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`.

Soln:

```
import heapq

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

def mergeKLists(lists):
    # Create a min-heap
    min_heap = []
    for i in range(len(lists)):
        if lists[i]:
            heapq.heappush(min_heap, (lists[i].val, i))

    # Create a dummy node and a pointer
    dummy = ListNode()
    curr = dummy

    while min_heap:
        # Pop the smallest element
        val, index = heapq.heappop(min_heap)
        node = lists[index]

        # Append the node to the merged list
        curr.next = node
        curr = curr.next

        # Move to the next node in the list
        lists[index] = lists[index].next
        if lists[index]:
            heapq.heappush(min_heap, (lists[index].val, index))

    return dummy.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`

Soln;

```
def countSmaller(nums):
    def mergeSort(nums, start, end):
        if start >= end:
            return []

        mid = (start + end) // 2
        count = 0

        left = mergeSort(nums, start, mid)
        right = mergeSort(nums, mid + 1, end)

        merged = []
        i, j = 0, 0
```

```
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            nums[start + i + j] = left[i]
            count += j
            i += 1
        else:
            merged.append(right[j])
            nums[start + i + j] = right[j]
            j += 1

    while i < len(left):
        merged.append(left[i])
        nums[start + i + j] = left[i]
        count += j
        i += 1

    while j < len(right):
        merged.append(right[j])
        nums[start + i + j] = right[j]
        j += 1

    return merged

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

---

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`

Soln:

```
def partition(nums, low, high):
    pivot = nums[high]
    i = low - 1

    for j in range(low, high):
        if nums[j] <= pivot:
            i += 1
            nums[i], nums[j] = nums[j], nums[i]

    nums[i + 1], nums[high] = nums[high], nums[i + 1]
    return i + 1


def quickSort(nums, low, high):
    if low < high:
        pivot = partition(nums, low, high)
```

```
        quickSort(nums, low, pivot - 1)
        quickSort(nums, pivot + 1, high)



def sortArray(nums):
    quickSort(nums, 0, len(nums) - 1)
    return nums
```

---

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};

Soln:

```
def moveZeroes(nums):
    left = 0
    right = 0


    while right < len(nums):
```

```
        if nums[right] != 0:
            nums[left], nums[right] = nums[right], nums[left]
            left += 1
        right += 1

    while left < len(nums):
        nums[left] = 0
        left += 1

    return nums
```

---

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}

Soln:

```
def rearrangeArray(nums):
```

```
    left = 0
    right = len(nums) - 1

    while left <= right:
        while left <= right and nums[left] < 0:
            left += 1
        while left <= right and nums[right] > 0:
            right -= 1
        if left <= right:
            nums[left], nums[right] = nums[right], nums[left]
            left += 1
            right -= 1

    positive_start = left
    negative_start = 0

    while positive_start < len(nums) and negative_start < positive_start and nums[negative_start]
< 0:
        nums[negative_start], nums[positive_start] = nums[positive_start], nums[negative_start]
        negative_start += 2
        positive_start += 1

    return nums
```

---

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}

Soln:

```
def mergeSortedArrays(arr1, arr2):
    n1 = len(arr1)
    n2 = len(arr2)
    arr3 = [0] * (n1 + n2)
    i = j = k = 0

    while i < n1 and j < n2:
        if arr1[i] <= arr2[j]:
            arr3[k] = arr1[i]
            i += 1
        else:
            arr3[k] = arr2[j]
            j += 1
        k += 1

    while i < n1:
```

```
        arr3[k] = arr1[i]
        i += 1
        k += 1

    while j < n2:
        arr3[k] = arr2[j]
        j += 1
        k += 1

    return arr3
```

---

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`

Soln:

```
def intersection(nums1, nums2):
    set1 = set(nums1)
    result = set()

    for num in nums2:
        if num in set1:
            result.add(num)
            set1.remove(num)

    return list(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`

Soln:

```
def intersect(nums1, nums2):
    freq = {}
    result = []

    for num in nums1:
        if num in freq:
            freq[num] += 1
        else:
            freq[num] = 1

    for num in nums2:
        if num in freq and freq[num] > 0:
            result.append(num)
            freq[num] -= 1

    return result
```