Algorithmen und Datenstrukturen (AD)
Wintersemester 2019/2020
Prof. Dr. Wolfgang Mühlbauer Blatt 01

Technische Hochschule Rosenheim
Technical University of Applied Sciences

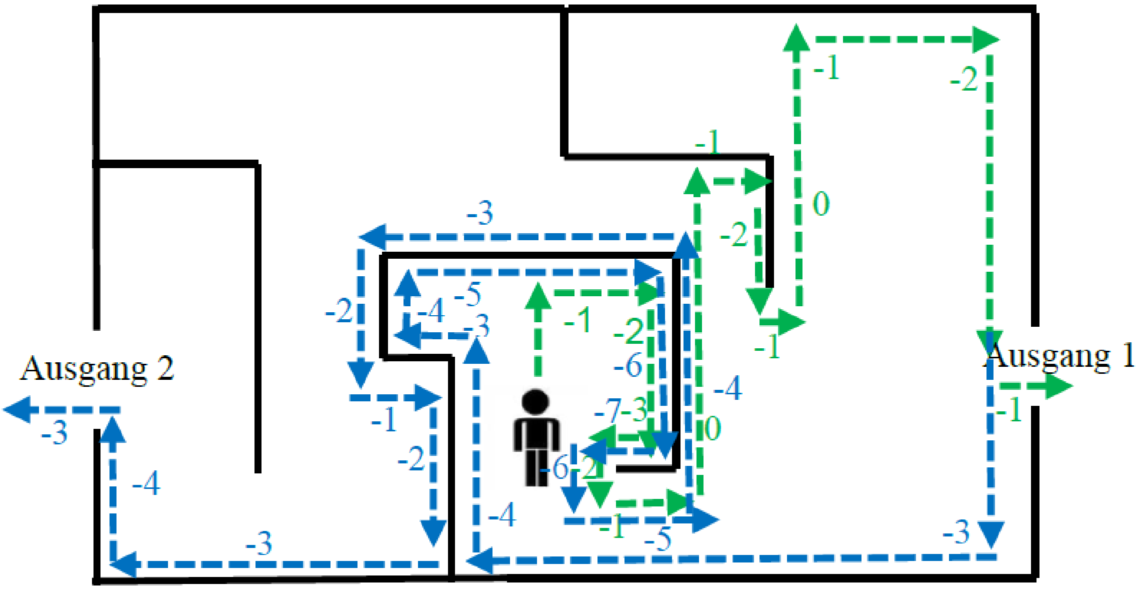
Lösung 01: Pledge, Bubblesort, Landau-Symbole

Aufgabe 1: Pledge Algorithmus

a) Der grüne Weg beschreibt den Fall, dass nur der Ausgang 1 vorhanden ist.

b) Für den Fall, dass nur der Ausgang 2 vorhanden ist, geht man zunächst den gleichen, grünen Weg bis zum verschlossenen Ausgang 1, der Rest des Weges ist in der Abbildung dann blau markiert.

Beachte: Beim Erreichen des Ausgangs muss der Umdrehungszähler nicht zwingend 0 sein. Man kann ja schon Tageslicht sehen.



Aufgabe 2: BubbleSort

a) <35, 22, 10, 51, 48>
<35, 22, 10, 48, 51>
<35, 10, 22, 48, 51>
<10, 35, 22, 48, 51> (jetzt ist die äußere for-Schleife einmal durchgelaufen)
<10, 22, 35, 48, 51> (jetzt ist die äußere for-Schleife das 2. Mal durchgelaufen)
In den weiteren Iterationen sind keine Vertauschungen mehr notwendig!

b) Nach der *i*.ten Iteration der äußeren for-Schleife ist das *i*.-kleinste Element an der richtigen Position (Invariante)

c) <35, 22, 10, 51, 48>
<22, 35, 10, 51, 48>
<10, 22, 35, 51, 48>
[<10, 22, 35, 51, 48>] (hier ändert sich nichts)
<10, 22, 35, 48, 51>

d) Der Worst Case tritt ein, falls das Array absteigend sortiert ist und aufsteigend sortiert werden soll. In diesem Fall sind in der *i*.-ten Iteration der äußeren for-Schleife (Zeile 1) jeweils $n - i$ Vertauschungen notwendig. Es ergibt sich mit der „Gauß-Formel" (ähnlich wie in Vorlesung):

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = n(n-1) - \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{n}{2}$$

Die Worst Case Laufzeit des Algorithmus ist also quadratisch: $\Theta(n^2)$.

e) Bezüglich der asymptotischen Laufzeit im Worst Case unterscheiden sich BubbleSort und InsertionSort kaum.
Zusatz: Ein Nachteil von BubbleSort ist, dass für jede einzelne Vertauschung eine Zuweisung zu einer temporären Variable notwendig ist. Bei InsertionSort muss das nur einmal für jede äußere for-Schleife gemacht werden. Insertionsort ist dann gut, wenn das Eingabearray bereits recht gut sortiert ist, hier könnte Bubblesort ungünstig sein.

f) Implementierung, siehe `BubbleSort.java im` IntelliJ Projekt, Source Verzeichnis. Für das Vertauschen ist eine temporäre Variable nötig. Aufpassen muss man mit den Indizes!

g) Man stellt fest, das eine Verdoppelung der Eingabegröße erhebliche Auswirkungen auf die Laufzeit hat, z.B. 94 ms (für Array mit 8000 Elementen) statt 31 ms (Array mit 4000 Elementen). Solche Messungen sind aus mehreren Gründen nicht unbedingt aussagekräftig:

- Abhängig von Rechnerhardware, Betriebssystem, Programmiersprache
- Außerdem heißt es nicht, dass während der gemessenen Zeit nur das Programm aktiv war. Das Betriebssystem kann auch anderen Prozessen Rechenzeit einräumen.
- Bei einer Verdoppelung von 4000 auf 8000 Elementen, würde man gemäß der Theorie erwarten, dass sich die Laufzeit vervierfacht. Das ist hier aber nicht zwingend der Fall, vielleicht ist die Eingabegröße einfach noch zu klein (nicht wirklich asymptotischer Fall).

Aufgabe 3: Asymptotisches Wachstum von Funktionen

a) $2^{n+1} = O(2^n)$ → Diese Aussage ist wahr!
Um das zu zeigen, muss man Konstanten $c, n_0 > 0$ finden, so dass gilt:
$0 \leq 2^{n+1} \leq c \cdot 2^n$ für alle $n \geq n_0$.
Da $2^{n+1} = 2 \cdot 2^n$ ist, ist diese Bedingung z.B. für $c = 2$ und $n_0 = 1$ erfüllt.

b) $2^{2n} = O(2^n)$ → Diese Aussage ist falsch!
Beweis durch Widerspruch:
Angenommen, es gibt Konstanten $c, n_0 > 0$ mit $0 \leq 2^{2n} \leq c \cdot 2^n$ für alle $n \geq n_0$.
Dann ergibt sich: $2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n \rightarrow 2^n \leq c$.
Es gibt aber keine Konstante c, die größer als **alle** 2^n (für alle n) ist. Die Annahme führt deshalb zu einem Widerspruch und die Behauptung war somit falsch.

c)

A	**B**	**O**	**Ω**	**Θ**
$n+1$	n	Ja	Ja	Ja
$1+\frac{1}{n}$	n^2	Ja	Nein	Nein
$2n^3 - 15n^2 + n$	$4n^{3,5}$	Ja	Nein	Nein
$4\log_2 n$	$\log_{10} n$	Ja	Ja	Ja