Assignment 18 Solution - Searching and Sorting| DSA

1. Merge Intervals

Given an array of `intervals` where `intervals[i] = [starti, endi]`, merge all overlapping intervals, and return an array of the non-overlapping intervals that cover all the intervals in the input.

Example 1:
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].

Example 2:
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.

Constraints:
- `1 <= intervals.length <= 10000`
- `intervals[i].length == 2`
- `0 <= starti <= endi <= 10000`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class MergeIntervals_1 {
    public static int[][] merge(int[][] intervals) {
        // Sort the intervals based on the start time
        Arrays.sort(intervals, Comparator.comparingInt(a -> a[0]));

        List<int[]> merged = new ArrayList<>();
        int[] currentInterval = intervals[0];

        for (int i = 1; i < intervals.length; i++) {
            int[] interval = intervals[i];

            if (interval[0] <= currentInterval[1]) {
                // Overlapping intervals, update the end time
                currentInterval[1] = Math.max(currentInterval[1], interval[1]);
            } else {
                // Non-overlapping interval, add the current interval to the result
                merged.add(currentInterval);
                currentInterval = interval;
            }
        }

        // Add the last interval to the result
        merged.add(currentInterval);

        return merged.toArray(new int[merged.size()][]);
    }

    public static void main(String[] args) {
        int[][] intervals = { { 1, 3 }, { 2, 6 }, { 8, 10 }, { 15, 18 } };
        int[][] mergedIntervals = merge(intervals);

        // Print the merged intervals
        for (int[] interval : mergedIntervals) {
            System.out.println(Arrays.toString(interval));
        }
    }
}
```

2. Sort Colors

Given an array `nums` with `n` objects colored red, white, or blue, sort them [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.
You must solve this problem without using the library's sort function.
Example 1:
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]

Example 2:
Input: nums = [2,0,1]
Output: [0,1,2]

Constraints:
- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.

Source Code:

```
package in.ineuron.pptAssignment18;

public class SortColors_2 {

    public static void sortColors(int[] nums) {
        int low = 0; // Pointer for the red color (0)
        int mid = 0; // Pointer for the white color (1)
        int high = nums.length - 1; // Pointer for the blue color (2)

        while (mid <= high) {
            if (nums[mid] == 0) {
            // Swap nums[mid] and nums[low] and move both pointers to the right
                swap(nums, low, mid);
                low++;
                mid++;
            } else if (nums[mid] == 1) {
        // Element is already in the correct position, move the mid pointer to the right
                mid++;
            } else {
            // Swap nums[mid] and nums[high] and move the high pointer to the left
                swap(nums, mid, high);
                high--;
            }
        }
    }
```

```
        private static void swap(int[] nums, int i, int j) {
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
        }

        public static void main(String[] args) {
                int[] nums = { 2, 0, 2, 1, 1, 0 };
                sortColors(nums);

                // Print the sorted array
                for (int num : nums) {
                        System.out.print(num + " ");
                }
        }
}
```

3. First Bad Version Solution

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:
Input: n = 5, bad = 4
Output: 4
Explanation:
call isBadVersion(3) -> false
call isBadVersion(5) -> true
call isBadVersion(4) -> true
Then 4 is the first bad version.

Example 2:
Input: n = 1, bad = 1
Output: 1

Constraints:
- `1 <= bad <= n <= 2^31 - 1`

Source Code:

```
package in.ineuron.pptAssignment18;

public class FirstBadVersion_3 {
    private static boolean isBadVersion(int version) {
        // API function that checks if the version is bad
        // Replace this with the actual API function provided
        return version >= 4;
    }

    public static int firstBadVersion(int n) {
        int left = 1;
        int right = n;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (isBadVersion(mid)) {
                right = mid;
                // The bad version is in the left half or is the current mid
            } else {
                left = mid + 1; // The bad version is in the right half
            }
        }

        return left; // The left pointer points to the first bad version
    }

    public static void main(String[] args) {
        int n = 5;
        int firstBad = firstBadVersion(n);
        System.out.println("The first bad version is: " + firstBad);
    }
}
```

4. Maximum Gap

Given an integer array `nums`, return the maximum difference between two successive elements in its sorted form. If the array contains less than two elements, return `0`.

You must write an algorithm that runs in linear time and uses linear extra space.

Example 1:

Input: nums = [3,6,9,1]
Output: 3
Explanation: The sorted form of the array is [1,3,6,9], either (3,6) or (6,9) has the maximum difference 3.

Example 2:
Input: nums = [10]
Output: 0
Explanation: The array contains less than 2 elements, therefore return 0.

Constraint
- `1 <= nums.length <= 10^5`
- `0 <= nums[i] <= 10^9`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.Arrays;

public class MaximumGap_4 {
    public static int maximumGap(int[] nums) {
        int n = nums.length;

        if (n < 2) {
            return 0;
        }

        // Find the maximum element in the array
        int maxNum = Arrays.stream(nums).max().getAsInt();

        int exp = 1; // Current digit position
        int[] sorted = new int[n];

        while (maxNum / exp > 0) {
            int[] count = new int[10];
                // Counting sort array to store the occurrence of each digit

            // Count the occurrences of each digit at the current digit position
            for (int i = 0; i < n; i++) {
                int digit = (nums[i] / exp) % 10;
                count[digit]++;
            }

            // Calculate the cumulative count for each digit
            for (int i = 1; i < 10; i++) {
                count[i] += count[i - 1];
            }

            // Build the sorted array based on the current digit position
            for (int i = n - 1; i >= 0; i--) {
                int digit = (nums[i] / exp) % 10;
                sorted[count[digit] - 1] = nums[i];
                count[digit]--;
            }

            // Copy the sorted array back to the original array
            System.arraycopy(sorted, 0, nums, 0, n);

            // Move to the next digit position
            exp *= 10;
        }

        // Calculate the maximum difference between two successive elements in the
        // sorted array
        int maxDiff = 0;
        for (int i = 1; i < n; i++) {
            int diff = nums[i] - nums[i - 1];
            maxDiff = Math.max(maxDiff, diff);
        }

        return maxDiff;
    }

    public static void main(String[] args) {
        int[] nums = { 3, 6, 9, 1 };
        int maxGap = maximumGap(nums);
        System.out.println("Maximum Gap: " + maxGap);
    }
}
```

5. Contains Duplicate

Given an integer array `nums`, return `true` if any value appears at least twice in the array, and return `false` if every element is distinct.

Example 1:
Input: nums = [1,2,3,1]
Output: true

Example 2:
Input: nums = [1,2,3,4]
Output: false

Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

Constraints:
- `1 <= nums.length <= 10^5`
- `109 <= nums[i] <= 10^9`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.HashSet;
import java.util.Set;

public class ContainsDuplicate_5 {
	public static boolean containsDuplicate(int[] nums) {
		Set<Integer> set = new HashSet<>();

		for (int num : nums) {
			if (set.contains(num)) {
				return true; // Found a duplicate element
			}
			set.add(num);
		}

		return false; // No duplicates found
	}

	public static void main(String[] args) {
		int[] nums = { 1, 2, 3, 1 };
		boolean containsDup = containsDuplicate(nums);
		System.out.println("Contains Duplicate: " + containsDup);
	}
}
```

6. Minimum Number of Arrows to Burst Balloons

There are some spherical balloons taped onto a flat wall that represents the XY-plane. The balloons are represented as a 2D integer array `points` where `points[i] = [xstart, xend]` denotes a balloon whose horizontal diameter stretches between `xstart` and `xend`. You do not know the exact y-coordinates of the balloons.

Arrows can be shot up directly vertically (in the positive y-direction) from different points along the x-axis. A balloon with `xstart` and `xend` is burst by an arrow shot at `x` if `xstart <= x <= xend`. There is no limit to the number of arrows that can be shot. A shot arrow keeps traveling up infinitely, bursting any balloons in its path.

Given the array `points`, return the minimum number of arrows that must be shot to burst all balloons.

Example 1:
Input: points = [[10,16],[2,8],[1,6],[7,12]]
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at x = 6, bursting the balloons [2,8] and [1,6].
- Shoot an arrow at x = 11, bursting the balloons [10,16] and [7,12].

Example 2:
Input: points = [[1,2],[3,4],[5,6],[7,8]]
Output: 4
Explanation: One arrow needs to be shot for each balloon for a total of 4 arrows.

Example 3:
Input: points = [[1,2],[2,3],[3,4],[4,5]]
Output: 2
Explanation: The balloons can be burst by 2 arrows:
- Shoot an arrow at x = 2, bursting the balloons [1,2] and [2,3].
- Shoot an arrow at x = 4, bursting the balloons [3,4] and [4,5].
Constraints:
- `1 <= points.length <= 10^5`
- `points[i].length == 2`
- `231 <= xstart < xend <= 2^31 - 1`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.Arrays;
import java.util.Comparator;

public class MinimumArrowsToBurstBalloons_6 {
        public static int findMinArrowShots(int[][] points) {
                if (points.length == 0) {
                        return 0;
                }

                // Sort the balloons based on their end coordinates in ascending order
                Arrays.sort(points, Comparator.comparingInt(a -> a[1]));

                int arrows = 1; // At least one arrow is needed
                int end = points[0][1]; // End coordinate of the first balloon

                // Iterate through the remaining balloons
                for (int i = 1; i < points.length; i++) {
                // If the current balloon's start coordinate is after the previous balloon's end
//coordinate, we need to shoot another arrow since the balloons are not overlapping
//anymore.
                        if (points[i][0] > end) {
                            arrows++;
                            end = points[i][1];
                        // Update the end coordinate to the current balloon's end coordinate
                        }
                }

                return arrows;
        }

        public static void main(String[] args) {
                int[][] points = { { 10, 16 }, { 2, 8 }, { 1, 6 }, { 7, 12 } };
                int minArrows = findMinArrowShots(points);
                System.out.println("Minimum Arrows: " + minArrows);
        }
}
```

7. Longest Increasing Subsequence

Given an integer array `nums`, return the length of the longest strictly increasing

subsequence
Example 1:
Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

Example 2:
Input: nums = [0,1,0,3,2,3]
Output: 4

Example 3:
Input: nums = [7,7,7,7,7,7,7]
Output: 1

Constraints:
- `1 <= nums.length <= 2500`
- `-10^4 <= nums[i] <= 10^4`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.Arrays;

public class LongestIncreasingSubsequence_7 {
    public static int lengthOfLIS(int[] nums) {
        int n = nums.length;
        int[] dp = new int[n];
        // dp[i] represents the length of the longest increasing subsequence ending at
        // index i

        Arrays.fill(dp, 1);
        // Initialize dp array with 1 since each element is a valid subsequence of
        // length 1

        int maxLen = 1; // Maximum length of the increasing subsequence

        for (int i = 1; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
// If the current number is greater than the previous number, we can extend the subsequence

                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
```

```
                maxLen = Math.max(maxLen, dp[i]);
            }

            return maxLen;
        }

        public static void main(String[] args) {
            int[] nums = { 10, 9, 2, 5, 3, 7, 101, 18 };
            int maxLength = lengthOfLIS(nums);
            System.out.println("Length of Longest Increasing Subsequence: " + maxLength);
        }
    }
```

8. 132 Pattern

Given an array of `n` integers `nums`, a 132 pattern is a subsequence of three integers `nums[i]`, `nums[j]` and `nums[k]` such that `i < j < k` and `nums[i] < nums[k] < nums[j]`.

Return `true` if there is a 132 pattern in `nums`, otherwise, return `false`.

Example 1:
Input: nums = [1,2,3,4]
Output: false
Explanation: There is no 132 pattern in the sequence.

Example 2:
Input: nums = [3,1,4,2]
Output: true
Explanation: There is a 132 pattern in the sequence: [1, 4, 2].

Example 3:
Input: nums = [-1,3,2,0]
Output: true
Explanation: There are three 132 patterns in the sequence: [-1, 3, 2], [-1, 3, 0] and [-1, 2, 0].

Constraints:
- `n == nums.length`
- `1 <= n <= 2  10^5`
- `-10^9 <= nums[i] <= 10^9`

Source Code:

```
package in.ineuron.pptAssignment18;

import java.util.Stack;

public class Pattern132_8 {

        public static boolean find132pattern(int[] nums) {
                int n = nums.length;
                Stack<Integer> stack = new Stack<>();
                int numK = Integer.MIN_VALUE; // Initialize numK to the minimum value

                for (int i = n - 1; i >= 0; i--) {
                        // Check if the current element is greater than numK
                        if (nums[i] < numK) {
                                return true; // Found a 132 pattern
                        }

        // Keep updating numK by popping elements from the stack that are less than the
```

```
                    // current element
                    while (!stack.isEmpty() && nums[i] > stack.peek()) {
                        numK = stack.pop();
                    }

                    // Push the current element to the stack
                    stack.push(nums[i]);
                }

                return false; // No 132 pattern found
            }

            public static void main(String[] args) {
                int[] nums = { 1, 2, 3, 4 };
                boolean has132Pattern = find132pattern(nums);
                System.out.println("Has 132 Pattern: " + has132Pattern);
            }
        }
```

GitHub Link : https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-18