**23. 合并K个升序链表**

给你一个链表数组，每个链表都已经按升序排列。

请你将所有链表合并到一个升序链表中，返回合并后的链表。

输入：lists = [[1,4,5],[1,3,4],[2,6]]

输出：[1,1,2,3,4,4,5,6]

解释：链表数组如下：

[

1->4->5,

1->3->4,

2->6

]

将它们合并到一个有序链表中得到。

1->1->2->3->4->4->5->6

输入：lists = []

输出：[]

输入：lists = [[]]

输出：[]

|  |
| --- |
| class Solution {  public:  ListNode\* mergeKLists(vector<ListNode\*>& lists) {  auto head = ListNode(0);  auto comp = [](ListNode\* const &a, ListNode\* const &b){return a->val > b->val;};  priority\_queue<ListNode\*, vector<ListNode\*>, decltype(comp)> q(comp);  for (auto &h : lists) if (h != nullptr) q.push(h);  auto p = &head;  while (!q.empty()) {  p->next = q.top();  p = p->next;  q.pop();  if (p->next != nullptr) q.push(p->next);  }  return head.next;  }  };  多路合并的时候初始化思路可以想到堆  把所有数据加入堆里，再pop出来构建新链表。 |

|  |
| --- |
| class Solution {  public ListNode mergeKLists(ListNode[] lists) {  if (lists == null || lists.length == 0) return null;  return merge(lists, 0, lists.length - 1);  }  private ListNode merge(ListNode[] lists, int left, int right) {  if (left == right) return lists[left];  int mid = left + (right - left) / 2;  ListNode l1 = merge(lists, left, mid);  ListNode l2 = merge(lists, mid + 1, right);  return mergeTwoLists(l1, l2);  }  private ListNode mergeTwoLists(ListNode l1, ListNode l2) {  if (l1 == null) return l2;  if (l2 == null) return l1;  if (l1.val < l2.val) {  l1.next = mergeTwoLists(l1.next, l2);  return l1;  } else {  l2.next = mergeTwoLists(l1,l2.next);  return l2;  }  }  } |

**215. 数组中的第K个最大元素**

在未排序的数组中找到第 k 个最大的元素。请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

输入: [3,2,1,5,6,4] 和 k = 2

输出: 5

输入: [3,2,3,1,2,4,5,5,6] 和 k = 4

输出: 4

|  |
| --- |
| class Solution {  public:  int findKthLargest(vector<int>& nums, int k) {  priority\_queue<int, vector<int>, greater<int>> q;  for\_each(nums.begin(), nums.end(), [&q, &k](int v) {  if (q.size() < k) {  q.push(v);  } else if (v > q.top()) {  q.pop();  q.push(v);  } else {  // do nothing  }  });  return q.top();  }  };  维护最大为K的堆(小的数在堆顶)。时间复杂度O(NlogK)。 |

**692. 前K个高频单词**

给一非空的单词列表，返回前 k 个出现次数最多的单词。

返回的答案应该按单词出现频率由高到低排序。如果不同的单词有相同出现频率，按字母顺序排序。

输入: ["i", "love", "leetcode", "i", "love", "coding"], k = 2

输出: ["i", "love"]

解析: "i" 和 "love" 为出现次数最多的两个单词，均为2次。

注意，按字母顺序 "i" 在 "love" 之前。

输入: ["the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is"], k = 4

输出: ["the", "is", "sunny", "day"]

解析: "the", "is", "sunny" 和 "day" 是出现次数最多的四个单词，

出现次数依次为 4, 3, 2 和 1 次。

|  |
| --- |
| class Solution {  public:  struct cmp{  bool operator()(const pair<string,int> &a,const pair<string,int> &b)  {  if(a.second!=b.second)return a.second<b.second;  else return a.first>b.first;  }  };  vector<string> topKFrequent(vector<string>& words, int k) {  map<string,int> mp;  for(string s:words)  mp[s]++;  priority\_queue<pair<string,int>,vector<pair<string,int>>,cmp> pq;  for(auto m:mp)  {  pq.push(m);  }  vector<string> res;  while(k--)  {  res.push\_back(pq.top().first);  pq.pop();  }  return res;  }  };  维护最大K的堆(从小到大排序)。时间复杂度O(NlogK)。 |

**1383. 最大的团队表现值**

公司有编号为 1 到 n 的 n 个工程师，给你两个数组 speed 和 efficiency ，其中 speed[i] 和 efficiency[i] 分别代表第 i 位工程师的速度和效率。请你返回由最多 k 个工程师组成的 最大团队表现值 ，由于答案可能很大，请你返回结果对 10^9 + 7 取余后的结果。

团队表现值 的定义为：一个团队中「所有工程师速度的和」乘以他们「效率值中的最小值」。

输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 2

输出：60

解释：

我们选择工程师 2（speed=10 且 efficiency=4）和工程师 5（speed=5 且 efficiency=7）。他们的团队表现值为 performance = (10 + 5) \* min(4, 7) = 60 。

输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 3

输出：68

解释：

此示例与第一个示例相同，除了 k = 3 。我们可以选择工程师 1 ，工程师 2 和工程师 5 得到最大的团队表现值。表现值为 performance = (2 + 10 + 5) \* min(5, 4, 7) = 68 。

输入：n = 6, speed = [2,10,3,1,5,8], efficiency = [5,4,3,9,7,2], k = 4

输出：72

|  |
| --- |
| class Solution {  public:  using LL = long long;  struct Staff {  int s, e;  bool operator < (const Staff& rhs) const {  return s > rhs.s;  }  };  int maxPerformance(int n, vector<int>& speed, vector<int>& efficiency, int k) {  vector<Staff> v;  priority\_queue<Staff> q;  for (int i = 0; i < n; ++i) {  v.push\_back({speed[i], efficiency[i]});  }  sort(v.begin(), v.end(), [] (const Staff& u, const Staff& v) { return u.e > v.e; });  LL ans = 0, sum = 0;  for (int i = 0; i < n; ++i) {  LL minE = v[i].e;  LL sumS = sum + v[i].s;  ans = max(ans, sumS \* minE);  q.push(v[i]);  sum += v[i].s;  if (q.size() == k) {  sum -= q.top().s;  q.pop();  }  }  return ans % (int(1E9) + 7);  }  }; |

题目要求我们最优化「速度和」和「效率最小值」的乘积。变化的量有两个，一个是「速度」，一个是「效率」，这看起来有些棘手。我们不妨采用「动一个，定一个」的策略——即我们可以枚举效率的最小值 e\_emin，在所有效率大于 e\_

min的工程师中选取不超过k−1 个，让他们的速度和最大。

**为什么是k−1 个而不是 k 个？**

因为最小值 e\_min代表的工程师是必选，加起来一共 kk 个，所以剩下只要选k−1 个。

**如何满足速度和最大？** 因为 speed[i] > 0，所以只需要选效率大于 e\_e

min中速度最大的k−1 个，如果效率大于 e\_emin的元素小于k−1，就全取。