Assignment 19 Solution - Searching and Sorting| DSA

1. Merge k Sorted Lists

You are given an array of `k` linked-lists `lists`, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.
Example 1:
Input: lists = [[1,4,5],[1,3,4],[2,6]]
Output: [1,1,2,3,4,4,5,6]
Explanation: The linked-lists are:
[
 1->4->5,
 1->3->4,
 2->6
]
merging them into one sorted list: 1->1->2->3->4->4->5->6

Example 2:
Input: lists = []
Output: []

Example 3:
Input: lists = [[]]
Output: []

Constraints:
- `k == lists.length`
- `0 <= k <= 10000`
- `0 <= lists[i].length <= 500`
- `-10000 <= lists[i][j] <= 10000`
- `lists[i]` is sorted in ascending order.
- The sum of `lists[i].length` will not exceed `10000`.

Source Code:

```
package in.ineuron.pptAssignment19;

import java.util.PriorityQueue;

class ListNode {
	int val;
	ListNode next;

	ListNode(int val) {
		this.val = val;
	}
}
```

```
public class MergekSortedLists_1 {
    public static void main(String[] args) {
        ListNode[] lists = new ListNode[3];
        lists[0] = createList(new int[] { 1, 4, 5 });
        lists[1] = createList(new int[] { 1, 3, 4 });
        lists[2] = createList(new int[] { 2, 6 });

        ListNode mergedList = mergeKLists(lists);
        printList(mergedList);
        // Output: 1 -> 1 -> 2 -> 3 -> 4 -> 4 -> 5 -> 6
    }

    public static ListNode mergeKLists(ListNode[] lists) {
        // Create a min-heap to store the nodes of the linked lists
        PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);

        // Add the head nodes of all linked lists to the min-heap
        for (ListNode list : lists) {
            if (list != null) {
                minHeap.offer(list);
            }
        }

        // Create a dummy node to build the merged list
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;

        // Process the nodes in the min-heap until it becomes empty
        while (!minHeap.isEmpty()) {
            ListNode node = minHeap.poll();
                // Extract the node with the minimum value
            curr.next = node; // Add the node to the merged list
            curr = curr.next; // Move the current pointer

            // If the extracted node has a next node, add it to the min-heap
            if (node.next != null) {
                minHeap.offer(node.next);
            }
        }

        return dummy.next; // Return the head of the merged list
    }

    // Utility method to create a linked list from an array
    public static ListNode createList(int[] nums) {
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;
```

```
            for (int num : nums) {
                curr.next = new ListNode(num);
                curr = curr.next;
            }
            return dummy.next;
        }

        // Utility method to print the elements of a linked list
        public static void printList(ListNode head) {
            ListNode curr = head;
            while (curr != null) {
                System.out.print(curr.val + " -> ");
                curr = curr.next;
            }
            System.out.println("null");
        }
    }
```

2. **Count of Smaller Numbers After Self**

Given an integer array `nums`, return an integer array `counts` where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example 1:
Input: nums = [5,2,6,1]
Output: [2,1,1,0]
Explanation:
To the right of 5 there are2 smaller elements (2 and 1).
To the right of 2 there is only1 smaller element (1).
To the right of 6 there is1 smaller element (1).
To the right of 1 there is0 smaller element.

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

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

Constraints:
- `1 <= nums.length <= 100000`
- `-10000 <= nums[i] <= 10000`

Source Code:

```
package in.ineuron.pptAssignment19;
import java.util.ArrayList;
import java.util.List;

public class CountSmallerNumbersAfterSelf_2 {
	public static void main(String[] args) {
		int[] nums = { 5, 2, 6, 1 };
		int[] counts = countSmaller(nums);
		for (int count : counts) {
			System.out.print(count + " ");
		}
		// Output: 2 1 1 0
	}

	public static int[] countSmaller(int[] nums) {
		int n = nums.length;
		int[] counts = new int[n];
		List<Integer> sorted = new ArrayList<>();

		for (int i = n - 1; i >= 0; i--) {
			int index = findIndex(sorted, nums[i]);
```

```
                counts[i] = index;
                sorted.add(index, nums[i]);
            }

            return counts;
        }

        private static int findIndex(List<Integer> sorted, int target) {
            int left = 0;
            int right = sorted.size() - 1;

            while (left <= right) {
                int mid = left + (right - left) / 2;
                if (sorted.get(mid) < target) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }

            return left;
        }
    }
```

3. Sort an Array

Given an array of integers `nums`, sort the array in ascending order and return it.

You must solve the problem without using any built-in functions in `O(nlog(n))` time complexity and with the smallest space complexity possible.

Example 1:
Input: nums = [5,2,3,1]
Output: [1,2,3,5]
Explanation: After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:
Input: nums = [5,1,1,2,0,0]
Output: [0,0,1,1,2,5]
Explanation: Note that the values of nums are not necessarily unique.

Constraints:
- `1 <= nums.length <= 5  10000`
- `-5  104 <= nums[i] <= 5  10000`

Source Code:

```
package in.ineuron.pptAssignment19;

public class SortByArray_3 {
	public static void main(String[] args) {
		int[] nums = { 5, 1, 1, 2, 0, 0 };
		sortArray(nums);
		for (int num : nums) {
			System.out.print(num + " ");
		}
		// Output: 1 2 3 5
	}

	public static void sortArray(int[] nums) {
		mergeSort(nums, 0, nums.length - 1);
	}

	private static void mergeSort(int[] nums, int left, int right) {
		if (left >= right) {
			return;
		}

		int mid = left + (right - left) / 2;
		mergeSort(nums, left, mid);
		mergeSort(nums, mid + 1, right);
		merge(nums, left, mid, right);
```

```
        }

        private static void merge(int[] nums, int left, int mid, int right) {
                int[] temp = new int[right - left + 1];
                int i = left;
                int j = mid + 1;
                int k = 0;

                while (i <= mid && j <= right) {
                        if (nums[i] <= nums[j]) {
                                temp[k++] = nums[i++];
                        } else {
                                temp[k++] = nums[j++];
                        }
                }

                while (i <= mid) {
                        temp[k++] = nums[i++];
                }

                while (j <= right) {
                        temp[k++] = nums[j++];
                }

                for (i = left; i <= right; i++) {
                        nums[i] = temp[i - left];
                }
        }
}
```

4. Move all zeroes to end of array

Given an array of random numbers, Push all the zero's of a given array to the end of the array. For example, if the given arrays is {1, 9, 8, 4, 0, 0, 2, 7, 0, 6, 0}, it should be changed to {1, 9, 8, 4, 2, 7, 6, 0, 0, 0, 0}. The order of all other elements should be same. Expected time complexity is O(n) and extra space is O(1).

Example:
Input : arr[] = {1, 2, 0, 4, 3, 0, 5, 0};
Output : arr[] = {1, 2, 4, 3, 5, 0, 0, 0};

Input : arr[] = {1, 2, 0, 0, 0, 3, 6};
Output : arr[] = {1, 2, 3, 6, 0, 0, 0};

Source Code:

```
package in.ineuron.pptAssignment19;

public class MoveAllZeroesToEndOfArray_4 {
	public static void main(String[] args) {
		int[] arr = { 1, 2, 0, 4, 3, 0, 5, 0 };
		moveZeroes(arr);
		for (int num : arr) {
			System.out.print(num + " ");
		}
		// Output: 1 2 4 3 5 0 0 0
	}

	public static void moveZeroes(int[] arr) {
		int n = arr.length;
		int count = 0; // Count of non-zero elements

		// Traverse the array and move non-zero elements to the beginning
		for (int i = 0; i < n; i++) {
			if (arr[i] != 0) {
				arr[count++] = arr[i];
			}
		}

		// Fill the remaining elements with zeroes
		while (count < n) {
			arr[count++] = 0;
		}
	}
}
```

5. Rearrange array in alternating positive & negative items with O(1) extra space

Given an array of positive and negative numbers, arrange them in an alternate fashion such that every positive number is followed by a negative and vice-versa maintaining the order of appearance. The number of positive and negative numbers need not be equal. If there are more positive numbers they appear at the end of the array. If there are more negative numbers, they too appear at the end of the array.

Examples:
Input: arr[] = {1, 2, 3, -4, -1, 4}
Output: arr[] = {-4, 1, -1, 2, 3, 4}

Input: arr[] = {-5, -2, 5, 2, 4, 7, 1, 8, 0, -8}
Output: arr[] = {-5, 5, -2, 2, -8, 4, 7, 1, 8, 0}

Source Code:

```
package in.ineuron.pptAssignment19;
public class RearrangeArray_5 {
	public static void main(String[] args) {
		int[] arr = { 1, 2, 3, -4, -1, 4 };
		rearrangeArray(arr);
		for (int num : arr) {
			System.out.print(num + " ");
		}
		// Output: -4 1 -1 2 3 4
	}
	public static void rearrangeArray(int[] arr) {
		int n = arr.length;

		// Find the index of the first positive number
		int posIndex = 0;
		while (posIndex < n && arr[posIndex] < 0) {
			posIndex++;
		}
		// Rearrange the array using the two-pointer approach
		int negIndex = 1;
		while (posIndex < n && negIndex < n) {
			// Swap the positive number with the negative number
			int temp = arr[posIndex];
			arr[posIndex] = arr[negIndex];
			arr[negIndex] = temp;

			// Move the pointers to the next positive and negative numbers
			posIndex += 2;
			negIndex += 2;
		}
	}
}
```

6. Merge two sorted arrays

Given two sorted arrays, the task is to merge them in a sorted manner.
Examples:

Input: arr1[] = { 1, 3, 4, 5}, arr2[] = {2, 4, 6, 8}
Output: arr3[] = {1, 2, 3, 4, 4, 5, 6, 8}

Input: arr1[] = { 5, 8, 9}, arr2[] = {4, 7, 8}
Output: arr3[] = {4, 5, 7, 8, 8, 9}

Source Code:

```
package in.ineuron.pptAssignment19;

public class MergeTwoSortedArrays_6 {
	public static void main(String[] args) {
		int[] arr1 = { 1, 3, 4, 5 };
		int[] arr2 = { 2, 4, 6, 8 };
		int[] mergedArray = mergeArrays(arr1, arr2);
		for (int num : mergedArray) {
			System.out.print(num + " ");
		}
		// Output: 1 2 3 4 4 5 6 8
	}

	public static int[] mergeArrays(int[] arr1, int[] arr2) {
		int n1 = arr1.length;
		int n2 = arr2.length;
		int[] mergedArray = new int[n1 + n2];
		int i = 0; // Index for arr1
		int j = 0; // Index for arr2
		int k = 0; // Index for mergedArray

		// Compare elements from both arrays and merge them in sorted order
		while (i < n1 && j < n2) {
			if (arr1[i] <= arr2[j]) {
				mergedArray[k++] = arr1[i++];
			} else {
				mergedArray[k++] = arr2[j++];
			}
		}

		// Copy any remaining elements from arr1
		while (i < n1) {
			mergedArray[k++] = arr1[i++];
		}

		// Copy any remaining elements from arr2
```

```
            while (j < n2) {
                mergedArray[k++] = arr2[j++];
            }

            return mergedArray;
        }
    }
```

7. Intersection of Two Arrays

Given two integer arrays `nums1` and `nums2`, return an array of their intersection. Each element in the result must be unique and you may return the result in any order.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [9,4]
Explanation: [4,9] is also accepted.

Constraints:
- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

Source Code:

```
package in.ineuron.pptAssignment19;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class IntersectionOfTwoArrays_6 {
	public static void main(String[] args) {
		int[] nums1 = { 1, 2, 2, 1 };
		int[] nums2 = { 2, 2 };
		int[] intersection = findIntersection(nums1, nums2);
		for (int num : intersection) {
			System.out.print(num + " ");
		}
		// Output: 2
	}

	public static int[] findIntersection(int[] nums1, int[] nums2) {
		Set<Integer> set1 = new HashSet<>();
		Set<Integer> set2 = new HashSet<>();

		// Add elements from nums1 to set1
		for (int num : nums1) {
			set1.add(num);
		}

		// Add elements from nums2 to set2
		for (int num : nums2) {
```

```
                set2.add(num);
            }

            // Find the intersection of set1 and set2
            List<Integer> intersection = new ArrayList<>();
            for (int num : set1) {
                if (set2.contains(num)) {
                    intersection.add(num);
                }
            }

            // Convert the List<Integer> to int[]
            int[] result = new int[intersection.size()];
            for (int i = 0; i < intersection.size(); i++) {
                result[i] = intersection.get(i);
            }

            return result;
        }
    }
```

8. Intersection of Two Arrays II

Given two integer arrays `nums1` and `nums2`, return an array of their intersection. Each element in the result must appear as many times as it shows in both arrays and you may return the result in any order.

Example 1:
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]

Example 2:
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Explanation: [9,4] is also accepted.

Constraints:
- `1 <= nums1.length, nums2.length <= 1000`
- `0 <= nums1[i], nums2[i] <= 1000`

Source Code:

```
package in.ineuron.pptAssignment19;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class IntersectionOfTwoArraysII_8 {
	public static void main(String[] args) {
		int[] nums1 = { 1, 2, 2, 1 };
		int[] nums2 = { 2, 2 };
		int[] intersection = findIntersection(nums1, nums2);
		for (int num : intersection) {
			System.out.print(num + " ");
		}
		// Output: 2 2
	}

	public static int[] findIntersection(int[] nums1, int[] nums2) {
		Map<Integer, Integer> countMap1 = new HashMap<>();
		Map<Integer, Integer> countMap2 = new HashMap<>();

		// Count the occurrences of each element in nums1
		for (int num : nums1) {
			countMap1.put(num, countMap1.getOrDefault(num, 0) + 1);
		}

		// Count the occurrences of each element in nums2
```

```
        for (int num : nums2) {
            countMap2.put(num, countMap2.getOrDefault(num, 0) + 1);
        }

        // Find the intersection by comparing the counts in both maps
        List<Integer> intersection = new ArrayList<>();
        for (int num : countMap1.keySet()) {
            if (countMap2.containsKey(num)) {
                int count = Math.min(countMap1.get(num), countMap2.get(num));
                for (int i = 0; i < count; i++) {
                    intersection.add(num);
                }
            }
        }

        // Convert the List<Integer> to int[]
        int[] result = new int[intersection.size()];
        for (int i = 0; i < intersection.size(); i++) {
            result[i] = intersection.get(i);
        }

        return result;
    }
}
```


GitHub Link : https://github.com/devavratwadekar/ineuron_ppt_ProgramAssignmentCode/tree/master/PPTAssignment-19