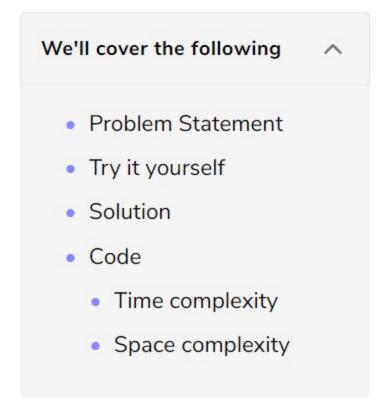


educative

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 character.
```

₿

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:

```
G C++
           Python3
                          JS JS
Java
    def sort character by frequency(str):
      # TODO: Write your code here
      return ""
    def main():
      print("String after sorting characters by frequency: " +
            sort_character_by_frequency("Programming"))
      print("String after sorting characters by frequency: " +
10
            sort_character_by_frequency("abcbab"))
11
12
13
14 main()
15
Run
                                                                                              Save
                                                                                                        Reset
```

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:

```
JS JS
                          G C++
Java
            Python3
1 from heapq import *
    def sort_character_by_frequency(str):
      # find the frequency of each character
      charFrequencyMap = {}
      for char in str:
        charFrequencyMap[char] = charFrequencyMap.get(char, 0) + 1
11
      maxHeap = []
      # add all characters to the max heap
      for char, frequency in charFrequencyMap.items():
        heappush(maxHeap, (-frequency, char))
14
15
      # build a string, appending the most occurring characters first
      sortedString = []
      while maxHeap:
        frequency, char = heappop(maxHeap)
        for _ in range(-frequency):
          sortedString.append(char)
21
22
      return ''.join(sortedString)
25
    def main():
26
27
      print("String after sorting characters by frequency: " +
            sort_character_by_frequency("Programming"))
      print("String after sorting characters by frequency: " +
           sort character by frequency("abcbab"))
 Run
                                                                                               Save
                                                                                                         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 ${\cal O}(N)$, as in the worst case, we need to store all the 'N' characters in the HashMap.

