CSCI3320 Programming Assignment #3

Problems Description: Given an array of integers (which can include both positive and negative values), you need to determine if there is any continuous sequence of elements in the array whose total sum equals **zero**. This sequence is formed by consecutive elements from the array, and the task is to check whether such a sequence exists and print the elements of the subarray whose sum is zero.

Example 1:
Array: [4, 2, -3, 1, 6]

- The sum of the **second**, **third**, and **fourth** elements is zero (i.e. $2 + (-3) + 1 = 0$). This means there exists a sequence of consecutive elements where the sum equals zero.

Output: Yes, there is a sequence where the sum of the elements equals zero.
Starting index: 1,
Ending index: 3
Sum: $2 + (-3) + 1 = 0$

Example 2:
Array: [1, -5, 3, 4, 5]

- None of these sums equal zero. Therefore, there is **no** sequence of consecutive numbers in this array whose sum is zero.

Output: No, there is no sequence where the sum of the elements equals zero.

Here is a solution with $O(n^2)$ complexity. **To earn points for the assignment, your algorithm must be more efficient than this one.**

Algorithm 1: Check the sum of all possible subarrays. If any sum is 0, return True.

```
def Alg1_brute(arr):
    n = len(arr)
    for i in range(n):
        sum = 0
        for j in range(i, n):
            sum += arr[j]
            if sum == 0:
                print(arr[i:j+1])
                return True
    return False
```

Your program should prompt the user for an input size (N) and generate a random sequence of N integers, with values ranging from -9999 to 9999. If N is less than 50, the program **must generate random numbers between -99 to 99**, **print all the randomly generated numbers**, **check if a subarray with a sum of zero exists, and display the elements of that subarray** if it does, as demonstrated in **Example 1** and **Example 2**.

Grading Criteria: Any solution with $O(n^2)$ or higher complexity will not earn any points.

- **Correctly finding indexes (50%)**: Your program must print all the random numbers if N is less than 50. Failure to do so will result in a penalty of at least 50%.
- **Solution complexity (30%)**:

- **Accuracy of complexity analysis (20%)**.

Deliverable: A **zipped file** including

- source codes with compiling/running instructions,
- a MS-Word document for the complexity analysis: Copy and paste your algorithm, and include a detailed complexity analysis of it.

I recommend using 'PyCharm' as your IDE. For the installation of PyCharm, please refer the following webpage: https://www.jetbrains.com/help/pycharm/installation-guide.html.

Do not use any non-standard Python library (such as NumPy).

For full credit, your code should be **well documented with comments**, and the style of your code should follow the guidelines given below:

Your programs must contain enough comments. **Programs without comments or with insufficient and/or vague comments may cost you 30%.**

Every user-defined method or function (that you added in the source files) should have a comment header describing inputs, outputs, and what it does. An example function comment is shown below:

```
def quadratic_max_subsequence(array: List[int]):
    """
    FUNCTION quadratic_max_subsequence:
        Find the subsequence in the array with the maximum sum
        using an algorithm with O(n^2) time complexity.

    INPUT_Parameters:
        array (List[int]): Array to evaluate for the largest subsequence.

    Returns:
        ArraySubSequence: The subsequence with the largest sum in the array.

    """
```

Inline comments should be utilized as necessary (but not overused) to **make algorithms clear** to the reader.

Not-compile programs receive 0 point. By **not-compile**, I mean any reason that could cause an unsuccessful compilation, including missing files, incorrect filenames, syntax errors in your programs, and so on. Double check your files before you submit, since I will **not** change your program to make it work.

Compile-but-not-run programs receive no more than 50%. Compile-but-not-run means you have attempted to solve the problem to certain extent, but you failed to make it working properly. A meaningless or vague program receives no credit even though it compiles successfully.

Programs delivering incorrect result, incomplete result, or incompatible output receive no more than 70%