DSA Remove Element Explanation

Give DSA notes for this along with refined code Give DSA notes for this program.

```
public int
removeElement(int[] nums, int val) {
int i=0,j=0;
for(;j<nums.length;j++){
if(nums[j]!=val){
nums[i]=nums[j];
i++;
}
}
return i;
}
```

Here's a refined version of your code along with DSA notes explaining the approach.

DSA Notes for `removeElement` Problem

Problem Statement

Given an array `nums` and a value `val`, remove all occurrences of `val` in-place and return the new length of the array. The relative order of the elements may be changed.

Approach: Two Pointers

- We use two pointers (`i` and `j`) to process the array efficiently.
- `j` iterates through the array.
- `i` tracks the position where the next valid (non-`val`) element should be placed.

Algorithm

1. Initialize `i = 0` (this keeps track of the index for valid elements).
2. Traverse the array using `j` from `0` to `nums.length - 1`.
3. If `nums[j]` is **not** equal to `val`, copy `nums[j]` to `nums[i]` and increment `i`.
4. Continue until `j` reaches the end of the array.
5. Return `i`, which represents the count of elements remaining after removal.

Time & Space Complexity

- **Time Complexity:** $O(N)$ (Single pass through the array)
- **Space Complexity:** $O(1)$ (In-place modification, no extra space used)

Refined Code

```java
public int removeElement(int[] nums, int val) {
    int i = 0; // Pointer for valid elements
    for (int j = 0; j < nums.length; j++) {
        if (nums[j] != val) {
            nums[i++] = nums[j]; // Copy only valid elements
        }
    }
    return i; // New length of the modified array
}
```

Key Improvements

- Removed unnecessary initialization of `j` outside the loop.
- Used `i++` inside the assignment to make it more concise.

Let me know if you need further clarifications! 🚀