
```

### Swift Solution:

```
class Solution {
    func minTotalDistance(_ grid: [[Int]]) -> Int {

    }
}
```

### Rust Solution:

```
// Problem: Best Meeting Point
// Difficulty: Hard
// Tags: array, math, sort
//
// 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 min_total_distance(grid: Vec<Vec<i32>>) -> i32 {

    }
}
```

### Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def min_total_distance(grid)

end
```

### PHP Solution:

```
class Solution {

    /**
     * @param Integer[][] $grid
     * @return Integer
     */
    function minTotalDistance($grid) {

    }

}
```

### Dart Solution:

```
class Solution {
  int minTotalDistance(List<List<int>> grid) {

  }

}
```

### Scala Solution:

```
object Solution {
  def minTotalDistance(grid: Array[Array[Int]]): Int = {

  }

}
```

### Elixir Solution:

```
defmodule Solution do
  @spec min_total_distance(grid :: [[integer]]) :: integer
  def min_total_distance(grid) do

  end
end
```

```
end
```

### Erlang Solution:

```
-spec min_total_distance(Grid :: [[integer()]]) -> integer().  
min_total_distance(Grid) ->  
.
```

### Racket Solution:

```
(define/contract (min-total-distance grid)  
  (-> (listof (listof exact-integer?)) exact-integer?)  
  )
```