Binary Search Fix

Let's go through this code line by line and break it down for better understanding.

🔹 Problem Statement Recap

We are given an **m × n matrix**. Our goal is to **sort each diagonal in ascending order** while keeping the relative positions of the diagonals intact.

A **diagonal** in a matrix is a set of elements that move **down-right** (`row++`, `col++`) starting from some cell.

🔹 Example

📌 Input Matrix

```
3  3  1  1
2  2  1  2
1  1  1  2
```

📌 Output Matrix (after sorting each diagonal)

```
1  1  1  1
1  2  2  2
1  2  3  3
```

🔹 Step-by-Step Code Walkthrough

🔹 Function: `diagonalSort(int[][] mat)`

This function initiates the diagonal sorting process.

java

```
public int[][] diagonalSort(int[][] mat) {
    int n = mat.length;       // Number of rows
    int m = mat[0].length;    // Number of columns
```

✅ `n` stores the total number of rows in the matrix.

✅ `m` stores the total number of columns in the matrix.

🔹 First Loop: Sorting Diagonals from the First Row (`row = 0`)

```java
// row = 0
for(int col = 0 ; col < m ; col++){
    sort(mat , 0 , col , n , m);
}
```

💡 **What does this loop do?**

- It starts at **row = 0** (first row).
- Iterates through **each column (col = 0 to m-1)**.
- Calls `sort(mat, 0, col, n, m)`, which sorts the diagonal **starting from** `(0, col)`.

Example

Start Position	Extracted Diagonal
`(0,0)` → `(1,1)` → `(2,2)`	[3, 2, 1]
`(0,1)` → `(1,2)` → `(2,3)`	[3, 1, 2]
`(0,2)` → `(1,3)`	[1, 2]
`(0,3)`	[1]

✅ This ensures **all diagonals starting from the first row are covered.**

🔹 Second Loop: Sorting Diagonals from the First Column (`col = 0`)

```java
// col = 0;
for(int row = 1 ; row < n ; row++){
    sort(mat , row , 0 , n , m);
}
```

💡 What does this loop do?

- It starts at **col = 0** (first column).
- Iterates through **each row (row = 1 to n-1)**.
- Calls `sort(mat, row, 0, n, m)`, which sorts the diagonal **starting from** `(row, 0)`.

Example

Start Position	Extracted Diagonal
`(1,0)` → `(2,1)`	[2, 1]
`(2,0)`	[1]

✅ This ensures **all diagonals starting from the first column are covered.**
✅ The `(0,0)` diagonal is already covered in the first loop, so we start from `row = 1`.

🔹 Function: `sort(int[][] mat, int row, int col, int n, int m)`

This function sorts a single diagonal starting from `(row, col)`.

```java
public void sort(int[][] mat , int row , int col , int n , int m){
    List<Integer> values = new ArrayList<>();
```

✅ Creates an **empty list** `values` to store diagonal elements.

🔹 Step 1: Extract Diagonal Elements

```java
```

```
int r = row , c = col;

while(r < n && c < m){
    values.add(mat[r][c]);
    r++;
    c++;
}
```

💡 **What happens here?**

- `(r, c)` starts at `(row, col)`.
- Moves **down-right** (`r++`, `c++`).
- Stores diagonal elements in `values`.

Example

Starting from `(0,0)`:

```ini
values = [3, 2, 1]
```

Starting from `(0,1)`:

```ini
values = [3, 1, 2]
```

🔹 Step 2: Sort the Extracted Values

```java
Collections.sort(values);
```

✅ Sorts the diagonal elements in **ascending order**.

For `(0,0) → (1,1) → (2,2)`, we get:

```less
```

```
Before Sorting: [3, 2, 1]
After Sorting:  [1, 2, 3]
```

🔹 Step 3: Write Back the Sorted Values

```java
r = row;
c = col;
int index = 0;

while(r < n && c < m){
    mat[r][c] = values.get(index++);
    r++;
    c++;
}
```

💡 What happens here?

- Starts at `(row, col)`.
- **Writes back the sorted values** into the matrix along the diagonal.

Example

For diagonal `(0,0) → (1,1) → (2,2)`:

```makefile
Original: [3, 2, 1]
Sorted:   [1, 2, 3]
```

✅ This replaces the **original** matrix values with the sorted diagonal.

🔹 Final Sorted Matrix

```
1  1  1  1
1  2  2  2
1  2  3  3
```

🔹 Time Complexity Analysis

For an `m × n` matrix:

1. **Extracting a diagonal → O(D)**
2. **Sorting a diagonal → O(D log D)**
3. **Writing back → O(D)**

Since there are **(n + m - 1) diagonals**, total complexity:

```mathematica
O((n + m) * D log D)
```

🔥 Final Summary

1️⃣ **First loop** covers **diagonals from the first row.**
2️⃣ **Second loop** covers **diagonals from the first column** (except `(0,0)`).
3️⃣ **Sort function:**

- **Extracts** diagonal elements
- **Sorts** them
- **Writes them back** into the matrix

✅ **Every diagonal is sorted independently.**
✅ **Final result maintains diagonal order but sorts the elements.**

🎯 Would You Like a Step-by-Step Visual Diagram?

I can illustrate it for better clarity! 🚀