Assignment Questions 19

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`.

code:-

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if(lists.length == 0){
            return null;
        }
        if(lists.length == 1){
            return lists[0];
        }

        ListNode mergedList = helperMerge(lists[0], lists[1]);

        for(int i = 2; i < lists.length; i++){
            mergedList = helperMerge(mergedList, lists[i]);
        }
        return mergedList;
    }

    public ListNode helperMerge(ListNode list1, ListNode list2){
        ListNode dummyNode = new ListNode();
        ListNode tail = dummyNode;

        while(list1 != null && list2 != null){
            if(list1.val < list2.val){
                tail.next = list1;
                list1 = list1.next;
                tail = tail.next;
            }else{
                tail.next = list2;
                list2 = list2.next;
                tail = tail.next;
            }
        }

        tail.next = (list1 != null) ? list1 : list2;

        return dummyNode.next;
    }
}
```

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`

code:-
```
public class Solution {
      public List<Integer> countSmaller(int[] nums) {
            List<Integer> res = new ArrayList<>();
            if(nums == null || nums.length == 0) return res;
            TreeNode root = new TreeNode(nums[nums.length - 1]);
            res.add(0);
            for(int i = nums.length - 2; i >= 0; i--) {
                  int count = insertNode(root, nums[i]);
                  res.add(count);
            }
            Collections.reverse(res);
            return res;
      }

      public int insertNode(TreeNode root, int val) {
            int thisCount = 0;
            while(true) {
                  if(val <= root.val) {
                        root.count++;
                        if(root.left == null) {
                              root.left = new TreeNode(val); break;
                        } else {
                              root = root.left;
                        }
                  } else {
                        thisCount += root.count;
                        if(root.right == null) {
                              root.right = new TreeNode(val); break;
                        } else {
                              root = root.right;
                        }
                  }
            }
            return thisCount;
      }
}

class TreeNode {
      TreeNode left;
      TreeNode right;
      int val;
      int count = 1;
      public TreeNode(int val) {
            this.val = val;
      }
}
```

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

code:-

```
class Solution {
    public int[] sortArray(int[] nums) {
        mergeSort(nums,0,nums.length-1);
        return nums;
    }
    public  void mergeSort(int[] nums,int start,int end){
        if(start<end){
            int mid=start+(end-start)/2;
            mergeSort(nums,start,mid);
            mergeSort(nums,mid+1,end);
            merge(nums,start,mid,end);
        }
    }
    public void merge(int[] nums,int start,int mid,int end){
        int[] arr1=Arrays.copyOfRange(nums,start,mid+1);
        int[] arr2=Arrays.copyOfRange(nums,mid+1,end+1);
        int i=0,j=0;
        int index=start;
        while(i<arr1.length && j<arr2.length){
            if(arr1[i]<=arr2[j]){
                nums[index]=arr1[i];
                i++;
            }else{
                nums[index]=arr2[j];
                j++;
            }
            index++;
        }
        while(i<arr1.length){
            nums[index]=arr1[i];
            i++;index++;
        }
        while(j<arr2.length){
            nums[index]=arr2[j];
            j++;index++;
        }
    }
}
```

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};
```

code:-

```
class Solution {
    public void moveZeroes(int[] arr) {
    int n = arr.length;
    if(n<=1) return;
    int s=0;
    int e=1;
    while(e<n){
        if(arr[s]==0 && arr[e]!=0){
            int temp = arr[s];
            arr[s] = arr[e];
            arr[e] = temp;
            s++;
            e++;
        }else if(arr[s]==0 && arr[e]==0){
            e++;
        }else{
            s++;
            e++;
        }
    }
    }
}
```

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}
```

code:-

```
class Solution {
    public int[] rearrangeArray(int[] nums) {
        int[] res= new int[nums.length];
        int[] p= new int[nums.length/2];
        int[] n= new int[nums.length/2];
        int i,j;
        i=j=0;

         for(int num:nums){
            if(num>0){
                p[i++]=num;

            }else n[j++]=num;
        }
        int k=0;
        for(int l=0;l<nums.length;l+=2){
            res[l]=p[k];
            res[l+1]=n[k];
            k++;
        }
        return res;
    }
}
```

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}

code:-
```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        //variables to work as pointers
        int i=m-1; // will point at m-1 index of nums1 array
        int j=n-1; // will point at n-1 index of nums2 array
        int k=nums1.length-1; //will point at the last position of the nums1 array

        // Now traversing the nums2 array
        while(j>=0){
            // If element at i index of nums1 > element at j index of nums2
            // then it is largest among two arrays and will be stored at k position
of nums1
            // using i>=0 to make sure we have elements to compare in nums1 array
            if(i>=0 && nums1[i]>nums2[j]){
                nums1[k]=nums1[i];
                k--;
                i--; //updating pointer for further comparisons
            }else{
                // element at j index of nums2 array is greater than the element at
i index of nums1 array
                // or there is no element left to compare with the nums1 array
                // and we just have to push the elements of nums2 array in the
nums1 array.
                nums1[k] = nums2[j];
                k--;
                j--; //updating pointer for further comparisons
            }
        }
    }
}
```

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`

code:-
```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {
    Set<Integer> x=new HashSet<>();
      Set<Integer> y=new HashSet<>();
      for(int i=0;i<nums1.length;i++){
          x.add(nums1[i]);
      }
      for(int i=0;i<nums2.length;i++){
          y.add(nums2[i]);
      }
      x.retainAll(y);
      int ans[]=new int[x.size()];
      int i = 0;
        for(Integer n : x) {
            ans[i] = n;
            i++;
        }
     return ans; }
}
```

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`

code:-
```
class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        int l1 = nums1.length;
        int l2 = nums2.length;
        int i = 0, j = 0, k = 0;
        Arrays.sort(nums1);
        Arrays.sort(nums2);
        while( i < l1 && j < l2)
        {
            if(nums1[i] < nums2[j])
            {
                i++;
            }
            else if(nums1[i] > nums2[j])
            {
                j++;
            }
            else
            {
                nums1[k++] = nums1[i++];
                j++;
            }
        }
        return Arrays.copyOfRange(nums1,0,k);
    }
}
```