INSTRUCTIONS

- You have 25 minutes to complete this quiz.
- The exam is closed book, closed notes, closed computer, closed calculator.
- Mark your answers on the quiz itself. We will not grade answers written on scratch paper.

<table>
<thead>
<tr>
<th>Last name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First name</td>
<td></td>
</tr>
<tr>
<td>Student ID number</td>
<td></td>
</tr>
<tr>
<td>Instructional account (cs61a-)</td>
<td></td>
</tr>
<tr>
<td>BearFacts email (@berkeley.edu)</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your left</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your right</td>
<td></td>
</tr>
</tbody>
</table>

All the work on this exam is my own.
(please sign)
1. (5 points) The Evil Empire

Let’s implement a data abstraction for basketball players. Our constructor takes in a name, a position (1, 2, 3, 4, or 5), and, optionally, a backup position. Our selectors retrieve information about a player.

```python
def player(name, position, backup=None):
    if backup:
        return {'name': name, 'position': position, 'backup': backup}
    return {'name': name, 'position': position}

def name(player):
    return player['name']

def position(player):
    return player['position']

def backup(player):
    return player['backup'] if 'backup' in player else None
```

When we make a basketball team, we want to make sure that there is at least one player for each position. So we define a function `check_team` that takes in a non-empty list of players. `check_team` returns `True` if there is at least one player per position, and `False` otherwise.

(a) (3 pt) The following implementation works, but it breaks abstraction barriers! Cross out each violation and, above the original line, write some replacement code that has no violations and maintains correctness.

```python
def check_team(players):
    """Make sure there is at least one player per position.
    Look on the next page for the players used in these doctests, and the implementation of the insert helper function.
    ""
    def checker(players, covered):
        if len(covered) == 5:
            return True
        elif len(players) == 0:
            return False
        p = players[0]
        in_main_role = checker(players[1:], insert(covered, p['position']))
        if 'backup' in p:
            in_backup_role = checker(players[1:], insert(covered, p['backup']))
            return in_main_role or in_backup_role
        return in_main_role
    return checker(players, [])
```

The doctest references these players, constructed for testing purposes:

```python
>>> steph = player('Steph Curry', 1)
>>> lebron = player('LeBron James', 3, 4)
>>> kd = player('Kevin Durant', 3, 4)
>>> klay = player('Klay Thompson', 2)
>>> iggy = player('Andre Iguodala', 4, 3)
>>> money = player('Draymond Green', 4, 5)
>>> wade = player('Dwyane Wade', 1)
>>> kyrie = player('Kyrie Irving', 1)
```

The `insert` helper function is also used in `check_team`:

```python
def insert(lst, elem):
    """Add elem to lst if elem is not already contained in lst."

    >>> insert([1, 2, 3], 5)
    [1, 2, 3, 5]
    >>> insert([1, 2, 3], 2)
    [1, 2, 3]
    """
    return lst if elem in lst else lst + [elem]
```

(b) (1 pt) Write a constructor and selectors that correctly implement the player abstraction, but would cause the original abstraction-violating code of `check_team` to error or have incorrect behavior.

```python
def player(name, position, backup=None):

def name(player):

def position(player):

def backup(player):
```

(c) (1 pt) If we call `check_team` with a list of \( n \) players, and every player in the list has a backup position, what is the order of growth on the runtime of `check_team` as a function of \( n \)? Assume that all built-in functions and operations run in constant time.

\[
\begin{align*}
\Theta(1) & \quad \Theta(\log n) & \quad \Theta(n) & \quad \Theta(n^2) & \quad \Theta(2^n)
\end{align*}
\]