Prof. Dr. Ir. Joost-Pieter Katoen

Jonathan Heinen, Sabrina von Styp

Hinweise:

- Die Übungsblätter sollen in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden.
- Die Lösungen müssen bis Montag, den 3. Mai um 11:00 Uhr in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Namen und Matrikelnummern der Studenten sowie die Nummer der Übungsgruppe sind auf jedes Blatt der Abgabe zu schreiben. Heften bzw. tackern Sie die Blätter!
- Bitte beachten Sie, dass am 4. Mai wegen der Fachschaftsvollversammlung von 10:00 - 14:00 Uhr keine Übungen stattfinden. Für die Gruppen 6-9 wird es Ausweichtermine geben, die ab Mittwoch den 28. April auf der Internetseite zur Vorlesung angekündigt werden.

Aufgabe 1 (Baumeigenschaften): (3+5+4 Punkte)

Beweisen Sie folgende, in der Vorlesung eingeführte, Fakten für Binärbäume:

a) Ein Binärbaum enthält höchstens 2^d Knoten in Ebene d.

b) Ein Binärbaum mit Höhe h kann maximal $2^{h+1} - 1$ Knoten enthalten.

c) Ein Binärbaum mit n Knoten hat mindestens die Höhe $\lceil \log_2(n+1) \rceil - 1$.

Aufgabe 2 (Alternative Queue-Implementierung): (10 Punkte)

Wir haben in der Vorlesung Queues kennengelernt. Bei der vorgestellten Implementierung kann der letzte freie Eintrag im Array nicht mehr gefüllt werden, da eine komplett volle Queue nicht von einer leeren zu unterscheiden wäre. Sie sollen nun eine alternative Implementierung entwickeln, die diese Einschränkung nicht besitzt. Ersetzten Sie hierzu die Blöcke `A`, `B`, `C`, `D` und `E` in der folgenden Java-Implementierung:

```
public class Queue {
  public Queue(int N) {
    data = new int[N];
  }
  public boolean isEmpty() {
    return head == tail;                          A
  }
  public boolean isFull() {
    return head == (tail + 1) % data.length;      B
  }
  public void enqueue(int e) {
    data[tail] = e;                               C
    tail = (tail + 1) % data.length;
  }
  public int dequeue() {
    int e = data[head];                           D
    head = (head + 1) % data.length;
    return e;
  }
  private int head, tail;                         E
  private int[] data;
}
```

Aufgabe 3 (Laufzeitanalyse): (6 Punkte)

Bestimmen Sie die Komplexitätsklasse für den Aufruf berechne(n) in Abhängigkeit von n. Gehen Sie davon aus, dass die Grundrechenarten +, -, *, / in konstanter Zeit O(1) ausgeführt werden, ebenso die Zuweisungen = und Vergleiche <=.

```
int berechne(int k){

    int result = k;

    for(int i = k; i > 0; i = i/2){
        result = k * result;
        for(int j = 0; j < i; j++){
            result++;
        }
    }

    return result;
}
```

Aufgabe 4 (Effiziente Implementierung): (10 Punkte)

Schreiben Sie einen Algorithmus der den kleinsten und größten Schlüsselwert eines übergebenen Arrays E zurückgibt. Hierbei sei n die Anzahl der Elemente im Array. Der Algorithmus soll im Worst-Case $W(n) = 1,5 \cdot n + c$ Schlüsselwerte (mit konstanten Wert c) vergleichen.

Hinweis:

- Es sollen nur Vergleiche zwischen Schlüsselwerten gezählt werden, nicht aber Vergleiche zwischen Zählervariablen wie sie in Schleifenbedingungen stattfinden.
- Versuchen Sie mit nur **vier** Vergleichen aus vier Werten den maximalen so wie minimalen herauszufinden und erweitern Sie diesen Ansatz dann auf beliebig viele Werte.