SoSe 2018

Prof. Dr. Margarita Esponda

Objektorientierte Programmierung

5. Übungsblatt

Ziel: Erste Auseinandersetzung mit dem Hoare-Kalkül.

1. Aufgabe (4 Punkte)

Definieren Sie eine **is_max_heap**-Funktion, die ein Array/Liste mit Zahlen als Argument bekommt und überprüft, ob die Zahlen als heap-Struktur sortiert sind bzw., ob die Liste ein Heap ist.

2. Aufgabe (6 Punkte)

a) Definieren Sie eine Variante des Countingsort-Algorithmus aus der Vorlesung, bei dem Sie die Elemente **in-Place** sortieren. Ihre Countingsort-Variante muss nicht stabil sein. Sie dürfen als einzigen zusätzlichen Speicher das Hilfsrarray C verwenden (siehe Vorlesungsfolien). Das Hilfsarray **B** darf nicht mehr verwendet werden.

b) Analysieren Sie die Komplexität ihres Algorithmus.

3. Aufgabe (5 Punkte)

Beantworten Sie die Frage, die Google an Obama im Jahr 2008 stellte.
"*What ist the most efficient way to sort a million 32-bit integers?*". Begründen Sie Ihre Antwort.
Implementieren Sie Ihre Lösung und testen Sie diese mit einer Million zufällig erzeugter 32-Bit Zahlen. Erläutern Sie Vor- und Nachteile Ihrer Lösung.

4. Aufgabe (5 Punkte)

Beweisen Sie die Gültigkeit der folgenden Programmformeln

$$\{P\} \equiv \{a > 0 \ \wedge \ b > 0 \ \wedge \ c < 0\}$$

```
a = a + b - c
d = b
b = a - b - c
c = - c
```

$$\{Q\} \equiv \{a > 0 \ \wedge \ b > 0 \ \wedge \ c > 0 \ \wedge \ b == a - d + c\}$$

5. Aufgabe (6 Punkte)

Seien **x, y, z** ganzzahlige Variablen (**int**) und **c** ein konstanter ganzer Wert größer 0.
Beweisen Sie die Gültigkeit der folgenden Programmformel.

$$\{P\} \equiv \{x \geq 0 \wedge (x * y + z) == c\}$$

```
if  x%2==0:
        y = y + y
        x = x//2
else:
        z = z + y
        x = x - 1
```

$$\{Q\} \equiv \{x \geq 0 \wedge (x * y + z) == c\}$$

6. Aufgabe (4 Punkte)

a) Ersetzen Sie die **for**-Schleife des **Insertsort**-Algorithmus der Vorlesung mit einer **while**-Schleife. Postulieren Sie aussagekräftige Invarianten für die äußeren und inneren Schleifen.

b) Testen Sie Ihre Invarianten mit Hilfe von **assert**-Anweisungen. Um Ihre Invarianten zu testen, dürfen Sie Hilfsfunktionen verwenden.

7. Aufgabe (4 Bonuspunkte)

Schreibe eine Funktion in Python, die mit linearem Aufwand (bezüglich der Anzahl der Elemente einer Eingabe-Matrix) das größte Quadrat der Matrix, das nur die Zahl Eins beinhaltet, findet. Die Elemente der Matrix bestehen nur aus Nullen und Einsen.

Das Ergebnis soll in einem Tupel aus zwei Tupeln zurückgegeben werden, die der oberen linken und der unteren rechten Koordinate des Quadrates entsprechen.

Beispiel:

```
0 0 1 1 0 1 1 1 0 0 0 0 0
0 0 1 0 1 0 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 1 1 0    =>    ((1,6), (4,9))
0 0 0 0 0 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 1 1 1 1 1 0
0 1 0 1 0 1 0 1 0 1 0 1 0
0 1 1 1 1 0 0 1 1 1 1 0 0
```

8. Aufgabe (6 Bonuspunkte)

a) Was ist die wichtigste Invariante der Schleife innerhalb folgender **partition**-Funktion?

```
def partition( A, low, high ):
    pivot = A[low]
    i = low
    for j in range(low+1,high+1):
        if ( A[j] < pivot ):
            i=i+1
            A[i], A[j] = A[j], A[i]
    A[i], A[low] = A[low], A[i]
    return i
```

b) Begründen Sie Ihre Antwort.

Schreiben Sie getrennte Test-Funktionen für alle Aufgaben für Ihren Tutor.

Wichtige Hinweise:

1) Verwenden Sie geeignete Namen für Ihre Variablen und Funktionsnamen, die den semantischen Inhalt der Variablen oder die Funktionalität der Funktionen darstellen.

2) Verwenden Sie vorgegebene Funktionsnamen, falls diese angegeben werden.

3) Kommentieren Sie Ihre Programme und schreiben Sie Hilfsdokumentation.

4) Verwenden Sie geeignete Hilfsvariablen und Hilfsfunktionen in Ihren Programmen.

5) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.

6) Schreiben Sie getrennte Test-Funktionen für alle 4 Aufgaben für Ihren Tutor.