1.
Sound: refer to ability of an analysis technique or tool to not report any FP
Complete: refer to the ability of an analysis technique or tool to detect al actual bugs and vulnerabilities that exist in the FN.

True positive: a test result that correctly indicates the presence of a condition or characteristic.
True negative: a test result that correctly indicates the absence of a condition or characteristic.
False positive: a test result which wrongly indicates that a particular condition or attribute is present.
False negative: a test result which wrongly indicates that a particular condition or attribute is absent.

When the meaning of positive changes:
Positive – find a bug: (same with TP and FP)
True positive means correctly find a bug.
False positive means find a bug that should not be there.
Positive – not find a bug: (same with TN and FN)
True positive means the error was not found correctly.
False positive means falsely believing that no error has been found, when in fact there is one.

2:
Python code:

```
import random

def random_test_case_generator(max_length=100, max_value=1000): # generate a
random test case
    length = random.randint(0, max_length) # generate a random length
    return [random.randint(0, max_value) for _ in range(length)] # generate a
random array

def quick_sort(arr): # quick sort algorithm
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        less_than_pivot = [x for x in arr[1:] if x <= pivot]
        greater_than_pivot = [x for x in arr[1:] if x > pivot]
        return quick_sort(less_than_pivot) + [pivot] +
quick_sort(greater_than_pivot)

for i in range(10):  # run with 10 different test cases
    test_case = random_test_case_generator() # generate a random test case
    print(f"Test case {i+1}:")
    print(f"Input array: {test_case}") # print the input array
    sorted_array = quick_sort(test_case.copy()) # sort the array
```

```
    print(f"Output array: {sorted_array}")
    print(f"Test passed: {sorted_array == sorted(test_case)}\n") # check if
the array is sorted correctly
```

First create a python (name.py) to copy some of the code into a file. Then, if you already have python installed on your computer, open cmd (for windows) and open the location of the file from the command line. Enter python3 name.py. That's how the Windows version works. Since I'm not using a mac, I'm not sure how it works on a mac, but I will copy a method from the Internet, but it may be outdated or no longer useful.

On MacOS, search for a program called terminal. You can do so by pressing the command key (⌘) + space bar. This will open up the Spotlight search bar, in which you start typing the word 'terminal'. Once you started the terminal, enter python3 to open the Python REPL. (from python land)

output:
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\tongs\Desktop\assign1 3p95> & 'C:\Users\tongs\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\tongs\.vscode\extensions\ms-python.python-2023.18.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '50226' '--' 'c:\Users\tongs\Desktop\assign1 3p95\random.py'
Test case 1:
Input array: [250, 727, 292, 461, 72, 309, 993, 450, 613, 962, 543, 84, 128, 666, 767, 30, 738, 73, 78, 963,
739, 836, 497, 522, 328, 743, 230, 938, 886, 197, 676, 601, 110, 624, 519, 208, 341, 560, 209, 608, 750, 711, 257, 251, 106, 970, 807, 5, 920, 958, 903, 10, 787, 820, 529, 40, 824, 286, 283, 68, 896, 251, 360, 974, 332, 294, 672, 411, 748, 867, 85, 218, 244, 896, 387, 730, 504, 121, 546, 543, 629, 890, 195, 826, 774, 697]
Output array: [5, 10, 30, 40, 68, 72, 73, 78, 84, 85, 106, 110, 121, 128, 195, 197, 208, 209, 218, 230, 244, 250, 251, 251, 257, 283, 286, 292, 294, 309, 328, 332, 341, 360, 387, 411, 450, 461, 497, 504, 519, 522, 529, 543, 543, 546, 560, 601, 608, 613, 624, 629, 666, 672, 676, 697, 711, 727, 730, 738, 739, 743, 748, 750, 767, 774, 787, 807, 820, 824, 826, 836, 867, 886, 890, 896, 896, 903, 920, 938, 958, 962, 963, 970, 974, 993]
Test passed: True

Test case 2:
Input array: [46, 849, 478, 8, 481, 175, 747, 426, 496, 452, 584, 830, 126, 814, 19, 768, 571, 859, 498, 575, 619, 667, 139, 908, 217, 710, 673, 831, 12, 258, 81, 80, 590, 578, 888, 936, 836, 275, 727, 755, 808, 32, 724, 113, 533, 459]

Output array: [8, 12, 19, 32, 46, 80, 81, 113, 126, 139, 175, 217, 258, 275, 426, 452, 459, 478, 481, 496, 498, 533, 571, 575, 578, 584, 590, 619, 667, 673, 710, 724, 727, 747, 755, 768, 808, 814, 830, 831, 836, 849, 859, 888, 908, 936]
Test passed: True

Test case 3:
Input array: [208, 664, 867, 993, 632, 344, 453, 119, 459, 589, 831, 561, 314, 645, 353, 707, 341, 437, 47, 179, 883, 961, 403, 170, 685, 771, 629, 810, 87, 960]
Output array: [47, 87, 119, 170, 179, 208, 314, 341, 344, 353, 403, 437, 453, 459, 561, 589, 629, 632, 645, 664, 685, 707, 771, 810, 831, 867, 883, 960, 961, 993]
Test passed: True

Test case 4:
Input array: [705, 784, 850, 207, 674, 433, 799]
Output array: [207, 433, 674, 705, 784, 799, 850]
Test passed: True

Test case 5:
Input array: [633, 629, 340, 924, 262, 60, 905, 192, 749, 310, 254, 865, 215, 879, 674, 439, 459, 839, 350, 727, 829, 537, 265, 128, 201, 858, 21, 723, 399, 754, 97, 150, 143, 38, 686]
Output array: [21, 38, 60, 97, 128, 143, 150, 192, 201, 215, 254, 262, 265, 310, 340, 350, 399, 439, 459, 537, 629, 633, 674, 686, 723, 727, 749, 754, 829, 839, 858, 865, 879, 905, 924]
Test passed: True

Test case 6:
Input array: [35, 64, 927, 734, 898, 575, 982, 761, 643, 854, 815, 802, 837, 621, 316, 534, 298, 378, 55, 261, 718, 217, 859, 26, 440, 326, 413, 786, 56, 917, 236, 386, 262, 175, 990, 386, 895, 821, 471, 906, 371, 548, 763, 306, 161, 100, 660, 834, 819, 761, 514, 790, 583, 546, 658, 850, 269, 361, 298, 527, 457, 686]
Output array: [26, 35, 55, 56, 64, 100, 161, 175, 217, 236, 261, 262, 269, 298, 298, 306, 316, 326, 361, 371, 378, 386, 386, 413, 440, 457, 471, 514, 527, 534, 546, 548, 575, 583, 621, 643, 658, 660, 686, 718, 734, 761, 761, 763, 786, 790, 802, 815, 819, 821, 834, 837, 850, 854, 859, 895, 898, 906, 917, 927, 982, 990]
Test passed: True

Test case 7:
Input array: [150, 82, 246, 138, 811, 838, 789, 127, 421, 972, 383, 877, 155, 0, 420, 749, 6, 925, 112, 80, 120, 63, 647, 55, 459, 417, 317, 800, 680, 50, 183]
Output array: [0, 6, 50, 55, 63, 80, 82, 112, 120, 127, 138, 150, 155, 183, 246, 317, 383, 417, 420, 421, 459, 647, 680, 749, 789, 800, 811, 838, 877, 925, 972]
Test passed: True

Test case 8:
Input array: [773, 927, 364, 426, 316, 661, 340, 874, 5, 28, 438, 207]

Output array: [5, 28, 207, 316, 340, 364, 426, 438, 661, 773, 874, 927]
Test passed: True

Test case 9:
Input array: [795, 139, 595, 802, 745, 167, 827, 514, 285, 801, 707, 419, 865, 905, 931, 455, 4, 322, 987, 262, 14, 673, 741, 261, 930, 665, 340, 641, 994, 777, 343, 269, 319, 812, 293, 528, 743, 252, 140, 217, 238, 839, 320, 675, 532, 899, 998, 22, 854, 23, 864, 820, 319, 272, 482, 662, 768, 328, 630, 936, 429, 3, 73, 315, 509, 461, 84, 67, 421, 313, 802, 996, 350, 163, 451, 324, 143, 694, 0, 142, 762, 588, 651, 0]
Output array: [0, 0, 3, 4, 14, 22, 23, 67, 73, 84, 139, 140, 142, 143, 163, 167, 217, 238, 252, 261, 262, 269, 272, 285, 293, 313, 315, 319, 319, 320, 322, 324, 328, 340, 343, 350, 419, 421, 429, 451, 455, 461, 482, 509, 514, 528, 532, 588, 595, 630, 641, 651, 662, 665, 673, 675, 694, 707, 741, 743, 745, 762, 768, 777, 795, 801, 802, 802, 812, 820, 827, 839, 854, 864, 865, 899, 905, 930, 931, 936, 987, 994, 996, 998]
Test passed: True

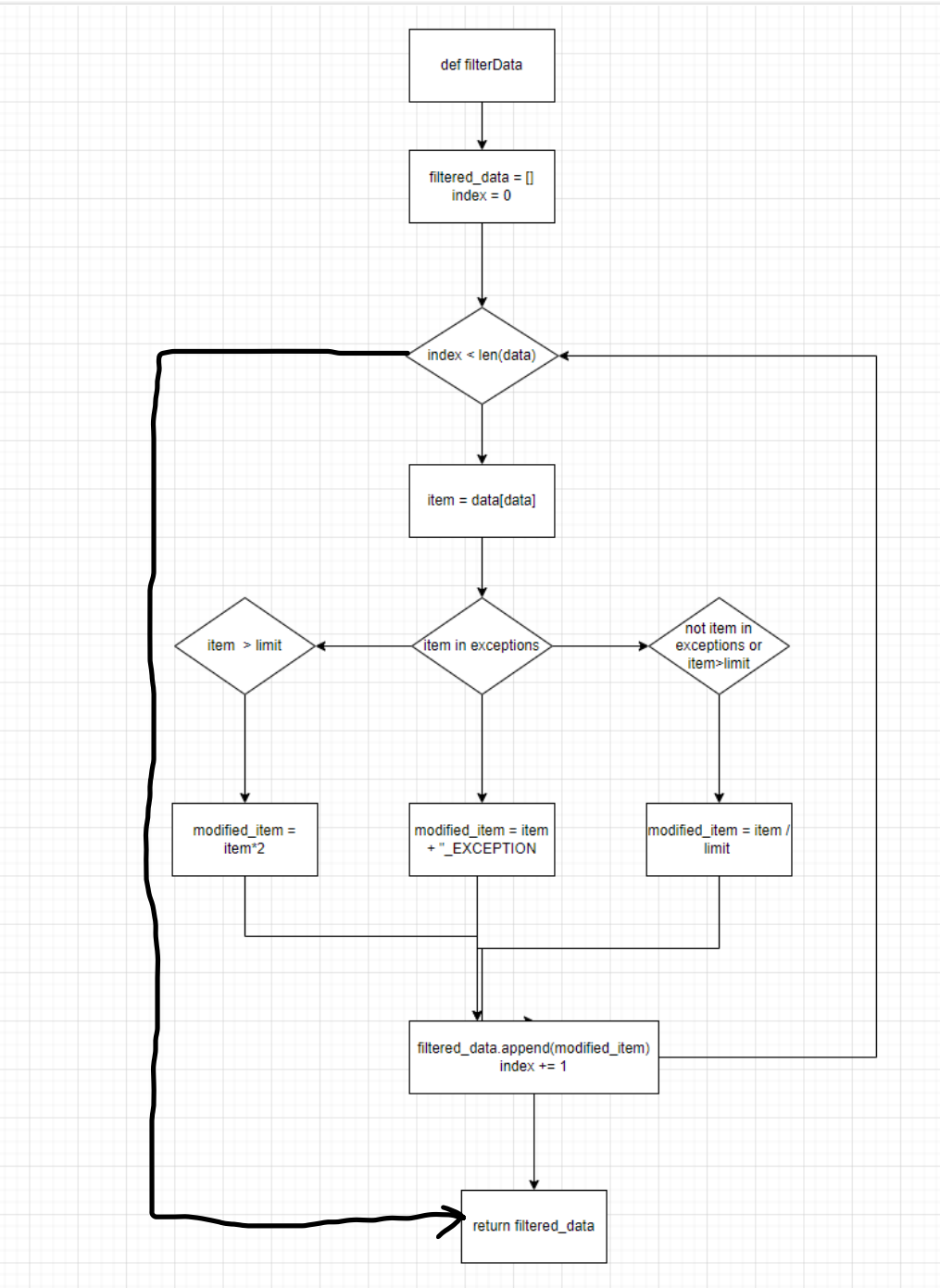
Test case 10:
Input array: [830, 564, 224, 37, 974, 786, 504, 997, 348, 958, 13, 837, 602, 756, 927, 723, 969, 797, 316, 888, 684, 906, 49, 476, 182, 132, 61, 25, 544, 183, 914, 357]
Output array: [13, 25, 37, 49, 61, 132, 182, 183, 224, 316, 348, 357, 476, 504, 544, 564, 602, 684, 723, 756, 786, 797, 830, 837, 888, 906, 914, 927, 958, 969, 974, 997]
Test passed: True

PS C:\Users\tongs\Desktop\assign1 3p95>

Context-free grammar:
Test case -> array
Array -> multiply elements

3:

A)CFG:



b)

We first need to generate random data to data, limit, and exceptions, making sure that none of them are null, and then run the program with the generated data. Verify the output.

If item is in the exceptions list, make sure the corresponding output item ends with "_EXCEPTION".

If item is greater than limit, make sure that the corresponding item is twice as large as item.

Otherwise, make sure the corresponding output item is item divided by limit.

Note those steps should be repeated several times

4:
A)
1)
Input:
Data = [](empty)
Limit = 1
Exceptions = []
Because data is empty, it cannot enter the loop at all.
4/13 = 0.30769 (31%)

2)
Input:
Data = [1]
Limite = 2
Exceptions = []
Goes to else part
8/13 = 0.6153 (62%)

3)
Input:
Data = [1,2,3]
Limite = 4
Exceptions = [2]
Goes to else and item in exceptions parts
11/13 = 0.8461 (85%)

4)
Input:
Data = [1.2.3,4]
Limit = 3
Exceptions = [3]
Goes through all
13/13 = 1.0 (100%)

B)
Since the modification only requires modifying a small part of the code, I only offer to modify the code.
1.
if item in exceptions: -> if item not in exceptions:
2.
elif item > limit: -> elif item < limit:
3:
elif item > limit: -> elif item >= limit:

4:
elif item > limit: -> elif item <= limit:
5:
while index < len(data): -> while index <= len(data):
6:
while index < len(data): -> while index > len(data):

c)
1. if item in exceptions: -> if item not in exceptions:
For this part, the part of if item in exceptions is modified, so we can use parts 3 and 4 of A, which have values for exception, so it is easier to find errors.

2. elif item > limit: -> elif item < limit:
3: elif item > limit: -> elif item >= limit:
4: elif item > limit: -> elif item <= limit:
For these three parts, we can use parts 2, 3, and 4 of A, which are all for elif item > limit modification.

5:while index < len(data): -> while index <= len(data):
6:while index < len(data): -> while index > len(data):
For these two parts, we can use parts 1 of A, which are all for while index < len(data): modification.

Ranking section:
4,3,2,1

D:
Path analysis: Test all of the above paths first with empty data to test like case1, then test the path in exception, then test item > limit the path and finally test the rest.
Branch analysis: Testing all branch paths mainly means testing exception, item greater than limit, and else cases. Make sure you test everything.
Statement analysis: You need to make sure that each one runs the same as case4.

5:
a.

```
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
```

Output_str += char.lower() -> output_str += char
The function of the program is to write smaller uppercase, lowercase change uppercase, the number is unchanged, by checking the function of the three branches found that the function of changing the case is no problem. If you look at the same number, you will see that the number has been multiplied by two (or entered twice).

b)

```
def is_failing(input_str):
    output_str = processString(input_str)
    # Add a condition to check whether the output is as expected or not
    # Return True if the output is incorrect (test fails), otherwise False

def delta_debugging(input_str):
    n = 2  # Initial granularity
    while n <= len(input_str):
        start_idx = 0
        subset_found = False
        while start_idx < len(input_str):
            subset = input_str[start_idx:start_idx+n]
            if is_failing(subset):
                input_str = subset
                n = max(n-1, 2)
                subset_found = True
                break
            start_idx += n
        if not subset_found:
            n += 1
    return input_str

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


# Test the delta_debugging function with the given input values
inputs = ["abcdefG1", "CCDDEExy", "1234567b", "8665"]
```

```
for input_str in inputs:
    minimized_input = delta_debugging(input_str)
    print(f"Minimized input for '{input_str}': '{minimized_input}'")
```

In this delta debug, is_failing checks whether the input will fail, and delta_debugging is the delta debugging algorithm, which looks for the smallest subset of tests that will fail.