23. Merge k Sorted Lists

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
import heapq
class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) -> Optional[ListNode]:
        heap = []
        i = 0
        for head in lists:
            while head:
                heapq.heappush(heap,(head.val,i,head))
                head = head.next
                i+=1

        temp = None
        head = None
        while heap:
            node = heapq.heappop(heap)[2]
            if not head:
                temp = node
                head = node
            else:
                temp.next =node
                temp = temp.next
        return head
```

215. Kth Largest Element in an Array

```
import heapq
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        heap = []
        for i in nums:
            heapq.heappush(heap,-i)

        for i in range(k-1):
            heapq.heappop(heap)
        return -heapq.heappop(heap)
```

---

295. Find Median from Data Stream

```
import heapq

class MedianFinder:

    def __init__(self):
        self.small = []
        self.large = []

    def addNum(self, num: int) -> None:
        #which heap to push into
        if not self.small or num <= -self.small[0]:
            #push small (max-heap) as negative
            heapq.heappush(self.small, -num)
        else:
            #push into large (min-heap)
            heapq.heappush(self.large, num)
        #rebalance the heaps
        if len(self.small) > len(self.large) + 1:
            #move max small to large
            val = -heapq.heappop(self.small)
            heapq.heappush(self.large, val)
        elif len(self.large) > len(self.small) + 1:
            #move min large to small
            val = heapq.heappop(self.large)
            heapq.heappush(self.small, -val)

    def findMedian(self) -> float:
        if len(self.small) == len(self.large):
            if not self.small:  # Edge case: no elements
                return 0.0
            return (-self.small[0] + self.large[0]) / 2
        elif len(self.small) > len(self.large):
            #small more element
            return float(-self.small[0])
        else:
            #large element
            return float(self.large[0])
```

347. Top K Frequent Elements

```
from collections import Counter
import heapq
class Solution:
    def topKFrequent(self,nums, k):
        c = Counter(nums)
        return [x for x, _ in heapq.nlargest(k, c.items(), key=lambda x: x[1])]
```

355. Design Twitter

```
from collections import defaultdict, deque

class Twitter:

    def __init__(self):
        self.follows = defaultdict(set)
        self.feed = deque()

    def postTweet(self, userId: int, tweetId: int) -> None:
        self.feed.appendleft((userId, tweetId))

    def getNewsFeed(self, userId: int) -> List[int]:
        return [tweetId for user, tweetId in self.feed if userId == user or user in
self.follows[userId]][:10]

    def follow(self, followerId: int, followeeId: int) -> None:
        self.follows[followerId].add(followeeId)

    def unfollow(self, followerId: int, followeeId: int) -> None:
        self.follows[followerId].discard(followeeId)
```

621. Task Scheduler

```
from collections import deque,defaultdict,Counter
class Solution:
    def leastInterval(self, tasks: List[str], n: int) -> int:
        cnt = Counter(tasks)
        maxHeap = [-num for num in cnt.values()]
        heapq.heapify(maxHeap)
        time = 0
        que = deque()
        while que or maxHeap:
            time+=1
            if maxHeap:
                count  = heapq.heappop(maxHeap)+1
                if count:que.append([count,time+n])
            if que and que[0][1] <= time:
                ele = que.popleft()[0]
                heapq.heappush(maxHeap,ele)
        return time
```

703. Kth Largest Element in a Stream

```
import heapq
class KthLargest:

    def __init__(self, k: int, nums: List[int]):
        self.size = k
        self.min_heap = nums
        heapq.heapify(self.min_heap)


    def add(self, val: int) -> int:
        heapq.heappush(self.min_heap,val)
        while len(self.min_heap) > self.size:
            heapq.heappop(self.min_heap)
        return self.min_heap[0]
```

846. Hand of Straights

```
from collections import Counter
import heapq
from typing import List

class Solution:
    def isNStraightHand(self, hand: List[int], groupSize: int) -> bool:
        if len(hand) % groupSize != 0:
            return False

        count = Counter(hand)
        min_heap = list(count.keys())
        heapq.heapify(min_heap)

        while min_heap:
            first = min_heap[0]
            for i in range(groupSize):
                num = first + i
                if count[num] == 0:
                    return False
                count[num] -= 1
                if count[num] == 0:
                    if num != min_heap[0]:
                        return False  # out-of-order deletion is not allowed
                    heapq.heappop(min_heap)

        return True
```