Given a string s, sort it in decreasing order based on the frequency of characters.

- The frequency of a character is the number of times it appears in the string.
- If multiple answers are possible, return any one.

🖊 **Example**

Input: s = "tree"

Output: "eert" (or "eetr")

Explanation:

- 'e' appears twice
- 't' and 'r' appear once
  So 'e' must come before 't' and 'r'.

□ Approach (Same Logic as Your Code)

We will **not use sorting or priority queue**.

Steps:

1. **Count frequency of each character** using HashMap
2. **Find the maximum frequency**
3. **From max frequency to 1**, append characters whose frequency matches the current value

---

✅ **Java Code (HashMap Version)**

```
import java.util.HashMap;
import java.util.Map;

class Solution {
    public String frequencySort(String s) {

        // Step 1: Count frequency of each character
        HashMap<Character, Integer> map = new HashMap<>();
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            map.put(ch, map.getOrDefault(ch, 0) + 1);
        }
```

```
        // Step 2: Find maximum frequency
        int max = 0;
        for (int freq : map.values()) {
            max = Math.max(max, freq);
        }

        // Step 3: Build result from max frequency to 1
        StringBuilder sb = new StringBuilder();

        while (max > 0) {
            for (Map.Entry<Character, Integer> entry : map.entrySet()) {
                if (entry.getValue() == max) {
                    int temp = max;
                    while (temp > 0) {
                        sb.append(entry.getKey());
                        temp--;
                    }
                }
            }
            max--;
        }

        return sb.toString();
    }
}
```

🔍 **Dry Run (Step-by-Step)**

Input:

s = "tree"

Step 1: Frequency Map

Character	Frequency
t	1
r	1
e	2

map = { t=1, r=1, e=2 }

Step 2: Find Maximum Frequency

max = 2

Step 3: Build Output

First iteration (max = 2)

- 'e' has frequency 2
- Append 'e' two times

sb = "ee"

Second iteration (max = 1)

- 't' → frequency 1 → append once
- 'r' → frequency 1 → append once

sb = "eetr"

Final Output

"eetr"

✓ Valid output
("eert" is also valid)

⏱ **Time Complexity**

Let n = length of string

Step	Complexity
Frequency counting	O(n)

Step	Complexity
Finding max frequency	O(n)
Building result	O(n)
Total Time	**O(n)**

Even though we loop over the map multiple times, **each character is appended exactly n times total**, so overall work is linear.

💾 Space Complexity

Structure	Space
HashMap	O(k) (unique characters)
StringBuilder	O(n)
Total Space	**O(n)**