COMPUTER SOCIETY OF INDIA (CSI)
COEP TECH STUDENT CHAPTER

PRESENTS

CODE QUEST 5.0

EDITIORIAL

EQUAL PIZZA PACT

EXPLANTION

The problem requires us to divide the given N pizza slices into exactly D equal portions by making the minimum number of radial cuts. Each slice already has a given angle, and we can cut it further to achieve uniformity. The challenge is to determine the best strategy to minimize the number of cuts while ensuring that exactly D equal slices are obtained.

STEPS:

1. Sorting the slices:
   - We first sort the given N pizza slices based on their internal angles. This helps in efficiently trying to redistribute slices by making cuts in a systematic way.

2. Iterating through possible target slice sizes:
   - We assume that the final equal slice size should be some factor of an existing slice size.
   - We iterate through potential slice sizes (denoted as m) based on existing slice values to check if we can split other slices to match this target.

EXPLANTION

3. Checking divisibility and making cuts:

- For each potential target size, we check how many cuts are needed to divide the slices accordingly.
- If a slice's angle can be evenly divided into the target size, we determine how many complete slices can be formed and count the cuts required.
- If a slice cannot be divided exactly, we approximate the best division possible while ensuring that D slices are created.

4. Tracking the minimum cuts:

- Across all possible equal slice sizes, we track the minimum number of cuts required to create exactly D slices.
- The approach ensures that we always minimize unnecessary extra cuts while maintaining the required count of slices.

EXPLANTION

Example Walkthrough:

Input
1
3 4
180000000000 90000000000 90000000000
- 1 test case
- 3 slices: 180 × 10^9 nanodegrees, 90 × 10^9 nanodegrees, 90 × 10^9 nanodegrees
- 4 friends need equal slices

Step 1: Sorting Slices
The given angles are already sorted:
[90 × 10^9, 90 × 10^9, 180 × 10^9]

Step 2: Trying Possible Slice Sizes
- Since we need 4 equal slices, we check what size each slice should ideally be.
- Possible candidates for the final slice size are:
  - 90 × 10^9 nanodegrees
  - 45 × 10^9 nanodegrees

EXPLANTION

Step 3: Dividing Slices

- If we take 90×10^9 as the target slice size:
  - The first two slices are already 90×10^9, so they don’t need cuts.
  - The last slice (180×10^9) can be split into two 90×10^9 slices with 1 cut.
  - Total slices now: 4 (matches required number).
  - Cuts required: 1.

- If we take 45×10^9 as the target slice size:
  - The first two slices (90×10^9) need to be split into two each (1 cut per slice 2 cuts total).
  - The last slice (180×10^9) needs to be split into four 45×10^9 slices (3 cuts).
  - Total slices now: 4 (matches required number).
  - Cuts required: 3.

Step 4: Choosing the Minimum Cuts

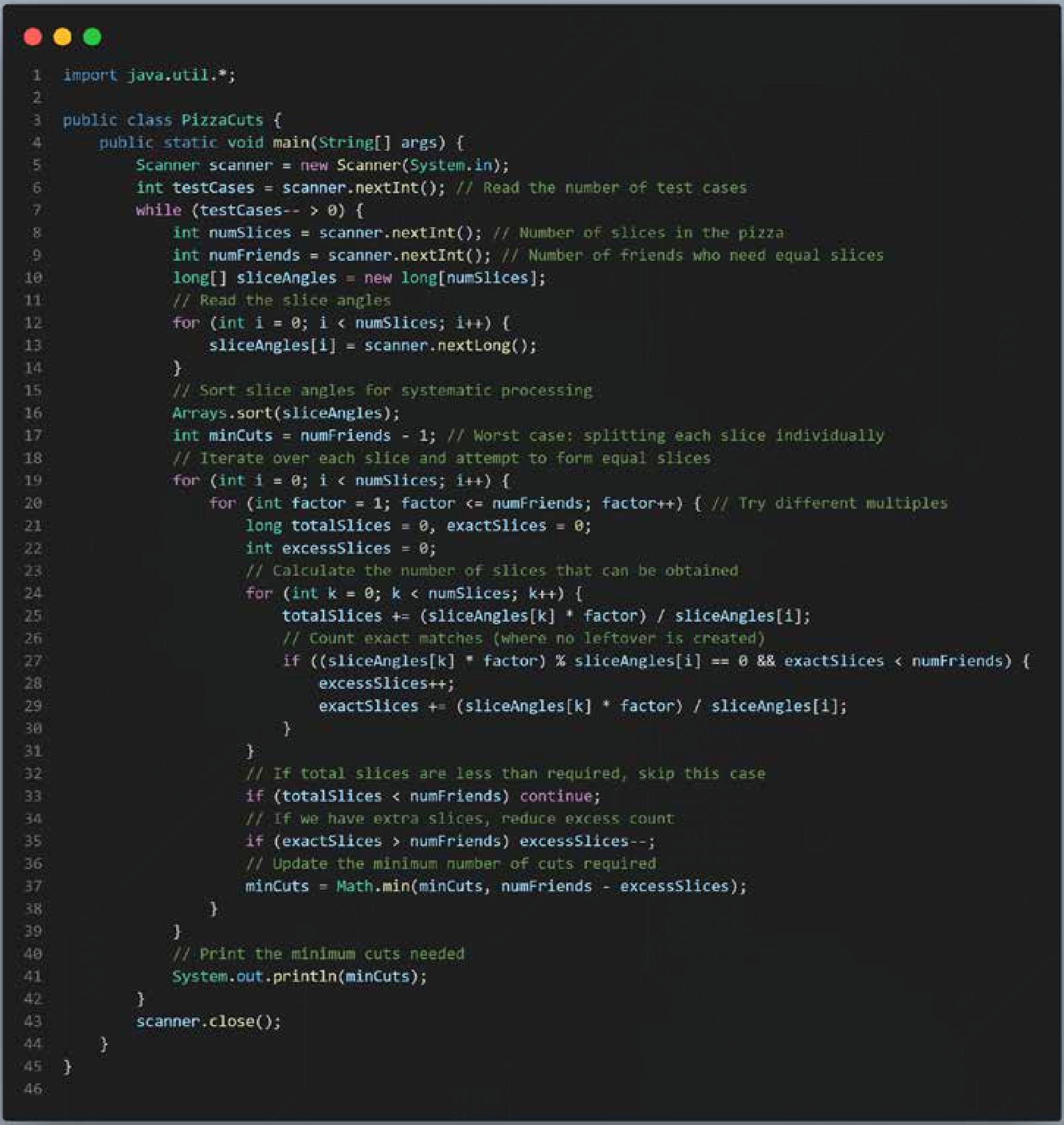
Among all tested slice sizes, the minimum number of cuts required is 1 cut when we choose 90×10^9 as the final equal slice size.

EXPLANTION

Output
1

This means that the minimum number of cuts required to create 4 equal slices is 1.

SOLUTION



```java
import java.util.*;

public class PizzaCuts {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int testCases = scanner.nextInt(); // Read the number of test cases
        while (testCases-- > 0) {
            int numSlices = scanner.nextInt(); // Number of slices in the pizza
            int numFriends = scanner.nextInt(); // Number of friends who need equal slices
            long[] sliceAngles = new long[numSlices];
            // Read the slice angles
            for (int i = 0; i < numSlices; i++) {
                sliceAngles[i] = scanner.nextLong();
            }
            // Sort slice angles for systematic processing
            Arrays.sort(sliceAngles);
            int minCuts = numFriends - 1; // Worst case: splitting each slice individually
            // Iterate over each slice and attempt to form equal slices
            for (int i = 0; i < numSlices; i++) {
                for (int factor = 1; factor <= numFriends; factor++) { // Try different multiples
                    long totalSlices = 0, exactSlices = 0;
                    int excessSlices = 0;
                    // Calculate the number of slices that can be obtained
                    for (int k = 0; k < numSlices; k++) {
                        totalSlices += (sliceAngles[k] * factor) / sliceAngles[i];
                        // Count exact matches (where no leftover is created)
                        if ((sliceAngles[k] * factor) % sliceAngles[i] == 0 && exactSlices < numFriends) {
                            excessSlices++;
                            exactSlices += (sliceAngles[k] * factor) / sliceAngles[i];
                        }
                    }
                    // If total slices are less than required, skip this case
                    if (totalSlices < numFriends) continue;
                    // If we have extra slices, reduce excess count
                    if (exactSlices > numFriends) excessSlices--;
                    // Update the minimum number of cuts required
                    minCuts = Math.min(minCuts, numFriends - excessSlices);
                }
            }
            // Print the minimum cuts needed
            System.out.println(minCuts);
        }
        scanner.close();
    }
}

```