Problem 3464: Maximize the Distance Between Points on a Square

Problem Information

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

Problem Description

You are given an integer

side

, representing the edge length of a square with corners at

(0, 0)

,

(0, side)

,

(side, 0)

, and

(side, side)

on a Cartesian plane.

You are also given a

positive

integer

k

and a 2D integer array

points

, where

points[i] = [x

i

, y

i

]

represents the coordinate of a point lying on the

boundary

of the square.

You need to select

k

elements among

points

such that the

minimum

Manhattan distance between any two points is

maximized

.

Return the

maximum

possible

minimum

Manhattan distance between the selected

k

points.

The Manhattan Distance between two cells

(x

i

, y

i

)

and

(x

j

, y

j

)

is

|x

i

- x

j

| + |y

i

- y

j

|

.

Example 1:

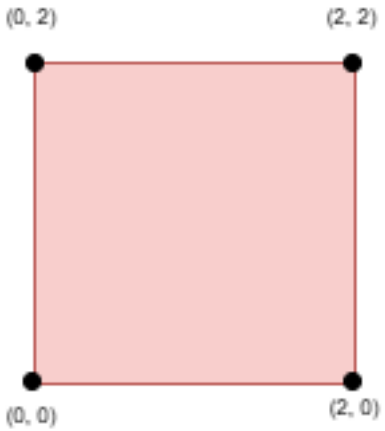
Input:

side = 2, points = [[0,2],[2,0],[2,2],[0,0]], k = 4

Output:

2

Explanation:



Select all four points.

Example 2:

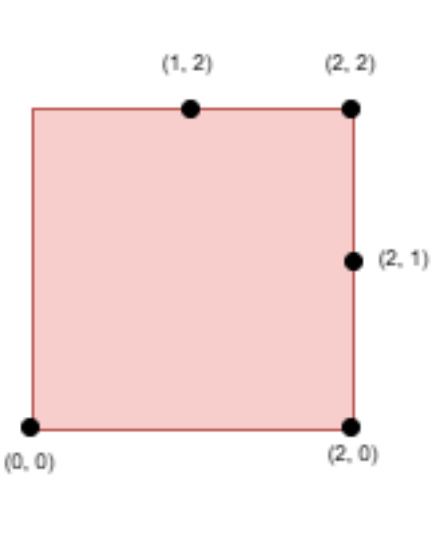
Input:

side = 2, points = [[0,0],[1,2],[2,0],[2,2],[2,1]], k = 4

Output:

1

Explanation:



Select the points

(0, 0)

,

(2, 0)

,

(2, 2)

, and

(2, 1)

.

Example 3:

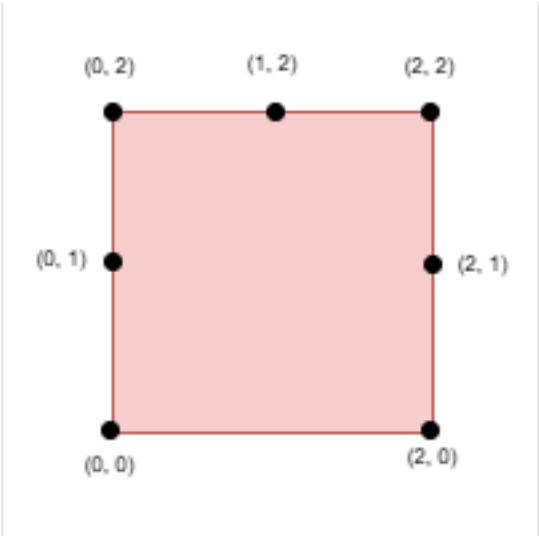
Input:

side = 2, points = [[0,0],[0,1],[0,2],[1,2],[2,0],[2,2],[2,1]], k = 5

Output:

1

Explanation:



Select the points

(0, 0)

,

(0, 1)

,

(0, 2)

,

(1, 2)

, and

(2, 2)

.

Constraints:

1 <= side <= 10

9

4 <= points.length <= min(4 * side, 15 * 10

3

)

points[i] == [xi, yi]

The input is generated such that:

points[i]

lies on the boundary of the square.

All

points[i]

are

unique

.

4 <= k <= min(25, points.length)

Code Snippets

C++:

```
class Solution {
public:
int maxDistance(int side, vector<vector<int>>& points, int k) {

}
};
```

Java:

```
class Solution {
public int maxDistance(int side, int[][] points, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number} side
* @param {number[][]} points
* @param {number} k
* @return {number}
*/
var maxDistance = function(side, points, k) {

};
```

TypeScript:

```
function maxDistance(side: number, points: number[][], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxDistance(int side, int[][] points, int k) {

}
}
```

C:

```
int maxDistance(int side, int** points, int pointsSize, int* pointsColSize,
int k) {

}
```

Go:

```
func maxDistance(side int, points [][]int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxDistance(side: Int, points: Array<IntArray>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func maxDistance(_ side: Int, _ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_distance(side: i32, points: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} side
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def max_distance(side, points, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $side
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function maxDistance($side, $points, $k) {

}
}
```

Dart:

```
class Solution {
int maxDistance(int side, List<List<int>> points, int k) {

}
}
```

Scala:

```
object Solution {
def maxDistance(side: Int, points: Array[Array[Int]], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_distance(side :: integer, points :: [[integer]], k :: integer) ::
integer
def max_distance(side, points, k) do

end
end
```

Erlang:

```
-spec max_distance(Side :: integer(), Points :: [[integer()]], K ::
integer()) -> integer().
max_distance(Side, Points, K) ->
.
```

Racket:

```
(define/contract (max-distance side points k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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:
int maxDistance(int side, vector<vector<int>>& points, int k) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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 int maxDistance(int side, int[][] points, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Distance Between Points on a Square
Difficulty: Hard
Tags: array, greedy, 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 maxDistance(self, side: int, points: List[List[int]], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxDistance(self, side, points, k):
```

```
"""
:type side: int
:type points: List[List[int]]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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} side
* @param {number[][]} points
* @param {number} k
* @return {number}
*/
var maxDistance = function(side, points, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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 maxDistance(side: number, points: number[][], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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 int MaxDistance(int side, int[][] points, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Distance Between Points on a Square
* Difficulty: Hard
* Tags: array, greedy, 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
*/

int maxDistance(int side, int** points, int pointsSize, int* pointsColSize,
int k) {

}
```

Go Solution:

```
// Problem: Maximize the Distance Between Points on a Square
// Difficulty: Hard
// Tags: array, greedy, 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 maxDistance(side int, points [][]int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxDistance(side: Int, points: Array<IntArray>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxDistance(_ side: Int, _ points: [[Int]], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize the Distance Between Points on a Square
// Difficulty: Hard
// Tags: array, greedy, 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 max_distance(side: i32, points: Vec<Vec<i32>>, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} side
# @param {Integer[][]} points
# @param {Integer} k
# @return {Integer}
def max_distance(side, points, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $side
* @param Integer[][] $points
* @param Integer $k
* @return Integer
*/
function maxDistance($side, $points, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxDistance(int side, List<List<int>> points, int k) {

}
}
```

Scala Solution:

```
object Solution {
def maxDistance(side: Int, points: Array[Array[Int]], k: Int): Int = {

}
```

```
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_distance(side :: integer, points :: [[integer]], k :: integer) ::
integer
def max_distance(side, points, k) do

end
end
```

Erlang Solution:

```
-spec max_distance(Side :: integer(), Points :: [[integer()]], K ::
integer()) -> integer().
max_distance(Side, Points, K) ->
.
```

Racket Solution:

```
(define/contract (max-distance side points k)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?
exact-integer?)
)
```