*Here is a complete documentation for the solution to the* **"Third Maximum Number"** *problem:*

## 1. Problem Statement

*Given an integer array nums, we are tasked with returning the third distinct maximum number in the array. If the third distinct maximum does not exist, return the maximum number.*

### Example 1:

- **Input:** nums = [3,2,1]
- **Output:** 1
- **Explanation:** The first distinct maximum is 3. The second distinct maximum is 2. The third distinct maximum is 1.

### Example 2:

- **Input:** nums = [1,2]
- **Output:** 2
- **Explanation:** The first distinct maximum is 2. The second distinct maximum is 1. The third distinct maximum does not exist, so we return the maximum (2).

### Example 3:

- **Input:** nums = [2,2,3,1]
- **Output:** 1
- **Explanation:** The first distinct maximum is 3. The second distinct maximum is 2 (both 2's are counted together). The third distinct maximum is 1.

## 2. Intuition

The problem asks us to find the third largest distinct number in an array. Given that duplicate numbers don't matter for this task, we can reduce the array to unique values and then sort them to identify the third largest.

## 3. Key Observations

1. **Distinct Numbers**: Duplicates are irrelevant. Only the distinct values of the array matter for finding the third maximum.
2. **Array Length**: If there are fewer than three distinct numbers, we return the maximum.
3. **Sorting**: Sorting the distinct numbers in descending order allows us to easily access the top 3 maximum values.
4. **Edge Cases**: Arrays with fewer than three distinct values, or arrays with negative or very large numbers, need special handling.

## 4. Approach

*We can solve this problem in three steps:*

1. **Remove duplicates**: Convert the input list to a set to retain only unique elements.
2. **Sort the set**: Sort the distinct numbers in descending order so that the largest values come first.
3. **Return the result**: If the sorted list contains at least three elements, return the third element. Otherwise, return the first element (the largest number).

## 5. Edge Cases

- **Array with fewer than 3 distinct numbers**: If there are fewer than 3 distinct elements, return the largest one.
- **Array with all identical elements**: In this case, there's only one distinct number, so return that.
- **Negative numbers**: Handle correctly since negative numbers can be part of the largest or smallest distinct numbers.
- **Empty array**: This is not a valid case based on the problem's constraints (1 <= nums.length), but if encountered, it would ideally return an error or a specific value.

## 6. Complexity Analysis

### Time Complexity

- ***O(n log n)***: Converting the array into a set takes O(n) time. Sorting the distinct set takes O(k log k), where k is the number of distinct elements, which can be at most n. In the worst case, k = n, so the time complexity is O(n log n).

### Space Complexity

- ***O(n)***: We use a set to store distinct numbers, so in the worst case, the space complexity is O(n) where n is the number of elements in the input array.

## 7. Alternative Approaches

1. **Heap-based Approach**: Use a min-heap to track the top 3 distinct maximum numbers. This would allow us to keep track of the largest numbers without sorting the entire array. However, this would still have a time complexity of O(n log 3), which simplifies to O(n).
2. **Tracking Maximums with Three Variables**: Instead of sorting or using a heap, we could track the three largest distinct numbers manually using three variables. This approach would have a time complexity of O(n) but could be harder to implement and understand compared to the sorting approach.

## 9. Test Cases

### Test Case 1:

***Input***: [3, 2, 1]
***Output***: 1
***Explanation***: The distinct numbers sorted in descending order are [3, 2, 1]. The third maximum is 1.

### Test Case 2:

***Input:*** [1, 2]

***Output:*** 2

***Explanation:*** The distinct numbers sorted in descending order are [2, 1]. There is no third distinct maximum, so the maximum 2 is returned.

### Test Case 3:

***Input:*** [2, 2, 3, 1]

***Output:*** 1

***Explanation:*** The distinct numbers sorted in descending order are [3, 2, 1]. The third maximum is 1.

### Test Case 4:

***Input:*** [5, 5, 5, 5]

***Output:*** 5

***Explanation:*** All elements are the same, so the third maximum does not exist. We return the only number 5.

### Test Case 5:

***Input:*** [-1, -2, -3, -4]

***Output:*** -3

***Explanation:*** The distinct numbers sorted in descending order are [-1, -2, -3, -4]. The third maximum is -3.

## 9. Final Thoughts

This solution efficiently handles the problem in O(n log n) time by leveraging sorting, while ensuring correctness through distinct number tracking. Alternative approaches like using a heap or manually managing the top 3 maximums could provide O(n) time complexity but may introduce more complexity in implementation. Given the constraints and simplicity, the sorting-based approach is an optimal solution for this problem.