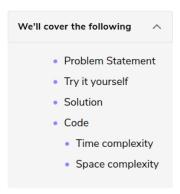
# Frequency Sort (medium)



# **Problem Statement**

Given a string, sort it based on the decreasing frequency of its characters.

#### Example 1:

```
Input: "Programming"
Output: "rrggmmPiano"
Explanation: 'r', 'g', and 'm' appeared twice, so they need to appear before any other characte r.
```

### Example 2:

```
Input: "abcbab"
Output: "bbbaac"
Explanation: 'b' appeared three times, 'a' appeared twice, and 'c' appeared only once.
```

# Try it yourself #

Try solving this question here:

# Solution #

This problem follows the Top 'K' Elements pattern, and shares similarities with Top 'K' Frequent Numbers.

We can follow the same approach as discussed in the Top 'K' Frequent Numbers problem. First, we will find

the frequencies of all characters, then use a max-heap to find the most occurring characters.

### Code #

Here is what our algorithm will look like:

```
Python3
                           ⊘ C++
👙 Java
                                        Js JS
          heapq import
    def sort_character_by_frequency(str):
       charFrequencyMap = {}
       for char in str:
        charFrequencyMap[char] = charFrequencyMap.get(char, 0) + 1
       for char, frequency in charFrequencyMap.items():
        heappush(maxHeap, (-frequency, char))
      \# build a string, appending the most occurring characters first {\bf sortedString} = []
       while maxHeap:
         frequency, char = heappop(maxHeap)
         for _ in range(-frequency):
           sortedString.append(char)
       return ''.join(sortedString)
      print("String after sorting characters by frequency: " +
      | | sort_character_by_frequency("Programming")) | print("String after sorting characters by frequency: " +
             sort_character_by_frequency("abcbab"))
                                                                                                           Reset []
```

### Time complexity

The time complexity of the above algorithm is O(D\*logD) where 'D' is the number of distinct characters in the input string. This means, in the worst case, when all characters are unique the time complexity of the algorithm will be O(N\*logN) where 'N' is the total number of characters in the string.

# Space complexity #

The space complexity will be O(N), as in the worst case, we need to store all the 'N' characters in the HashMap.

