Java Day10

🔃 3. Sorting Techniques

🔷 Bubble Sort — O(n²)

Repeatedly swap adjacent elements if they're in the wrong order.

```
package DSA.Day10;

import java.util.Arrays;

public class BubbleSort {
    public static void main(String[] args) {
        int[] nums={9,6,8,4,2};
        int n=nums.length;
        for(int i=0;i<n-1;i++){
            boolean isSwapped=false;
            for(int j=0;j<n-1-i;j++){
                if(nums[j]>nums[j+1]){
                    int temp=nums[j];
                    nums[j]=nums[j+1];
                    nums[j+1]=temp;
                    isSwapped=true;
                }
            }
            if(!isSwapped){
                break;
            }
        }
        System.out.println(Arrays.toString(nums));
    }
}
```

🔷 Selection Sort — O(n²)

Find the minimum element and place it at the beginning.

```
package DSA.Day10;

import java.util.Arrays;

public class SelectionSort {
    public static void main(String[] args) {
        int[] nums={8,5,7,3,2};
        int n=nums.length;
        for(int i=0;i<n-1;i++){
            int min=i;
            for(int j=i+1;j<n;j++){
                if(nums[j]<nums[min]){
                    min=j;
                }
            }
            int temp=nums[min];
            nums[min]=nums[i];
            nums[i]=temp;
        }
        System.out.println(Arrays.toString(nums));
    }

}
```

🔷 Insertion Sort — O(n^2)

Build the sorted array one element at a time.

```
package DSA.Day10;

import java.util.Arrays;

public class InsertionSort {
    public static void main(String[] args) {
        int[] nums={8,5,7,3,2};
        int n=nums.length;
        for(int i=1;i<n;i++){
            int j=i-1;
            int temp=nums[i];
            while(j>-1 && nums[j]>temp){
                nums[j+1]=nums[j];
                j--;
            }
            nums[j+1]=temp;
        }
        System.out.println(Arrays.toString(nums));
    }
}
```

🔷 Merge Sort — O(n log n)

Divide the array, sort each half, and merge.

```
package DSA.Day10;

import java.util.Arrays;

public class MergeSortExample {
    public static void merge(int[] arr, int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;

        // Create temporary arrays
        int[] L = new int[n1];
        int[] R = new int[n2];

        // Copy data to temp arrays
        for (int i = 0; i < n1; i++)
            L[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
            R[j] = arr[mid + 1 + j];

        // Merge the temp arrays
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
```

```
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Recursive merge sort function
public static void mergeSort(int[] arr, int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        // Sort first and second halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the sorted halves
        merge(arr, left, mid, right);
    }
}

// Main method to test the program
public static void main(String[] args) {
        int[] marks = {45, 78, 12, 89, 34, 67};

        System.out.println("Before sorting:");
        for (int mark : marks) {
            System.out.print(mark + " ");
        }

        mergeSort(marks, 0, marks.length - 1);

        System.out.println(Arrays.toString(marks));
    }
}
```

🔷 Quick Sort — O(n log n) average, O(n²) worst

Divide and conquer using a pivot element.

```
package DSA.Day10;

import java.util.Arrays;

public class QuickSort {
    public static void main(String[] args) {
        int[] nums={10,16,8,12,15,6,3,9,5,75};
        quickSort(nums,0,nums.length-1);
        System.out.println(Arrays.toString(nums));
    }
    public static void quickSort(int[] nums,int start,int end){
        if(start<end){
            int j=partition(nums,start,end);
//            System.out.println(j);
            quickSort(nums,start,j-1);
            quickSort(nums,j+1,end);
//            System.out.println(Arrays.toString(nums));
        }
    }
    public static int partition(int[] nums,int start,int end){
        int pivot=nums[start];
        int i=start;
        int j=end;
        while(i<j){
            while(i<=j && nums[i]<=pivot){
                i++;
            }
            while(i<=j && nums[j]>=pivot){
                j--;
            }
            if(i<j) {
                swap(nums, i, j);
            }
        }
        swap(nums,j,start);
        return j;
    }
    public static void swap(int[] nums,int a,int b){
        int temp=nums[a];
        nums[a]=nums[b];
        nums[b]=temp;
    }
}
```

Leetcode problem solving

283. Move Zeroes

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

Constraints:

- `1 <= nums.length <= 104`
- `231 <= nums[i] <= 231 - 1`

Follow up:

Could you minimize the total number of operations done?

```
class Solution {
    public void moveZeroes(int[] nums) {
        int j=0;
        for(int i=0;i<nums.length;i++){
            if(nums[i]!=0) nums[j++]=nums[i];
        }
        while(j<nums.length){
            nums[j++]=0;
        }
    }
}
```

704. Binary Search

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

Constraints:

- `1 <= nums.length <= 104`
- `104 < nums[i], target < 104`
- All the integers in `nums` are **unique**.
- `nums` is sorted in ascending order.

```
class Solution {
    public int search(int[] nums, int target) {
        int start=0;
        int end=nums.length-1;
        while(start<=end){
            int mid=(start+end)/2;
            if(nums[mid]==target){
                return mid;
            }else if(nums[mid]<target){
                start=mid+1;
            }else{
                end=mid-1;
            }
        }
        return -1;
    }
}
```

26. Remove Duplicates from Sorted Array

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**.

The **relative order** of the elements should be kept the **same**.

Consider the number of *unique elements* in `nums` to be `k`. After removing duplicates, return the number of unique elements `k`.

The first `k` elements of `nums` should contain the unique numbers in **sorted order**. The remaining elements beyond index `k - 1` can be ignored.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

```
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements o
f nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are un
derscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements o
f nums being 0, 1, 2, 3, and 4 respectively.
It does not matter what you leave beyond the returned k (hence they are un
derscores).
```

Constraints:

- `1 <= nums.length <= 3 * 104`
- `100 <= nums[i] <= 100`
- `nums` is sorted in **non-decreasing** order.

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int j=1;
        for(int i=1;i<nums.length;i++) {//1,1,2,3,4,4,5//1 2 3 4 5 6 7
            if(nums[i]==nums[i-1]) {
                continue;
            }
            else {
                nums[j++]=nums[i];//1 2 3 4 5 4 5
            }//2 3 4 5
        }
        return j;

    }
}
```