**COSC 3P95- Software Analysis & Testing**

**Assignment 1**

**Questions:**

**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: Jerome Uwaneme Student ID: 7141270**

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)**

While a sound program analysis will not report any false positives, a complete analysis technique is one that detects all possible bugs and issues within a program i.e it prevents false negatives.

False positives in program analysis occur when the tool or technique detects issues and the issues do not exist.

False Negatives in program analysis occurs when the tool does not detect issues that exist within the program

True positive in program analysis occurs when the tool or technique detects issues that indeed exist within the program.

True Negatives in program analysis occur when the tools or techniques do not detect any issues and no issues actually exist.

While I feel the definition above would still be valid if positive meant finding a bug, they will change when positive means not finding a bug.

False positives now will occur when the tool or technique did not find any bugs or issues but they exist within the program.

False negative will be when the tool finds bugs in the program but said bugs do not exist within the program.

True positive: now occurs when the tool says there are no bugs in the program and truly are not any bugs

True negative occurs when the tool detects bugs in the program and there are indeed bugs

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)**

function testCaseGen(){

//set the length of the array to a random number between 1 and 150

let length = Math.floor(Math.random()\*150)+1;

let test\_arr = [];//declare empty array to house random value

for(let i=0;i<length;i++){

//adds a random a value to the array of random length at every iteration of the for loop

test\_arr.push(Math.floor(Math.random()\*151));

}

return test\_arr;

}

function Selection\_Sort (arr){

let length = arr.length;

for(let i=0;i<length-1;i++){ //iterate to n-1 because at that point the array is sorted

let smallest\_index = i;//Assume the first value is the smallest

for(let j=i+1; j<length;j++){// iterate from second element to the end to find the smallest

if(arr[j]<arr[smallest\_index]){//if value at j is smaller than first

smallest\_index = j// set smallest index to j

}

}

//Using a temp value swap number at index j and number at first position or index i

if(smallest\_index!== i){

let temp = arr[i];

arr[i] = arr[smallest\_index];

arr[smallest\_index] = temp

}

}

return arr;

}

function check(){

let new\_arr = testCaseGen();

let sorted\_arr = Selection\_Sort(new\_arr);

let sys\_sorted\_arr = new\_arr.sort((a,b)=>a-b);//array sorted with javascript built in sort function instead of Selection sort

console.log('Selection Sort sorted array : ',sorted\_arr);

console.log('Same array sorted by built js sort function: ',sys\_sorted\_arr);

if(sys\_sorted\_arr == sorted\_arr){

console.log("TRUE PASSED");

}

else{

console.log('FAlSE FAILED');

}

}

check();

//2B Explanation

// I wrote a random test case generator to produce arrays of random various sizes. Following the generator is a selection sort program that sorts the produced array from smallest to largest value and returns the sorted array. I went with the selection sort because I feel it is the easiest to implement. I further implemented a method that sorts a randomly generated array using javascripts built in sorting function and my selection sort, comparing them to see if the Selection sort output is correct

//2C code is commented

//2D Download node.js from nodejs.dev

//open terminal and type in node test.js

//2E Logs generated form print statements capturing both input array, output array for each run of the program.

// Original Array: [

// 103, 46, 133, 24, 97, 8, 93, 16, 72, 40,

// 118, 64, 52, 46, 74, 34, 30, 118, 105, 103,

// 110, 4, 77, 150, 135, 117, 10, 38, 19, 111,

// 85, 53, 141, 34, 15, 9, 45, 48, 92, 105,

// 130, 11, 26, 108, 119

// ]

// Sorted Array: [

// 4, 8, 9, 10, 11, 15, 16, 19, 24, 26,

// 30, 34, 34, 38, 40, 45, 46, 46, 48, 52,

// 53, 64, 72, 74, 77, 85, 92, 93, 97, 103,

// 103, 105, 105, 108, 110, 111, 117, 118, 118, 119,

// 130, 133, 135, 141, 150

// ]

// 2F

// Selection Sort sorted array : [

// 6, 11, 14, 17, 19, 19, 20, 20, 21,

// 22, 24, 41, 48, 48, 48, 54, 61, 69,

// 70, 76, 77, 78, 81, 83, 94, 97, 108,

// 123, 126, 139, 146

// ]

// Same array sorted by built js sort function: [

// 6, 11, 14, 17, 19, 19, 20, 20, 21,

// 22, 24, 41, 48, 48, 48, 54, 61, 69,

// 70, 76, 77, 78, 81, 83, 94, 97, 108,

// 123, 126, 139, 146

// ]

// TRUE PASSED

//2 PART B

// CFG for all possible test cases

/\* <array> ::= [] | [<array-with-elements>]

<array-with-elements> ::= <number> | <number>, <array-with-elements>

<number> ::= 0 | 1 | ... | 150 \*/

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*

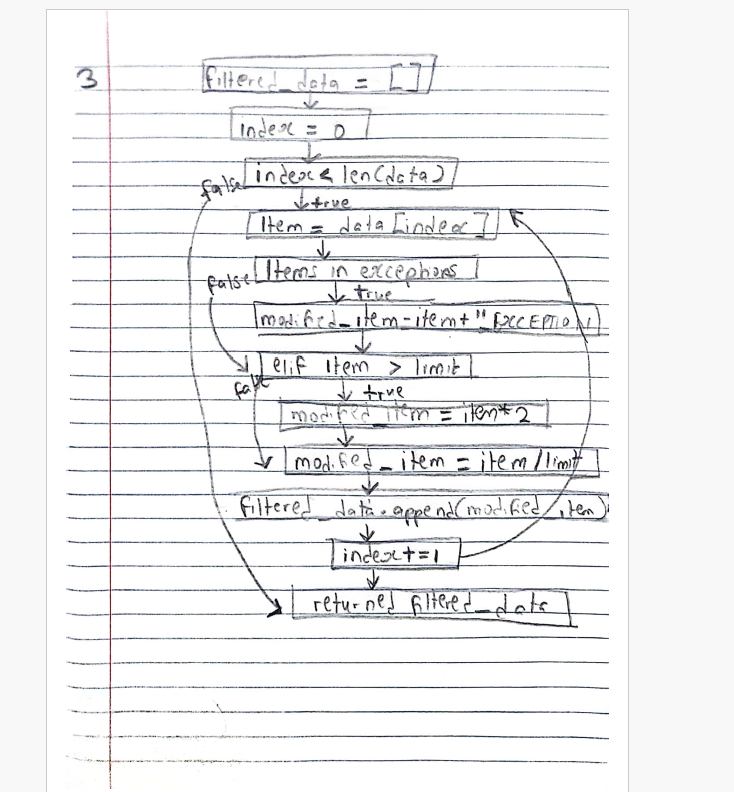
*return filtered\_data*

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)**

Random testing on the above code would be done by setting the variables data, limit,and the array exceptions to random values. A method that calls the filtered\_data method multiple times for the random inputs can also be created to ensure it works for all instances of the variable. The bugs and issues should then be recorded.

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.

Test case 1

data = [5 , 7]

limit = 2

Exception = [7]

Statement Coverage = 7/9\*100= 77%

Branch Coverage = ⅔\*100 = 67%

Test case 2

Data = [ 45,70]

Limit = 3

Exception = [ 1, 45, 67]

Statement Coverage = 9/9\*100 =100%

Branch Coverage = 100%

Test case3

Data = [1]

Limit = 1

Exception = [1]

Statement Coverage = 7/9\*100 = 77%

Branch Coverage = ⅓\*100 = 33.33%

Test 4

Data = [3,4,5]

Limit = 4

Exception = [4,5]

Statement Coverage = 100%

Branch Coverage = 100%

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

Mutation 1

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index > len(data): //mutation*

*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*

Mutation 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 //Mutation*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Mutation 3

*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 data //Mutation*

Mutation 4

*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: //Mutation*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Mutation 5

*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 //Mutation*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Mutation 6

*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 Mutation*

*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.

Mutation 1

*def filterData(data, limit, exceptions):*

*filtered\_data = []*

*index = 0*

*while index > len(data): //mutation*

*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*

Mutation Analysis:

All test cases will detect this mutation as the method returns an empty array i.e the loop never runs.

Mutation 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 //Mutation*

*filtered\_data.append(modified\_item)*

*index += 1*

Mutation Analysis:

Only test case 4 detects the mutation 2

Mutation 3

*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 data //Mutation*

Mutation Analysis:

All test cases will detect this mutation as the final return statement return the original array

Mutation 4

*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: //Mutation*

*modified\_item = item \* 2*

*else:*

*modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

*return filtered\_data*

Mutation Analysis:

Test cases 1 ,2 and 4 detect this mutation but case 3 does not.

Mutation 5

*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 //Mutation*

*else: modified\_item = item / limit*

*filtered\_data.append(modified\_item)*

*index += 1*

Mutation analysis

Test cases 1 and 2 detect mutation 5 but others do not.

Mutation 6

*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 Mutation*

*return filtered\_data*

Mutation Analysis:

All test cases detect this mutation as it creates an infinite loop after the first iteration.

Ranking Test cases according to mutation score

Test case 1 ⅚

Test case 2 ⅚

Test case 4 5/6

Test case 3 3/6

I have ranked the test cases according to their mutation scores which is the ratio of the number of killed mutants(detected changes) to the total number of mutants . Test cases 1 and 2 have the highest score and as such are the most effective test cases.

D) Discuss how you would use path, branch, and statement static analysis to evaluate/analyse the above code. **(4 \* 8 = 32 pts)**

Static analysis would involve reviewing the source code to find all kinds of syntactic and semantic errors. In this case it would simply mean going through the function checking variables and assignments for any bugs.

I would apply proper path analysis by constructing a control flow graph of the function to ensure the program begins and stops where it should.

I would do proper Branch testing by going through all branch and conditional statements ensuring I know the outcome of each condition and testing all possibilities at least once.

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 utilise a random testing tool or implement a 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 bug in this program is at line (output\_str+=char\*2). At this line after checking if the current character is a number, it multiplies said number by 2 and then concatenates it with the output\_str. I found this bug by going through the source code and understanding what each statement meant.

b. Implement Delta Debugging, in your preferred programming language to minimise 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)**

**function processString(input\_str){**

**let output\_str = '';**

**//console.log(input\_str.length);**

**for (let i = 0; i < input\_str.length; i++) {**

**let letter = input\_str[i];**

**if (!isNaN(letter)) { // If the letter is a number**

**output\_str += letter + letter; // Add it twice to replicate bug in python code**

**} else if (letter === letter.toUpperCase()) {**

**output\_str += letter.toLowerCase();**

**} else {**

**output\_str += letter.toUpperCase();**

**}**

**}**

**return output\_str;**

**}**

**function test(input\_str){// checks if the number in the string is printed twice**

**let new\_String = processString(input\_str);**

**for(let i=0; i<input\_str.length;i++){**

**let char = input\_str[i];**

**if(!isNaN(char)){**

**if(new\_String[i+1] === char){**

**return true;**

**}**

**}**

**}**

**return false;**

**}**

**function Delta\_debug(input\_str){**

**let length =input\_str.length;**

**while(length>=1){**

**let split\_length= input\_str.length/2;**

**let one = input\_str.substring(0,split\_length);**

**let two = input\_str.substring(split\_length);**

**if(test(one)==true){**

**input\_str = one;**

**}**

**else if(test(two)){**

**input\_str = two;**

**}**

**length= length/2;**

**}**

**return input\_str**

**}**

**let input= ['abcdefG1','CCDDEExy', '1234567b', '8665']**

**for(let i of input){**

**console.log(Delta\_debug(i));**

**}**

**//console.log(Delta\_debug('8665'));**

**// //Log of 5b test Cases**

**// 1**

**// CCDDEExy**

**// 1**

**// 8**

This Delta debugging algorithm attempts to continuously divide the input string into 2 halves, testing both to find which half has the bug using the test method which returns true only when the output of the given input string contains the bug. It does this until the input string length is less than or equal to one returning said character as the source of the bug.