Top K Frequent Elements

Difficulty	Medium
Category	Arrays
Question	https://leetcode.com/problems/top-k-frequent-elements/
Solution	https://www.youtube.com/watch?v=YPTqKIgVk-k
Status	Done

- [ ] Solve the problem from https://leetcode.com/problems/top-k-frequent-elements/

```
from collections import defaultdict
from typing import List

class Solution:
    def topKFrequent(self, nums: List[int], k: int) -> List[int]:
        # Create a defaultdict with lists as default values to store numbers grouped by frequency
        num_dict = defaultdict(list)

        # Initialize an empty list to store the k most frequent numbers
        most_common = []

        # Group numbers by their frequency in the num_dict dictionary
        for num in nums:
            num_dict[num].append(num)

        # Sort the values (lists of numbers) in descending order of length (frequency)
        val_list = sorted(num_dict.values(), key=len, reverse=True)

        # Iterate through the sorted values and pick the first element (most frequent) from each group
        for group in val_list[:k]:
            most_common.append(group[0])

        # Sort the most_common list to return the result in any order
        return sorted(most_common)
```

Explanation

1. We import necessary modules and define a class `Solution` with a method `topKFrequent` that takes a list of integers `nums` and an integer `k` as input and returns a list of integers.

2. We create a `defaultdict` named `num_dict` with lists as default values. This dictionary will be used to group numbers by their frequency. The key is the number, and the value is a list of that number repeated as many times as its frequency.
3. We initialize an empty list named `most_common` to store the k most frequent numbers.
4. We iterate through the `nums` list. For each number `num`, we append it to the list associated with its frequency in the `num_dict` dictionary.
5. We sort the values of the `num_dict` dictionary (lists of numbers) in descending order of length, which effectively sorts them by frequency. This step is done using the `sorted` function and the `key` argument, where `len` is used as the key function to sort by the length of each list.
6. We iterate through the sorted `val_list` and pick the first element (most frequent) from each group. We append these most frequent numbers to the `most_common` list.
7. Finally, we sort the `most_common` list to return the result in any order as required by the problem statement.

The time complexity of this code is O(n * log(n)), where n is the length of the `nums` list. The sorting step dominates the time complexity. The space complexity is O(n) due to the use of the `num_dict` dictionary, which can have at most n keys, and the `most_common` list.