2733. Neither Minimum nor Maximum

Easy Array Sorting

Problem Description

The problem presents us with an integer array called `nums`, which contains distinct positive integers. This means that all the elements in the array are unique and greater than zero. The task is to identify a number in this array that is neither the smallest (minimum) nor the largest (maximum) value present. If such a number exists, it should be returned as the output. If no such number can be found, which would likely be the case if the array has too few elements to have a middle value, then the function should return `-1`.

The problem specifies that the function should return any such "middle" value. This means that if there are multiple numbers satisfying the condition of being neither the minimum nor the maximum, any one of them can be chosen as the correct answer.

Intuition

To solve this problem, we can follow a straightforward approach. Since we are given that all the numbers in the array are distinct, we know that there will be one unique minimum value and one unique maximum value.

The first step is to find these two values, the smallest and the largest number in the array, which we can do efficiently by using the `min` and `max` functions respectively.

Once we have the smallest and largest values, we can then iterate through the array to find an element that is neither of these. As soon as we find such an element, we can return it, since the problem does not require us to do anything other than find one such 'middle' number.

- We use a loop to check each number.
- For each number, if it is not equal to the minimum and not equal to the maximum, we have found a valid "middle" number and we return it immediately.
- If the loop completes without finding such an element, this means that all elements are either the minimum or the maximum, and we return `-1` as per the problem's instructions.

This approach ensures that we look at each element of the array at most once, making the solution efficient and straightforward.

Solution Approach

The implementation of the solution uses a very basic algorithmic approach with simple control structures. No advanced data structures or design patterns are required, just the built-in Python list and the straightforward use of the `min` and `max` functions.

The core parts of the solution can be broken down as follows:

1. **Finding the Minimum and Maximum**: We first find the smallest (`mi`) and largest (`mx`) numbers in the array using the `min(nums)` and `max(nums)` functions. These functions traverse the array, and each one completes in `O(n)` time where `n` is the number of elements in the array.

```
mi, mx = min(nums), max(nums)
```

2. **Iterating Through the Array**: Next, we iterate through each element `x` in the array `nums`. This is done using a for loop.

```
for x in nums:
```

3. **Checking the Condition**: In each iteration, we check if the current element `x` is not equal to the minimum value `mi` and not equal to the maximum value `mx`.

```
if x != mi and x != mx:
```

4. **Returning the Middle Value**: If `x` is neither the minimum nor maximum, we have found a valid number that fits the problem's criteria, and we can return it immediately.

```
return x
```

5. **Returning -1 If No Number Found**: If we go through the entire array without finding a number that satisfies the condition, it implies that no such number exists (for example, the array might only contain the minimum and maximum values). In that case, after the loop concludes, we return `-1`.

```
return -1
```

This algorithm is linear in nature since it involves a single pass through the array to find `min` and `max` and another pass to find a non-minimum and non-maximum element. Hence, the overall time complexity is `O(n)` because the array is traversed at most twice.

Note: The assumption is that the input array `nums` provided to the solution function is valid as per the problem description and contains distinct positive integers only.

Example Walkthrough

Let's illustrate the solution approach with an example:

Assume our input array `nums` is `[3, 1, 4, 2]`.

1. **Finding the Minimum and Maximum**: We find the smallest number (`mi`) is `1` and the largest number (`mx`) is `4`.

```
mi, mx = min(nums), max(nums) # mi = 1, mx = 4
```

2. **Iterating Through the Array**: We iterate through each element `x` in the array `[3, 1, 4, 2]`, using a for loop.

3. **Checking the Condition**:
   - First iteration: `x = 3`, `x != mi` is `True`, and `x != mx` is also `True`. Since both conditions are satisfied, we have found a "middle" number.

```
for x in nums:
    if x != mi and x != mx:
        return x
```

4. **Returning the Middle Value**: We return `3` immediately because it satisfies the condition of being neither the minimum nor the maximum value in the array.

In this case, the function would have returned `3` on the first iteration, illustrating that the algorithm efficiently finds a number that is neither the smallest nor the largest in the array without needing to check the rest of the elements. If the array had been `[2, 1]`, the loop would have completed without finding a suitable middle number, so the function would return `-1`.

Solution Implementation

Python

```
from typing import List

class Solution:
    def findNonMinOrMax(self, nums: List[int]) -> int:
        """
        This method returns the first element from the input list 'nums'
        that is not the minimum or maximum value in the list.
        If all elements are the minimum or maximum, -1 is returned.

        :param nums: List[int] - List of integers to search through
        :return: int - The first non-minimum and non-maximum value, or -1 if not found
        """

        # Find the minimum and maximum values in the list
        min_value, max_value = min(nums), max(nums)

        # Iterate over the list to find an element that is not minimum or maximum
        for num in nums:
            if num != min_value and num != max_value:
                return num  # Return the first non-min/max number

        # If all elements are the same (i.e., min equals max), return -1
        return -1
```

Java

```
class Solution {
    // Method to find an element which is neither the minimum nor the maximum in the array.
    public int findNonMinOrMax(int[] nums) {
        // Initialize minimum and maximum with extreme values which are beyond
        // the expected range of elements in the array.
        int minimum = Integer.MAX_VALUE;
        int maximum = Integer.MIN_VALUE;

        // Loop through all elements in the array to find the minimum and maximum values.
        for (int num : nums) {
            minimum = Math.min(minimum, num);
            maximum = Math.max(maximum, num);
        }

        // Loop through the array again to find an element that is not equal to
        // the minimum or maximum value.
        for (int num : nums) {
            if (num != minimum && num != maximum) {
                return num; // Return the first non min/max element found.
            }
        }

        // If all elements are either min or max or there's only one element, return -1.
        return -1;
    }
}
```

C++

```
class Solution {
public:
    // Function to find an element in the vector that is not the minimum or maximum
    int findNonMinOrMax(vector<int>& nums) {
        // Find the minimum element in nums using min element algorithm
        int minElement = *min_element(nums.begin(), nums.end());
        // Find the maximum element in nums using max element algorithm
        int maxElement = *max_element(nums.begin(), nums.end());
        // Iterate through all elements in the vector
        for (int element : nums) {
            // Check if current element is neither the minimum nor the maximum
            if (element != minElement && element != maxElement) {
                // Return the first element found that is not a min or max
                return element;
            }
        }
        // If there is no such element, return -1
        return -1;
    }
};
```

TypeScript

```
// Import statements for necessary functionalities
import { min, max } from 'lodash';

// Function to find an element in the array that is not the minimum or maximum
function findNonMinOrMax(nums: number[]): number {
    // Find the minimum element in nums
    const minElement = min(nums);
    // Find the maximum element in nums
    const maxElement = max(nums);

    // Iterate through all elements in the array
    for (const element of nums) {
        // Check if current element is neither the minimum nor the maximum
        if (element !== minElement && element !== maxElement) {
            // Return the first element found that is not a min or max
            return element;
        }
    }

    // If there is no such element, return -1
    return -1;
}
```

```
from typing import List

class Solution:
    def findNonMinOrMax(self, nums: List[int]) -> int:
        """
        This method returns the first element from the input list 'nums'
        that is not the minimum or maximum value in the list.
        If all elements are the minimum or maximum, -1 is returned.

        :param nums: List[int] - List of integers to search through
        :return: int - The first non-minimum and non-maximum value, or -1 if not found
        """

        # Find the minimum and maximum values in the list
        min_value, max_value = min(nums), max(nums)

        # Iterate over the list to find an element that is not minimum or maximum
        for num in nums:
            if num != min_value and num != max_value:
                return num  # Return the first non-min/max number

        # If all elements are the same (i.e., min equals max), return -1
        return -1
```

Time and Space Complexity

// The time complexity of the `findNonMinOrMax` method is `O(n)`, where `n` is the number of elements in the input list `nums`. This is because `min(nums)` and `max(nums)` both iterate over the entire list individually, each taking `O(n)` time. The subsequent for-loop also iterates over the list once, leading to a linear time complexity overall.

// The space complexity of the method is `O(1)` since it only uses a fixed amount of additional space for the variables `mi`, `mx`, and `x` regardless of the size of the input list.