151. Reverse Words in a String
We will split the original string and reverse it. In the end, we will use blank to join these characters. TC is O(n)
```
class Solution:
    def reverseWords(self, s: str) -> str:
        return ' '.join(reversed(s.split()))
```

14. Longest Common Prefix
We will compare our result, initially strs[0], to compare one by one and reset result as common prefix. If during the process, either our result or s is ‘’, we will return ‘’ immediately. TC is O(n * length).
```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
            return ''

        result = strs[0]
        for s in strs[1:]:
            if not s or not result:
                return ''
            length = min(len(result), len(s))
            for i in range(length):
                if result[i] != s[i]:
                    result = result[:i]
                    break
                if i == length - 1:
                    result = result[:length]
        return result
```

489. Robot Room Cleaner
We will use dfs to traverse all accessible cells. We will use move to check whether each neighbor cell accessible. If it’s accessible and not visited, we will bfs from that point. We will use two leftTurn, one move, two leftTurn to get back to original point and bfs other three directions. TC is O(n)
```
class Solution(object):
    def cleanRoom(self, robot):
        """
        :type robot: Robot
        :rtype: None
        """
        self.dfs(0, 0, 0, 1, set(), robot)
    def dfs(self, x, y, d_x, d_y, visited, robot):
        robot.clean()
        visited.add((x, y))
        for i in range(4):
            new_x, new_y = x + d_x, y + d_y
            if (new_x, new_y) not in visited and robot.move():
                self.dfs(new_x, new_y, d_x, d_y, visited, robot)
                robot.turnLeft()
                robot.turnLeft()
                robot.move()
                robot.turnLeft()
                robot.turnLeft()
            robot.turnLeft()
            d_x, d_y = -d_y, d_x
```

362. Design Hit Counter

We will use a sum and deque to maintain all hits within 5 mins. Every hit we will first check the last element's timestamp equal to its current timestamp, if they are equal, we will add it directly, or we will append a new (timestamp, times) to the end of deque. For get hits, we will check from the beginning and its timestamp is less than the current one by 300, we will popleft it and reduce this number from our sum at the same time. Until all front element's timestamp meets our requirement, we will return our sum. TC is O(2), O(n)

```
from collections import deque
class HitCounter:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.sum = 0
        self.q = deque()


    def hit(self, timestamp: int) -> None:
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds granularity).
        """
        if self.q and self.q[-1][0] == timestamp:
            t, v = self.q.pop()
            self.q.append((t, v + 1))
        else:
            self.q.append((timestamp, 1))
        self.sum += 1
    def getHits(self, timestamp: int) -> int:
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds granularity).
        """
        while self.q and timestamp - self.q[0][0] >= 300:
            self.sum -= self.q.popleft()[1]
        return self.sum
```

819. Most Common Word

We will split paragraph by non letters and filter ‘’. And we will counter each word in the array and if word is in banned set, we will continue. In the end, we get the maximum one. TC is O(n)

```
import re
class Solution:
    def mostCommonWord(self, paragraph: str, banned: List[str]) -> str:
        memo = {}
        banned_set = set(banned)
        for word in map(lambda a: a.lower(), re.split('[^a-zA-Z]', paragraph)):
            if word in banned_set or not word:
                continue
            if word not in memo:
                memo[word] = 1
            else:
                memo[word] += 1
        return max(memo.items(), key=lambda a: a[1])[0]
```