Student name: Sk Md Shariful Islam Arafat	**Student ID:** a1983627
Problem name: LetterBar	**Solution number - 1**

Q1.1. What are the errors/issues you identified and what is your recommendations/suggestions to improve the solutions to pass all the test cases? Please describe your approach to write the solution for the given problem.
In case you used ChatGPT, what are errors, recommendations/suggestions ChatGPT provided to improve the solution to pass all the test cases? what is your takeaway from these identified errors and recommendations?

Errors/Issues Identified

In Solution 1, the main issue is **inefficiency**. The code generates all possible substrings ($O(N^2)$) and then checks each one for duplicates in $O(N)$, resulting in **$O(N^3)$** time complexity. While this still works for the problem constraints ($N \leq 50$), it is not scalable. Additionally:

- Used a `HashMap<Character, Integer>` when a `HashSet<Character>` would have been simpler and more efficient, since only uniqueness matters.
- The variable `maxLen` was initialized to `-1`, which is unnecessary. It should start at `0`, since the minimum possible valid substring length is 0.

Recommendations/Suggestions

1. Replace `HashMap` with a `HashSet` to simplify uniqueness checks.
2. Initialize `maxLen = 0` instead of `-1` for clarity.
3. If efficiency is important, switch to the **sliding window approach** (Solution 2 or 3). That reduces complexity to $O(N)$ using two pointers and a `HashSet` to maintain the current window of unique characters.

Approach to Writing the Solution

```
For each i in [0..n-1]:
    For each j in [i+1..n]:
        ▪ Let s = letters.substring(i, j).
        ▪ Check uniqueness: iterate s, insert into a HashSet<Character>. If an insert fails, it has a duplicate.
        ▪ If unique, update maxLen = max(maxLen, s.length()).
```

Takeaway from Errors/Recommendations

- The brute-force approach is correct but inefficient ($O(N^3)$), which is acceptable only because of the small input size constraint ($N \leq 50$).
- Using a `HashMap` for duplicate checking was unnecessary; a `HashSet` is simpler and more efficient when only uniqueness matters.
- Initializing `maxLen` to `-1` was misleading; starting at `0` is safer and more logical.

Q1.2. What challenges have you faced in identifying a complete solution to the given problem? What type of help would you need to solve similar problems like this? What kind of feedback would you like to receive from your teaching team to improve your programming skills?

Challenges Faced

- Understanding that the problem is essentially finding the *longest substring without repeating characters* rather than removing characters arbitrarily.
- Ensuring that all possible substrings were checked without missing edge cases (e.g., all characters same, single character input, long unique strings).

Help Needed

- Guidance on recognizing when brute-force is acceptable versus when optimization is required.
- More exposure to standard algorithmic techniques like **sliding window**, which would naturally lead to more efficient solutions.

Feedback Wanted from Teaching Team

- Feedback on code efficiency and clarity, not just correctness, so I can improve my programming style.
- Suggestions on better data structures or patterns that fit the problem more elegantly.

Q1.3. What are your suggestions to improve the feedback provided by AI tools like ChatGPT to cater to your learning needs and to guide you in programming tasks which would help you to be a skilled programmer?

Suggestions to Improve AI Feedback (e.g., ChatGPT)

- Provide **step-by-step reasoning** before giving the final solution, so I can understand the thought process instead of just copying code.
- Highlight **common mistakes** (like using `HashMap` instead of `HashSet`, or initializing variables incorrectly) and explain why they matter.
- Offer **multiple solution strategies** (e.g., brute-force, optimized sliding window) with comparisons of time and space complexity.
- Encourage me to **analyze trade-offs** (correctness vs efficiency) rather than focusing only on passing test cases.
- Give **hints first**, instead of the full code, so I can attempt solving problems independently before seeing a complete implementation.
- Suggest **coding best practices** (naming conventions, readability, removing debug prints) alongside algorithmic improvements.
- Provide **targeted practice problems** after giving feedback, to reinforce concepts (e.g., other problems solvable with sliding window).

Problem name: LetterBar	**Solution number - 2**

Q2.1. What are the errors/issues you identified and what is your recommendations/suggestions to improve the solutions to pass all the test cases? Please describe your approach to write the solution for the given problem.
In case you used ChatGPT, what are errors, recommendations/suggestions ChatGPT provided to improve the solution to pass all the test cases? What is your takeaway from these identified errors and recommendations?

Errors/Issues Identified

- The solution is correct and passes all test cases, but it still has some **efficiency drawbacks**:
    - Using `ArrayList.contains()` and `ArrayList.indexOf()` makes certain operations O(N).
    - Overall complexity is closer to **O(N^2)** instead of O(N).
- Managing duplicates by removing elements with `subList.clear()` works, but it's not the most elegant way — a `HashSet` with two pointers would be simpler and more efficient.
- Readability could be improved by clearly separating the **"move left pointer"** logic from the **"add right pointer"** logic.

Recommendations/Suggestions

- Replace `ArrayList` with a `HashSet<Character>` for O(1) duplicate checks.
- Use **two pointers (`left`, `right`)** explicitly to track the sliding window rather than clearing sublists.
- Keep code modular: extracting "check and shrink window" into a helper function can make the logic cleaner.
- Add explanatory comments to highlight the invariant: *the window always contains unique characters*.

Approach to Write the Solution

1. **Problem Understanding**: We need the maximum contiguous substring without repeating characters → this is a classic sliding window problem.
2. **Window Maintenance**: Use a dynamic data structure to represent the current substring (`charList` in your version).
3. **Duplicate Handling**:
    - If the new character already exists in the window, remove characters from the start until the duplicate is gone.
    - Then add the new character at the end.
4. **Max Length Update**: After each step, compare the current window size with the maximum length so far.
5. **Result**: At the end, the maximum recorded window size is the answer.

Takeaway from Errors/Recommendations

- Even though this solution is much more efficient than brute-force, it still can be optimized further.
- The main lesson is to **match data structures with problem requirements** — using a set instead of a list reduces complexity from O(N^2) to O(N).
- It also showed the importance of **writing clean, modular code** to make algorithms easier to reason about.
- By practicing with sliding window, I gained a deeper understanding of a reusable technique that applies to many substring/subarray problems.

Q2.2. What challenges have you faced in identifying a complete solution to the given problem? What type of help would you need to solve similar problems like this? What kind of feedback would you like to receive from your teaching team to improve your programming skills?

Challenges Faced

- Translating the sliding window idea into working code without falling back into brute-force logic.
- Handling duplicates correctly without breaking the unique window invariant.

Help Needed

- Guidance on selecting the most efficient data structure when multiple options (ArrayList vs HashSet) can solve the same problem.
- More practice with common patterns like sliding window, two pointers, and hashing.
- Feedback on how to balance code clarity and efficiency, especially for algorithmic tasks.

Feedback I'd Like from Teaching Team

- Detailed feedback on **algorithm design choices**, not just correctness.
- Suggestions on how to identify when my solution is sub-optimal, even if it passes all test cases.
- Encouragement to **analyze time/space complexity** explicitly in submissions.

Q2.3. What are your suggestions to improve the feedback provided by AI tools like ChatGPT to cater to your learning needs and to guide you in programming tasks which would help you to be a skilled programmer?

Suggestions to Improve AI Feedback

- Provide **step-by-step sliding window walkthroughs** on test cases, showing how `left` and `right` pointers move, so I can visualize why the algorithm works.
- Highlight **data structure trade-offs** (e.g., using `ArrayList` vs `HashSet`) and explain how each affects complexity.
- When my solution is correct but not optimal, give **gentle hints first** ("your solution is $O(N^2)$, can you think of a way to reduce contains/indexOf lookups?") instead of jumping directly to final code.
- Emphasize the importance of **time/space complexity analysis** and ask me to calculate it myself before providing the answer.
- Suggest **incremental refactoring tasks**: e.g., "try rewriting your code with a HashSet" → "now try explicit two pointers."
- Give **best practice reminders** (clear invariants, modular methods, no unnecessary list clearing) to reinforce good habits.
- Recommend **related problems** (e.g., "Longest substring without repeating characters," "Longest subarray with distinct elements") to practice applying the sliding window pattern.
- Encourage me to **compare solutions** (Solution 1 brute-force vs Solution 2 sliding window) and reflect on efficiency gains.

Write your solution below:

```java
import java.util.*;

public class LetterBar {
    public static int maxLength(String letters) {
        List<Character> charList = new ArrayList<>();
        int maxLength = 0;

        for (int right = 0; right < letters.length(); right++) {
            char current = letters.charAt(right);

            // Duplicate found => remove all char up to its previous index
            if (charList.contains(current)) {
                int index = charList.indexOf(current);
                charList.remove(index);
                if (index > 0) {
                    charList.subList(0, index).clear();
                }
            }

            // Add current character to the window
            charList.add(current);

            // Update maximum length
            maxLength = Math.max(maxLength, charList.size());
        }

        return maxLength;
    }

    public static void main(String[] args) {
        System.out.println(maxLength("srm")); // 3
        System.out.println(maxLength("dengklek")); // 6
        System.out.println(maxLength("haha")); // 2
        System.out.println(maxLength("www")); // 1
    }
}
```