Problem 1200: Minimum Absolute Difference

Problem Information

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

Problem Description

Given an array of

distinct

integers

arr

, find all pairs of elements with the minimum absolute difference of any two elements.

Return a list of pairs in ascending order(with respect to pairs), each pair

[a, b]

follows

a, b

are from

arr

a < b

b - a

equals to the minimum absolute difference of any two elements in

arr

Example 1:

Input:

arr = [4,2,1,3]

Output:

[[1,2],[2,3],[3,4]]

Explanation:

The minimum absolute difference is 1. List all pairs with difference equal to 1 in ascending order.

Example 2:

Input:

arr = [1,3,6,10,15]

Output:

[[1,3]]

Example 3:

Input:

arr = [3,8,-10,23,19,-4,-14,27]

Output:

[[-14,-10],[19,23],[23,27]]

Constraints:

2 <= arr.length <= 10

5

-10

6

<= arr[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> minimumAbsDifference(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> minimumAbsDifference(int[] arr) {

}
}
```

Python3:

```
class Solution:
def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumAbsDifference(arr: number[]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> MinimumAbsDifference(int[] arr) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** minimumAbsDifference(int* arr, int arrSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func minimumAbsDifference(arr []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun minimumAbsDifference(arr: IntArray): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func minimumAbsDifference(_ arr: [Int]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_abs_difference(arr: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer[][]}
def minimum_abs_difference(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[][]
*/
function minimumAbsDifference($arr) {

}
}
```

Dart:

```
class Solution {
List<List<int>> minimumAbsDifference(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def minimumAbsDifference(arr: Array[Int]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_abs_difference(arr :: [integer]) :: [[integer]]
def minimum_abs_difference(arr) do

end
end
```

Erlang:

```
-spec minimum_abs_difference(Arr :: [integer()]) -> [[integer()]].
minimum_abs_difference(Arr) ->
.
```

Racket:

```
(define/contract (minimum-abs-difference arr)
(-> (listof exact-integer?) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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:
vector<vector<int>> minimumAbsDifference(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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 List<List<Integer>> minimumAbsDifference(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Absolute Difference
Difficulty: Easy
Tags: array, 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 minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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[]} arr
* @return {number[][]}
*/
var minimumAbsDifference = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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 minimumAbsDifference(arr: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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 IList<IList<int>> MinimumAbsDifference(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Absolute Difference
* Difficulty: Easy
* Tags: array, 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** minimumAbsDifference(int* arr, int arrSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Minimum Absolute Difference
// Difficulty: Easy
// Tags: array, 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 minimumAbsDifference(arr []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumAbsDifference(arr: IntArray): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func minimumAbsDifference(_ arr: [Int]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Minimum Absolute Difference
// Difficulty: Easy
// Tags: array, 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 minimum_abs_difference(arr: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer[][]}
def minimum_abs_difference(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[][]
*/
function minimumAbsDifference($arr) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> minimumAbsDifference(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def minimumAbsDifference(arr: Array[Int]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_abs_difference(arr :: [integer]) :: [[integer]]
def minimum_abs_difference(arr) do

end
end
```

Erlang Solution:

```
-spec minimum_abs_difference(Arr :: [integer()]) -> [[integer()]].
minimum_abs_difference(Arr) ->
.
```

Racket Solution:

```
(define/contract (minimum-abs-difference arr)
(-> (listof exact-integer?) (listof (listof exact-integer?)))
)
```