Algorithmik

1 Pseudocode

- Entwerfen Sie einen Algorithmus, um Minimum, Maximum, Summe und Durchschnitt aus einer Liste zu berechnen. Implementieren Sie diesen!
- Bilden Sie folgendes Verhalten in Pseudocode ab: Ein Roboter kann nach vorne gehen, sich um 90 Grad drehen und Hindernisse *vor* sich erkennen. Beschreiben Sie einen Algorithmus, der ihn zwischen zwei Wänden hin und her gehen lässt.

2 Komplexität

- Finden Sie ein Best Case und ein Worst Case Beispiel für Insertion Sort. Warum ist genau dieser Fall der Beste/Schlechteste?
- Finden Sie ein Best Case und ein Worst Case Beispiel für den Sortieralgorithmus, den Sie in Übung 2 entworfen haben. Wieviel Speicher verbraucht ihr Algorithmus im Best Case, im Average Case, und im Worst Case? Wie viele Vertauschung und wie viele Vergleiche gibt es? Modifizieren Sie ihre Implementierung, sodass Vertauschungen und Vergleiche gezählt werden, und prüfen Sie damit ihre Behauptung.

3 Zusatzübungen

- Entwerfen Sie einen Algorithmus, mit dem Sie alle Primzahlen kleiner n berechnen können. Sollten Sie keine Idee dazu haben, erkundigen Sie sich über das Sieb des Eratosthenes.
- (SEHR SCHWER!) Implementieren Sie ein Programm, das Graphen darstellt und berechnen Sie nach folgendem Algorithmus (Quelle: `http://en.wikipedia.org/wiki/Bellman_Ford`) den kürzesten Weg von einem Knoten zu allen anderen Knoten:

```
procedure BellmanFord(list vertices, list edges, vertex source)
    // This implementation takes in a graph, represented as lists
    // of vertices and edges, and fills two arrays (distance and
    // predecessor) with shortest-path information

    // Step 1: initialize graph
    for each vertex v in vertices:
        if v is source then distance[v] := 0
        else distance[v] := infinity
        predecessor[v] := null

    // Step 2: relax edges repeatedly
    for i from 1 to size(vertices)-1:
        for each edge (u, v) with weight w in edges:
            if distance[u] + w < distance[v]:
                distance[v] := distance[u] + w
                predecessor[v] := u

    // Step 3: check for negative-weight cycles
```

```
for each edge (u, v) with weight w in edges:
    if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
```