# Problem 624: Maximum Distance in Arrays

## Problem Information

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

## Problem Description

You are given

m

arrays

, where each array is sorted in

ascending order

.

You can pick up two integers from two different arrays (each array picks one) and calculate the distance. We define the distance between two integers

a

and

b

to be their absolute difference

|a - b|

.

Return

the maximum distance

.

Example 1:

Input:

arrays = [[1,2,3],[4,5],[1,2,3]]

Output:

4

Explanation:

One way to reach the maximum distance 4 is to pick 1 in the first or third array and pick 5 in the second array.

Example 2:

Input:

arrays = [[1],[1]]

Output:

0

Constraints:

m == arrays.length

2 <= m <= 10

5

1 <= arrays[i].length <= 500

-10

4

<= arrays[i][j] <= 10

4

arrays[i]

is sorted in

ascending order

.

There will be at most

10

5

integers in all the arrays.

## Code Snippets

**C++:**

```
class Solution {
public:
int maxDistance(vector<vector<int>>& arrays) {

}
};
```

**Java:**

```
class Solution {
public int maxDistance(List<List<Integer>> arrays) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function maxDistance(arrays: number[][]): number {

};
```

**C#:**

```
public class Solution {
public int MaxDistance(IList<IList<int>> arrays) {

}
}
```

**C:**

```
int maxDistance(int** arrays, int arraysSize, int* arraysColSize) {

}
```

**Go:**

```
func maxDistance(arrays [][]int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxDistance(arrays: List<List<Int>>): Int {

}
}
```

**Swift:**

```
class Solution {
func maxDistance(_ arrays: [[Int]]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_distance(arrays: Vec<Vec<i32>>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} arrays
# @return {Integer}
def max_distance(arrays)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $arrays
* @return Integer
*/
function maxDistance($arrays) {

}
}
```

**Dart:**

```
class Solution {
int maxDistance(List<List<int>> arrays) {

}
}
```

**Scala:**

```
object Solution {
def maxDistance(arrays: List[List[Int]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_distance(arrays :: [[integer]]) :: integer
def max_distance(arrays) do

end
end
```

**Erlang:**

```
-spec max_distance(Arrays :: [[integer()]]) -> integer().
max_distance(Arrays) ->
.
```

**Racket:**

```
(define/contract (max-distance arrays)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 maxDistance(vector<vector<int>>& arrays) {

}
};
```

**Java Solution:**

```
/**
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 maxDistance(List<List<Integer>> arrays) {
}
}
```

## Python3 Solution:

```
"""
Problem: Maximum Distance in Arrays
Difficulty: Medium
Tags: array, greedy, 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 maxDistance(self, arrays: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

## Python Solution:

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

## JavaScript Solution:

```
/**
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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[][]} arrays
* @return {number}
*/
var maxDistance = function(arrays) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 maxDistance(arrays: number[][]): number {

};
```

**C# Solution:**

```
/*
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 MaxDistance(IList<IList<int>> arrays) {

}

}
```

**C Solution:**

```
/*
* Problem: Maximum Distance in Arrays
* Difficulty: Medium
* Tags: array, greedy, 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 maxDistance(int** arrays, int arraysSize, int* arraysColSize) {

}
```

**Go Solution:**

```
// Problem: Maximum Distance in Arrays
// Difficulty: Medium
// Tags: array, greedy, 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 maxDistance(arrays [][]int) int {

}
```

**Kotlin Solution:**

```
class Solution {
fun maxDistance(arrays: List<List<Int>>): Int {

}
}
```

**Swift Solution:**

```
class Solution {
func maxDistance(_ arrays: [[Int]]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Maximum Distance in Arrays
// Difficulty: Medium
// Tags: array, greedy, 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 max_distance(arrays: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} arrays
# @return {Integer}
def max_distance(arrays)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $arrays
* @return Integer
*/
function maxDistance($arrays) {

}
}
```

**Dart Solution:**

```
class Solution {
int maxDistance(List<List<int>> arrays) {

}
}
```

**Scala Solution:**

```
object Solution {
def maxDistance(arrays: List[List[Int]]): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec max_distance(arrays :: [[integer]]) :: integer
def max_distance(arrays) do

end
end
```

**Erlang Solution:**

```
-spec max_distance(Arrays :: [[integer()]]) -> integer().
max_distance(Arrays) ->
.
```

**Racket Solution:**

```
(define/contract (max-distance arrays)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```