**Julio Pochet Edmead  
06/05/2025  
Module 2 Discussion Board**

**Binary Search & The Call Stack: Breaking Down Recursion**

When I first learned recursion, it felt confusing, like a method calling itself just seemed weird. But once I saw it used in something like **binary search**, it started to click.

Binary search is used to find a specific value in a **sorted array**. Instead of going through every item, it checks the middle one first. If that’s not it, the method calls itself again by either looking in the left or right half depending on the value. Here’s a basic version:

public static int binarySearch(int[] arr, int target, int low, int high) {

if (low > high) return -1;

int mid = (low + high) / 2;

if (arr[mid] == target) return mid;

else if (arr[mid] > target)

return binarySearch(arr, target, low, mid - 1);

else

return binarySearch(arr, target, mid + 1, high);

}

Now, here’s how recursion works under the hood: every time this method calls itself, that new call gets pushed onto the **call stack**. The call stack keeps track of all the active method calls and their variables. It’s like stacking plates. Once the base case is hit (target found or not found), Java starts popping calls off the stack in reverse order and returns the result up the chain.

Understanding this made recursion way easier for me. Without a base case, the stack just keeps growing and crashes with a **stack overflow**. So yeah, binary search is a clean way to practice recursion, and learning how the call stack handles it helped me debug and understand it better.