1200. Minimum Absolute Difference

Easy Array Sorting

Leetcode Link

Problem Description

The problem provides an array of distinct integers named `arr`. The objective is to find all pairs of elements that have the minimum absolute difference when compared to all other possible element pairs. In other words, we're looking for pairs of numbers that are closer together than any other pairs.

To satisfy the conditions of the problem, there are a few requirements:

- Each pair must be constructed from elements of `arr`.
- In each pair `[a, b]`, `a` should be less than `b`.
- The difference `b - a` must be equal to the smallest absolute difference of any two elements in `arr`.

The result should be a list of these pairs in ascending order. Ascending order here applies to the order of the pairs themselves, meaning the first elements of each pair should be in ascending order, and in the case of a tie, the second elements should be in ascending order.

Intuition

To find the pairs with the minimum absolute difference, our approach involves sorting the array and comparing adjacent elements. The reason for sorting is that the smallest difference will always be between adjacent elements; no element can have a smaller difference with a non-adjacent element due to the properties of real numbers.

Here's the step-by-step intuition:

1. **Sort the Array**: By sorting `arr`, we ensure that we're only comparing differences between neighbors, which simplifies the problem significantly. We know that the minimum difference can only occur between neighboring numbers because the array consists of distinct integers.
2. **Find the Minimum Difference**: We iterate through the sorted array, considering each pair of adjacent elements, to find the minimum difference that exists in the array. This step is key to identifying what the "minimum absolute difference" actually is.
3. **Gather All Pairs with Minimum Difference**: Once we know the minimum difference, we iterate through the array again, this time collecting all pairs of adjacent elements that have this minimum difference. We construct a list of lists, where each inner list contains a pair of numbers.

The code provided utilizes a Pythonic way of dealing with the adjacent elements through the use of the `pairwise()` function, which when provided an iterable, yields tuples containing pairs of consecutive elements. This function is a cleaner and more readable way of accessing adjacent pairs without having to manage indexes manually.

Altogether, the solution efficiently gathers the pairs in a single pass after sorting, keeping the process succinct and the code readable.

Solution Approach

To implement the solution to this problem, we are taking advantage of the Python language features and some common algorithmic patterns. Here is a detailed explanation of how the solution approach works:

1. **Sorting the Array**:
   The very first step is to sort the array, `arr.sort()`. Sorting is a foundational algorithm in computer science, which organizes elements of a list in a certain order – in this case, ascending numerical order. Since the array consists of `distinct` integers, sorting will arrange the numbers such that if there's any pair with a minimum absolute difference, it will have neighboring elements.

2. **Finding the Minimum Absolute Difference**:
   With the sorted array, we employ a generator expression within the `min` function: `min(b - a for a, b in pairwise(arr))`. The `pairwise` function is used to create an iterator that will return paired tuples (e.g., (arr[0], arr[1]), (arr[1], arr[2]), ...). The `min` function then finds the smallest difference between these successive pairs.

3. **List Comprehension to Collect Pairs with Minimum Difference**:
   After finding the minimum absolute difference, we step through the array again with a list comprehension `[ [a, b] for a, b in pairwise(arr) if b - a == mi ]`. This comprehension checks each adjacent pair again to see if their difference is equal to the previously found minimum difference (`mi`). When a pair matches, it is added to the output list.

4. **Returning the Result**:
   The list of collected pairs that satisfy the minimum absolute difference condition is returned as the final result.

The algorithms and patterns used in this solution are:

- **Sorting**: A fundamental operation that reorders the items of a list (or any other sequence) to make subsequent operations (like finding a minimum difference) more efficient.
- **List Comprehension**: A convenient and readable way to construct lists in Python, used here for collecting the required pairs.
- **Generator Expression**: Used to generate values on the fly; in this case, it helps to find the minimum absolute difference without creating an intermediate list of differences.
- **Itertools pairwise() Utility**: Although `pairwise` is not an inbuilt function in Python versions prior to 3.10, it can be found in the `itertools` module or be custom implemented. It's a utility to abstract away the complexity of iterating over pairs of consecutive elements.

By breaking down the problem into these steps, the implementation remains clean, efficient, and easy to understand.

Example Walkthrough

Let's walk through a small example to demonstrate how the solution approach is applied. Suppose we have the following array of distinct integers:

```
1 arr = [4, 2, 1, 3]
```

We are tasked with finding all pairs of elements that have the minimum absolute difference. Here's how we will apply the explained solution approach to this array:

1. **Sort the Array**:

   First, we will sort the array:

   ```
   1 arr.sort() => arr = [1, 2, 3, 4]
   ```

   After sorting, the array's elements are arranged in ascending order, making it easier to find pairs with the minimum absolute difference, which will be between neighbors.

2. **Find the Minimum Absolute Difference**:

   Next, we examine adjacent pairs to find the smallest difference. Using the `pairwise()` function (or iterating through neighboring pairs manually), we get:

   ```
   1 Pairs: [1, 2], [2, 3], [3, 4]
   2 Differences: 1, 1, 1
   ```

   The minimum difference here is `1`. This value is determined by iterating through the pairs and finding the smallest difference, which is achieved using:

   ```
   1 min(b - a for a, b in pairwise(arr))
   ```

3. **Collect Pairs with Minimum Difference**:

   Now that we know the minimum difference is `1`, we look for all pairs with this difference:

   ```
   1 [ [a, b] for a, b in pairwise(arr) if b - a == 1 ]
   2 Resulting Pairs: [[1, 2], [2, 3], [3, 4]]
   ```

   Each of these pairs has the same difference of `1`, which is the minimum we found in the previous step.

4. **Returning the Result**:

   Lastly, we return the resultant pairs:

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

This is the final result, containing all pairs of elements in `arr` which have the smallest absolute difference of any two elements, formatted according to the problem requirements. The pairs are already in ascending order due to the initial sorting of `arr`.

Through this example, the implementation follows the steps laid out in the solution approach and demonstrates the effectiveness of sorting, generator expressions, and list comprehensions to solve the problem with minimum absolute difference pairs efficiently.

Python Solution

```python
from itertools import pairwise  # Import the pairwise function from itertools

class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        arr.sort()  # Sort the array
        # Find the minimum absolute difference between any two consecutive elements
        min_diff = min(b - a for a, b in pairwise(arr))

        # Create a list of pairs that have the minimum absolute difference
        result = [[a, b] for a, b in pairwise(arr) if b - a == min_diff]

        return result  # Return the list of pairs with the minimum absolute difference

# Note: If the `pairwise` function is not available, you can replace it with a direct implementation as shown below.
# This could be important if you are using a Python version older than 3.10, where `pairwise` was introduced.
```

If you're using a Python version older than 3.10, which might not have the `pairwise` utility available from `itertools`, you can rewrite the `pairwise` functionality yourself as follows:

```python
class Solution:
    def minimumAbsDifference(self, arr: List[int]) -> List[List[int]]:
        arr.sort()  # Sort the array to obtain elements in ascending order

        # Helper function to iterate over the array in pairs
        def pairwise(iterable):
            a, b = iter(iterable)
            next(b, None)
            return zip(a, b)

        # Compute the minimum difference between consecutive elements
        min_diff = min(b - a for a, b in pairwise(arr))

        # Create a list of pairs with the minimum difference
        result = [[a, b] for a, b in pairwise(arr) if b - a == min_diff]

        return result  # Return the resulting list of pairs

# This code is compatible with Python versions older than 3.10, because it defines a custom pairwise function.
```

Java Solution

```java
import java.util.Arrays; // Import necessary classes
import java.util.List;
import java.util.ArrayList;

class Solution {
    public List<List<Integer>> minimumAbsDifference(int[] arr) {
        // Sort the array to ensure that the pairs with the smallest absolute difference are adjacent
        Arrays.sort(arr);
        int n = arr.length; // Total number of elements in the array
        int minDifference = Integer.MAX_VALUE; // Initialize minimum difference with the maximum possible integer value

        // Find the minimum absolute difference between consecutive elements
        for (int i = 0; i < n - 1; ++i) {
            minDifference = Math.min(minDifference, arr[i + 1] - arr[i]);
        }

        // Prepare a list to hold pairs with the minimum absolute difference
        List<List<Integer>> result = new ArrayList<>();

        // Iterate over the array again to find all pairs with the minimum absolute difference
        for (int i = 0; i < n - 1; ++i) {
            // If the absolute difference between the minimum difference found, add the pair to the result list
            if (arr[i + 1] - arr[i] == minDifference) {
                result.add(List.of(arr[i], arr[i + 1]));
            }
        }

        // Return all pairs with the minimum absolute difference
        return result;
    }
}
```

C++ Solution

```cpp
class Solution {
public:
    vector<vector<int>> minimumAbsDifference(vector<int>& arr) {
        // Sort the input array
        sort(arr.begin(), arr.end());

        // Initialize the minimum absolute difference as a large value.
        int minDifference = INT_MAX;

        // Calculate the size of the input array.
        int n = arr.size();

        // Loop through the sorted array to find the minimum absolute difference.
        for (int i = 0; i < n - 1; ++i) {
            minDifference = min(minDifference, arr[i + 1] - arr[i]);
        }

        // Initialize an empty vector to store pairs with the minimum absolute difference.
        vector<vector<int>> result;

        // Loop through the array once more to find all pairs with the minimum absolute difference.
        for (int i = 0; i < n - 1; ++i) {
            // If the current pair has the minimum absolute difference,
            if (arr[i + 1] - arr[i] == minDifference) {
                // Add the pair to the result vector.
                result.push_back({arr[i], arr[i + 1]});
            }
        }

        // Return the vector containing all pairs of integers with the minimum absolute difference.
        return result;
    }
};
```

Typescript Solution

```typescript
function minimumAbsDifference(arr: number[]): number[][] {
    // Sort the input array in non-decreasing order
    arr.sort((a, b) => a - b);

    // Initialize the minimum difference to a large number
    let minDifference = Number.MAX_SAFE_INTEGER;

    // Calculate the number of elements in the array
    const n = arr.length;

    // Find the minimum absolute difference between any two adjacent elements
    for (let i = 0; i < n - 1; ++i) {
        minDifference = Math.min(minDifference, arr[i + 1] - arr[i]);
    }

    // Prepare an array to store pairs with the minimum absolute difference
    const results: number[][] = [];

    // Populate the result array with pairs of numbers that have the minimum difference
    for (let i = 0; i < n - 1; ++i) {
        if (arr[i + 1] - arr[i] === minDifference) {
            results.push([arr[i], arr[i + 1]]);
        }
    }

    // Return the array with all pairs having the minimum absolute difference
    return results;
}
```

Time and Space Complexity

The given Python code snippet defines a function `minimumAbsDifference` that finds all the pairs of elements in a list with the smallest absolute difference. Here is the analysis of its computational complexity:

Time Complexity

1. Sorting the array: `arr.sort()` is used, which has a time complexity of $O(n \log n)$ where n is the number of elements in `arr`.
2. Finding the minimum absolute difference: The comprehension `min(b - a for a, b in pairwise(arr))` iterates over the array once, which gives a time complexity of $O(n-1)$, simplifying to $O(n)$.
3. Generating the list of pairs with the smallest absolute difference: The list comprehension `[[a, b] for a, b in pairwise(arr) if b - a == mi]` also iterates over the array once, resulting in the time complexity of $O(n)$.

Since the sorting step dominates the overall time complexity, the final time complexity of the entire function is $O(n \log n)$.

Space Complexity

1. The sorted list: In-place sorting is used so no additional space complexity is introduced for sorting beyond the variable reassignments, which is $O(1)$.
2. The minimum absolute difference: The minimum value `mi` is found with a generator expression, so the space complexity for this step is $O(1)$.
3. The output list: The space complexity is proportional to the number of pairs with the smallest absolute difference. In the worst case, all pairs have the same minimum absolute difference, thus the space complexity is $O(n)$.

So, the space complexity for the function is $O(n)$ with respect to the output size.