# Problem 1131: Maximum of Absolute Value Expression

## Problem Information

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

## Problem Description

Given two arrays of integers with equal lengths, return the maximum value of:

|arr1[i] - arr1[j]| + |arr2[i] - arr2[j]| + |i - j|

where the maximum is taken over all

0 <= i, j < arr1.length

.

Example 1:

Input:

arr1 = [1,2,3,4], arr2 = [-1,4,5,6]

Output:

13

Example 2:

Input:

arr1 = [1,-2,-5,0,10], arr2 = [0,-2,-1,-7,-4]

Output:

20

Constraints:

2 <= arr1.length == arr2.length <= 40000

-10^6 <= arr1[i], arr2[i] <= 10^6

## Code Snippets

**C++:**

```
class Solution {
public:
int maxAbsValExpr(vector<int>& arr1, vector<int>& arr2) {

}
};
```

**Java:**

```
class Solution {
public int maxAbsValExpr(int[] arr1, int[] arr2) {

}
}
```

**Python3:**

```
class Solution:
def maxAbsValExpr(self, arr1: List[int], arr2: List[int]) -> int:
```

**Python:**

```
class Solution(object):
def maxAbsValExpr(self, arr1, arr2):
"""
:type arr1: List[int]
```

```
:type arr2: List[int]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} arr1
* @param {number[]} arr2
* @return {number}
*/
var maxAbsValExpr = function(arr1, arr2) {

};
```

**TypeScript:**

```
function maxAbsValExpr(arr1: number[], arr2: number[]): number {

};
```

**C#:**

```
public class Solution {
public int MaxAbsValExpr(int[] arr1, int[] arr2) {

}
}
```

**C:**

```
int maxAbsValExpr(int* arr1, int arr1Size, int* arr2, int arr2Size) {

}
```

**Go:**

```
func maxAbsValExpr(arr1 []int, arr2 []int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxAbsValExpr(arr1: IntArray, arr2: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func maxAbsValExpr(_ arr1: [Int], _ arr2: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_abs_val_expr(arr1: Vec<i32>, arr2: Vec<i32>) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer}
def max_abs_val_expr(arr1, arr2)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer
*/
function maxAbsValExpr($arr1, $arr2) {

}
```

```
}
```

**Dart:**

```
class Solution {
int maxAbsValExpr(List<int> arr1, List<int> arr2) {

}
}
```

**Scala:**

```
object Solution {
def maxAbsValExpr(arr1: Array[Int], arr2: Array[Int]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_abs_val_expr(arr1 :: [integer], arr2 :: [integer]) :: integer
def max_abs_val_expr(arr1, arr2) do

end
end
```

**Erlang:**

```
-spec max_abs_val_expr(Arr1 :: [integer()], Arr2 :: [integer()]) ->
integer().
max_abs_val_expr(Arr1, Arr2) ->
.
```

**Racket:**

```
(define/contract (max-abs-val-expr arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum of Absolute Value Expression
* Difficulty: Medium
* 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 maxAbsValExpr(vector<int>& arr1, vector<int>& arr2) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximum of Absolute Value Expression
* Difficulty: Medium
* 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 maxAbsValExpr(int[] arr1, int[] arr2) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum of Absolute Value Expression
```

```
Difficulty: Medium
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:
def maxAbsValExpr(self, arr1: List[int], arr2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def maxAbsValExpr(self, arr1, arr2):
"""
:type arr1: List[int]
:type arr2: List[int]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Maximum of Absolute Value Expression
* Difficulty: Medium
* 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
*/

/**
* @param {number[]} arr1
* @param {number[]} arr2
* @return {number}
*/
var maxAbsValExpr = function(arr1, arr2) {
```

```
};
```

### TypeScript Solution:

```
/**
* Problem: Maximum of Absolute Value Expression
* Difficulty: Medium
* 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 maxAbsValExpr(arr1: number[], arr2: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Maximum of Absolute Value Expression
* Difficulty: Medium
* 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 MaxAbsValExpr(int[] arr1, int[] arr2) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum of Absolute Value Expression
```

```
* Difficulty: Medium
* 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 maxAbsValExpr(int* arr1, int arr1Size, int* arr2, int arr2Size) {

}
```

### Go Solution:

```
// Problem: Maximum of Absolute Value Expression
// Difficulty: Medium
// 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 maxAbsValExpr(arr1 []int, arr2 []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxAbsValExpr(arr1: IntArray, arr2: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func maxAbsValExpr(_ arr1: [Int], _ arr2: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Maximum of Absolute Value Expression
// Difficulty: Medium
// 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 max_abs_val_expr(arr1: Vec<i32>, arr2: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @return {Integer}
def max_abs_val_expr(arr1, arr2)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @return Integer
*/
function maxAbsValExpr($arr1, $arr2) {

}
}
```

### Dart Solution:

```
class Solution {
int maxAbsValExpr(List<int> arr1, List<int> arr2) {

}
}
```

### Scala Solution:

```
object Solution {
def maxAbsValExpr(arr1: Array[Int], arr2: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec max_abs_val_expr(arr1 :: [integer], arr2 :: [integer]) :: integer
def max_abs_val_expr(arr1, arr2) do

end
end
```

### Erlang Solution:

```
-spec max_abs_val_expr(Arr1 :: [integer()], Arr2 :: [integer()]) ->
integer().
max_abs_val_expr(Arr1, Arr2) ->
.
```

### Racket Solution:

```
(define/contract (max-abs-val-expr arr1 arr2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```