## Reorganize String

Given a string `S`, check if the letters can be rearranged so that two characters that are adjacent to each other are not the same.

If possible, output any possible result.  If not possible, return the empty string.

### Example 1:

```
Input: S = "aab"
Output: "aba"
```

### Example 2:

```
Input: S = "aaab"
Output: ""
```

### Note:

- `S` will consist of lowercase letters and have length in range `[1, 500]`.

## Solution 1

```
class Solution {
    public String reorganizeString(String S) {
        // Create map of each char to its count
        Map<Character, Integer> map = new HashMap<>();
        for (char c : S.toCharArray()) {
            int count = map.getOrDefault(c, 0) + 1;
            // Impossible to form a solution
            if (count > (S.length() + 1) / 2) return "";
            map.put(c, count);
        }
        // Greedy: fetch char of max count as next char in the result.
        // Use PriorityQueue to store pairs of (char, count) and sort by count DESC.
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> b[1] - a[1]);
        for (char c : map.keySet()) {
            pq.add(new int[] {c, map.get(c)});
        }
        // Build the result.
        StringBuilder sb = new StringBuilder();
        while (!pq.isEmpty()) {
            int[] first = pq.poll();
            if (sb.length() == 0 || first[0] != sb.charAt(sb.length() - 1)) {
                sb.append((char) first[0]);
                if (--first[1] > 0) {
                    pq.add(first);
                }
            } else {
                int[] second = pq.poll();
                sb.append((char) second[0]);
                if (--second[1] > 0) {
                    pq.add(second);
                }
                pq.add(first);
            }
        }
        return sb.toString();
    }
}
```

written by shawngao original link here

## Solution 2

We can solve this problem using priority queue. Another solution is using sort.

```
The solution sort the string by occurrence, i.e. the character with most occurrence
is at front. For example, aaaaabbcc.

Next, let i = 0 at the beginning, j = (n-1)/2+1 in the middle.
We can build the answer by appending s[i++] and s[j++] sequentially.
```

### Sort

```
class Solution {
public:
    string reorganizeString(string S) {
        vector<int> mp(26);
        int n = S.size();
        for (char c: S) ++mp[c-'a'];
        vector<pair<int, char>> charCounts;
        for (int i = 0; i < 26; ++i) {
            if (mp[i] > (n+1)/2) return "";
            if (mp[i]) charCounts.push_back({mp[i], i+'a'});
        }
        sort(charCounts.rbegin(), charCounts.rend());
        string strSorted;
        for (auto& p: charCounts)
            strSorted += string(p.first, p.second);
        string ans;
        for (int i = 0, j = (n-1)/2+1; i <= (n-1)/2; ++i, ++j) {
            ans += strSorted[i];
            if (j < n) ans += strSorted[j];
        }
        return ans;
    }
};
```

### Priority Queue

```cpp
class Solution {
public:
    string reorganizeString(string S) {
        vector<int> mp(26);
        int n = S.size();
        for (char c: S)
            ++mp[c-'a'];
        priority_queue<pair<int, char>> pq;
        for (int i = 0; i < 26; ++i) {
            if (mp[i] > (n+1)/2) return "";
            if (mp[i]) pq.push({mp[i], i+'a'});
        }
        queue<pair<int, char>> myq;
        string ans;
        while (!pq.empty() || myq.size() > 1) {
            if (myq.size() > 1) {
                auto cur = myq.front();
                myq.pop();
                if (cur.first != 0) pq.push(cur);
            }
            if (!pq.empty()) {
                auto cur = pq.top();
                pq.pop();
                ans += cur.second;
                cur.first--;
                myq.push(cur);
            }
        }
        return ans;
    }
};
```

written by zestypanda original link here

## Solution 3

Put the *least* common letters at the *odd* indexes and put the *most* common letters at the *even* indexes (both from left to right in order of frequency). The task is only impossible if some letter appears too often, in which case it'll occupy all of the even indexes and at least the last odd index, so I check the last two indexes.

```
def reorganizeString(self, S):
    a = sorted(sorted(S), key=S.count)
    h = len(a) / 2
    a[1::2], a[::2] = a[:h], a[h:]
    return ''.join(a) * (a[-1:] != a[-2:-1])
```

written by StefanPochmann original link here