Constantin Lazari, Marco Wettstein 9. Oktober 2013

1. Grundlagen zu Laufzeitkomplexität

   (a) Nennen Sie ein paar Grössen, von welchen das Laufzeitverhalten eines Algorithmuses abhängen kann

   Lösung:

   - von der Eingabegrösse bzw. Eingabemenge
   - vom Algorithmus selbst
   - vom benötigtem Zeitaufwand der einzelnen Anweisungen

   (b) Wie werden bestehende Abhängigkeiten zwischen der Problemgrösse n und der Laufzeit t eines Algorithmuses angegeben?

   Lösung:
   Mit Hilfe der $\mathcal{O}$-Notation. Kleinere Terme werden dabei vernachlässigt.

   (c) Welche Einflüsse, abgesehen vom Algorithmus, können sich auf die Laufzeit auswirken?

   Lösung:

   - Andere laufende Prozesse
   - Prozessor
   - Temperatur
   - verfügbarer Speicher
   - andere Hardware

   (d) Erklären Sie die 90/10 Regel in eigenen Worten

   Lösung:
   Die Regel besagt im wesentlichen zwei Dinge:

   - 90% der Laufzeit eines Programms werden von nur 10% des Programmcodes verursacht
   - 90% der Entwicklungszeit wird mit Code verbracht, der nur 10% der Anwendungsfälle zum Einsatz kommt (z. B. Abfangen von fehlerhaften Eingaben)

   (e) Ist ein Algorithmus der ein Problem in linearer Zeit löst immer (d. h. für jede Eingabe) schneller als ein Algorithmus der dasselbe Problem in exponentieller Zeit bew ltigt? Begründen Sie ihre Antwort.

Lösung:
Nein, für (sehr) kleine Eingabegrössen, kann der exponentielle durchaus schneller sein. (Bsp: Linear mit der Steigung 5, d. h. $t_P = 5 \cdot n$ erst ab $n \geq 6$ schneller als ein Algorithmus für den $t_P = n^2$ gilt.

2. Landau Notation

   Es sind die Funktionen $G, F : \mathbb{N} \to ?$ gegeben, deren Wachstumsverhalten sich wie folgt abschätzen lässt:

   - $F \in \mathcal{O}(G)$
   - $G \in \mathcal{O}(2^{2^x})$

   Welche der folgenden Aussagen ist wahr, welche sind falsch und von welchen ist es nicht klar, ob sie wahr oder falsch sind. Begründen Sie Ihre Aussagen.

   (a) $F \notin \mathcal{O}(2^{2^x})$

   Lösung:
   Falsch, weil $F \in \mathcal{O}(G)$ und $G \in \mathcal{O}(2^{2^x})$ somit $F \in \mathcal{O}(2^{2^x})$

   (b) $F \in \mathcal{O}(2^x)$

   Lösung:
   Falsch, denn über F ist nur bekannt, dass es nicht wesentlich schneller wächst als 2^{2^x}. F kann (muss aber nicht) schneller wachsen als 2^x

   (c) $G \notin \mathcal{O}(1)$

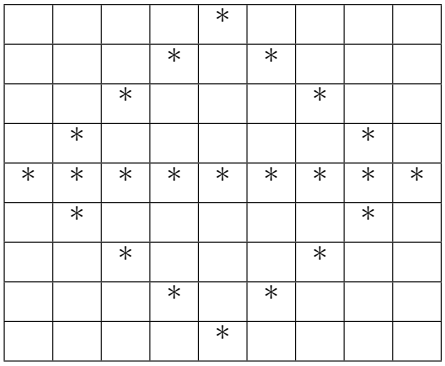
   Lösung:
   Keine Aussage möglich. G wächst zwar nicht wesentlich schneller als 2^{2^x}, eine „untere Schranke“ ist aber nicht gegeben. G könnte also auch konstant sein. ($\mathcal{O}(1) \in \mathcal{O}(2^{2^x})$)

   (d) $\mathcal{O}(F) \subseteq \mathcal{O}(G)$

   Lösung:
   Ja, das kann durchaus so sein.

3. Diamant

   (a) Erstellen Sie einen Algorithmus, der ein gegebenes, zweidimensionales mit `char[][]` - Array mit dem Muster eines auf der Kante stehenden Rechtecks füllt. Das Rechteck soll mit * und ohne Verwendung von Graphikbibliotheken gezeichnet werden.



Das Ihnen übergebene Array ist von der Dimension $n \times n$, wobei n eine ungerade Zahl ist, die in der Grösse variieren kann.Das Array ist an jeder Stelle bereits mit ␣(Leerzeichen) vorinstalliert.

Erstellen Sie einen möglchen schnellen Algorithmus und führen Sie eine Feinanalyse durch.

Vergleichen Sie Ihre Resultate mit denen anderer Studenten. Stellen Sie den Zusammenhang von Arraygrösse zur Lufzeit Ihres Algorithmus graphisch dar. Präsentieren Sie Ihre Analyse in der Übungsstunde.

Lösung:

Das Programm in Coffee-Script:

```
half = (n-1)/2

for x:= 0 to half
  matrix[x, half] = "x"
  matrix[x, half + x] = "x"
  matrix[x, half - x] = "x"
  opposite = n - x - 1
  matrix[opposite, half] = "x"
  matrix[opposite, half + x] = "x"
  matrix[opposite, half - x] = "x"
end
```

Analyse:
(Z = Zuweisung (1.0), AS = Addition/Subtraktion (1.4), M = Multiplikation (2.3), D = Division (8.0), V = Vergleich inkl. Sprung (1.5), I = Indizierung (4.2))

Anweisung	**Anzahl**	**Schritte**
`half = (n - 1) / 2`	1	1 Z, 1 AS, 1 D
`for x:= 0 to half`	$n/2$	$n/2$ V
`matrix[x, half] = "x"`	$n/2$	$n/2$ (Z + I)
`matrix[x, half + x] = "x"`	$n/2$	$n/2$ (Z + I + AS)
`matrix[x, half - x] = "x"`	$n/2$	$n/2$ (Z + I + AS)
`oposite = n - x -1`	$n/2$	$n/2$ (Z + AS + AS)
`matrix[opposite, half] = "x"`	$n/2$	$n/2$ (Z + I)
`matrix[opposite, half + x] = "x"`	$n/2$	$n/2$ (Z + I + AS)
`matrix[opposite, half - x] = "x"`	$n/2$	$n/2$ (Z + I + AS)
`end`		

Der Algorithmus wächst um den Faktor $n/2$ das heisst linear.

4. Fibonacci reloaded

   Benutzen Sie Ihren Algorithmus zur Berechnung der Fibonacci Folge, um die Summe aller Folgenglieder, die kleiner als 4 000 000 und gerade Zahlen sind zu berechnen.

Lösung: