Testat 3

Technische Hochschule Ingolstadt

Prüfungssemester: SoSe 2026
Studiengang: User Experience Design (UXD)
Prüfungsfach: Softwareentwicklung 2 (SWE2)
Dozent: Prof. Dr. Simon Nestler

Name	Datum	Uhrzeit

Anmerkungen

Hinweise zur Bearbeitung

- Die Bearbeitungszeit beträgt 45 Minuten, die maximal erreichbare Punktzahl beträgt 40 Punkte.
- Bitte beantworten Sie die einzelnen Fragen direkt in dem jeweils für die Beantwortung vorgesehenen Antwortfeld.
- Der Ihnen für die Beantwortung einer Frage zur Verfügung gestellte Platz wurde anhand der erwarteten Ausführlichkeit Ihrer Beantwortung bemessen.

- **Bitte verwenden Sie keine Hilfsmittel (Entwicklungsumgebung, ChatGPT, Google, Lösung, etc.) um eine realistsiche Bewertung Ihres Leistungsstandes zu erhalten.**

- bitte eigenständig (im Rahmen der Besprechung ausfüllen –

Punkte	Prozent	Note

Durch das Unterschreiben mit meinem Namen (in Druckbuchstaben) bestätige ich, dass ich mich bei der Selbstbewertung fair und realistisch eingeschätzt habe.

Aufgabe 1 (20P): In dieser Aufgabe beschäftigen Sie sich mit einem einfachen Sortierverfahren: *InsertSort*. Ziel ist es, eine Liste von *Integer*-Werten schrittweise zu sortieren, indem jedes Element an die richtige Position eingefügt wird. Zusätzlich wird die Laufzeit der Sortierung gemessen. Die Klasse *Sortierer* stellt dafür grundlegende Funktionalitäten bereit, während die konkrete Sortierlogik in der Klasse *InsertSortierer* umgesetzt wird.

a) Warum wird vor der Sortierung eine Kopie der Liste erzeugt, anstatt direkt auf der übergebenen Liste zu arbeiten? (4P)

b) Was passiert bei *InsertSort* im schlimmsten Fall, wenn die Liste absteigend sortiert ist? Beschreiben Sie kurz das Verhalten des Algorithmus. (4P)

c) Was misst die Methode *System.currentTimeMillis()* und warum eignet sie sich für die Messung der Sortierdauer? (2P)

d) Implementieren Sie die Methode *sortiere(List<Integer> liste)* mit dem *InsertSort*-Verfahren. Sie müssen innerhalb der Methode die Liste nicht mehr kopieren, sondern können die Liste direkt aufsteigend sortieren. (8P)

e) Warum ist *InsertSort* für kleine oder nahezu sortierte Listen oft effizient? (2P)

Aufgabe 2 (20P): In dieser Aufgabe implementieren Sie das effizientere Sortierverfahren *QuickSort*. Dabei wird eine Liste in kleinere Teilprobleme zerlegt, die rekursiv gelöst werden. Ein zentrales Konzept von *QuickSort* ist das sogenannte *Pivot*-Element.

a) Was bedeutet es, dass ein Algorithmus rekursiv arbeitet? Beschreiben Sie das Prinzip kurz. (4P)

b) Welche Aufgabe hat das *Pivot*-Element im *QuickSort*-Algorithmus? (4P)

c) Warum muss in der *QuickSort*-Rekursion ein Abbruchfall (Liste mit Länge 0 oder 1) definiert werden? (2P)

d) Implementieren Sie den Anfang der Methode *sortiere(List<Integer> liste).* Erzeugen Sie eine Kopie der Liste und geben Sie diese direkt zurück, wenn sie höchstens ein Element enthält. (2P)

e) Ergänzen Sie die Methode so, dass das erste Element als *Pivot* gewählt und aus der Liste entfernt wird. Legen Sie außerdem die Listen *ersteListe*, *zweiteListe* und *sortierteListe* an. (4P)

f) Implementieren Sie die Aufteilung der Elemente: Alle Werte kleiner oder gleich dem *Pivot* kommen in *ersteListe*, alle anderen in *zweiteListe*. (4P)