COSC2469 Algorithms and Analysis
COSC2658 Data Structures and Algorithms

GROUP PROJECT (30%)

1.0 Project Overview

In this project, your team will design and implement an efficient algorithm that guesses a hidden secret code using a feedback mechanism. The objective is to recover the secret code in as few guesses and as little time as possible. The project is designed to assess your group's ability to balance algorithmic problem-solving, software development, and responsible AI usage. It is divided into two parts:

- In the first part, your group will collaborate to design and optimize your guessing algorithm with the support of AI tools. You are expected to:
  - Apply knowledge of algorithms and data structures.
  - Use AI tools (e.g., Val Gen AI (recommended), ChatGPT, GitHub Copilot, etc.) to support the development.
  - Evaluate AI-generated suggestions critically, verifying and adapting them for correctness, efficiency and relevance.
  - Improve your prompt engineering skills by asking clear and useful questions.
  - Document your AI usage, including prompts, reflections and logs.
  - Understand and clearly explain your algorithm in your own words (not just repeat what AI suggested).
  - Collaborate effectively in a group to deliver both the solution and the analysis.
- In the second part, each team member will independently reflect on their personal contribution, learning experience and challenges encountered during the project, and participate in a group video explanation to explain the algorithm, AI involvement and group roles.

The goal is to assess both your technical proficiency and your ability to use AI ethically and critically.

1.1 What should you NOT do?

- Blindly copy-paste AI-generated code without understanding it.
- Ask AI to complete your entire report or code for you.
- Use AI as a shortcut to avoid algorithmic thinking.
- Use the classes/interfaces defined in the Java Collections Framework and external libraries for solving the problem.

2.0 Problem Statement

You have encrypted your most valuable notes from the Algorithms course using a secret code inspired by your favourite beverage: “bạc xỉu”. Unfortunately, you forgot the secret code. All you remember is that it only contains the letters 'B', 'A', 'C', 'X', 'I', 'U', and you don't remember how long the secret code is. You are provided with a feedback function, which returns the number of characters that are correct and in the correct position for any guess.

Your goal is to first determine the correct length of the secret code, and then to discover the entire secret code using as few guesses as possible. This problem is challenging because the secret code length is unknown and must be inferred from the feedback function, and the total number of guesses will directly affect your algorithm's performance.

3.0 Technical Description

A basic brute force solution is provided on the Group Project page in the course Canvas. This solution includes logic to deduce the secret code length and then attempts all possible combinations in lexicographic order.

Provided source code files:

- **SecretCodeGuesser.java:** You will modify this class, especially the start() method, to implement your guessing strategy.
- **SecretCode.java:** This class simulates the hidden secret code and provides the guess() method. ***Do not modify this file.***

Your solution will be assessed not just on whether it recovers the secret code correctly, but how efficiently it does so, measured by the total number of calls made to guess() method. You may only use the guess() method for feedback, and therefore, are not allowed to inspect the correct secret code or modify the SecretCode class.

Within the start() method, your algorithm should:

1. Create an instance of the provided SecretCode class.
2. Use the guess() method iteratively to:
   a. First determine the correct secret code length,
   b. Then identify the secret code content.
3. Stop once the full secret code is found. Output the secret code, the number of guesses taken, and the total execution time of your algorithm.

Important: Every call to the guess() method, including those used to determine the secret code length, will be counted in the total guess count.

Testing procedure:

1. The secret code is hidden. Your program will not have direct access to it.
2. Your program will be tested with three different secret codes, each of unknown length, where length <= 18.
3. Each call to guess() method, including those used to determine the secret code length, will increase the counter. This is referred to as the guess count.
4. Your algorithm will be evaluated based on the total number of guesses across all three test runs and the total execution time. Therefore, the lower the guess count and execution time, the better your algorithm's performance.

4.0 Deliverables

1. Technical report: Submit a technical report with the following five sections.
   A. Algorithms and Data Structures
      - Describe the data structures and algorithms you applied or designed.

  - Explain their logic, purpose and how they improve performance.
  - Include visuals (e.g., flowcharts, diagrams) where helpful.

B. Complexity Analysis
  - Provide time complexity analysis of your algorithm.
  - Compare your algorithm to the brute force benchmark.

C. AI Usage and Reflection
  - Explain how AI tools (e.g., Val Gen AI (recommended), ChatGPT, GitHub Copilot) were used to support the development of your algorithm.
  - Include at least two meaningful prompt-response pairs with analysis:
    - What prompt you gave.
    - What response you received.
    - Whether the response helped or not, and why.
    - How you adapted or improved the response (if applicable).
  - Critically reflect on the benefits and limitations of using AI tools.
  - Include your full AI usage logs in the Appendix.

D. Experimental Evaluation
  - State your system specifications: CPU, RAM, OS, Java version.
    Example: All experiments were conducted on a laptop with an Intel Core i7-1165G7 CPU @ 2.80 GHz, 16GB RAM, running Windows 11 (64-bit), using JDK 21.
  - Describe how you evaluated the correctness and efficiency of your algorithm experimentally.
  - Test your algorithm on at least three distinct test cases:
    - Best case scenario.
    - Average case scenario.
    - Worst case scenario.
  - For each test case, report total number of guesses, total execution time and short interpretation of the results (e.g., how the test case reflects algorithm's strengths or limitations).

E. Conclusion and Reflection
  - Summarize the strengths and limitations of your solution.
  - Reflect on what you would improve with more time, knowledge or tools.

2. Source Code
   - Submit your full Java implementation of SecretCodeGuesser using your custom algorithm (not the brute force version).
   - Follow best practices (clear comments, meaningful variable names, readable code, and well-structured).
3. Video Explanation: Upload your video to YouTube and present its URL in README.txt.
   - Length: 7 - 10 minutes.
   - Introduce your team.
   - Explain how your algorithm works and its performance (with examples).
   - Describe what AI tools were used and how they supported development.
   - Clarify who did what in the team.
4. Reflective Report (Individual – each member must submit)
   - Length: 400–500 words.

- Describe your personal contribution and learning.
- Reflect on challenges faced, your use of AI tools, and your problem-solving experience.

5. README.txt
   - State the contribution score of each member (see Section 5.0 Contribution Score)
   - Provide the link to your project video as described in the Video Explanation.

Submission instruction:

- Technical report (PDF format), Java source code, reflective reports (PDF format) and README.txt file should be placed and submitted as a single zip file on Canvas.
- The name format of the zip file: Group_no_Assessment3.zip, e.g. Group_00_Assessment3.zip if your group number is 0.
- Only one member needs to submit the zip file.

5.0 Contribution Score

Each member starts with 3 points. The team can adjust points (+/-) based on actual contribution, but the total point of the whole team remains as (number of members x 3). Additional rules:

- The maximum point for a member is 5.
- A member who gets 0 points receives 0 for the entire project. In this case, the total point of the whole group = (the number of members whose scores > zero) * 3.
- The contribution score must be agreed upon by all members. If there are disagreements, you must inform the lecturer before the due time.
- The maximum score for the whole project is 30. If you get more than 30 (due to a high contribution score), the final score is capped at 30.

6.0 AI Use and Referencing Requirements

You must acknowledge and reference your usage of generative AI tools (e.g., Val Gen AI, ChatGPT, Copilot) in your source code, report, or reflection. This includes:

- Clearly stating the AI tools used and the specific model (e.g., GPT-4) in your source code comments, written report and/or individual reflection.
- Maintaining a log of your interactions with AI tools (prompts and responses) and submitting this as an Appendix.
- Critically reflecting on how AI suggestions were used, adapted, or rejected, and your rationale behind your decisions.

Ensure your referencing follows the official RMIT Library guidelines on citing generative AI tools @ https://rmit.libguides.com/referencing_AI_tools#s-lg-box-22377324

Failure to properly cite or document the use of generative AI may be considered a breach of academic integrity.

Example 1: Parts of the initial draft of the algorithm were generated using Val Gen AI (OpenAI's GPT-4 (o4) model as of July 2025). Prompts such as "What is a better search strategy than brute-force for a secret code of unknown length?" were used. All output was verified, edited, and adapted by <student name>. The full AI interaction log is included in the Appendix.

Example 2: We used ChatGPT (GPT-4, July 2025 version) to refine our approach for minimizing the number of guesses. We asked: "How can I determine the length of a secret string using a feedback function?" ChatGPT suggested incrementally guessing increasing string lengths until feedback plateaued. While the initial logic was correct, we had to refine it further for edge cases. The full AI interaction log is included in the Appendix.

Appendix A: Example output using the brute force solution

All experiments were conducted on a laptop with an Intel Core i5-1135G7 CPU @ 2.40 GHz, 16GB RAM, running Windows 10 (64-bit), using JDK 21

```
Number of guesses: 3132550996
I found the secret code. It is BACXIUBACXIUBA
Time taken: 401162 ms
```

Appendix B: Working with AI in programming

- **AI Prompts for brute force optimization:** Students can use AI to do the following:
  - ✓ Improve brute force by analyzing the feedback and reducing unnecessary guesses.
  - ✓ Explore heuristics or probabilistic strategies (e.g., narrowing down character positions using feedback scores).
  - ✓ Example AI prompts:
    - "Suggest a more efficient strategy than brute-force to guess unknown character code when feedback tells how many characters are in correct position."
    - "How can I update only specific character positions in a string based on feedback from the guess function?"

- **Reflection:** Critically evaluate the AI responses.
  - ✓ Are the changes logically correct?
  - ✓ Does it reduce the number of guesses?
  - ✓ What assumptions did the AI make, and are they valid in this context?

- Get the most out of AI coding support in terms of structure queries for debugging or idea generation, evaluation of AI-generated code for correctness and performance.
    - Prompt Engineering Guide for Students
    - ChatGPT for Code Improvement
    - ChatGPT for Developers Guide
    - How to Use AI to Improve Code Quality