Diagonal Sum Problem🎯

> The **Diagonal Sum Problem** involves calculating the sum of elements along the **primary diagonal** and **secondary diagonal** of a square matrix. A square matrix has equal rows and columns (e.g., 3×3, 4×4).

🎯 Key Points:

- **Primary Diagonal**: Elements where the

  (row index = column index) e.g., [0][0], [1][1], [2][2].
- **Secondary Diagonal**: Elements where the

  (row index + column index = matrix size - 1) e.g., [0][2], [1][1], [2][0] in a 3×3 matrix.
- Sometimes, the problem requires adding both diagonals or just one!

🧩 Example:

For a 3×3 matrix:

```
1 2 3
4 5 6
7 8 9
```

- Primary diagonal sum: 1 + 5 + 9 = **15**
- Secondary diagonal sum: 3 + 5 + 7 = **15**
- Total diagonal sum (if both are added): **30**

📐 Understanding the Matrix

A **matrix** is a 2D grid of numbers. In Java, it's represented as a 2D array (`int[][]`).

🔷 Key Concepts:

- **Rows**: Horizontal lines (e.g., row 0, row 1).
- **Columns**: Vertical lines (e.g., column 0, column 1).
- **Indices**: Start at 0 (like in arrays).

🧮 Example:

```
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

- `matrix[0][0]` = 1 (first row, first column)
- `matrix[1][2]` = 6 (second row, third column)

🔄 Steps to Solve the Problem

Here's how to approach the Diagonal Sum Problem:

📋 Step-by-Step:

1. **Initialize a sum variable** (e.g., `int sum = 0`).
2. **Loop through each element** of the matrix using nested loops (rows and columns).
3. **Check if the element is on the primary diagonal**:

  - If `row == column`, add it to the sum.

4. **Check if the element is on the secondary diagonal**:
   - If `row + column == matrix.length - 1`, add it to the sum.
5. **Return the total sum**.

⚠️ Note:

- In odd-sized matrices (e.g., 3×3), the center element (`matrix[1][1]`) is counted **twice**. Some problems require subtracting it once to avoid duplication.

💻 Java Code Example

Here's a simple Java program to calculate the diagonal sum:

```
public class DiagonalSum {
    public static void main(String[] args) {
        int[][] matrix = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        int sum = calculateDiagonalSum(matrix);
        System.out.println("Diagonal Sum: " + sum); // Output: 30
    }

    public static int calculateDiagonalSum(int[][] matrix) {
        int sum = 0;
        int n = matrix.length; // Size of the square matrix
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                // Check primary diagonal
                if (i == j) {
                    sum += matrix[i][j];
                }
                // Check secondary diagonal
                if (i + j == n - 1) {
                    sum += matrix[i][j];
                }
            }
        }
        return sum;
    }
}
```

🧪 Sample Output:

For the above matrix, the output is **30** (1+5+9 + 3+5+7).

🌍 Real-Life Applications

The Diagonal Sum Problem isn't just theoretical! It can be used in:

- **Game Development**: Calculating scores on a game board where diagonal moves matter.
- **Data Analysis**: Summing values on diagonals in datasets (e.g., stock prices over time).
- **Image Processing**: Analyzing diagonal patterns in pixel data.

⚠️ Common Mistakes to Avoid

1. **Incorrect Indexing**: Forgetting that indices start at 0.
2. **Double Counting**: Not handling the center element in odd-sized matrices.
3. **Wrong Comparison**: Using `i != j` instead of `i == j` for the primary diagonal.

🧪 Self-Check Questions

1. What is the index of the element at the second row, third column in a 3×3 matrix?
2. Why might the center element be counted twice in the diagonal sum?
3. How would you modify the code to calculate only the primary diagonal sum?

🔚 Quick Recap

- The Diagonal Sum Problem involves summing elements on the primary and secondary diagonals of a square matrix.
- Use nested loops to iterate through the matrix and check diagonal conditions.
- Be careful with odd-sized matrices to avoid double-counting the center element.
- Java code uses `matrix[i][j]` to access elements and `i == j` / `i + j == n-1` for diagonals.