Ta Quang Tung - 104222196

SWE30009 - Assignment 2

Task 1

From the program description, the properties of its input and output can be summarized as:

- An input is considered **valid** if it is a **non-empty list of up to 20 positive (non-zero) and possibly duplicated integers (*)**.
- The output consists of two lists of integers such that:
    1. One list only contains the odd integers of the input list while the other only contains the even integers.
    2. Both lists are sorted in ascending order.
    3. Both lists do not contain any duplicate numbers.

The program is considered faulty if one or more of the above output properties are not satisfied. If we treat each unsatisfied property as a failure, the program can have up to 3 different failures. For each failure, we can define a corresponding testing objective:

O1	Detect the incorrect separation of odd and even integers.
O2	Detect the incorrect sorting of integers.
O3	Detect the incorrect removal of duplicate integers.

To construct a concrete valid test case for the program, first select the objectives that it should achieve, then select a **valid (see *)** list of integers that satisfies the constraints set by the chosen objectives. Each testing objective enforces a constraint on the test cases that achieve it. More specifically:

- O1 requires that the test cases **include one or more odd numbers, one or more even numbers, or both**. If a test case includes both types of number, the expected output will be an odd list and an even list, which can confirm the separation of integers. If a test case only has one type of number, the expected output will contain an empty list and a non-empty list, which still confirms the separation of integers. Clearly, this constraint is always satisfied by any **valid (*)** test case because the list is bound to contain either an odd or even integer. Thus, this constraint can be ignored.
- O2 requires that the test cases **include different odd numbers, different even numbers, or both (C2)**. Only when there are different odd or even numbers is the effect of sorting apparent.
- O3 requires that the test cases **include duplicate integers (C3)**. Only when there are duplicate integers is the effect of removing duplicates apparent.

The following table presents 6 test cases constructed following the described process, along with their expected outputs:

Case	Objective(s) achieved	Constraints of input	Input	Expected output
#1	O1	None	[1, 2]	Odd: [1] Even: [2]
#2	O1, O2	C2	[5, 4, 2, 1, 3, 6]	Odd: [1, 3, 5] Even: [2, 4, 6]
#3	O1, O3	C3	[1, 1, 1, 2, 2, 2]	Odd: [1] Even: [2]
#4	O1, O2, O3	C2, C3	[2, 3, 1, 1, 4, 4, 6, 7, 7, 5]	Odd: [1, 3, 5, 7] Even: [2, 4, 6]
#5	O1, O2, O3	C2, C3	[7, 5, 1, 5, 3, 9]	Odd: [1, 3, 5, 7, 9] Even: [ ]
#6	O1, O2, O3	C2, C3	[10, 4, 6, 6, 8, 4, 2]	Odd: [ ] Even: [2, 4, 6, 8, 10]

Task 2

If only one test case can be chosen, it makes sense for it to cover as many testing objectives as possible. In the above table, only test cases #4, #5, and #6 achieve all 3 testing objectives, making them ideal candidates. However, **the best test case is #4** because it contains both odd and even numbers, thus being able to reveal errors in the event that the sort and duplicate removal logic is not the same for the two types of numbers. Suppose that the program's author first writes this logic for one type of number, then incorrectly copies it over to the other type. In this case, the logic is different for the two types of numbers and the program is bound to be faulty. As test cases #5 and #6 only cover either odd or even numbers, they may overlook the faulty logic of the other type. Given that the code of the program cannot be viewed by the tester, it is reasonable to assume this scenario.

Task 3

The following table shows the result of executing the split_and_sort() function in the assignment2.py script with the 6 above test cases:

Case	Input	Expected output	Actual output	Status
#1	[1, 2]	Odd: [1]	Odd: [1]	Passed

		Even: [2]	Even: [2]	
#2	[5, 4, 2, 1, 3, 6]	Odd: [1, 3, 5] Even: [2, 4, 6]	Odd: [1, 3, 5] Even: [2, 4, 6]	Passed
#3	[1, 1, 1, 2, 2, 2]	Odd: [1] Even: [2]	Odd: [1, 1, 1] Even: [2, 2, 2]	Failed
#4	[2, 3, 1, 1, 4, 4, 6, 7, 7, 5]	Odd: [1, 3, 5, 7] Even: [2, 4, 6]	Odd: [1, 1, 3, 5, 7, 7] Even: [2, 4, 4, 6]	Failed
#5	[7, 5, 1, 5, 3, 9]	Odd: [1, 3, 5, 7, 9] Even: [ ]	Odd: [1, 3, 5, 5, 7, 9] Even: [ ]	Failed
#6	[10, 4, 6, 6, 8, 4, 2]	Odd: [ ] Even: [2, 4, 6, 8, 10]	Odd: [ ] Even: [2, 4, 4, 6, 6, 8, 10]	Failed

The test results show that only 2 test cases passed while 4 failed. The actual outputs of the failed test cases indicate that duplicate integers have not been removed by the program. The reason these test cases can reveal the failure is that they are designed to achieve objective O3. In contrast, the passed test cases cannot because they are not designed to achieve O3.

Inspecting the program's code reveals the problematic portion of the split_and_sort function that causes the failure:

```
# remove duplicates and sort
odd_nums = sorted(odd_nums)
even_nums = sorted(even_nums)
```

This code sorts but forgets to remove duplicates from the two lists. One correct solution would be:

```
# remove duplicates and sort
odd_nums = list(set(sorted(odd_nums)))
even_nums = list(set(sorted(even_nums)))
```

This code converts the sorted odd and even lists into sets, effectively removing duplicate values from both. It then converts them back into lists, which is the expected type of the output. Despite not being the most efficient and readable solution to the problem, this code ensures that the output correctly matches the program's description.