

Frequency Sort (medium)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

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:

JavaPython3JS C++

```
1 import java.util.*;
2
3 class FrequencySort {
4
5     public static String sortCharacterByFrequency(String str) {
6         // TODO: Write your code here
7         return "";
8     }
9
10    public static void main(String[] args) {
11        String result = FrequencySort.sortCharacterByFrequency("Programming");
12        System.out.println("Here is the given string after sorting characters by frequency: " + result);
13
14        result = FrequencySort.sortCharacterByFrequency("abcbab");
15        System.out.println("Here is the given string after sorting characters by frequency: " + result);
16    }
17 }
18
```

RunSaveReset

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:

JavaPython3C++JS

```
1 import java.util.*;
```

```
2
3 class FrequencySort {
4
5     public static String sortCharacterByFrequency(String str) {
6         // find the frequency of each character
7         Map<Character, Integer> characterFrequencyMap = new HashMap<>();
8         for (char chr : str.toCharArray()) {
9             characterFrequencyMap.put(chr, characterFrequencyMap.getOrDefault(chr, 0) + 1);
10        }
11
12        PriorityQueue<Map.Entry<Character, Integer>> maxHeap = new PriorityQueue<Map.Entry<Character, Integer>>
13            ((e1, e2) -> e2.getValue() - e1.getValue());
14
15        // add all characters to the max heap
16        maxHeap.addAll(characterFrequencyMap.entrySet());
17
18        // build a string, appending the most occurring characters first
19        StringBuilder sortedString = new StringBuilder(str.length());
20        while (!maxHeap.isEmpty()) {
21            Map.Entry<Character, Integer> entry = maxHeap.poll();
22            for (int i = 0; i < entry.getValue(); i++)
23                sortedString.append(entry.getKey());
24        }
25        return sortedString.toString();
26    }
27
28    public static void main(String[] args) {
```

Time complexity

The time complexity of the above algorithm is $O(D * \log D)$ 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 $O(N * \log N)$ 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.



[← Back](#)

[Next →](#)

Top 'K' Frequent Numbers (medium)

Kth Largest Number in a Stream (medi...

☒ Mark as Completed

 Report an Issue  Ask a Question