SS 2014
Prof. Dr. Margarita Esponda
ALP2
3. Übungsblatt (Abgabe: Mi., den 14.05.)

Ziel: Auseinandersetzung mit Sortieralgorithmen und Analysen der Klassenkomplexität.

1. Aufgabe (8 Punkte)

a) Erläutern Sie mit einem konkreten Zahlenbeispiel, warum der Quicksort-Algorithmus aus der Vorlesung nicht stabil ist.

b) Können Sie den Algorithmus aus der Vorlesung stabil machen? Erläutern Sie Ihre Antwort, indem Sie die entsprechende Variante in Python programmieren. Oder erläutern Sie warum es nicht möglich ist.

c) Wie könnten Sie, falls eine Lösung existiert, mit entsprechenden Testdaten automatisch kontrollieren, dass dieser Algorithmus funktioniert?

d) Wie oft kann während der Ausführung des Quicksort-Algorithmus das größte Element maximal bewegt werden? Begründen Sie Ihre Antwort.

2. Aufgabe (6 Punkte)

Ein Element einer Liste von ***n*** Objekten stellt die absolute Mehrheit der Liste dar, wenn das Element mindestens $\left(\frac{n}{2}+1\right)$ mal in der Liste vorkommt. Definieren Sie

eine **majority**-Funktion, die das Majority-Element der Liste findet, wenn eines existiert oder sonst "keine Majority" als Ergebnisse zurückgibt.

Analysieren Sie die Komplexität Ihres Algorithmus.

3. Aufgabe (18 Punkte)

In dieser Aufgabe möchten wir ein Spiel simulieren, das analog zum *Minesweeper*-Spiel funktionieren soll. Wir benutzen nur Löcher statt Bomben. Das Spielfeld soll mit Hilfe einer Matrix modelliert werden.

a) Schreiben Sie zuerst eine **new_play** Funktion, die nach Eingabe von **p** eine **n**x**m** Matrix initialisiert, in dem mit Wahrscheinlichkeit **p** in jeder beliebigen (x,y)-Position ein Loch vorkommen kann. Sie können dabei mit einem '**.**'-Zeichen die Positionen ohne Löcher und mit einem '**O**'-Zeichen die Positionen mit Löchern markieren.

b) Schreiben Sie eine Funktion **generate_solution**, mit der in jeder Position des Feldes, an der kein Loch existiert, die Anzahl der benachbarten Löcher ausgegeben wird. Wenn keine Nachbarn vorhanden sind, soll das Punkt-

Zeichen hinterlassen werden.

c) Schreiben Sie eine Funktion **print_field**, die das Feld ausgibt. Eine Beispielausgabe könnte folgendermaßen aussehen:

```
o 2 1 1 2 2 1 . .
2 o 1 2 o o 1 . .
1 1 1 2 o 3 1 1 1
. . . 1 1 1 . 1 o
```

d) Schreiben Sie eine Funktion **start_play**, die bei Eingabe einer Wahrscheinlichkeit **p** und einer Spielfeldgröße (**n**, **m**) eine Matrix mit entsprechender Lösung intern produziert und diese zuerst mit gedeckten Feldern ausgibt.

```
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
X X X X X X X X X X
```

e) Schreiben Sie zum Schluss eine Funktion **play**, die das Spiel mit folgenden Regeln startet:

1) Der Spieler wird jedes mal aufgefordert eine **x**, **y** Position einzugeben.

2) Wenn in der eingegebenen Position ein Loch existiert, fällt der Spieler in das Loch und das Spiel ist beendet.

3) Wenn an der gewählten **x**, **y** Position kein Loch ist, aber mindestens ein Loch in der Nachbarschaft existiert, wird nur diese Position mit entsprechender Zahl aufgedeckt.

4) Wenn die gewählte Position keinen Nachbarn mit einem Loch hat (Position mit '.' - Zeichen), wird diese aufgedeckt und alle seine Nachbarn (so lange noch welche vorhanden sind) werden auch automatisch aufgedeckt. Die neu aufgedeckten Nachbarn, die selber keinen Nachbarn mit Loch haben (Nachbarn ohne Zahlen), verursachen wiederum die Aufdeckung ihrer Nachbarn. Diese Kettenreaktion wird so lange kein Loch, kein Nachbar mit Zahlen und kein Rand vorkommt, wiederholt.

5) Wenn alle Positionen aufgedeckt sind und der Spieler noch am Leben ist, hat er gewonnen!