

## Solution Review: Problem Challenge 1

### We'll cover the following ^

- Rearrange String K Distance Apart (hard)
- Solution
- Code
  - Time complexity
  - Space complexity

## Rearrange String K Distance Apart (hard) #

Given a string and a number 'K', find if the string can be rearranged such that the same characters are at least 'K' distance apart from each other.

### Example 1:

```
Input: "mmp", K=2
Output: "mpm" or "pmp"
Explanation: All same characters are 2 distance apart.
```

### Example 2:

```
Input: "Programming", K=3
Output: "rgmPrmgiano" or "gmrngmrPoa" or "gmrPagimnor" and a few more
Explanation: All same characters are 3 distance apart.
```

### Example 3:

```
Input: "aab", K=2
Output: "aba"
Explanation: All same characters are 2 distance apart.
```

### Example 4:

```
Input: "aappa", K=3
Output: ""
Explanation: We cannot find an arrangement of the string where any two 'a' are 3 distance apart.
```

## Solution #

This problem follows the [Top 'K' Numbers](#) pattern and is quite similar to [Rearrange String](#). The only difference is that in the 'Rearrange String' the same characters need not be adjacent i.e., they should be at least '2' distance apart (in other words, there should be at least one character between two same characters), while in the current problem, the same characters should be 'K' distance apart.

Following a similar approach, since we were inserting a character back in the heap in the next iteration, in this problem, we will re-insert the character after 'K' iterations. We can keep track of previous characters in a queue to insert them back in the heap after 'K' iterations.

## Code #

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1 from heapq import *
2 from collections import deque
3
4
5 def reorganize_string(str, k):
6     if k <= 1:
7         return str
8
9     charFrequencyMap = {}
10    for char in str:
11        charFrequencyMap[char] = charFrequencyMap.get(char, 0) + 1
12
13    maxHeap = []
14    # add all characters to the max heap
15    for char, frequency in charFrequencyMap.items():
16        heappush(maxHeap, (-frequency, char))
17
18    queue = deque()
19    resultString = []
20    while maxHeap:
21        frequency, char = heappop(maxHeap)
22        # append the current character to the result string and decrement its count
23        resultString.append(char)
24        # decrement the frequency and append to the queue
25        queue.append((char, frequency+1))
26        if len(queue) == k:
27            char, frequency = queue.popleft()
28            if -frequency > 0:
29                heappush(maxHeap, (frequency, char))
30
31    # if we were successful in appending all the characters to the result string, return it
32    return ''.join(resultString) if len(resultString) == len(str) else ""
33
34
35 def main():
36     print("Reorganized string: " + reorganize_string("Programming", 3))
37     print("Reorganized string: " + reorganize_string("mmpp", 2))
38     print("Reorganized string: " + reorganize_string("aab", 2))
39     print("Reorganized string: " + reorganize_string("aapa", 3))
40
41
42 main()
43
```

Run Save Reset

### Time complexity #

The time complexity of the above algorithm is  $O(N * \log N)$  where 'N' is the number of characters in the input 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

Problem Challenge 1

Next →

Problem Challenge 2

✓ Completed

⚠ Report an Issue    ? Ask a Question