CSC148 - Object-Oriented Design Considerations

Recall our `Tweet` class:

```
class Tweet:
    """A tweet, like in Twitter.

    === Attributes ===
    content: the contents of the tweet.
    userid: the id of the user who wrote the tweet.
    created_at: the date the tweet was written.
    likes: the number of likes this tweet has received.
    """
    # Attribute types
    content: str
    userid: str
    created_at: date
    likes: int

    def __init__(self, who: str, when: date, what: str) -> None:
        """Initialize a new Tweet.
        """
        self.userid = who
        self.content = what
        self.created_at = when
        self.likes = 0

    def like(self, n: int) -> None:
        """Record the fact that this tweet received <n> likes.

        These likes are in addition to the ones <self> already has.
        """
        self.likes += n

    def edit(self, new_content: str) -> None:
        """Replace the contents of this tweet with the new message.
        """
        self.content = new_content[:280]
```

1. Write code that creates a tweet called `misbehaved` that is in some way nonsensical. There are at least two ways to do this.

2. Describe a property (something that should be true) that your `misbehaved` instance has violated.

3. Modify the `Tweet` class above to prevent your methods from violating this property.

4. Here's a `Tournament` class describing a sports tournament, including a method for reporting statistics. Method bodies are omitted.

```python
class Tournament:
    """A sports tournament.

    === Attributes ===
    teams:
        The names of the teams in this tournament.
    team_stats:
        The history of each team in this tournament. Each key is a team name,
        and each value is a list storing two non-negative integers:
        the number of games played and the number won.

    === Sample usage ===

    >>> t = Tournament(['a', 'b', 'c'])
    >>> t.record_game('a', 'b', 10, 4)
    >>> t.record_game('a', 'c', 5, 1)
    >>> t.record_game('b', 'c', 2, 0)
    >>> t.best_percentage()
    'a'
    """
    # Attribute types
    teams: List[str]
    team_stats: Dict[str, List[int]]

    def __init__(self, teams: List[str]) -> None:
        """Initialize a new Tournament among the given teams.

        Note: Does not make an alias to <teams>.
        """

    def record_game(self, team1: str, team2: str,
                    score1: int, score2: int) -> None:
        """Record the fact that <team1> played <team2> with the given scores.

        <team1> scored <score1> and <team2> scored <score2> in this game.

        Precondition: team1 and team2 are both in this tournament.
        """

    def best_percentage(self) -> str:
        """Return the team name with the highest percentage of games won.

        If no team has won a game, return the empty string.
        Otherwise if there is a tie for best percentage, return the name of any
        of the tied teams.
        """
```

(a) Are the instance attributes sufficient in order to implement method `best_percentage`? Explain.

(b) Identify another statistic that could be reported and for which the instance attributes are insufficient. How would you change the instance attributes to support it?

(c) What negative consequences might ensue if you changed the instance attributes?