1. Odd String Difference

Python
```
def difference_array(word):
  return [ord(b) - ord(a) for a, b in zip(word, word[1:])]

def find_odd_string(words):
  # Use a set to efficiently check for duplicate difference arrays
  unique_diffs = set(difference_array(word) for word in words)
  for word in words:
    if difference_array(word) not in unique_diffs:
      return word

# Example usage
words = ["adc", "wzy", "abc"]
odd_string = find_odd_string(words)
print(odd_string)  # Output: abc
```

2. Words Within Two Edits of Dictionary

Python
```
def min_edits(word1, word2):
  m, n = len(word1), len(word2)
  # Create a DP table to store edit distances
  dp = [[0] * (n + 1) for _ in range(m + 1)]
  for i in range(m + 1):
    dp[i][0] = i  # Base case: delete all characters from word1
  for j in range(n + 1):
    dp[0][j] = j  # Base case: insert all characters from word2
  for i in range(1, m + 1):
    for j in range(1, n + 1):
      if word1[i - 1] == word2[j - 1]:
        dp[i][j] = dp[i - 1][j - 1]  # No edit needed if characters match
      else:
        dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1
        # Minimum edit distance: replace, insert, or delete
  return dp[m][n]

def find_words_within_two_edits(queries, dictionary):
  results = []
  for query in queries:
    for word in dictionary:
      if min_edits(query, word) <= 2:
        results.append(query)
        break  # Only need to find one matching word in dictionary
  return results

# Example usage
queries = ["word", "note", "ants", "wood"]
dictionary = ["wood", "joke", "moat"]
words_within_two_edits = find_words_within_two_edits(queries, dictionary)
print(words_within_two_edits)  # Output: ["word", "note", "wood"]
```

3. Destroy Sequential Targets

Python
```
def destroy_targets(nums, seed, space):
  count = 0
  for target in nums:
```

```
    if target % (seed + space) == 0:
      count += 1
  return count

def find_best_seed(nums, space):
  best_seed = float('inf')  # Initialize with positive infinity
  best_destruction = 0
  for seed in nums:
    destruction = destroy_targets(nums, seed, space)
    if destruction > best_destruction:
      best_destruction = destruction
      best_seed = seed
  return best_seed

# Example usage
nums = [3,7,8,1,1,5]
space = 2
best_seed = find_best_seed(nums, space)
print(best_seed)  # Output: 1
```

4. Minimum Addition to Make Integer Beautiful

Python

```
def make_beautiful(n, target):
  digit_sum = sum(int(d) for d in str(n))
  diff = target - digit_sum
  return max(0, diff)  # Return 0 if the number is already beautiful

# Example usage
n = 16
target = 6
min_addition = make_beautiful(n, target)
print(min_addition)  # Output: 4
```

Use code with caution.
content_copy

5. Sort Array by Moving Items to Empty Space

Python

```
def sort_by_moving(nums):
  n = len(nums)
  for i in range(n):
    # Check if the current element is not at its correct position
    if nums[i] != i and nums[i] != 0:
      # Find the correct position for the current element
      correct_index = nums[i]
      # Swap the current element with the element at the correct position
      while nums[correct_index] != 0:
        correct_index = nums[correct_index]
        nums[i], nums[correct_index] = nums[correct_index], nums[i]
      # Place the current element at its correct position
      nums[i], nums[correct_index] = nums[correct_index], nums[i]

# Example usage
nums = [4, 2, 0, 3, 1]
sort_by_moving(nums)
print(nums)  # Output: [1, 2, 3, 4, 0]
```

6. Apply Operations to an Array

Python

```
def apply_operations(nums):
  n = len(nums)
  for i in range(n - 1):
    if nums[i] == nums[i + 1]:
      nums[i] *= 2
      nums[i + 1] = 0
  # Shift zeroes to the end of the array using two pointers
  j = 0
  for i in range(n):
    if nums[i] != 0:
      nums[i], nums[j] = nums[j], nums[i]
      j += 1
  return nums

# Example usage
nums = [1, 2, 2, 1, 1, 0]
result = apply_operations(nums)
print(result)  # Output: [1, 4, 2, 0, 0, 0]
```