COSC 3P95- Software Analysis & Testing

Assignment 1

Due date: Monday, Oct 16th, 2023, at **23:59** (11:59 pm)

Delivery method: This is an individual assignment. Each student should submit one PDF through Brightspace.

Attention: This assignment is worth 10% of the course grade. Please also check the Late Assignment Policy.

Name: Charlotte Kerrigan Student ID: 7266117

Questions:

1- **Explain the difference between "sound" and "complete" analysis in software analysis. Then, define what true positive, true negative, false positive, and false negative mean. How would these terms change if the goal of the analysis changes, particularly when "positive" means finding a bug, and then when "positive" means not finding a bug. (10 pts)**

Soundness guarantees there will be no false positives, while a complete software analysis guarantees there will be no false negatives. So, soundness will guarantee that every problem found in the analysis is a problem in the software, while completeness guarantees that every problem in the software is found in the analysis.

True positive: A true positive is when a tool does report an error/vulnerability that exists in the software.

True negative: A true negative is when a tool doesn't report an error/vulnerability that doesn't exist.

False positive: A false positive is when a tool reports an error/vulnerability when there is none in the software.

False negative: A false negative is when a tool doesn't report when a error/vulnerability which exists in the software.

a) Finds a bug.
   - True positive: The tool found a bug that exists.
   - False Positive: The tool incorrectly finds a bug where there isn't a bug.

b) Not finding a bug
   - False negative: The tool didn't find a bug where there is a bug.
   - True negative: The tool didn't find a bug where there was no bug to be found.

2- Using your preferred programming language, implement a random test case generator for a sorting algorithm program that sorts integers in ascending order. The test case generator should be designed to produce arrays of integers with random lengths, and values for each sorting method.

A) Your submission should consist of:

a. Source code files for the sorting algorithm and the random test case generator.
b. Explanation of how your method/approach works and a discussion of the results (for example, if and how the method was able to generate or find any bugs, etc.). You can also include bugs in your code and show your method is able to find the input values causing that.
c. Comments within the code for better understanding of the code.
d. Instructions for compiling and running your code.
e. Logs generated by the print statements, capturing both input array, output arrays for each run of the program.
f. Logs for the random test executions, showing if the test was a pass and the output of the execution (e.g., exception, bug message, etc.).

B) Provide a context-free grammar to generate all the possible test-cases. **(18 + 8 = 26 pts)**

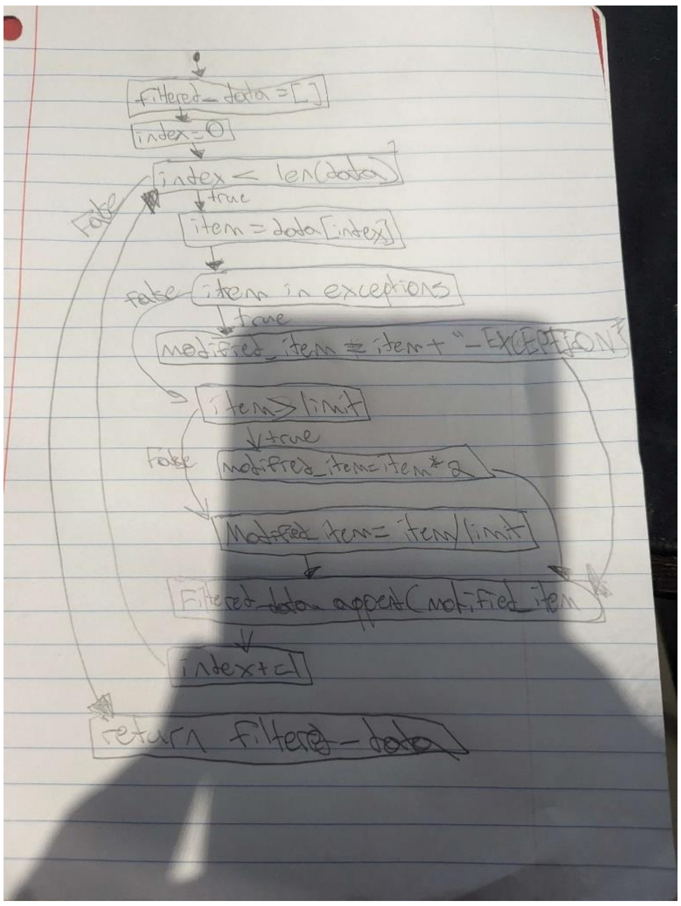
In the QuickSort.**py on line 34 you can find "return i + 1" which tells the program where it left off and where it should start from again. Leaving the +1 uncommented allows the program to be bug free to get the first 2 examples in my log. While to get the bug you must comment the '+1' to create the bug. (ie change the line from 'return i +1' to 'return I'**

3- A) For the following code, manually draw a control flow graph to represent its logic and structure.

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```



The code is supposed to perform the followings:

a. If an item is in the exceptions list, the function appends "_EXCEPTION" to the item.
b. If an item is greater than a given limit, the function doubles the item.
c. Otherwise, the function divides the item by 2.

B) Explain and provide detailed steps for “random testing” the above code. No need to run any code, just present the coding strategy or describe your testing method in detail. **(8 + 8 = 16 pts)**

I would start by looking at the input variables for the code and by generating random inputs ie random data array, a random limit and a random array of exceptions. The run the program with the generated inputs and see if the returned filter_data had errors, or if the function threw exceptions and to log any errors found.

4- A) Develop 4 distinct test cases to test the above code, with code coverage ranging from 30% to 100%. For each test-case calculate and mention its code coverage.

1.
data=[0], limit = 1, exceptions =[]
1 element in the data which is lower than the limit.

2.
data=[1], limit = 0, exceptions =[]
1 element in the data which is greater than the limit.

3.
data=[1,2], limit = 1, exceptions =[1]
1 element in the data which is greater than the limit, and another element in the exception.

4.
data=[1,2,3], limit = 2, exceptions =[3]
1 element in the data which is greater than the limit, and another element in the exception and another element that doesn't fit either of the requirements.

B) Generate 6 modified (mutated) versions of the above code.

1. *modified_item = item / 2*

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item / 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1
return filtered_data
```

2. *modified_item = item * limit*

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item * limit

        filtered_data.append(modified_item)
index += 1
return filtered_data

        index += 1
   return filtered_data
```
3. while index > len(data):
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index > len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```
4. index = 1
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 1
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```
5. filtered_data.clear()
```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION
        elif item > limit:
            modified_item = item * 2
        else:
            modified_item = item / limit

        filtered_data.clear()
        index += 1

    return filtered_data
```

6. modified_item = item * 2 + limit

```
def filterData(data, limit, exceptions):
    filtered_data = []
    index = 0
    while index < len(data):
        item = data[index]
        if item in exceptions:
            modified_item = item + "_EXCEPTION”
        elif item > limit:
            modified_item = item * 2 + limit
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

C) Assess the effectiveness of the test cases from part A by using mutation analysis in conjunction with the mutated codes from part B. Rank the test-cases and explain your answer.

4.
data=[0], limit = 1, exceptions =[]
1 element in the data which is lower than the limit.
This is the worst as it detects the least number of mutations. Detects 2,3,4,5 mutations.

3.
data=[1], limit = 0, exceptions =[]
1 element in the data which is greater than the limit.
This is also the worst as it detects the least number of mutations. Detects 1,3,4,5 mutations.

2.
data=[1,2], limit = 1, exceptions =[1]
1 element in the data which is greater than the limit, and another element in the exception.
Detects 1,3,4,5,6 mutations.
This is one is better than the previous ones as it detects more of the mutations.

1.
data=[1,2,3], limit = 2, exceptions =[3]

1 element in the data which is greater than the limit, and another element in the exception and another element that doesn't fit either of the requirements.
Detects 1,2,3,4,5,6 mutations.
This one is the best as it has 100% coverage so it detects all of the mutations.

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 * 8 = 32 pts)**
I would make a test case that went into all possible branches to test any statements in the code. For example I would make a test case with an element in data that was in exceptions another which was over or at the limit and then another which doesn't fall into either of the others if statements to test the if else statements:

```
if item in exceptions:
    modified_item = item + "_EXCEPTION
elif item > limit:
    modified_item = item * 2
else:
    modified_item = item / limit
```

5- The code snippet below aims to switch uppercase characters to their lowercase counterparts and vice versa. Numeric characters are supposed to remain unchanged. The function contains at least one known bug that results in incorrect output for specific inputs.

```
def processString(input_str):
    output_str = ""
    for char in input_str:
        if char.isupper():
            output_str += char.lower()
        elif char.isnumeric():
            output_str += char * 2
        else:
            output_str += char.upper()

    return output_str
```

In this assignment, your tasks are:

a. Identify the bug(s) in the code. You can either manually review the code (a form of static analysis) or run it with diverse input values (a form of manual random testing). If you are unable to pinpoint the bug using these methods, you may utilize a random testing tool or implement random test case generator in code. Provide a detailed explanation of the bug, identify the line of code causing it, and describe your strategy for finding it.

   The code snippet doesn't leave numeric numbers unchanged it doubles the count of them ie '1' -> '11'

b. Implement Delta Debugging, in your preferred programming language to minimize the input string that reveals the bug. Test your Delta Debugging code for the following input values provided.

i. “abcdefG1”
ii. “CCDDEExy”
iii. “1234567b”
iv. “8665”

Briefly explain your delta-debugging algorithm and its implementation and provide the source code in/with your assignment. **(4 + 12 = 16 pts)**

My delta-debugging algorithm iterates through all of the characters in the given test case and checks if they are needed for the bug to occur by comparing the output from the bugged function to a fixed function.

6- Extra Credit Assignment: Create a GitHub repository to host all the elements of this assignment. This includes source codes, test data, and any screenshots or logs you have generated. Submit the GitHub link along with your main submission through Brightspace. **(5 pts)**

https://github.com/maikeeb/Assignment-1-COSC-3P95

Marking Scheme:

Marks will be awarded for completeness and demonstration of understanding of the material. It is important that you fully show your knowledge when providing solutions in a concise manner. Quality and conciseness of solutions are considered when awarding marks. Lack of clarity may lead you to lose marks, so keep it simple and clear.

Submission:

The submission is expected to contain a sole word-processed document. The document can be in either ***DOC or PDF*** *format; it should be a single column, at least single-spaced, and at least in font 11. It is strongly recommended to use the assignment questions to facilitate marking: answer the questions just below them for easier future reference.*

Late Assignment Policy:

A one-time penalty of 25% will be applied on late assignments. Late assignments are accepted until the Late Assignment Date, four days after the Assignment Due Date. No excuses are accepted for missing deadlines. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion. However, deadline extensions may be granted under extenuating circumstances, such as medical or physical conditions; please note that granting the extension is under the instructor’s discretion.

Plagiarism:

Students are expected to respect academic integrity and deliver evaluation materials that are only produced by themselves. Any copy of content, text or code, from other students, books, web, or any other source is not tolerated. If there is any indication that an activity contains any part copied from any source, a case will be open and brought to a plagiarism committee’s attention. In case plagiarism is determined, the activity will be canceled, and the author(s) will be subject to university regulations. For further information on this sensitive subject, please refer to the document below: ***https://brocku.ca/node/10909***