

# Algorithmen und Datenstrukturen

Dr. M. Lüthi, Dr. G. Röger  
Frühjahrssemester 2018

Universität Basel  
Fachbereich Informatik

## Übungsblatt 6

**Abgabe: 20. April 2018**

Für dieses Übungsblatt benötigen Sie eine Java Umgebung, wie in Übungsblatt 1 beschrieben.  
Falls sie keine IDE verwenden, können Sie ihre Programme von der Kommandozeile mittels

```
.\gradlew -PmainClass="exercise6.SymbolTableExperiments" execute  
.gradlew -PmainClass="exercise6.BSTExperiments" execute
```

starten. Das komplizieren und ausführen der Testfälle erfolgt wie in den vorhergehenden Übungen durch `gradlew build` respektive `gradlew test`.

Als Grundlage für Ihre Implementation verwenden Sie das Projekt im Zip-File "blatt06.zip", welches Sie auf dem Adam-Workspace finden. Die Lösungen zu den Theoriefragen geben Sie als Dokument im PDF Format, mit dem Namen `loesung-6.pdf` ab. Bitte fügen sie diese PDF Datei zur Projektstruktur hinzu. Laden Sie Ihre Lösungen als zip Datei auf Courses hoch.

### Aufgabe 6.1 (Binäre Suchbäume (1 Punkt))

Angenommen, ein bestimmter binärer Suchbaum hat ganze Zahlen zwischen 1 und 10 als Schlüssel und wir suchen nach der 5. Welche der folgenden Sequenzen können nicht die untersuchte Schlüsselfolge sein?

- a. 10, 9, 8, 7, 6, 5
- b. 4, 10, 8, 7, 3, 5
- c. 1, 10, 2, 9, 3, 8, 4, 7, 6, 5
- d. 2, 7, 3, 8, 4, 5
- e. 1, 2, 10, 4, 8, 5

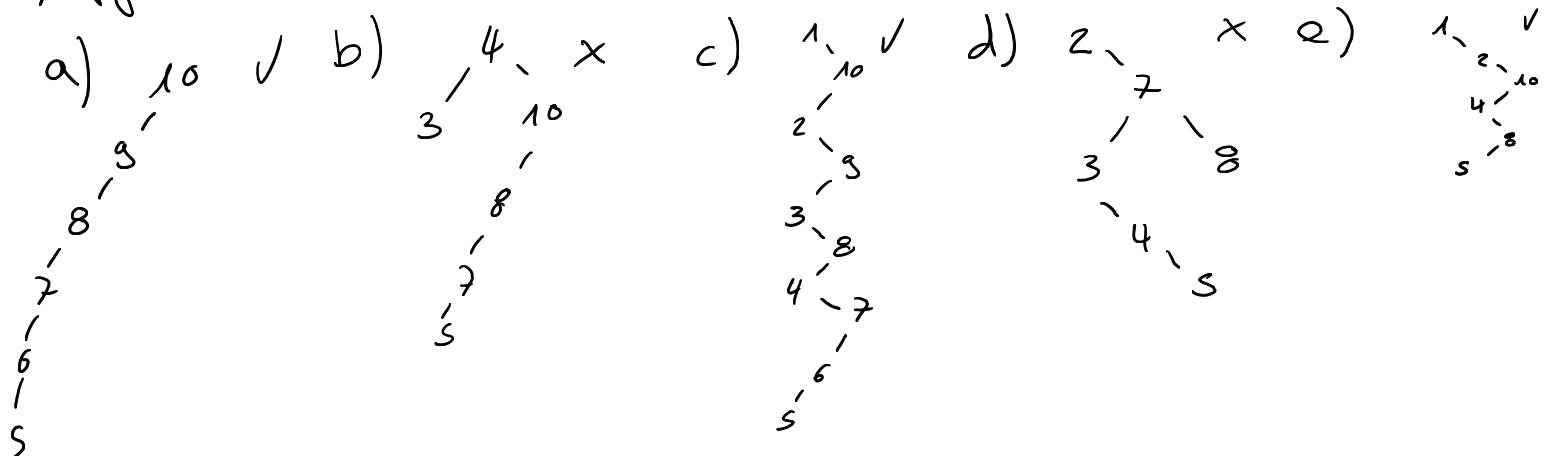
Begründen Sie ihre Antwort.

### Aufgabe 6.2 (Komplexität der Symboltabellen Implementationen (1 + 1 Punkte))

Schreiben Sie ein Programm, welches  $N$  zufällige Schlüssel in drei verschiedene Symboltabellen-Implementationen einfügt: SequentialSearch, BinarySearch und BinarySearchTree. Die Implementation der Symboltabellen sind in den Klassen `SequentialSearchST.java`, `BinarySearchST.java` und `BST.java` gegeben.

- Finden sie Werte von  $N$ , so dass die binäre Suche etwa 10, 100 und 1000 mal schneller ist als die sequentielle Suche. Diskutieren Sie die Ergebnisse. Entsprechen diese Ihren Erwartungen? Stimmen Sie mit den erwarteten Ergebnissen von der Komplexitätsanalyse überein?
- Finden Sie die Werte von  $N$ , für die die Verwendung eines binären Suchbaums zur *Erstellung* einer Symboltabelle von  $N$  zufälligen double-Schlüsseln etwa 10-, 100- und 1000-mal schneller ist als die binäre Suche. Diskutieren sie die Ergebnisse. Entsprechen diese Ihren Erwartungen? Stimmen Sie mit den erwarteten Ergebnissen von der Komplexitätsanalyse überein?

# Aufgabe 1



2a) BS  $\log_2 N$  SS  $N$

$$10: \log_2(512) \approx 9$$

$$512/9 \approx 56 \text{ J.}$$

$$100: \log_2(8000) \approx 12$$

$$8000/12 \approx 666 \text{ J.}$$

$$1000: \log_2(131072) \approx 17$$

$$131072/17 \approx 7710 \text{ J.}$$

$N$  so gross wie  $2^n$  als input für  $N$  siehe Code

2b) BS  $\frac{N}{2}$  BST  $1,39 \log_2 N$

$$10: \frac{300}{2} \quad 1,39 \log(300) \approx 12$$

$$\Rightarrow 12,5 \text{ J.}$$

$$100: \frac{3600}{2} \quad 1,39 \log(3600) \approx 16,5$$

$$\Rightarrow 16,5 \text{ J.}$$

$$1000: \frac{45000}{2} \quad 1,39 \log(45000) \approx 21,5$$

$$\Rightarrow 1040 \text{ J.}$$

Schreiben Sie den Testcode in die Klasse `SymbolTableExperiments.java`. Nutzen Sie die Klasse `exercise6.utils.Stopwatch` um die Zeit zu messen.

*Tip: Um verlässliche Zeitmessungen zu bekommen, müssen Sie die Operationen mehrmals durchführen und den Durchschnitt berechnen.*

**Aufgabe 6.3** (Unit Tests für ordnungsbasierte Methoden, 2 + 1 Punkte)

- Implementieren Sie sinnvolle Unitests für die Methoden `rank`, `floor`, `select` und `keys`, indem Sie die Datei `BSTTests.java` ergänzen.
- In der Implementation haben sich 2 Fehler versteckt. Finden und korrigieren sie diese. Falls keiner ihrer Unitests diesen Fehler abdeckt, schreiben sie einen.

**Aufgabe 6.4** (Binäre Suchbäume und Binärsuche, 1 Punkt)

- Schreiben Sie ein Programm, welches einen Satz von Schlüsseln in einen anfänglich leeren binären Suchbaum einfügt, so dass der erzeugte Baum äquivalent der Binärsuche ist, und zwar insofern als die Folge der Vergleiche bei der Suche nach einem beliebigen Schlüssel im binären Suchbaum die Gleiche ist wie die Folge der Vergleiche bei der Binärsuche für den gleichen Satz von Schlüsseln.

Bitte schreiben Sie Ihre Lösung in die Methode `BSTExperiments.binarySearch`.

**Aufgabe 6.5** (Durchschnittliche Suchpfadlänge, 1 Punkt)

Ergänzen Sie die Implementation der Methode `BST.averagePathLength`, welche die durchschnittliche Suchpfadlänge berechnet. Die durchschnittliche Suchpfadlänge ist definiert als die Summe der Tiefe aller inneren Knoten, dividiert durch die Anzahl innerer Knoten. Testen Sie ihr Programm mittels den entsprechenden Unit Tests.

**Aufgabe 6.6** (Löschen in einem Binärbaum, 2 Punkte)

- Implementieren Sie die Methode `BSTExperiments.testDelete`, welche für einen gegebenen Integer  $N$ , einen zufälligen binären Suchbaum der Größe  $N$  aufbaut. Messen Sie die durchschnittliche Länge eines Pfades im Baum und geben sie diese aus. Danach soll Ihre Methode in eine Schleife eintrete, in der es einen zufällig ausgewählten Schlüssel löscht und dafür anschliessend einen zufälligen Schlüssel einfügt. Die Schleife soll  $N^2$ -mal durchlaufen werden. Nach Abarbeitung der Schleife messen Sie nochmals die durchschnittliche Länge eines Pfades im Baum und geben sie diese aus. Führen Sie Ihr Programm für  $N = 10^2$ ,  $10^3$  und  $10^4$ . Was beobachten Sie? Diskutieren Sie die Ergebnisse.