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

**Assignment 2**

**Due date**: Monday, March 21st, 2023, at **23:59** (11:59 pm)

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

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

**Name: Reet Chahil Student ID: 6465314**

**Questions:**

1. Consider the following three code snippets:

**Code 1:**

public static int binarySearch(int[] arr, int l, int r, int x) {

if (r >= l) {

int mid = l + (r - l) / 2;

if (arr[mid] == x) {

return mid;

} else if (arr[mid] < x) {

return binarySearch(arr, mid + 1, r, x);

} else {

return binarySearch(arr, l, mid, x);

}

} else {

return -1;

}

}

**Code 2:**

public static int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n-1);

}

}

int result = factorial(5);

System.out.println(result);

**Code 3:**

public static void bubbleSort(int[] arr, int limit) {

int n = arr.length;

for (int i = 0; i < n; i++) {

for (int j = 0; j < n-i-1; j++) {

if (arr[j] > limit) {

return;

}

if (arr[j] > arr[j+1]) {

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

}

}

All three codes contain a bug that causes them to produce incorrect output or raise an error for some input values. If you are unable to identify the bug, you can optionally try using a techniques like random testing to generate inputs that trigger the bug.

Now, choose **one of** the above code snippets and use statistical debugging to isolate the root cause of the bug. Use and show the Failure, Context, and Increase functions for the selected/defined predicate(s).

In your answer, describe the bug you found, the inputs that trigger it, and your detailed diagnostic process using statistical debugging. **(20 pts + up to 10 pts bonus)**

**Extra-credit**, If you are able to identify the bugs in the other two code snippets, using the statistical debugging, you will receive up to 10 extra credit points (5 points per code)

Binary Search

The line in the else statement (return binarySearch(arr, l, mid, x);) must be changed to

return binarySearch(arr, l, mid - 1, x);

This bug caused the algorithm to miscalculate the midpoint for some inputs. The right index does not decrease and become smaller than the left index. So for many arrays where we searched for a key that did not exist in the array, the algorithm entered an infinite recursive loop, causing a stack overflow.

I created a list of interesting predicates.

1. Arr[mid] == x
2. Arr[mid] < x
3. Arr[mid] > x

I created hashmaps to track when:

1. The predicate was observed
2. The run passed and the predicate was true
3. The run failed and the predicate was true
4. The run succeeded and the predicate was observed
5. The run failed and the predicate was observed

I ran the binary search on the 3 sample arrays and tested 10 search keys for each array. I collected data from 30 runs. Here are the summarized results.

|  |  |  |  |
| --- | --- | --- | --- |
| Predicate | arr[mid] == x | arr[mid] < x | arr[mid] > x |
| Successful Runs | 16 | 12 | 7 |
| Failure Runs | 0 | 7 | 10 |
| Failure | 0.00 | 0.37 | 0.59 |
| Context | 0.33 | 0.33 | 0.33 |
| Increase | -0.33 | 0.04 | 0.25 |

arr[mid] > x has the highest correlation with failing runs, so it is highly likely that the bug is in the else branch of the method. After fixing the bug in that branch, I ran the tests again and they all passed.

Factorial

There is no safeguard if the user inputs a negative number. If the user inputs a negative number, the method will enter an infinite recursive loop, causing a stack overflow.

I created a list of interesting predicates.

1. n < 0
2. n = 0
3. n > 0

I created hashmaps to track the same fields as the previous example.

I computed the factorial of numbers ranging from [-10, 10]. I collected data from 21 runs. Here are the summarized results.

|  |  |  |  |
| --- | --- | --- | --- |
| Predicate | n < 0 | n = 0 | n > 0 |
| Successful Runs | 0 | 1 | 10 |
| Failure Runs | 10 | 0 | 0 |
| Failure | 1.00 | 0.00 | 0.00 |
| Context | 0.48 | 0.48 | 0.48 |
| Increase | 0.52 | -0.48 | -0.48 |

n < 0 has the highest correlation with failing runs whereas the other predicates have a high correlation with successful runs. Therefore, the bug corresponds with negative inputs. This can be fixed by changing the n = 0 to n <= 0 or by throwing an exception when a negative number is given.

1. In this assignment, you will learn how to instrument a system using OpenTelemetry and visualize the collected data. **(50 pts)**
2. Select a system to instrument and visualize. You can either choose an existing system such as spring-boot-todo-app (<https://github.com/sohamds1/spring-boot-todo-app>), or any other systems from GitHub. Alternatively, you can develop a simple system or use your previously implemented systems. Your system should be written in JAVA, Python or a language that is supported by OpenTelemetry. You can find a list of supported languages and frameworks on the OpenTelemetry website (<https://opentelemetry.io/>).
3. Instrument your system with OpenTelemetry: Use the OpenTelemetry SDK to instrument your system and collect telemetry data. Instrument the system with trace, metric, and log instrumentation to capture and trace requests, collect performance metrics, and capture log events, respectively, on all or most important components/functions.
4. Visualize the collected telemetry data: Use a visualization tool to display the collected telemetry data. You can use any tool that is compatible with OpenTelemetry, such as Jaeger, Zipkin, Grafana or any other compatible tools. In the visualization, try to display trace data, metrics, and log events with their details including but not limited to spans and their durations, involved components/functions, error/exceptions, count, rate, distribution, severity, source, and content.

**Deliverables** (in a single compressed file):

1. **Source code**: Submit the source code for your instrumented system, including all relevant components and instrumentation code.
2. **Visualization**: Submit screenshots or a video of your telemetry data visualization.
3. **Report:** Submit a short report (1/2 page to 2 pages) describing an overview of the system, a description of instrumentation, a description of telemetry data visualization, etc.

**Extra-credit**, you will get up to **5 extra points** if you organize the solution to this question in a public GitHub repository (including the source code, visualization and report) and give its URL address in your assignment solution file.

1. How would you implement a custom sampling strategy in OpenTelemetry? Provide an example of a simple custom sampling strategy that could be used to sample requests based on their HTTP method (e.g. GET, POST, PUT, DELETE, etc.). Implement the custom sampler in OpenTelemetry using your example and provide the source code. **(30 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 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***