1. You are given two 0-indexed integer arrays nums1 and nums2, each of size n, and an integer diff. Find the number of pairs (i, j) such that: 0 <= i < j <= n - 1 and nums1[i] - nums1[j] <= nums2[i] - nums2[j] + diff. Return the number of pairs that satisfy the conditions. Example 1: Input: nums1 = [3,2,5], nums2 = [2,2,1], diff = 1 Output: 3 Explanation: There are 3 pairs that satisfy the conditions: 1. i = 0, j = 1: 3 - 2 <= 2 - 2 + 1. Since i < j and 1 <= 1, this pair satisfies the conditions. 2. i = 0, j = 2: 3 - 5 <= 2 - 1 + 1. Since i < j and -2 <= 2, this pair satisfies the conditions. 3. i = 1, j = 2: 2 - 5 <= 2 - 1 + 1. Since i < j and -3 <= 2, this pair satisfies the conditions
   Program:-
   ```
   from sortedcontainers import SortedList

   def countPairs(nums1, nums2, diff):
       n = len(nums1)
       arr = [nums1[i] - nums2[i] for i in range(n)]
       sorted_list = SortedList()
       count = 0

       for i in range(n):
           # Find how many elements in sorted_list are <= arr[i] + diff
           count += sorted_list.bisect_right(arr[i] + diff)
           # Insert current element in sorted_list for future comparisons
           sorted_list.add(arr[i])

       return count

   # Example usage
   nums1 = [3, 2, 5]
   nums2 = [2, 2, 1]
   diff = 1
   print(countPairs(nums1, nums2, diff))  # Output: 3
   ```

2. Given an integer n, return the nth digit of the infinite integer sequence [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...]. Example 1: Input: n = 3 Output: 3 Example 2: Input: n = 11 Output: 0 Explanation: The 11th digit of the sequence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ... is a 0, which is part of the number 10.

```
def findNthDigit(n):

    # Initialize the length of digits, start number, and count of numbers

    length = 1

    count = 9

    start = 1


    # Determine the range that contains the nth digit

    while n > length * count:
```

```
        n -= length * count
        length += 1
        count *= 10
        start *= 10

    # Determine the exact number that contains the nth digit
    start += (n - 1) // length
    s = str(start)

    # Return the exact digit within that number
    return int(s[(n - 1) % length])

# Example usage:
n1 = 3
print(findNthDigit(n1))  # Output: 3

n2 = 11
print(findNthDigit(n2))  # Output: 0
```

3. A string s is nice if, for every letter of the alphabet that s contains, it appears both in uppercase and lowercase. For example, "abABB" is nice because 'A' and 'a' appear, and 'B' and 'b' appear. However, "abA" is not because 'b' appears, but 'B' does not. Given a string s, return the longest substring of s that is nice. If there are multiple, return the substring of the earliest occurrence. If there are none, return an empty string. Example 1: Input: s = "YazaAay" Output: "aAa" Explanation: "aAa" is a nice string because 'A/a' is the only letter of the alphabet in s, and both 'A' and 'a' appear. "aAa" is the longest nice substring.

```
def longestNiceSubstring(s):
    # Helper function to check if a character c is "nice" in the string s
    def isNice(c, s):
        return c.lower() in s and c.upper() in s

    # Recursive function to find the longest nice substring
    def helper(s):
        if len(s) < 2:
            return ""

        for i in range(len(s)):
            if not isNice(s[i], s):
                # Split the string and recur on both halves
                left = helper(s[:i])
                right = helper(s[i+1:])
                # Return the longer nice substring
                return left if len(left) >= len(right) else right

        # If every character is "nice", return the string itself
        return s

    # Initial call to the recursive helper function
    return helper(s)

# Example usage:
s1 = "YazaAay"
print(longestNiceSubstring(s1))  # Output: "aAa"

s2 = "Bb"
print(longestNiceSubstring(s2))  # Output: "Bb"

s3 = "c"
print(longestNiceSubstring(s3))  # Output: ""
```

4. Given a sentence that consists of some words separated by a single space, and a searchWord, check if searchWord is a prefix of any word in sentence. Return the index of the word in sentence (1-indexed) where searchWord is a prefix of this word. If searchWord is a prefix of more than one word, return the index of the first word (minimum index). If there is

no such word return 1. A prefix of a string s is any leading contiguous substring of s. Example 1: Input: sentence = "i love eating burger", searchWord = "burg" Output: 4 Explanation: "burg" is prefix of "burger" which is the 4th word in the sentence.

```
def isPrefixOfWord(sentence, searchWord):
    # Split the sentence into words
    words = sentence.split()

    # Iterate through the words
    for i, word in enumerate(words):
        # Check if searchWord is a prefix of the current word
        if word.startswith(searchWord):
            return i + 1  # Return 1-indexed position

    # If no word is found with searchWord as a prefix, return -1
    return -1

# Example usage
sentence1 = "i love eating burger"
searchWord1 = "burg"
print(isPrefixOfWord(sentence1, searchWord1))  # Output: 4

sentence2 = "this problem is an easy problem"
searchWord2 = "pro"
print(isPrefixOfWord(sentence2, searchWord2))  # Output: 2

sentence3 = "i am tired"
searchWord3 = "you"
print(isPrefixOfWord(sentence3, searchWord3))  # Output: -1
```

5. you are given an integer array nums and two integers indexDiff and valueDiff.Find a pair of indices (i, j) such that: i != j, abs(i - j) <= indexDiff. abs(nums[i] - nums[j]) <= valueDiff, and Return true if such pair exists or false otherwise. Example 1: Input: nums = [1,2,3,1], indexDiff = 3, valueDiff = 0 Output: true Explanation: We can choose (i, j) = (0, 3). We satisfy the three conditions: i != j --> 0 != 3 abs(i - j) <= indexDiff --> abs(0 - 3) <= 3 abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0

   program :-

```
from sortedcontainers import SortedList

def containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff):
    if indexDiff < 0 or valueDiff < 0:
        return False
    sorted_list = SortedList()

    for i in range(len(nums)):
        # Remove the element that is out of the sliding window
        if i > indexDiff:
            sorted_list.remove(nums[i - indexDiff - 1])

        # Find the position where nums[i] - valueDiff would be inserted
        pos1 = sorted_list.bisect_left(nums[i] - valueDiff)

        # Check if there's any number in the sorted_list within the valueDiff range
        if pos1 < len(sorted_list) and abs(sorted_list[pos1] - nums[i]) <= valueDiff:
            return True

        # Add the current number to the sorted list
        sorted_list.add(nums[i])

    return False

# Example usage
nums = [1, 2, 3, 1]
indexDiff = 3
valueDiff = 0
print(containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff))  # Output: True

nums2 = [1, 5, 9, 1, 5, 9]
indexDiff2 = 2
valueDiff2 = 3
print(containsNearbyAlmostDuplicate(nums2, indexDiff2, valueDiff2))  # Output: False
```