TU Kaiserslautern – FB Informatik
Dr. Patrick Michel

19.11.2019

# Algorithmen und Datenstrukturen (WS 2019)

# Aufgabenblatt 3

**zu bearbeiten bis: 25.11.19 - 27.11.19**

---

**Aufgabe 3.1 (ArrayHelper - Praxis/Theorie)**
Wir betrachten die Klasse `ArrayHelper`, die dabei hilft mit einem Array (`data`) zu arbeiten. Insbesondere verwaltet die Klasse einen Zähler `next`, der besagt wie viele Elemente im Array gerade gültig sind, bzw. ab welchem Index noch Platz im Array frei ist.

- Man kann ein **bestehendes** Array von einem `ArrayHelper` verwalten lassen, indem man es entweder dem Konstruktor übergibt oder das Array einer bestehenden Instanz mit Hilfe von `handleArray` austauscht.
- Alternativ kann man ein neues, **leeres** Array erstellen, indem man dem Konstruktor die maximale Kapazität übergibt.

Mit Hilfe des `ArrayHelper` kann man u.a.

- schauen wie viele Elemente gerade im Array sind und wie viel es maximal halten kann:
  `size()`, `reserved()`, `empty()`
- auf Elemente im genutzten Teil des Arrays zugreifen:
  `first()`, `last()`, `get(i)`, `min()`
- ein Element in den genutzten Teil des Array an verschiedenen Stellen einfügen:
  `addFront(p)`, `addBack(p)`, `insert(i, p)`, `addSorted(p)`
- ein Element aus dem genutzten Teil des Array entfernen:
  `removeFirst()`, `removeLast()`, `delete(i)`, `removeMin()`
- Elemente im genutzten Teil des Arrays tauschen:
  `swap(a, b)`, `reverse()`
- schauen ob das Array sortiert ist (sucht ein Gegenbeispiel zweier benachbarter Elemente):
  `outOfOrder()`

Wir werden uns nun mit der Klasse, den Operationen und ihren Komplexitäten beschäftigen.

a) Vervollständigen Sie die Methoden, die noch nicht implementiert sind.

b) Annotieren Sie an jede Methode die **best-** und die **worst-case** Komplexität in Abhängigkeit der Menge `n` der bereits enthaltenen Elemente. Wir achten nur auf Array Zugriffe und Vergleiche von Elementen.

**Aufgabe 3.2 (Sortieren - Praxis)**
Implementieren Sie die folgenden Sortiermethoden mit Hilfe der Klasse `ArrayHelper`.

a) `selectionSort` und `insertionSort`: Legen Sie ein neues Ergebnis Array (`result`) an und übertragen Sie im Sinne des Verfahrens die Elemente aus der Eingabe. Beide Arrays verwalten Sie mit Hilfe eines `ArrayHelper`.

b) `bubbleSort`: Verwenden Sie `ArrayHelper`, um das Eingabearray *in-place* zu sortieren.

Um zu sehen, ob größere Beispiele wirklich korrekt sortiert wurden, können Sie die Methode `last_result_out_of_order()` verwenden.
**Anmerkung für die Abgabe:** Laden Sie die Datei `ArrayHelper` in dieser Aufgabe nicht hoch, sie haben im System eine funktionierende Version zur Verfügung.

**Aufgabe 3.3 (ArrayList - Theorie)**
Anstatt begrenzter Arrays wollen wir *beliebig lange* Listen mit einer verbesserten Version von `ArrayHelper` verwalten: `ArrayList`. Wir können dies erreichen, indem wir beim Überschreiten der Kapazität anstatt eine `CapacityReachedException` zu werfen das interne Array vergrößern. Nehmen Sie also an, dass wir das interne Array `data` in der Methode `checkCapacity` in seiner Länge **verdoppeln**, sobald die Kapazität ausgeschöpft ist. Die Methode muss natürlich die Werte aus dem alten Array in das neue übertragen.

a) Welche **best/worst-case** Komplexität hätte dann die Methode `checkCapacity()`?

b) Wie änderen sich die **best/worst-case** Komplexitäten der Methoden `addFront(p)` bzw. `addBack(p)`?

c) Was ist nun die **average-case** Komplexität von `addBack(p)`?
Überlegen Sie sich dazu was passiert, wenn Sie immer mehr und mehr Elemente anfügen und dann die Kosten mitteln.