Problem 1584: Min Cost to Connect All Points

Problem Information

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

Problem Description

You are given an array

points

representing integer coordinates of some points on a 2D-plane, where

points[i] = [x

i

, y

i

]

.

The cost of connecting two points

[x

i

, y

i

]

and

[x

j

, y

j

]

is the

manhattan distance

between them:

|x

i

- x

j

| + |y

i

- y

j

|

, where

|val|

denotes the absolute value of

val

.

Return

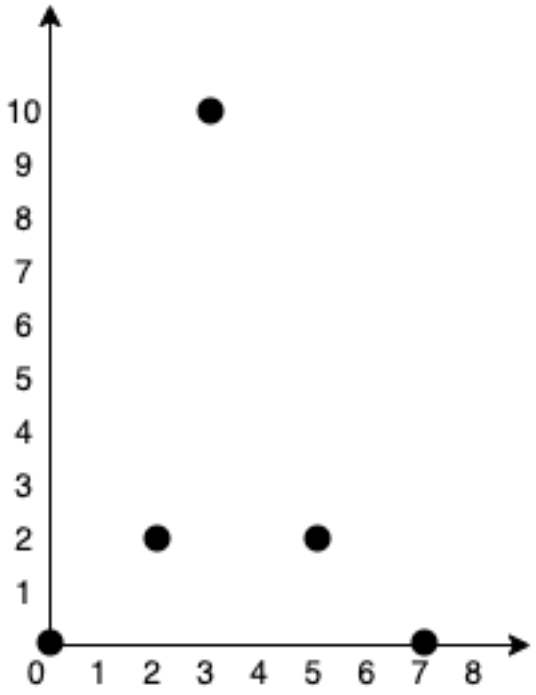
the minimum cost to make all points connected.

All points are connected if there is

exactly one

simple path between any two points.

Example 1:



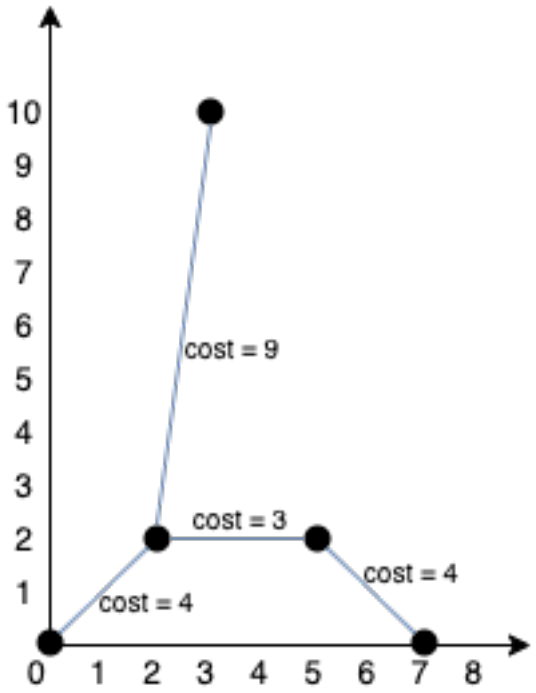
Input:

points = [[0,0],[2,2],[3,10],[5,2],[7,0]]

Output:

20

Explanation:



We can connect the points as shown above to get the minimum cost of 20. Notice that there is a unique path between every pair of points.

Example 2:

Input:

points = [[3,12],[-2,5],[-4,1]]

Output:

18

Constraints:

1 <= points.length <= 1000

-10

6

<= x

i

, y

i

<= 10

6

All pairs

(x

i

, y

i

)

are distinct.

Code Snippets

C++:

```
class Solution {
public:
int minCostConnectPoints(vector<vector<int>>& points) {

}
};
```

Java:

```
class Solution {
public int minCostConnectPoints(int[][] points) {

}
}
```

Python3:

```
class Solution:
def minCostConnectPoints(self, points: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCostConnectPoints(points: number[][]): number {
```

```
};
```

C#:

```
public class Solution {
public int MinCostConnectPoints(int[][] points) {

}
}
```

C:

```
int minCostConnectPoints(int** points, int pointsSize, int* pointsColSize) {

}
```

Go:

```
func minCostConnectPoints(points [][]int) int {

}
```

Kotlin:

```
class Solution {
fun minCostConnectPoints(points: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func minCostConnectPoints(_ points: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost_connect_points(points: Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer[][]} points
# @return {Integer}
def min_cost_connect_points(points)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function minCostConnectPoints($points) {

}
}
```

Dart:

```
class Solution {
int minCostConnectPoints(List<List<int>> points) {

}
}
```

Scala:

```
object Solution {
def minCostConnectPoints(points: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost_connect_points(points :: [[integer]]) :: integer
def min_cost_connect_points(points) do

end
end
```

Erlang:

```
-spec min_cost_connect_points(Points :: [[integer()]]) -> integer().
min_cost_connect_points(Points) ->
.
```

Racket:

```
(define/contract (min-cost-connect-points points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minCostConnectPoints(vector<vector<int>>& points) {

}
};
```

Java Solution:

```
/**
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minCostConnectPoints(int[][] points) {

}
}
```

Python3 Solution:

```
"""
Problem: Min Cost to Connect All Points
Difficulty: Medium
Tags: array, tree, graph

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minCostConnectPoints(self, points: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[][]} points
* @return {number}
*/
var minCostConnectPoints = function(points) {

};
```

TypeScript Solution:

```
/**
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minCostConnectPoints(points: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinCostConnectPoints(int[][] points) {

}
}
```

C Solution:

```
/*
* Problem: Min Cost to Connect All Points
* Difficulty: Medium
* Tags: array, tree, graph
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minCostConnectPoints(int** points, int pointsSize, int* pointsColSize) {

}
```

Go Solution:

```
// Problem: Min Cost to Connect All Points
// Difficulty: Medium
// Tags: array, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minCostConnectPoints(points [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCostConnectPoints(points: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCostConnectPoints(_ points: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Min Cost to Connect All Points
// Difficulty: Medium
// Tags: array, tree, graph
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn min_cost_connect_points(points: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} points
# @return {Integer}
def min_cost_connect_points(points)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $points
* @return Integer
*/
function minCostConnectPoints($points) {

}
}
```

Dart Solution:

```
class Solution {
int minCostConnectPoints(List<List<int>> points) {

}
}
```

Scala Solution:

```
object Solution {
def minCostConnectPoints(points: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost_connect_points(points :: [[integer]]) :: integer
def min_cost_connect_points(points) do

end
end
```

Erlang Solution:

```
-spec min_cost_connect_points(Points :: [[integer()]]) -> integer().
min_cost_connect_points(Points) ->
.
```

Racket Solution:

```
(define/contract (min-cost-connect-points points)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```