<https://leetcode.com/problems/top-k-frequent-words/>

Given an array of strings words and an integer k, return *the* k *most frequent strings*.

Return the answer **sorted** by **the frequency** from highest to lowest. Sort the words with the same frequency by their **lexicographical order**.

**Example 1:**

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

Output: ["i","love"]

Explanation: "i" and "love" are the two most frequent words.

Note that "i" comes before "love" due to a lower alphabetical order.

**Example 2:**

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

Output: ["the","is","sunny","day"]

Explanation: "the", "is", "sunny" and "day" are the four most frequent words, with the number of occurrence being 4, 3, 2 and 1 respectively.

**Constraints:**

1 <= words.length <= 500

1 <= words[i].length <= 10

words[i] consists of lowercase English letters.

k is in the range [1, The number of **unique** words[i]]

**Follow-up:** Could you solve it in O(n log(k)) time and O(n) extra space?

**Attempt 1: 2023-02-06**

**Solution 1:  Hash Table + maxPQ (30 min)**

**Style 1: Conventional maxPQ + helper class Node**

class Solution {

public List<String> topKFrequent(String[] words, int k) {

Map<String, Integer> freq = new HashMap<String, Integer>();

for(String word : words) {

freq.put(word, freq.getOrDefault(word, 0) + 1);

}

PriorityQueue<Node> maxPQ = new PriorityQueue<Node>(new Comparator<Node>() {

public int compare(Node a, Node b) {

if(a.freq == b.freq) {

return a.word.compareTo(b.word);

}

return b.freq - a.freq;

}

});

for(Map.Entry<String, Integer> entry : freq.entrySet()) {

maxPQ.offer(new Node(entry.getKey(), entry.getValue()));

}

List<String> result = new ArrayList<String>();

for(int i = 0; i < k; i++) {

result.add(maxPQ.poll().word);

}

return result;

}

}

class Node {

String word;

int freq;

public Node(String word, int freq) {

this.word = word;

this.freq = freq;

}

}

Time Complexity: O(nlogk)

Space Complexity: O(n)

**Style 2: lambda expression maxPQ without helper class**

class Solution {

public List<String> topKFrequent(String[] words, int k) {

Map<String, Integer> freq = new HashMap<String, Integer>();

for(String word : words) {

freq.put(word, freq.getOrDefault(word, 0) + 1);

}

PriorityQueue<Map.Entry<String, Integer>> maxPQ = new PriorityQueue<Map.Entry<String, Integer>>((a, b) -> a.getValue() == b.getValue() ? a.getKey().compareTo(b.getKey()) : b.getValue() - a.getValue());

for(Map.Entry<String, Integer> entry : freq.entrySet()) {

maxPQ.offer(entry);

}

List<String> result = new ArrayList<String>();

for(int i = 0; i < k; i++) {

result.add(maxPQ.poll().getKey());

}

return result;

}

}

Time Complexity: O(nlogk)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/top-k-frequent-words/solutions/108346/my-simple-java-solution-using-hashmap-priorityqueue-o-nlogk-time-o-n-space/>

The idea is to keep a count of each word in a HashMap and then insert in a Priority Queue.

While inserting in pq, if the count of two words is same then insert based on string compare of the keys.

class Solution {

public List<String> topKFrequent(String[] words, int k) {

List<String> result = new LinkedList<>();

Map<String, Integer> map = new HashMap<>();

for(int i=0; i<words.length; i++)

{

if(map.containsKey(words[i]))

map.put(words[i], map.get(words[i])+1);

else

map.put(words[i], 1);

}

PriorityQueue<Map.Entry<String, Integer>> pq = new PriorityQueue<>(

(a,b) -> a.getValue()==b.getValue() ? b.getKey().compareTo(a.getKey()) : a.getValue()-b.getValue()

);

for(Map.Entry<String, Integer> entry: map.entrySet())

{

pq.offer(entry);

if(pq.size()>k)

pq.poll();

}

while(!pq.isEmpty())

result.add(0, pq.poll().getKey());

return result;

}

}