

Remove Duplicates Example



```
public class RemoveDuplicates {
    public static int removeDuplicates(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }

        // Pointer to the position of the last unique element
        int uniqueIndex = 0;

        for (int i = 1; i < nums.length; i++) {
            // If the current element is not equal to the last unique element
            if (nums[i] != nums[uniqueIndex]) {
                uniqueIndex++;
                nums[uniqueIndex] = nums[i];
            }
        }

        // Return the count of unique elements
        return uniqueIndex + 1;
    }

    public static void main(String[] args) {
        int[] nums = {0, 0, 1, 1, 2, 2, 3, 4, 4, 5};
        int uniqueCount = removeDuplicates(nums);

        System.out.println("Number of unique elements: " + uniqueCount);
        System.out.print("Modified array: ");
        for (int i = 0; i < uniqueCount; i++) {
            System.out.print(nums[i] + " ");
        }
    }
}

}. Dru run tis for {1,2,2,3}
```



Here is the output when you run the program for the input `{1, 2, 2, 3}`:

```
java

int[] nums = {1, 2, 2, 3};
int uniqueCount = removeDuplicates(nums);
```

Output:

```
sql
```

```
Number of unique elements: 3
Modified array: 1 2 3
```

Explanation:

- The input array `{1, 2, 2, 3}` has duplicates.
- The algorithm moves unique elements to the start of the array.
- After processing, the array becomes `{1, 2, 3, ...}` where the elements beyond index `uniqueCount - 1` are irrelevant.
- `uniqueCount = 3` because there are three unique elements.



dry run



Let's do a **dry run** of the code step by step for the input `{1, 2, 2, 3}`:

Initial setup:

- `nums = {1, 2, 2, 3}`
- `uniqueIndex = 0` (points to the last unique element)
- `i` starts at 1 (to compare with the element at `uniqueIndex`)

Step-by-step iterations:

Iteration 1: (`i = 1`)

- Compare `nums[i]` (2) with `nums[uniqueIndex]` (1):
 - `nums[1] != nums[0]`, so a new unique element is found.
 - Increment `uniqueIndex` to `1`.
 - Update `nums[uniqueIndex]` with `nums[i]` (no change, as it's already 2).
- `nums = {1, 2, 2, 3}`, `uniqueIndex = 1`.

Iteration 2: (`i = 2`)

- Compare `nums[i]` (2) with `nums[uniqueIndex]` (2):
 - `nums[2] == nums[1]`, so it's a duplicate.
 - Do **not** increment `uniqueIndex`.
- `nums = {1, 2, 2, 3}`, `uniqueIndex = 1`.

Iteration 3: (`i = 3`)

- Compare `nums[i]` (3) with `nums[uniqueIndex]` (2):
 - `nums[3] != nums[1]`, so a new unique element is found.
 - Increment `uniqueIndex` to `2`.
 - Update `nums[uniqueIndex]` with `nums[i]` (set `nums[2] = 3`).
 - `nums = {1, 2, 3, 3}`, `uniqueIndex = 2`.
-

Final state:

- `uniqueIndex = 2`, meaning there are `uniqueIndex + 1 = 3` unique elements.
 - Modified `nums = {1, 2, 3, ...}` (ignore elements beyond index 2).
-

Output:

```
sql
```

```
Number of unique elements: 3  
Modified array: 1 2 3
```