2740. Find the Value of the Partition

Medium Array Sorting

Problem Description

In this problem, we are presented with an array of positive integers named `nums`. Our task is to divide this array into two non-empty sub-arrays, `nums1` and `nums2`, in such a way that the difference between the maximum value in `nums1` and the minimum value in `nums2` is as small as possible. This difference is referred to as the "value of the partition," expressed mathematically as `|max(nums1) - min(nums2)|`. We need to find and return the smallest possible value of this partition.

Intuition

The intuition behind the solution is based on the fact that the smallest difference between the maximum of one subset and the minimum of another is achieved when the elements in both subsets are as close to each other as possible. Since `nums` contains only positive integers, the optimal partition would place consecutive elements from the sorted order of `nums` into `nums1` and `nums2`. Sorting `nums` ensures that any two adjacent numbers have the smallest possible difference. After sorting, we can iterate through `nums`, taking one element to `nums1` and the next to `nums2`, while keeping track of the minimum value of `|nums[i+1] - nums[i]|` for all `i`. This minimum value will occur between two consecutive elements in the sorted array, forming the optimal boundary between `nums1` and `nums2`. Using the `pairwise` function from Python, we can easily iterate through adjacent pairs of elements to find this boundary, resulting in the minimum partition value.

Solution Approach

The solution to this problem uses sorting and a simple iteration to determine the minimum partition value. The algorithm involves the following steps:

1. **Sorting**: The first action is to sort the input array `nums`. Sorting is a fundamental step because it allows us to consider elements in their natural order to find the smallest difference between adjacent elements. Python's built-in `.sort()` method sorts the array in place in ascending order. The time complexity of the sorting operation is O(N log N), where N is the number of elements in `nums`.
2. **Iterating with Adjacent Pairs**: Once the array is sorted, the next step is to iterate through the array and consider each pair of adjacent elements. This is where the minimum possible partition value will occur. To do this, we can use the `pairwise` function, which is a handy tool for iterating over a list in overlapping pairs. This function simplifies the process of comparing each element with its subsequent neighbor without the need for complex index management.
3. **Calculating the Minimum Partition Value**: For each pair of adjacent elements `(a, b)` generated by `pairwise(nums)`, we calculate the difference `b - a`. The partition value is the absolute difference between the maximum of one partition and the minimum of the second partition, so by considering differences between consecutive elements, we are directly calculating potential partition values. The minimum partition value is then found by taking the minimum of all these differences using the `min` function.
4. **Return the Result**: Finally, the function returns the minimum partition value that was calculated.

No additional data structures are required other than the space needed for sorting. The `pairwise` function generates a tuple for each pair of elements, which uses only a constant amount of additional space. Overall, the algorithm is efficient and straightforward, with the sorting step dominating the overall time complexity.

Here's how the solution might look in Python:

```python
from itertools import pairwise

class Solution:
    def findValueOfPartition(self, nums: List[int]) -> int:
        nums.sort()
        return min(b - a for a, b in pairwise(nums))
```

In this code, `pairwise(nums)` is an iterable that gives us each adjacent pair `(a, b)` from the sorted `nums`, and the generator expression `min(b - a for a, b in pairwise(nums))` calculates the minimum difference between any two consecutive elements in the array, which corresponds to the minimum partition value.

Example Walkthrough

Let's go through a simple example to illustrate the solution approach described above. Consider the following array of numbers:

```
nums = [4, 2, 5, 1]
```

Following the provided solution approach, here is how we would find the smallest possible value of the partition:

1. **Sorting**: We start by sorting the array.

   Before sorting: `nums = [4, 2, 5, 1]`

   After sorting: `nums = [1, 2, 4, 5]`

   Sorting ensures that we consider the elements in increasing order to find the smallest difference between adjacent elements.
2. **Iterating with Adjacent Pairs**: We use the `pairwise` function to iterate through the sorted array in adjacent pairs:

   Adjacent pairs: `(1, 2), (2, 4), (4, 5)`
3. **Calculating the Minimum Partition Value**: We calculate the difference between each pair of adjacent numbers:
   - For the pair `(1, 2)`, the difference is `2 - 1 = 1`.
   - For the pair `(2, 4)`, the difference is `4 - 2 = 2`.
   - For the pair `(4, 5)`, the difference is `5 - 4 = 1`.

   We are looking for the minimum of these differences.
4. **Return the Result**: The smallest difference from our pairs is `1` (which occurs between the pairs `(1, 2)` and `(4, 5)`).

Therefore, the smallest possible value of the partition for the input `nums = [4, 2, 5, 1]` is `1`. When we apply the provided Python code to this array, the function `findValueOfPartition` returns `1` as the result of the calculation.

Solution Implementation

Python

````
from itertools import tee

class Solution:
    def find_value_of_partition(self, nums: List[int]) -> int:
        # Sort the input list in ascending order
        nums.sort()

        # The 'pairwise' utility is not available in Python standard library
        # until Python 3.10. A custom implementation is needed for earlier versions.
        # Here is a custom implementation of pairwise utility using tee and zip.
        def pairwise(iterable):
            "s -> (s0,s1), (s1,s2), (s2, s3), ..."
            a, b = tee(iterable)
            next(b, None)
            return zip(a, b)

        # Calculate the minimum difference between consecutive elements in the sorted list
        # This difference represents the smallest partition value
        return min(b - a for a, b in pairwise(nums))
```

Please note that you need to import the `List` typing from the `typing` module to use list type hints. Here's how you include it:

```python
from typing import List
````

Java

```java
class Solution {
    public int findValueOfPartition(int[] nums) {
        // Sort the array to bring similar values closer.
        Arrays.sort(nums);

        // Initialize the minimum difference to a large value.
        // Instead of 1 << 30, Integer.MAX_VALUE is used for readability.
        int minValueOfPartition = Integer.MAX_VALUE;

        // Loop through the sorted array starting from the second element
        for (int i = 1; i < nums.length; ++i) {
            // Update the minValueOfPartition with the smallest difference found
            // between adjacent elements in the sorted array.
            minValueOfPartition = Math.min(minValueOfPartition, nums[i] - nums[i - 1]);
        }

        // Return the minimum value of the partition found.
        return minValueOfPartition;
    }
}
```

C++

```cpp
#include <vector> // Necessary for using std::vector
#include <algorithm> // Necessary for using std::sort and std::min

class Solution {
public:
    // Function to find the minimum difference between any two elements after sorting the array.
    int findValueOfPartition(std::vector<int>& nums) {
        // Sort the array in non-decreasing order.
        std::sort(nums.begin(), nums.end());

        // Initialize the answer with a large value.
        // INT_MAX from limits.h could also be used for maximum allowable integer.
        int minDifference = INT_MAX;

        // Iterate over the sorted array to find the smallest difference
        // between consecutive elements.
        for (int i = 1; i < nums.size(); ++i) {
            // Update minDifference with the smallest difference found so far.
            minDifference = std::min(minDifference, nums[i] - nums[i - 1]);
        }

        // Return the minimum difference found.
        return minDifference;
    }
};
```

TypeScript

```typescript
// Function to find the minimum difference between any two elements after sorting the array.
function findValueOfPartition(nums: number[]): number {
    // Sort the array in non-decreasing order.
    nums.sort((a, b) => a - b);

    // Initialize the answer with a large value. TypeScript's maximum safe integer can be used.
    let minDifference: number = Number.MAX_SAFE_INTEGER;

    // Iterate over the sorted array to find the smallest difference
    // between consecutive elements.
    for (let i = 1; i < nums.length; i++) {
        // Update minDifference with the smallest difference found so far.
        minDifference = Math.min(minDifference, nums[i] - nums[i - 1]);
    }

    // Return the minimum difference found.
    return minDifference;
}
```

````
from itertools import tee

class Solution:
    def find_value_of_partition(self, nums: List[int]) -> int:
        # Sort the input list in ascending order
        nums.sort()

        # The 'pairwise' utility is not available in Python standard library
        # until Python 3.10. A custom implementation is needed for earlier versions.
        # Here is a custom implementation of pairwise utility using tee and zip.
        def pairwise(iterable):
            "s -> (s0,s1), (s1,s2), (s2, s3), ..."
            a, b = tee(iterable)
            next(b, None)
            return zip(a, b)

        # Calculate the minimum difference between consecutive elements in the sorted list
        # This difference represents the smallest partition value
        return min(b - a for a, b in pairwise(nums))
```

Please note that you need to import the `List` typing from the `typing` module to use list type hints. Here's how you include it:

```python
from typing import List
````

Time and Space Complexity

Time Complexity

The given code has two main operations that dictate the time complexity: sorting the list `nums` and then computing the minimum difference between consecutive elements.

The sorting of a list of `n` elements has a time complexity of `O(n log n)` using the Timsort algorithm, which is the default sorting algorithm in Python.

The `pairwise` function generates tuples containing pairs of consecutive elements in the sorted list. Iterating through the `nums` list to find the minimum difference is a linear operation with a time complexity of `O(n - 1)`, which simplifies to `O(n)`.

Combining both, the overall time complexity is `O(n log n) + O(n)`. As `O(n log n)` is the dominating factor, the final time complexity is: `O(n log n)`

Space Complexity

The space complexity of the code is determined by the additional space required for sorting and the space required for the `pairwise` iterator.

The sort operation can be done in-place, so it does not significantly add to the space complexity, thus being `O(1)`.

The `pairwise` function, however, creates a new iterator that generates pairs of consecutive elements without generating all pairs at once. Therefore, the space complexity due to `pairwise` is `O(1)`.

Combining both the space complexities from sorting and `pairwise`, the overall space complexity is: `O(1)`