

educative

Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

13% completed

Search Course

We'll cover the following

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

## 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:

JavaPython3JS C++

```
1 def sort_character_by_frequency(str):
2     # TODO: Write your code here
3     return ""
4
5
6 def main():
7
8     print("String after sorting characters by frequency: " +
9           sort_character_by_frequency("Programming"))
10    print("String after sorting characters by frequency: " +
11          sort_character_by_frequency("abcbab"))
12
13
14 main()
15
```

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 from heapq import *
2
3
4 def sort_character_by_frequency(str):
5
6     # find the frequency of each character
7     charFrequencyMap = {}
8     for char in str:
9         charFrequencyMap[char] = charFrequencyMap.get(char, 0) + 1
10
11     maxHeap = []
12     # add all characters to the max heap
13     for char, frequency in charFrequencyMap.items():
14         heappush(maxHeap, (-frequency, char))
15
16     # build a string, appending the most occurring characters first
17     sortedString = []
18     while maxHeap:
19         frequency, char = heappop(maxHeap)
20         for _ in range(-frequency):
21             sortedString.append(char)
22
23     return ''.join(sortedString)
24
25
26 def main():
27
28     print("String after sorting characters by frequency: " +
29           sort_character_by_frequency("Programming"))
30     print("String after sorting characters by frequency: " +
31           sort_character_by_frequency("abcbab"))
```

RunSaveReset

### 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