



Aufgabe 1

Wiederholung Rekurrenzen

Betrachten Sie das folgende Programm:

```
import Bits

radix_sort xs = radix_sort1 xs 0 (maximum xs)

radix_sort1 xs i m = if 2^i > m then rs
                    else radix_sort1 (ls++rs) (i+1) m
  where ls = [x | x <- xs, testBit x i]
        rs = [x | x <- xs, not (testBit x i)]
```

Welchen Aufwand hat das Sortierverfahren? Beweisen Sie Ihre Aussage.

Lösung zu Aufgabe 1

Der Aufwand ist unbeschränkt!

Angenommen der Aufwand des Programms sei $O(f(n))$.

Sei nun t die Anzahl der durchgeführten `testBit` Aufrufe. Da ein Aufruf von `testBit` Aufwand verursacht genügt es ein Beispiel zu finden mit $t > f(n)$.

Betrachte die Liste $[1, 2^{f(n)+2}]$: Obwohl die Liste nur zwei Elemente enthält, sind $f(n) + 1$ `testBit` Überprüfungen nötig um die Sortierung herzustellen.

Trotzdem ist der Algorithmus in der Praxis oft sehr gut, z.B. zum Sortieren von 64-Bit Integer Zahlen.

Aufgabe 2

Wiederholung Rekurrenzen

Bestimmen Sie eine geschlossenen Form für die folgenden Rekurrenzen und beweisen Sie wenn nötig Ihre Aussage.

- a) $g_0 = 0, g_n = g_{n-1} + n^2$
- b) $g_0 = 0, g_1 = 0, g_n = g_{n-1} + g_{n-2} + 1$

Lösung zu Aufgabe 2

- a) $g_n = \sum_{i=0}^n i^2$ Der Beweis erfolgt einfach durch Induktion.
- b) Betrachtet man die Differenzen der Folge, so erhält man die Fibonacci-Folge F_n . Offensichtlich gilt:

$$g_{n+1} - g_n = F_n.$$

Die Rekurrenz für $n + 1$ eingesetzt ergibt dann:

$$g_n + g_{n-1} + 1 - g_n = F_n$$

also ist

$$g_n = F_{n+1} - 1.$$

Also

$$g_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^{n+1} - 1$$

Aufgabe 3

Wiederholung O-Kalkül

Beweisen oder widerlegen Sie die folgenden Aussagen:

b) $O\left(\frac{n}{\log_2 n}\right) = O(n)$

Lösung zu Aufgabe 3

a) Es gilt offensichtlich:

$$\sqrt{n} \leq c\sqrt[3]{n} \Leftrightarrow n \leq c^6$$

Also falsch!

b) Hier ist

$$\frac{n}{\log_2 n} = cn \Leftrightarrow n \leq 2^c$$

Ebenfalls falsch!

Aufgabe 4

Java, Programmierprojekt

In der Datei `Spielregeln` (enthalten in `Robo.tar.gz`) finden Sie eine Regelbeschreibung eines Spiels. Dieses Spiel sollen Sie in Java implementieren. Um Ihnen die Aufgabe zu erleichtern ist für diese Programmieraufgabe Gruppenarbeit mit bis zu drei Teilnehmern erwünscht. Das Spiel gliedert sich in zwei Teilbereiche: Einen Spielserver und einen Spielclient. Jede Gruppe hat die Wahl welchen Teil sie implementieren will.

Hinweise:

- Sie bekommen für beide Teile ein Programmgerüst auf das Sie aufbauen sollen in `Robo.tar.gz`.

Aufgabe 5

Java, Vererbung

Betrachten Sie das folgende Programm:

```
class A {
    Number n;
    A(Number n) { this.n = n; }
    void method_1() { System.out.println("A_1: " + n); }
    void method_2(Number param) { System.out.println("A_2: " + param); }
}
class B extends A {
    B(Double d) { super(d); }
    void method_1() { System.out.println("B_1: " + n); }
    void method_2(Double param) { System.out.println("B_2: " + param); }
}
class Test {
    public static void main(String[] args) {
        A variableA = new A(new Double(1.0));
        B variableB = new B(new Double(2.0));
        variableA.method_1();
        variableB.method_1();
        ((A)variableB).method_1();
        System.out.println();
        variableA.method_2(new Double(3.0));
        variableB.method_2(new Double(4.0));
        ((A)variableB).method_2(new Double(5.0));
    }
}
```

Interpretieren Sie die Ausgaben des Programms.

Lösung zu Aufgabe 5

fehlt noch!

Aufgabe 6

Es seien A und B Anweisungsfolgen, in denen Ausnahmen des Typs `Exception` ausgelöst werden können. Welche Unterschiede bestehen im Verhalten der folgenden Methoden?

```
void m1() throws Exception {
    A;
    B;
}
void m2() throws Exception {
    try {A;}
    catch (Exception e) {
        B;
    }
    return;
}
void m3() throws Exception {
    try {A;}
    finally {B;}
}
void m4() throws Exception {
    try {A;}
    catch (Exception e) {B;}
    finally { return;}
}
```

Lösung zu Aufgabe 6

- Weder A noch B löst Ausnahme aus: m_1, m_2, m_3 führen erst A , dann B aus. m_4 führt nur A aus.
- A löst Ausnahme aus, B nicht: m_2, m_4 führen erst A , dann B aus, m_3 gibt zusätzlich die Ausnahme an den Aufrufer durch, m_1 führt A aus und gibt die Ausnahme an den Aufrufer durch.
- B löst Ausnahme aus, A nicht: m_1, m_2, m_3 führen erst A , dann B aus, und geben die Ausnahme an den Aufrufer weiter. m_4 führt A aus.
- A, B lösen beide die Ausnahme aus: m_1 führt A aus, und gibt die Ausnahme an den Aufrufer weiter. m_2, m_3, m_4 führen erst A , dann B aus. m_2 und m_3 geben die Ausnahme an den Aufrufer weiter.

Aufgabe 7

Sortieren durch Einfügen

Zeigen Sie, daß der maximale Aufwand von Sortieren durch Einfügen linear ist, d.h. $O(n)$, falls die gegebene Reihung $A[0 : n - 1]$ derart in Teilreihungen $A[j_0 : j_1 - 1], A[j_1 : j_2 - 1], \dots, A[j_{p-1} : j_p]$ (mit $j_0 = 0, j_p = n - 1$) zerlegt werden kann (mit $k = \max\{(j_{i+1} - j_i) \mid i \in \{0, \dots, p-1\}\}$, und $k \ll n, k$ fest), so daß stets gilt:

$$\forall i \in \{0, \dots, p-2\} \forall l \in \{j_i, \dots, j_{i+1} - 1\} \forall m \in \{j_{i+1}, \dots, j_{i+2} - 1\} : A[l] \leq A[m]$$

Lösung zu Aufgabe 7

Der maximale Aufwand für die Anzahl der Vergleiche ist

$$C_{max}(n) = \sum_{l=1}^{n-1} l = \frac{n-1}{2}n$$

für eine Folge der Länge n . Für jede Teilfolge gilt also:

$$C_{max} = \sum_{l=1}^{j_{i+1}-j_i-1} l \leq \sum_{l=1}^k l = \frac{k-1}{2}k.$$

Desweiteren gibt es höchstens n Teilfolgen, daher gilt:

$$C_{max_{ges}} \in O\left(n \cdot \frac{k-1}{2}k\right) = O(n).$$