# Documentation for the Maximum Gap Problem Solution

## Problem Statement:

Given an integer array nums, the task is to find the maximum difference between two successive elements in its sorted form. If the array contains fewer than two elements, the function should return 0. The solution must be implemented in linear time (O(n)) and use linear extra space (O(n)).

## Key Concepts:

### 1. Edge Case:

- If the array has fewer than two elements, no meaningful gap can be computed. Therefore, the function should return 0 immediately for this case.

### 2. Minimum and Maximum Calculation:

- The minimum (min_num) and maximum (max_num) elements in the array are identified. These values are critical for determining the range of the data and setting up the buckets for the next steps.

### 3. Bucket Sort Principle:

- The algorithm is based on the bucket sort technique, which distributes the numbers into a series of buckets. The idea is that the maximum gap will likely be found between two consecutive buckets rather than within the same bucket.

**4. Bucket Size Calculation:**

- *The bucket size is determined using the formula:*

  [ text{Bucket Size} = max(1, frac{text{max} text{min}}{n 1})]

- This formula ensures that the numbers are distributed across the buckets efficiently, minimizing the chances that the maximum gap will be within a single bucket.

**5. Bucket Count:**

- *The number of buckets is calculated as:*

  [text{Number of Buckets} = frac{text{max} text{min}}{text{Bucket Size}} + 1]

- This accounts for all possible values in the range from min_num to max_num.

**6. Bucket Initialization:**

- Each bucket is initialized to hold the minimum and maximum values encountered for that bucket. Initially, these are set to None to indicate that they are empty.

**7. Distributing the Elements:**

- Each element in the array is placed into its appropriate bucket based on its value. *The placement is determined by the formula:*

  [text{Bucket Index} = frac{text{num} text{min_num}}{text{Bucket Size}}]

- The bucket's minimum and maximum values are updated accordingly as elements are assigned.

8. **Computing the Maximum Gap:**

- The maximum gap is found by iterating through the buckets and calculating the difference between the minimum value of the current bucket and the maximum value of the previous non-empty bucket.
- The global maximum gap is updated whenever a larger gap is found.

9. **Result:**

- The final maximum gap is returned as the output, which represents the largest difference between successive elements in the sorted form of the array.

## Complexity Analysis:

1. **Time Complexity:**

- The algorithm runs in linear time, O(n), as it only requires a few passes through the array for finding the min/max, distributing elements to buckets, and calculating the maximum gap.

2. **Space Complexity:**

- The algorithm uses linear extra space, O(n), mainly for the bucket storage, making it suitable for large input sizes.

## Benefits of this Approach:

- **Efficiency:** The algorithm meets the problem's requirement of linear time complexity, making it suitable for large datasets.

- **Simplicity:** By using the bucket sort technique, the problem of finding the maximum gap is simplified to operations on buckets, which are easier to manage and compute.

- **Scalability:** The solution scales well with increasing array sizes due to its linear characteristics in both time and space.

## Assumptions and Constraints:

**Input Size:** The algorithm assumes that the input array size n is within the range [1, $10^5$].

**Element Values:** The values in the array are non-negative integers within the range [0, $10^9$].

**Output:** The output is always a non-negative integer representing the maximum gap between successive elements in the sorted array.

## Example 1:

- **Input:** nums = [3, 6, 9, 1]
- **Output:** 3
- **Explanation:** The sorted form of the array is [1, 3, 6, 9]. The maximum gap is between 6 and 9.

## Example 2:

- **Input:** nums = [10]
- **Output:** 0
- **Explanation:** The array has fewer than two elements, so the output is 0.

## Conclusion:

This approach efficiently solves the problem by leveraging the bucket sort technique to ensure that the solution is both time and space efficient, aligning with the problem's constraints. The use of buckets helps to isolate the maximum gap between successive elements, which is the primary goal of the algorithm.