SWE30009 – Software Testing and Reliability

Assignment 2 – Do Duy Hung – 104182779

Task 1: In Task 1, I will divide it into 6 concrete test cases corresponding to 6 different equivalence classes. The purpose of these 6 test cases is to help verify the classification of even and odd numbers (test cases 1, 2, 3, 4, 5), check the removal of duplicate elements (test case 1), and evaluate complexity in ascending order (test case 1, 4), check when working with big numbers (test case 6). Below, I will analyze them:

1, A sequence with some duplicate elements, which is expected removed in the output and 2 lists of output print odd and even number contains in ascending order

Input: [4, 9, 12, 7, 17, 5, 11, 9, 14, 6, 20, 2, 10, 5, 13]

Expected output: Even numbers: [2, 4, 6, 10, 12, 14, 20]
Odd numbers: [5, 7, 9, 11, 13, 17]

2, A sequence of all odd numbers, which has appeared all in the odd numbers output as an ascending order. However, the list of even numbers is empty.

Input: [21, 15, 7, 31]

Expected output: Even numbers: []
Odd numbers: [7, 15, 21, 31]

3, A sequence of all even numbers, which has appeared all in the even numbers output as an ascending order. However, the list of odd numbers is empty.

Input: [22, 8, 14, 10]

Expected output: Even numbers: [8, 10, 14, 22]
Odd numbers: []

4, A sequence containing 20 integers randomly without duplicate elements. The output does not remove any elements and split to 2 lists of even and odd numbers in ascending order obviously.

Input: [3, 17, 12, 8, 21, 4, 9, 15, 22, 7, 18, 1, 10, 5, 14, 6, 13, 2, 11, 16]

Expected output: Even numbers: [2, 4, 6, 8, 10, 12, 14, 16, 18, 22]
Odd numbers: [1, 3, 5, 7, 9, 11, 13, 15, 17, 21]

5, A sequence with only one element. That means the output will not need to be sorted in ascending order, and there will be one list containing one element and one empty list. In this case, I will take the odd number.

Input: [9]

Expected output: Even numbers: []
Odd numbers: [9]

6, A big number in sequence that check for working overloaded. Expected that the program work

Input: [999, 101, 234, 1029, 10001, 23456, 19281]

Expected output: Even numbers: [234, 23456]
Odd numbers: [101, 999, 1029, 10001, 19281]

Task 2: Among with task 1, if I must choose 1 test case, I will choose the first test case that a sequence with some duplicate elements. You can see the input above and there are some reasons:

1, **Comprehensiveness Coverage**: This test case includes multiple duplicate elements in the input ([9, 5]), which is critical for verifying whether the program can correctly eliminate duplicate values and produce unique elements in the output, as proper handling of duplicates is a key aspect of the program's functionality. Additionally, the input sequence contains both odd and even numbers, allowing the program to demonstrate its ability to correctly split numbers into two separate lists, thereby testing the full range of conditions for handling both odd and even numbers and ensuring the program works for various number types. Finally, the program is expected to sort the even and odd numbers in ascending order, and this test case will validate whether the program correctly sorts the two lists after filtering out the duplicate elements.

2. **Difficulty and Complexity**: Due to its comprehensiveness, this test case has sufficient difficulty and complexity to serve as a concrete test case. It will help identify errors more clearly and accurately through the following factors: identifying and classifying the output into odd and even numbers, removing duplicate elements, and sorting them, as the test case is not in any specific order but completely random. Beside that there is a non-trival size input, which may be take more time to process it and consider exactly the efficiency of both program and test case.

3. **Comprehensiveness in Benefit**: Comprehensiveness here means that test case 1 covers most of the conditions from other test cases (such as distinguishing between odd and even numbers, handling duplicates, etc.). I can fully use it for general purposes and indirectly verify the correctness of the program through other test cases. For example, test cases 2 and 3, as mentioned in part 1, are used to accurately classify odd or even numbers, or test case 4, where there are no duplicate elements. Test case 1 meets all of these conditions. Moreover, if test case 1 fails, I can easily analyze the output lists, select the corresponding test case, and clarify the error. For instance, if I replicate case 3, where I am only looking for even numbers, there is a likelihood that the odd number list will be empty as expected, producing a specific type of output.

4. **Enhancing the Program**: With this concrete test case, the critical components of the program will be tested, and there is a likelihood of detecting various errors, allowing for timely identification of mistakes and corrections.

Task 3: Let's conduct testing for some of the test cases I created in Task 1 and Task 2 using the Python program provided in the Appendix

1, **Test Case 1**: [4, 9, 12, 7, 17, 5, 11, 9, 14, 6, 20, 2, 10, 5, 13]

```
Output: Odd numbers: [5, 5, 7, 9, 9, 11, 13, 17]
        Even numbers: [2, 4, 6, 10, 12, 14, 20]
```

2, **Test Case 2**: [21, 15, 7, 31]
Output: Odd numbers: [7, 15, 21, 31]
Even numbers: []

3, **Test Case 3**: [22, 8, 14, 10]
Output: Odd numbers: []
Even numbers: [8, 10, 14, 22]

4, **Test Case 4**: [3, 17, 12, 8, 21, 4, 9, 15, 22, 7, 18, 1, 10, 5, 14, 6, 13, 2, 11, 16]
Output: Odd numbers: [1, 3, 5, 7, 9, 11, 13, 15, 17, 21]
Even numbers: [2, 4, 6, 8, 10, 12, 14, 16, 18, 22]

5, **Test Case 5**: [9]
Output: Odd numbers: [9]
Even numbers: []

6, **Test Case 6**: [999, 101, 234, 1029, 10001, 23456, 19281]
Output: Odd numbers: [234, 23456]
Even numbers: [101, 999, 1029, 10001, 19281]

We can easily see that, in essence, this program has successfully solved the problem of splitting the two lists into even and odd numbers, as well as sorting them in ascending order. However, in test case 1, we noticed that the duplicate elements were not removed as expected, such as the numbers 5 and 9. Therefore, we can conclude that the program only accomplished 2 out of the 3 tasks.
Therefore, I checked the Python program and realized that the error of not removing the duplicate elements lies in lines 15 and 16. I suggest using the `set()` function within the `sorted()` function to ensure that the program correctly categorizes odd and even numbers, sorts them, and removes the duplicate elements. Additionally, I noticed that in this program, when the sequence contains more than 20 elements or includes the value 0, the error is not alerted, and instead, a `ValueError` occurs. This happens because there is no function to handle and print these errors. To fix this, I replaced the variable on line 21 with `result` and added an `if` statement to call the `result` variable afterward. This ensures that if I provide a list of numbers falling into either of the error cases, an alert will appear instead of a `ValueError`.

Line 15 – 16:

```
odd_nums = sorted(set(odd_nums)) # Old programming: odd_nums = sorted(odd_nums)
even_nums = sorted(set(even_nums)) # Old programming: even_nums =
sorted(even_nums)
```

Line 21 – end:

```
result = split_and_sort(nums) # Old programming: odd_nums, even_nums =
split_and_sort(nums)
if isinstance(result, str):  # Check if the result is an error message (a string)
    print(result)
else:
    odd_nums, even_nums = result # Print() function in last 2 lines are the same
```