ASSIGNMENT 19

1. 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.

```
import heapq

class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def mergeKLists(lists):
    min_heap = []
    dummy = ListNode(0)
    curr = dummy

    # Insert the heads of linked lists into the min-heap
    for head in lists:
        if head:
            heapq.heappush(min_heap, (head.val, head))

    while min_heap:
        val, node = heapq.heappop(min_heap)
        curr.next = node
        curr = curr.next

        if node.next:
            heapq.heappush(min_heap, (node.next.val, node.next))

    return dummy.next
```

2. 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]`.

```
def countSmaller(nums):
    counts = [0] * len(nums)

    def mergeSort(nums, left, right):
        if left >= right:
            return

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

    def merge(nums, left, mid, right):
        i, j, k = left, mid + 1, 0
        merged = [0] * (right - left + 1)
        while i <= mid and j <= right:
            if nums[i] <= nums[j]:
                counts[i] += j - (mid + 1)
                merged[k] = nums[i]
                i += 1
            else:
                merged[k] = nums[j]
                j += 1
            k += 1

        while i <= mid:
            counts[i] += j - (mid + 1)
            merged[k] = nums[i]
            i += 1
            k += 1

        while j <= right:
            merged[k] = nums[j]
            j += 1
            k += 1

        nums[left:right + 1] = merged

    mergeSort(nums, 0, len(nums) - 1)
    return counts
```

3. 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.

```
def sortArray(nums):
    def mergeSort(nums, low, high):
        if low >= high:
            return

        mid = (low + high) // 2
        mergeSort(nums, low, mid)
        mergeSort(nums, mid + 1, high)
        merge(nums, low, mid, high)

    def merge(nums, low, mid, high):
        left = nums[low:mid+1]
        right = nums[mid+1:high+1]
        i = j = 0
        k = low
```

```
 while i < len(left) and j < len(right):
         if left[i] <= right[j]:
            nums[k] = left[i]
            i += 1
         else:
            nums[k] = right[j]
            j += 1
         k += 1

      while i < len(left):
         nums[k] = left[i]
         i += 1
         k += 1

      while j < len(right):
         nums[k] = right[j]
         j += 1
         k += 1

   mergeSort(nums, 0, len(nums) - 1)
   return nums
```

5. 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.

```
def alternatePositiveNegative(nums):
   positive = []
   negative = []

   for num in nums:
      if num >= 0:
         positive.append(num)
      else:
         negative.append(num)

   result = []
   p_len = len(positive)
   n_len = len(negative)
   shorter_list = positive if p_len < n_len else negative
   longer_list = negative if p_len < n_len else positive

   for i in range(min(p_len, n_len)):
      result.append(shorter_list[i])
      result.append(longer_list[i])

   result.extend(longer_list[min(p_len, n_len):])
   result.extend(shorter_list[min(p_len, n_len):])

   return result
```

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

```
def mergeSortedArrays(arr1, arr2):
    merged = []
    i = j = 0

    while i < len(arr1) and j < len(arr2):
        if arr1[i] <= arr2[j]:
            merged.append(arr1[i])
            i += 1
        else:
            merged.append(arr2[j])
            j += 1

    while i < len(arr1):
        merged.append(arr1[i])
        i += 1

    while j < len(arr2):
        merged.append(arr2[j])
        j += 1

    return merged
```

7. 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**.

```
def intersection(nums1, nums2):
    nums1.sort()
    nums2.sort()
    p1, p2 = 0, 0
    result = []

    while p1 < len(nums1) and p2 < len(nums2):
        if nums1[p1] == nums2[p2]:
            result.append(nums1[p1])
            p1 += 1
            p2 += 1
        elif nums1[p1] < nums2[p2]:
            p1 += 1
        else:
            p2 += 1

    return result
```

8. 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**.

```
def intersection(nums1, nums2):
    nums1.sort()
    nums2.sort()
    p1, p2, pres = 0, 0, 0
    result = []

    while p1 < len(nums1) and p2 < len(nums2):
        if nums1[p1] == nums2[p2]:
            result.append(nums1[p1])
            p1 += 1
            p2 += 1
        elif nums1[p1] < nums2[p2]:
            p1 += 1
        else:
            p2 += 1

    return result
```