Übungsblatt 5

Willkommen zum Praktikum zu Algorithmen und Datenstrukturen.

Hinweis 1. Dies ist die erste bewertete Übung. Alle Aufgaben sind zu lösen und abzugeben.

- *Abgabe:* Spätester Abgabetermin ist Mittwoch, der **04.06.25, 23:55 Uhr**. Bitte geben Sie über das read.mi System ab. Die Quellen der Lösung geben sie in der Paketstruktur ab. Alle Gruppenmitglieder müssen auf der Lösung draufstehen!
- *Gruppenarbeit:* Bilden sie eine Gruppe mit zwei oder maximal drei Teilnehmern. *Jeder* Gruppenpartner muss sich in *allen* Aufgaben und deren Lösung genau auskennen. Bei Zweifeln kann es Rücksprachen und gegebenenfalls Punktabzug geben.
- *Kopien:* Die Aufgaben müssen je Gruppe individuell und eigenständig gelöst werden. Identische Lösungen verschiedener Abgaben werden bestraft.

Hinweis 2. In dieser Abgabe beschäftigen wir uns mit Sortierverfahren. Lesen Sie bitte dazu auch Hinweis 1 von Blatt3, da Sie für die Abgabe auch die `Sort`-Klasse verwenden. Es ist hilfreich zur Vorbereitung zunächst

Aufgabe 1. Realisieren Sie ein Sortierverfahren das darauf spezialisiert ist nur positive Werte bis zu einer oberen Grenze m zu haben. Es gilt also ein Feld `a` zu sortieren, in dem alle Werte `a[i]` größer gleich 0 und kleiner m sind. Dies kann man erreichen indem man ein neues Feld der Größe m anlegt. In diesem Feld zählen wir an der Stelle e, wie oft ein Element e in `a` vorkommt. Das heißt für jedes Element e in `a` erhöhen wir `a[e]` um 1.

Beispiel: Sortieren Sie das folgende Feld mit 12 Werten.

```
5 3 4 2 0 2 1 4 2 7 3 2
```

Wir haben Werte zwischen 0 und 7, also gilt $m = 8$. Wir initialisieren ein Feld mit den Indizes 0–7 je Eintrag auf den Wert 0. Dann führen wir schrittweise aus `a[5] += 1`, `a[3] += 1`, `a[4] += 1`, und so weiter. Wir erhalten schließlich das folgende Feld.

```
a[0]=1, a[1]=1 , a[2]=4, a[3]=2 , a[4]=2 a[5]=1, a[6]=0, a[7]=1
```

Dabei ist die Summer der Werte dieses Felds 12. Wir schreiben dann in das ursprüngliche Feld die Werte wieder zurück. Dabei schreiben wir so häufig wie der Wert den Index in das nächste Feld. Wir erhalten das folgende Feld.

```
0 1 2 2 2 2 3 3 4 4 5 7
```

Realisieren Sie das Verfahren und testen Sie das Verfahren mit allen kleinen (`runSmall`) und großen Problemen (`runLarge`) aus `Sort`. Notieren Sie die gemessene Zeitdauer für verschiedene obere Grenzen m mit $m \in \{10^k \mid k \in \mathbb{N}, 3 \leq k \leq 8\}$. Beachten Sie, dass die Anzahl der Vergleiche und Vertauschungen nicht relevant ist, da man weder vergleicht noch vertauscht.

Bestimmen Sie Laufzeit und Komplexität ausführlich. Berücksichtigen Sie den besten, schlechtesten und mittleren Fall. Bewerten Sie das Sortierverfahren im Vergleich zu Ihnen bekannten Verfahren. Unterstützen Sie Ihre Aussagen mit den empirischen Ergebnissen.

Aufgabe 2. Realisieren Sie ein Sortierverfahren durch Einfügen, das versucht die weiten Wege des klassischen Sortierens durch Einfügen zu vermeiden. Dazu zerlegt das Verfahren die zu sortierende Sequenz in mehrere Teilsequenzen, die jeweils mit einem Sortieren durch Einfügen sortiert werden. Die Elemente einer zu sortierenden Sequenz sind allerdings keine direkten Nachbarn, sondern Nachbarn, die einen festen Abstand von einander haben. Dabei werden initial hohe Abstände verwendet, was in vielen zu sortierenden Teilsequenzen mündet, später immer kleiner werdende Abstände, bis zum Abstand 1. Der Abstand 1 stellt sicher, dass das Feld am Ende des Verfahrens sortiert ist. Die hohen Abstände vorher sollen helfen große Werte zügig an den Anfang des Felds zu transportieren und umgekehrt kleine Werte schnell ans Ende. Dadurch, dass initial hohe Abstände verwendet werden soll das Sortierverfahren schneller als ein klassisches Sortieren durch Einfügen werden.

Beispiel: Sortieren Sie das folgende Feld mit 12 Werten.

```
98 23 77 33 76 24 95 18 76 23 66 11
```

Initial erhalten wir 6 Teilsequenzen mit je zwei Werten, die wir mit Sortieren durch Einfügen sortieren.

```
98 95 → 95 98    23 18 → 18 23    77 76 → 76 77
33 23 → 23 33    76 66 → 66 76    24 11 → 11 24
```

Dabei werden die Werte nicht umkopiert, sondern bleiben an Ihrer Stelle im Feld. Betrachten wir das ursprüngliche Feld, dann sehen wir

```
95 18 76 23 66 11 98 23 77 33 76 24
```

Nach dem Sortieren mit Abstand 3 von Feldern der Länge 4 erhalten wir

```
23 18 11 33 23 24 95 66 76 98 76 77
```

Nach dem Sortieren mit Abstand 1 von einem Feld der Länge 12 erhalten wir

```
11 18 23 23 24 33 66 76 76 77 95 98
```

Wir sehen an dem Beispiel, dass die 11 sich recht schnell nach links bewegt, mit insgesamt $1+1+2=4$ Vertauschungen. Direkt mit dem Sortieren durch Einfügen hätte man 11 Vertauschungen benötigt. Wir haben dagegen nichts gewonnen bei der zweiten 76, für die wir $1+0+2=3$ Vertauschungen benötigt haben, die jedoch ohne Vertauschungen beim direkten Verfahren am Platz geblieben wäre.

Realisieren Sie das Verfahren und wählen Sie als Abstände zunächst die halbe Feldgröße, dann viertel Feldgröße, und so weiter jeweils nach unten gerundet, bis der Abstand 1 ist. Es ist möglich, dass nicht alle Felder gleich lang sind. Testen Sie das Verfahren mit allen kleinen (`runSmall`) und großen Problemen (`runLarge`) aus `Sort`. Notieren Sie die gemessene Zeitdauer.

Bestimmen Sie nachvollziehbar die Komplexität des Algorithmus. Bewerten Sie das Sortierverfahren im Vergleich zu den Ihnen bekannten Verfahren.

Hinweis 3. Verwenden Sie als Datenstrukturen nur Grundtypen und Felder. Vermeiden Sie globale Variablen (Klassenvariablen). Versuchen Sie unnötige Speicherallokation zu vermeiden. Importieren Sie `adstool.jar` als einzige JAR-Datei in Ihr Projekt. Für die Analysen ist es sinnvoll und ausreichend handschriftliche Anlagen zu verwenden.

```
ReadMI, Medieninformatik, 2.Semester, Algorithmen und Datenstrukturen
```