# Problem 1213: Intersection of Three Sorted Arrays

## Problem Information

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

## Problem Description

Given three integer arrays

arr1

,

arr2

and

arr3

sorted

in

strictly increasing

order, return a sorted array of

only

the integers that appeared in

all

three arrays.

Example 1:

Input:

arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

Output:

[1,5]

Explanation:

Only 1 and 5 appeared in the three arrays.

Example 2:

Input:

arr1 = [197,418,523,876,1356], arr2 = [501,880,1593,1710,1870], arr3 = [521,682,1337,1395,1764]

Output:

[]

Constraints:

1 <= arr1.length, arr2.length, arr3.length <= 1000

1 <= arr1[i], arr2[i], arr3[i] <= 2000

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> arraysIntersection(vector<int>& arr1, vector<int>& arr2,
vector<int>& arr3) {

}
};
```

**Java:**

```
class Solution {
public List<Integer> arraysIntersection(int[] arr1, int[] arr2, int[] arr3) {

}
}
```

**Python3:**

```
class Solution:
def arraysIntersection(self, arr1: List[int], arr2: List[int], arr3:
List[int]) -> List[int]:
```

**Python:**

```
class Solution(object):
def arraysIntersection(self, arr1, arr2, arr3):
"""
:type arr1: List[int]
:type arr2: List[int]
:type arr3: List[int]
:rtype: List[int]
"""
```

**JavaScript:**

```
/**
* @param {number[]} arr1
* @param {number[]} arr2
* @param {number[]} arr3
* @return {number[]}
*/
var arraysIntersection = function(arr1, arr2, arr3) {
```

```
};
```

**TypeScript:**

```
function arraysIntersection(arr1: number[], arr2: number[], arr3: number[]):
number[] {

};
```

**C#:**

```
public class Solution {
public IList<int> ArraysIntersection(int[] arr1, int[] arr2, int[] arr3) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* arraysIntersection(int* arr1, int arr1Size, int* arr2, int arr2Size,
int* arr3, int arr3Size, int* returnSize) {

}
```

**Go:**

```
func arraysIntersection(arr1 []int, arr2 []int, arr3 []int) []int {

}
```

**Kotlin:**

```
class Solution {
fun arraysIntersection(arr1: IntArray, arr2: IntArray, arr3: IntArray):
List<Int> {

}
}
```

**Swift:**

```
class Solution {
func arraysIntersection(_ arr1: [Int], _ arr2: [Int], _ arr3: [Int]) -> [Int]
{

}
}
```

**Rust:**

```
impl Solution {
pub fn arrays_intersection(arr1: Vec<i32>, arr2: Vec<i32>, arr3: Vec<i32>) ->
Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @param {Integer[]} arr3
# @return {Integer[]}
def arrays_intersection(arr1, arr2, arr3)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $arr1
* @param Integer[] $arr2
* @param Integer[] $arr3
* @return Integer[]
*/
function arraysIntersection($arr1, $arr2, $arr3) {

}
}
```

**Dart:**

```
class Solution {
List<int> arraysIntersection(List<int> arr1, List<int> arr2, List<int> arr3)
{

}
}
```

**Scala:**

```
object Solution {
def arraysIntersection(arr1: Array[Int], arr2: Array[Int], arr3: Array[Int]):
List[Int] = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec arrays_intersection(arr1 :: [integer], arr2 :: [integer], arr3 ::
[integer]) :: [integer]
def arrays_intersection(arr1, arr2, arr3) do

end
end
```

**Erlang:**

```
-spec arrays_intersection(Arr1 :: [integer()], Arr2 :: [integer()], Arr3 ::
[integer()]) -> [integer()].
arrays_intersection(Arr1, Arr2, Arr3) ->
.
```

**Racket:**

```
(define/contract (arrays-intersection arr1 arr2 arr3)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
vector<int> arraysIntersection(vector<int>& arr1, vector<int>& arr2,
vector<int>& arr3) {

}
};
```

### Java Solution:

```
/**
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public List<Integer> arraysIntersection(int[] arr1, int[] arr2, int[] arr3) {

}
}
```

### Python3 Solution:

```
"""
Problem: Intersection of Three Sorted Arrays
Difficulty: Easy
Tags: array, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def arraysIntersection(self, arr1: List[int], arr2: List[int], arr3:
List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def arraysIntersection(self, arr1, arr2, arr3):
"""
:type arr1: List[int]
:type arr2: List[int]
:type arr3: List[int]
:rtype: List[int]
"""
```

### JavaScript Solution:

```
/**
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} arr1
* @param {number[]} arr2
```

```
* @param {number[]} arr3
* @return {number[]}
*/
var arraysIntersection = function(arr1, arr2, arr3) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function arraysIntersection(arr1: number[], arr2: number[], arr3: number[]):
number[] {

};
```

**C# Solution:**

```
/*
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public IList<int> ArraysIntersection(int[] arr1, int[] arr2, int[] arr3) {

}
}
```

### C Solution:

```
/*
* Problem: Intersection of Three Sorted Arrays
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* arraysIntersection(int* arr1, int arr1Size, int* arr2, int arr2Size,
int* arr3, int arr3Size, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Intersection of Three Sorted Arrays
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func arraysIntersection(arr1 []int, arr2 []int, arr3 []int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun arraysIntersection(arr1: IntArray, arr2: IntArray, arr3: IntArray):
List<Int> {

}
}
```

### Swift Solution:

```
class Solution {
func arraysIntersection(_ arr1: [Int], _ arr2: [Int], _ arr3: [Int]) -> [Int]
{

}
}
```

### Rust Solution:

```
// Problem: Intersection of Three Sorted Arrays
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn arrays_intersection(arr1: Vec<i32>, arr2: Vec<i32>, arr3: Vec<i32>) ->
Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} arr1
# @param {Integer[]} arr2
# @param {Integer[]} arr3
# @return {Integer[]}
def arrays_intersection(arr1, arr2, arr3)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr1
```

```
* @param Integer[] $arr2
* @param Integer[] $arr3
* @return Integer[]
*/
function arraysIntersection($arr1, $arr2, $arr3) {

}
}
```

**Dart Solution:**

```
class Solution {
List<int> arraysIntersection(List<int> arr1, List<int> arr2, List<int> arr3)
{

}
}
```

**Scala Solution:**

```
object Solution {
def arraysIntersection(arr1: Array[Int], arr2: Array[Int], arr3: Array[Int]):
List[Int] = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec arrays_intersection(arr1 :: [integer], arr2 :: [integer], arr3 ::
[integer]) :: [integer]
def arrays_intersection(arr1, arr2, arr3) do

end
end
```

**Erlang Solution:**

```
-spec arrays_intersection(Arr1 :: [integer()], Arr2 :: [integer()], Arr3 ::
[integer()]) -> [integer()].
```

```
arrays_intersection(Arr1, Arr2, Arr3) ->
.
```

**Racket Solution:**

```
(define/contract (arrays-intersection arr1 arr2 arr3)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?)
(listof exact-integer?))
)
```