Problem Solving

Solution for Question 1

Description:

The goal is to get the count of the total number of valid arrays formed with the given secret array, including the lower and upper bounds. As the requirement is limited to the count of arrays formed, rather than generating the actual arrays. The solution is to get the size of the window and calculate the count of windows that can uniquely fit in the given lower and upper bounds.

Time and Space Complexity:

- Time: O (N)
- Space: O (1)

Code:

```
public static int countAnalogousArrays(int[] consecutiveDifference, int
lowerBound, int upperBound) {
   long curDiff = 0, windowStart = Long.MAX_VALUE, windowEnd = Long.MIN_VALUE;

   for (long d: consecutiveDifference) {
       curDiff -= d;
       windowStart = Math.min(curDiff, windowStart);
       windowEnd = Math.max(curDiff, windowEnd);
   }
   long windowSpread = windowEnd - windowStart;
   long totalOptions = (long) (upperBound - lowerBound) - windowSpread + 1;

   return totalOptions < 0 ? 0 : (int) totalOptions;
}
```

System Design

Solution for Question 1

add_product

```
private static final String SQL_INSERT_PRODUCT = """
   INSERT INTO products(product_id, name, price, stock)
   VALUES (?, ?, ?, ?)
   """;

// Assuming conn is valid
public boolean add_product(int product_id, String name, double price, int
stock) {
   try (PreparedStatement ps = conn.prepareStatement(SQL_INSERT_PRODUCT)) {
       ps.setInt(1, product_id);
       ps.setString(2, name);
       ps.setDouble(3, price);
       ps.setInt(4, stock);
       ps.executeUpdate();
       return true;
   } catch (SQLException e) {
       // Duplicate key or CHECK constraint failure
       return false;
   }
}
```

record_sale

```
private static final String SQL_GUARDED_DECREMENT = """
   UPDATE products
      SET stock = stock - ?
    WHERE product_id = ?
      AND stock >= ?
   """;

private static final String SQL_INSERT_SALE = """
   INSERT INTO sales(product_id, quantity_sold)
   VALUES (?, ?)
   """;

public boolean record_sale(int product_id, int quantity_sold) {
   if (quantity_sold <= 0) return false;

   boolean priorAuto;
   try {
       priorAuto = conn.getAutoCommit();  // Assuming conn is valid
```

```
        conn.setAutoCommit(false);

        int affected;
        try (PreparedStatement dec =
conn.prepareStatement(SQL_GUARDED_DECREMENT)) {
            dec.setInt(1, quantity_sold);
            dec.setInt(2, product_id);
            dec.setInt(3, quantity_sold);
            affected = dec.executeUpdate();
        }

        if (affected == 0) {
            conn.rollback();
            conn.setAutoCommit(priorAuto);
            return false; // insufficient stock or product not found
        }

        try (PreparedStatement ins = conn.prepareStatement(SQL_INSERT_SALE)) {
            ins.setInt(1, product_id);
            ins.setInt(2, quantity_sold);
            ins.executeUpdate();
        }

        conn.commit();
        conn.setAutoCommit(priorAuto);
        return true;

    } catch (SQLException e) {
        try { conn.rollback(); } catch (SQLException ignored) {}
        try { conn.setAutoCommit(true); } catch (SQLException ignored) {}
        return false;
    }
}
```

top_selling_products

```
private static final String SQL_TOP_SELLERS = """
    SELECT p.name, SUM(s.quantity_sold) AS total_qty, p.product_id
      FROM sales s
      JOIN products p ON p.product_id = s.product_id
     GROUP BY s.product_id
     ORDER BY total_qty DESC, p.product_id ASC
     LIMIT 3
    """;

public List<String> top_selling_products() {
```

```
    List<String> out = new ArrayList<>();
    try (PreparedStatement ps = conn.prepareStatement(SQL_TOP_SELLERS);
         ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            String name = rs.getString(1);
            int qty = rs.getInt(2);
            out.add(name + "," + qty);
        }
    } catch (SQLException ignored) {}
    return out;
}
```

SQL

Solution for Question 1

Description:

Here, there are three things that need to be identified.

1. ROOT - this is fairly simple, where the P_ID is NULL
2. LEAF - This needs an inner query to identify all the parent nodes, and then see if the current node is not in the parent list
3. Inner - If the above two do not satisfy, then the node is an inner node

Once we get the node type, then we ORDER BY ID in ascending order

SQL Query:

```
SELECT
    ID,
    CASE
        WHEN P_ID IS NULL THEN 'ROOT'
        WHEN ID NOT IN (SELECT P_ID FROM TREE WHERE P_ID IS NOT NULL) THEN
'LEAF'
        ELSE 'INNER'
    END AS TYPE
FROM TREE
ORDER BY ID;
```

REST API

Solution for Question 1

Code: https://github.com/mohitchhapru/getTopRatedFoodOutlets