

# Problem 1266: Minimum Time Visiting All Points

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

On a 2D plane, there are

$n$

points with integer coordinates

`points[i] = [x`

`i`

`, y`

`i`

`]`

. Return

the

minimum time

in seconds to visit all the points in the order given by

points

You can move according to these rules:

In

1

second, you can either:

move vertically by one unit,

move horizontally by one unit, or

move diagonally

$\sqrt{2}$

units (in other words, move one unit vertically then one unit horizontally in

1

second).

You have to visit the points in the same order as they appear in the array.

You are allowed to pass through points that appear later in the order, but these do not count as visits.

Example 1:



Input:

```
points = [[1,1],[3,4],[-1,0]]
```

Output:

7

Explanation:

One optimal path is

[1,1]

-> [2,2] -> [3,3] ->

[3,4]

-> [2,3] -> [1,2] -> [0,1] ->

[-1,0]

Time from [1,1] to [3,4] = 3 seconds Time from [3,4] to [-1,0] = 4 seconds Total time = 7 seconds

Example 2:

Input:

points = [[3,2],[-2,2]]

Output:

5

Constraints:

points.length == n

1 <= n <= 100

points[i].length == 2

-1000 <= points[i][0], points[i][1] <= 1000

## Code Snippets

C++:

```
class Solution {
public:
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {
        }
};
```

Java:

```
class Solution {  
    public int minTimeToVisitAllPoints(int[][] points) {  
  
    }  
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

### TypeScript:

```
function minTimeToVisitAllPoints(points: number[][]): number {  
  
};
```

### C#:

```
public class Solution {  
    public int MinTimeToVisitAllPoints(int[][] points) {  
  
    }  
}
```

**C:**

```
int minTimeToVisitAllPoints(int** points, int pointsSize, int* pointsColSize)
{
}
```

**Go:**

```
func minTimeToVisitAllPoints(points [][]int) int {
}
```

**Kotlin:**

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

**Swift:**

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

**Rust:**

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

**Ruby:**

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

```
end
```

### PHP:

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

### Dart:

```
class Solution {  
int minTimeToVisitAllPoints(List<List<int>> points) {  
  
}  
}
```

### Scala:

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

### Elixir:

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

### Erlang:

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

## Racket:

```
(define/contract (min-time-to-visit-all-points points)  
(-> (listof (listof exact-integer?)) exact-integer?)  
)
```

# Solutions

## C++ Solution:

```
/*  
 * Problem: Minimum Time Visiting All Points  
 * Difficulty: Easy  
 * 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:  
    int minTimeToVisitAllPoints(vector<vector<int>>& points) {  
        }  
};
```

## Java Solution:

```
/**  
 * Problem: Minimum Time Visiting All Points  
 * Difficulty: Easy  
 * 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
```

```
*/\n\n\nclass Solution {\n    public int minTimeToVisitAllPoints(int[][] points) {\n\n        }\n    }\n}
```

### Python3 Solution:

```
'''\n\nProblem: Minimum Time Visiting All Points\nDifficulty: Easy\nTags: array, math\n\nApproach: Use two pointers or sliding window technique\nTime Complexity: O(n) or O(n log n)\nSpace Complexity: O(1) to O(n) depending on approach\n'''
```

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

### Python Solution:

```
class Solution(object):\n    def minTimeToVisitAllPoints(self, points):\n\n        '''\n        :type points: List[List[int]]\n        :rtype: int\n        '''
```

### JavaScript Solution:

```
/**\n * Problem: Minimum Time Visiting All Points\n * Difficulty: Easy\n * Tags: array, math\n */
```

```

* 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[][]} points
* @return {number}
*/
var minTimeToVisitAllPoints = function(points) {
};

```

### TypeScript Solution:

```

/**
* Problem: Minimum Time Visiting All Points
* Difficulty: Easy
* 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 minTimeToVisitAllPoints(points: number[][]): number {
}

```

### C# Solution:

```

/*
* Problem: Minimum Time Visiting All Points
* Difficulty: Easy
* 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 int MinTimeToVisitAllPoints(int[][] points) {  
  
    }  
}
```

### C Solution:

```
/*  
 * Problem: Minimum Time Visiting All Points  
 * Difficulty: Easy  
 * 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  
 */  
  
int minTimeToVisitAllPoints(int** points, int pointsSize, int* pointsColSize)  
{  
  
}
```

### Go Solution:

```
// Problem: Minimum Time Visiting All Points  
// Difficulty: Easy  
// 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 minTimeToVisitAllPoints(points [][]int) int {  
  
}
```

### Kotlin Solution:

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

```
}
```

```
}
```

### Swift Solution:

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

### Rust Solution:

```
// Problem: Minimum Time Visiting All Points  
// Difficulty: Easy  
// 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 min_time_to_visit_all_points(points: Vec<Vec<i32>>) -> i32 {  
  
    }  
}
```

### Ruby Solution:

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

### PHP Solution:

```
class Solution {  
  
    /**
```

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

}
}
```

### Dart Solution:

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

}
```

### Scala Solution:

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

}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (min-time-to-visit-all-points points)
  (-> (listof (listof exact-integer?)) exact-integer?)
)
```