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: __Quinn Guernsey____ **Student ID:** ___7214075___

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 refers to an analysis's ability to avoid reporting false positives. This means that if the analysis says something is true or that there is a particular bug in the code, you know for sure that it is there.

Completeness refers to an ability of an analysis program to prevent reporting false negatives. This means that if you know for sure a bug is in the code, the complete analysis will tell you about it and not say that everything is fine when you know that it is not.

True Positive: This is a true identification of a bug. When analysis tells you that it found a bug, there is a bug in the program.

True Negative: This is the non-identification of a bug. When analysis tells you it didn't find a bug, there is no bug in the program.

False Positive: This is the false identification of a bug. This is when analysis tells you that it found a bug, but there isn't a bug in the program.

False Negative: This is falsely identifying that there is no bug in the program. This is when analysis tells you that the program doesn't have a bug in it, but there is a bug in the program.

If the goal of the analysis changes so that positive means not finding a bug compared to when positive means finding a bug, true positive and true negative definitions are reversed as well as and false positive and false negative definitions are reversed.

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.).

To compile and run the code:

Save the sorting algorithm code in a file called Sort.java.

Save the random test case generator code in a file called RandomTestCaseGenerator.java.

Compile both Java files using javac:

```
javac <file path>\Sort.java
javac <file path>\RandomTestCaseGenerator.java
```

Run the test case generator:

```
java RandomTestCaseGenerator
```

SAMPLE LOG:

```
Test Case 1:
Input Array: [290, 667, 184, 402, 812, 129, 292, 373, 723, 464, 41, 369,
750, 674, 567, 475, 501, 3, 889, 248, 157, 114, 870, 814, 284, 319, 253]
Sorted Array: [3, 41, 114, 129, 157, 184, 248, 253, 284, 290, 292, 319,
369, 373, 402, 464, 475, 501, 567, 667, 674, 723, 750, 812, 814, 870, 889]

Test Case 2:
Input Array: [587, 581, 428, 463, 14, 272, 894]
Sorted Array: [14, 272, 428, 463, 581, 587, 894]

Test Case 3:
Input Array: [740, 967, 839, 216, 840, 791, 800, 410, 865, 263, 269, 769,
250, 885, 606, 321, 134, 655, 478, 142, 738, 567, 413, 867, 27, 923, 232,
982, 408, 911, 78, 488, 160, 697, 920, 678, 961, 132, 418, 519, 690, 47,
149, 639, 475, 210, 466, 701, 914, 241, 190, 563, 327, 216, 945, 850, 350,
777, 722, 975, 199, 962, 403, 894, 733, 576, 852, 304, 686, 231, 648, 843,
559, 10, 289, 887, 24, 491, 293, 82, 960, 179, 185, 58, 868, 632, 885, 432,
 459, 184, 60, 699, 602, 522, 253, 428, 382]
```

Sorted Array: [10, 24, 27, 47, 58, 60, 78, 82, 132, 134, 142, 149, 160, 179, 184, 185, 190, 199, 210, 216, 216, 231, 232, 241, 250, 253, 263, 269, 289, 293, 304, 321, 327, 350, 382, 403, 408, 410, 413, 418, 428, 432, 459, 466, 475, 478, 488, 491, 519, 522, 559, 563, 567, 576, 602, 606, 632, 639, 648, 655, 678, 686, 690, 697, 699, 701, 722, 733, 738, 740, 769, 777, 791, 800, 839, 840, 843, 850, 852, 865, 867, 868, 885, 885, 887, 894, 911, 914, 920, 923, 945, 960, 961, 962, 967, 975, 982]

Test Case 4:
Input Array: [584, 963, 173, 21, 529, 703, 315, 875, 811, 845, 554, 72, 217, 158, 490, 383, 431, 264, 381, 525, 924, 751, 948, 412, 688, 226, 801, 626, 594, 13, 255, 820, 448, 572, 966, 611, 170, 38, 143, 361, 512, 115, 888
, 773, 154, 434, 759, 695, 872, 451, 628, 7, 346, 72, 843, 692, 111, 496, 659, 462, 555, 245, 756, 443, 56, 609, 93, 478, 154, 148, 962, 984, 951, 381, 569, 533, 443, 382, 913, 461, 658, 209, 379, 435, 546, 641, 29]
Sorted Array: [7, 13, 21, 29, 38, 56, 72, 72, 93, 111, 115, 143, 148, 154, 154, 158, 170, 173, 209, 217, 226, 245, 255, 264, 315, 346, 361, 379, 381, 381, 382, 383, 412, 431, 434, 435, 443, 443, 448, 451, 461, 462, 478, 490, 496, 512, 525, 529, 533, 546, 554, 555, 569, 572, 584, 594, 609, 611, 626, 628, 641, 658, 659, 688, 692, 695, 703, 751, 756, 759, 773, 801, 811, 820, 843, 845, 872, 875, 888, 913, 924, 948, 951, 962, 963, 966, 984]

Test Case 5:
Input Array: [246, 428, 764, 580, 708, 342, 15, 849, 954, 223, 430, 986, 150, 979, 410, 26, 826, 847, 994, 831, 156, 214, 928, 703, 232, 156, 357, 627, 844, 908, 148, 674, 35, 320, 295, 126, 233, 631, 146, 476, 999, 904, 91, 11, 113, 300, 891, 285, 987, 111, 937, 307, 872, 376, 180, 133, 821, 781, 496, 525, 382, 327, 875, 946, 436, 826, 864, 825, 735, 285, 37, 309, 644, 936, 790, 675, 407, 850, 669, 807, 400, 242, 88, 739, 358, 177, 534]
Sorted Array: [11, 15, 26, 35, 37, 88, 91, 111, 113, 126, 133, 146, 148, 150, 156, 156, 177, 180, 214, 223, 232, 233, 242, 246, 285, 285, 295, 300, 307, 309, 320, 327, 342, 357, 358, 376, 382, 400, 407, 410, 428, 430, 436, 476, 496, 525, 534, 580, 627, 631, 644, 669, 674, 675, 703, 708, 735, 739, 764, 781, 790, 807, 821, 825, 826, 826, 831, 844, 847, 849, 850, 864, 872, 875, 891, 904, 908, 928, 936, 937, 946, 954, 979, 986, 987, 994, 999]

Test Case 6:
Input Array: [405, 492, 272, 452]
Sorted Array: [272, 405, 452, 492]

Test Case 7:
Input Array: [833, 175, 288, 352, 694, 939, 660, 566, 460, 691, 404, 673, 838, 343, 388, 649, 249, 786, 348, 952, 480, 161, 502, 237, 383, 173, 566, 923, 264, 750, 758, 796, 459, 149, 803, 13, 511, 556, 70, 588, 303, 618, 281, 941, 880, 40, 1, 455, 788, 131, 356, 468, 984, 721, 35, 811, 627, 469, 435, 589, 945, 912, 395, 138, 867, 592, 792, 694, 756, 663, 86, 790, 902, 794, 369, 826, 964, 572, 79, 710, 262, 889, 130, 16, 528, 412, 749, 339, 217, 176, 118, 379]
Sorted Array: [1, 13, 16, 35, 40, 70, 79, 86, 118, 130, 131, 138, 149, 161, 173, 175, 176, 217, 237, 249, 262, 264, 281, 288, 303, 339, 343, 348, 352, 356, 369, 379, 383, 388, 395, 404, 412, 435, 455, 459, 460, 468, 469, 480, 502, 511, 528, 556, 566, 566, 572, 588, 589, 592, 618, 627, 649, 660, 663,

673, 691, 694, 694, 710, 721, 749, 750, 756, 758, 786, 788, 790, 792, 794,
796, 803, 811, 826, 833, 838, 867, 880, 889, 902, 912, 923, 939, 941,
945, 952, 964, 984]

Test Case 8:
Input Array: [980, 700, 19, 92, 610, 626, 904, 433, 599, 799, 332, 251,
219, 256, 904, 33, 292, 237, 494, 86, 28, 942, 404, 311, 605, 931, 386,
490, 578, 944, 498, 317, 1, 281, 351, 531, 171, 799, 188, 75, 942, 691,
456, 175, 576, 463, 244, 735]
Sorted Array: [1, 19, 28, 33, 75, 86, 92, 171, 175, 188, 219, 237, 244,
251, 256, 281, 292, 311, 317, 332, 351, 386, 404, 433, 456, 463, 490, 494,
498, 531, 576, 578, 599, 605, 610, 626, 691, 700, 735, 799, 799, 904, 904,
931, 942, 942, 944, 980]

Test Case 9:
Input Array: [894, 580, 399, 65, 319, 475, 57, 669, 210, 102, 293, 367,
574, 995, 291, 76, 800, 816, 344, 778, 313, 107, 804, 370, 441, 543, 911,
932, 97, 903, 6, 78, 978, 123, 710, 535, 577, 121, 603, 635, 214, 562, 24,
973, 277, 503, 184, 326, 716]
Sorted Array: [6, 24, 57, 65, 76, 78, 97, 102, 107, 121, 123, 184, 210,
214, 277, 291, 293, 313, 319, 326, 344, 367, 370, 399, 441, 475, 503, 535,
543, 562, 574, 577, 580, 603, 635, 669, 710, 716, 778, 800, 804, 816, 894,
903, 911, 932, 973, 978, 995]

Test Case 10:
Input Array: [549, 454, 770, 605, 862, 654, 964, 73, 923, 406, 675, 830,
31, 659, 503, 670, 141, 964, 137, 972, 599, 214, 430, 965, 410, 11, 141,
800, 483, 315, 468, 253, 823, 706, 421, 113, 141, 132, 514, 747, 722, 799,
113, 132, 763, 785, 719, 129, 973, 189, 111, 399, 216, 202]
Sorted Array: [11, 31, 73, 111, 113, 113, 129, 132, 132, 137, 141, 141,
141, 189, 202, 214, 216, 253, 315, 399, 406, 410, 421, 430, 454, 468, 483,
503, 514, 549, 599, 605, 654, 659, 670, 675, 706, 719, 722, 747, 763, 770,
785, 799, 800, 823, 830, 862, 923, 964, 964, 965, 972, 973]

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

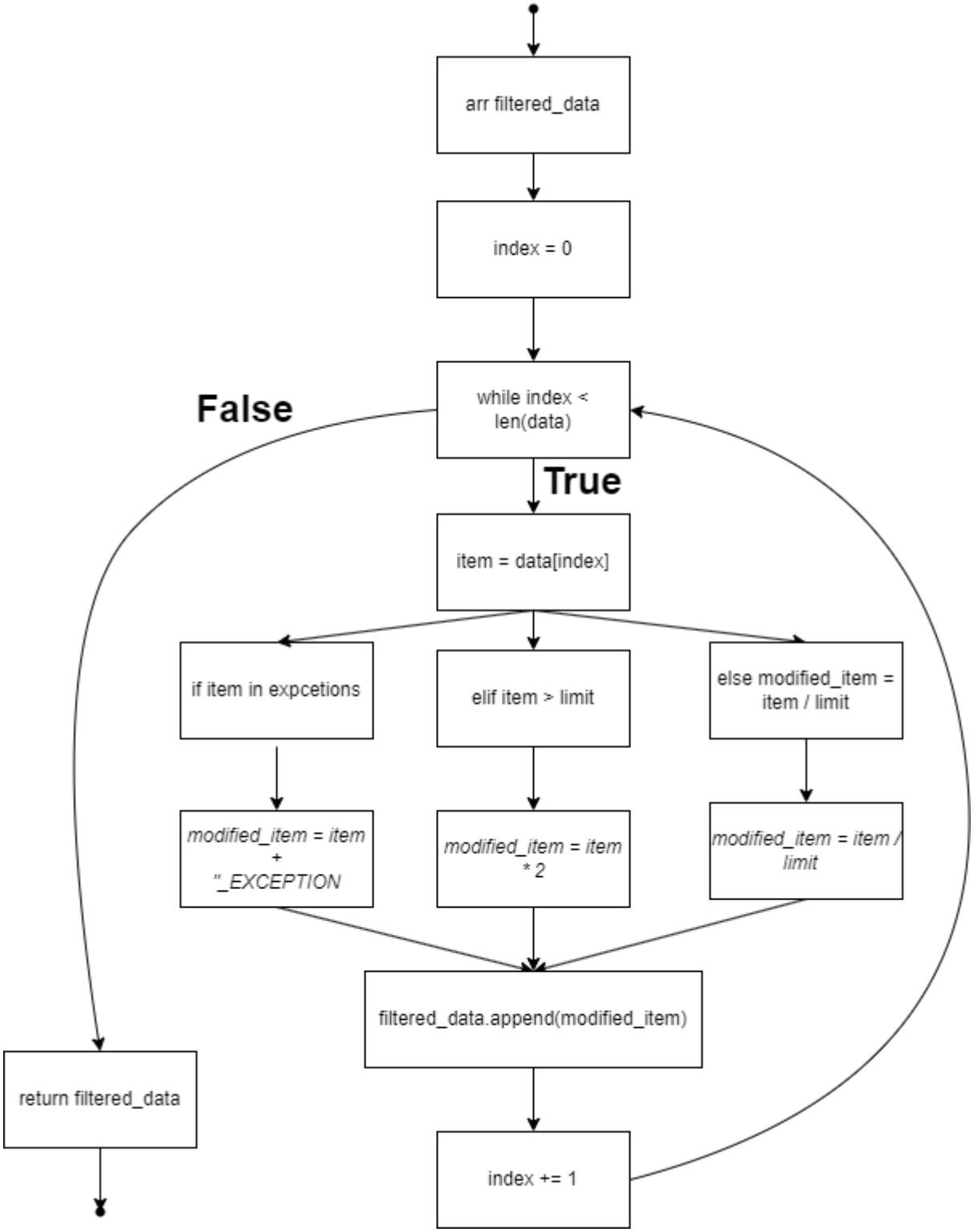
3- A) For the following code, manually draw a control flow graph to represent its logic and structure.

```
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
```

14 lines



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)**

First I would define the kind of inputs for each of the parameters data, limit, and exceptions. To keep things simple we should define the data to be an array of integers, limit to be some large integer we don't want to exceed, and exceptions to be items in data that we don't want to see so that we can mark them with _EXECEPTION

Next, I would generate integers in a wide range to test the code is working correctly. Negative integers, large integers, zero, and positive integer lists for both *data* and *exceptions* and random values for *limit*. Collect the results from *filtered_data*.

After that, for each test case, verify that the output follows to the expected behavior based on the functionality described in the code comments a, b, and c and that they've been modified correctly and analyze the results to identify patterns or unexpected behaviors.

I may also perform boundary testing using extreme values like positive or negative INFINITY, large empty data or exceptions lists, and large limits.

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.

The code has 14 lines total. Code coverage tells us how many we visit.

Test case 1: Empty array

```
data = []
limit = 5
exceptions = []
result = filterData(data, limit, exceptions)
# Expected Result: []
```

5/ 14 lines = 35.7% coverage when input is empty

Test case 2: basic case with simple ints below the limit

```
data = [3, 2, 1]
limit = 5
exceptions = []
result = filterData(data, limit, exceptions)
# Expected Result: [0.6, 0.4, 0.2]
```

10/14 lines = 71% coverage when input is simple with no exceptions and below the limit

Test case 3: Exceptions

```
data = [3, "butt", 10]
limit = 5
exceptions = ["butt"]
result = filterData(data, limit, exceptions)
# Expected Result: [0.6, butt_EXCEPTION', 20]
```

14/14 lines = 100% when input contains below and above limit and an exception

Test case 4: Edge Cases

```
data = [5, 5]
limit = 5
exceptions = []
```

```
result = filterData(data, limit, exceptions)
# Expected Result: [1, 1]
```

Where item = limit code coverage is 10/14 lines = 71%

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

1.

```
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 null
```

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
```

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 filtered_data
```

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:
            modified_item = item / 2
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

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
        else:
            modified_item = item / limit

        filtered_data.append(modified_item)
        index += 1

    return filtered_data
```

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

        return 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.

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

Path Analysis: Identify different execution paths in the code, considering various combinations of conditions and loops. Ensure that your test cases cover different paths. Add additional test cases to get full path coverage.

Branch Analysis: Identify conditional branches in the code and check if your test cases cover all if-else branches.

Statement Analysis: Count the total number of statements in the code and track how many are executed by the test cases. The code coverage tools can provide this information and analyze which statements are untested.

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 bug is contained on the 2 lines "*elif char.isnumeric():"* and "*output_str += char * 2*". I found the bug by compiling the code in Python and adding an random input like "5f4dsaKFHNJSudgFSsaad65s" and checking the input.

The output ends up being "55F44DSAkfhnjsUDGfsSAAD6655S" which is not defined in the problem. Upper and lower case letters are swapped correctly, but numeric value which are supposed to remain unchanged are duplicated as seen in the output. It was found by comparing the length of the output compared to the input. You can see it is longer on any inputs that contain numeric values.

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)**

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)**

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