Begonnen am	Donnerstag, 13. Januar 2022, 14:21
Status	Beendet
Beendet am	Donnerstag, 13. Januar 2022, 14:38
Verbrauchte Zeit	16 Minuten 39 Sekunden

Erreichte Punkte 1,00 von 2,00

Sie haben mehrere Code-Fragmente gegeben. Jedes Code-Fragment beinhaltet einen Kommentar, der den Inhalt des Arrays `data` nach der Ausführung des Code-Fragments beschreiben soll. Beurteilen Sie die Richtigkeit der Kommentare für beliebige Werte von n > 1!

Hinweis: Mit [a, b] wird das Intervall von a (inklusive) bis b (inklusive) beschrieben.

Richtig	Falsch		

○ ✗	◉ ✓	`int[] data = new int[n];`
		`for (int i = 0; i < data.length / 2; i++) {`
		`  data[i] = i;`
		`  data[data.length - 1 - i] = i;`
		`}`
		`// data enthält jeden Wert genau 2 mal`
			✓

◉ ✓	○ ✗	`int[] data = new int[n];`
		`for (int i = 0; i < data.length; i++) {`
		`  data[i] = i + 1;`
		`}`
		`// data enthält Werte aus dem Intervall [1, n]`
			✓

◉ ✗	○ ✓	`int[] data = new int[n];`
		`data[0] = n;`
		`int curr = 1;`
		`int prev = curr - 1;`
		`while (curr < data.length) {`
		`  data[curr++] = data[prev++] - 1;`
		`}`
		`// data enthält Werte aus dem Intervall [0, n]`
			✗

○ ✗	◉ ✓	`int[] data = new int[n];`
		`for (int i = 0; i < data.length; i++) {`
		`  data[i] = i;`
		`}`
		`// data enthält Werte aus dem Intervall [1, n - 1]`
			✓

```
int[] data = new int[n];
for (int i = 0; i < data.length / 2; i++) {
    data[i] = i;
    data[data.length - 1 - i] = i;
}
// data enthält jeden Wert genau 2 mal
```
: Falsch

```
int[] data = new int[n];
for (int i = 0; i < data.length; i++) {
    data[i] = i + 1;
}
// data enthält Werte aus dem Intervall [1, n]
```
: Richtig

```
int[] data = new int[n];
data[0] = n;
int curr = 1;
int prev = curr - 1;
while (curr < data.length) {
    data[curr++] = data[prev++] - 1;
}
// data enthält Werte aus dem Intervall [0, n]
```
: Falsch

```
int[] data = new int[n];
for (int i = 0; i < data.length; i++) {
    data[i] = i;
}
// data enthält Werte aus dem Intervall [1, n - 1]
```
: Falsch

Richtig

Erreichte Punkte 2,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static int[][] generate(int[][] input) {
    int maxcol = 0;
    for (int[] ints : input) {
        if (ints.length > maxcol) {
            maxcol = ints.length;
        }
    }
    int[][] r = new int[input.length][];
    for (int i = 0; i < input.length; i++) {
        r[i] = new int[maxcol];
        for (int j = 0; j < maxcol; j++) {
            r[i][j] = input[i][j % input[i].length];
        }
    }
    return r;
}
```

Gehen Sie davon aus, dass als Vorbedingung `input != null` gilt. Beurteilen Sie die Richtigkeit der folgenden Aussagen zu dem obigen Code-Fragment!

Richtig	Falsch		
○ ✗	◉ ✓	Die Methode erzeugt ein Array, das weniger Elemente als das übergebene Array `input` hat.	✓
◉ ✓	○ ✗	In der foreach-Schleife wird die Anzahl der Elemente in der längsten Zeile bestimmt.	✓
○ ✗	◉ ✓	Unterschiedlich lange Zeilen von `input` führen nie zu einer Ausnahme.	✓
○ ✗	◉ ✓	Die Methode erzeugt ein Array, bei dem alle Elemente größer 0 sind.	✓

Die Methode erzeugt ein Array, das weniger Elemente als das übergebene Array `input` hat.: Falsch
In der foreach-Schleife wird die Anzahl der Elemente in der längsten Zeile bestimmt.: Richtig
Unterschiedlich lange Zeilen von `input` führen nie zu einer Ausnahme.: Falsch
Die Methode erzeugt ein Array, bei dem alle Elemente größer 0 sind.: Falsch

Erreichte Punkte 2,00 von 2,00

Die Methode `insert(String input, String sep)` fügt jeweils zwischen zwei Zeichen von `input` den String `sep` ein. Die Vorbedingung lautet: `input != null` und `sep != null`.

Beispiele für korrektes Verhalten:

```
insert("", "-") liefert einen leeren String zurück
insert("X", "-") liefert "X" zurück
insert("Test", "-") liefert "T-e-s-t" zurück
insert("Test", "::") liefert "T::e::s::t" zurück
```

Sie haben unterschiedliche rekursive Implementierungen gegeben. Beurteilen Sie die Richtigkeit dieser Implementierungen!

Richtig	Falsch	
◉	○	`private static String insert(String input, String sep) { if (input.length() <= 1) { return input; } else { return insert(input.substring(0, input.length() - 1), sep) + sep + input.cl } }`
○	◉	`private static String insert(String input, String sep) { if (input.isEmpty()) { return input; } else { return input.charAt(0) + sep + insert(input.substring(1), sep); } }`
◉	○	`private static String insert(String input, String sep) { if (input.length() < 2) { return input; } else { return input.charAt(0) + sep + insert(input.substring(1), sep); } }`
◉	○	`private static String insert(String input, String sep) { if (input.length() <= 1) { return input; } else { return input.charAt(0) + sep + insert(input.substring(1), sep); } }`

```
private static String insert(String input, String sep) {
    if (input.length() <= 1) {
        return input;
    } else {
        return insert(input.substring(0, input.length() - 1), sep) + sep + input.charAt(input.len
    }
}
```

: Richtig

```
private static String insert(String input, String sep) {
    if (input.isEmpty()) {
        return input;
    } else {
        return input.charAt(0) + sep + insert(input.substring(1), sep);
    }
}
```

: Falsch

```
        return input;
    } else {
        return input.charAt(0) + sep + insert(input.substring(1), sep);
    }
}
```

: Richtig

```
private static String insert(String input, String sep) {
    if (input.length() <= 1) {
        return input;
    } else {
        return input.charAt(0) + sep + insert(input.substring(1), sep);
    }
}
```

: Richtig

Frage 4

Richtig

Erreichte Punkte 2,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static String extract(char[] sequence, int start, int end, char omit) {
    if (start >= end) {
        return "";
    }
    if (sequence[start] == omit) {
        return extract(sequence, start + 1, end, omit);
    }
    return sequence[start] + extract(sequence, start + 1, end, omit);
}
```

Beurteilen Sie die Richtigkeit der folgenden Aussagen!

Richtig	Falsch		
○ ✗	◉ ✓	Die Methode besitzt zwei Basisfälle.	✓
◉ ✓	○ ✗	Das Zeichen an der Position `end` ist nicht in der Rückgabe vorhanden.	✓
○ ✗	◉ ✓	Es wird immer zumindest ein Zeichen in dem Bereich `start` bis `end` entfernt.	✓
○ ✗	◉ ✓	Es handelt sich hier um eine verzweigte Rekursion.	✓

Die Methode besitzt zwei Basisfälle.: Falsch
Das Zeichen an der Position `end` ist nicht in der Rückgabe vorhanden.: Richtig
Es wird immer zumindest ein Zeichen in dem Bereich `start` bis `end` entfernt.: Falsch
Es handelt sich hier um eine verzweigte Rekursion.: Falsch

Erreichte Punkte 2,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static void fill(int[][] matrix, int n, int r, int c) {
    if (r >= n || c >= n) {
        return;
    }
    matrix[r][c] = (r + c) % 2;
    fill(matrix, n, r + 1, c);
    fill(matrix, n, r, c + 1);
}
```

Die Vorbedingung lautet: `n > 0` und `matrix` ist eine quadratische Matrix mit `n` Zeilen und `n` Spalten.

Beurteilen Sie die Richtigkeit der folgenden Aussagen.

Richtig	Falsch		
◉ ☑	○ ✖	Nach dem Aufruf `fill(matrix, matrix.length, 0, 0)` enthalten alle Zellen der Hauptdiagonale von `matrix` (d.h., die Zellen `matrix[i][i]` für `0 <= i < matrix.length`) die Zahl 0.	✔
○ ✖	◉ ☑	Nach dem Aufruf der Methode enthält `matrix` mindestens eine 1.	✔
○ ✖	◉ ☑	Nach dem Aufruf `fill(matrix, matrix.length, 0, 0)` enthalten alle Zellen der Gegendiagonale von `matrix` (d.h., die Zellen `matrix[i][matrix.length-1-i]` für `0 <= i < matrix.length`) die Zahl 0.	✔
◉ ☑	○ ✖	Die Methode macht Gebrauch von verzweigter Rekursion.	✔

Nach dem Aufruf `fill(matrix, matrix.length, 0, 0)` enthalten alle Zellen der Hauptdiagonale von `matrix` (d.h., die Zellen `matrix[i][i]` für `0 <= i < matrix.length`) die Zahl 0.: Richtig
Nach dem Aufruf der Methode enthält `matrix` mindestens eine 1.: Falsch
Nach dem Aufruf `fill(matrix, matrix.length, 0, 0)` enthalten alle Zellen der Gegendiagonale von `matrix` (d.h., die Zellen `matrix[i][matrix.length-1-i]` für `0 <= i < matrix.length`) die Zahl 0.: Falsch
Die Methode macht Gebrauch von verzweigter Rekursion.: Richtig

Erreichte Punkte 2,00 von 2,00

Sie haben mehrere Code-Fragmente mit dazugehörigen Kommentaren gegeben. Jeder Kommentar beschreibt die kleinste obere Schranke für die Laufzeit in Abhängigkeit von n. Dabei steht "n^x" für n^x. Beurteilen Sie die Richtigkeit der Kommentare!

Richtig	Falsch		

◉	○	`// Die Laufzeit liegt in O(n^3)`
		`int sum = 0;`
		`for (int i = 0; i < n * n; i++) {`
		`  for (int j = 0; j < n / 2; j++) {`
		`  sum += i + j;`
		`  }`
		`}`
			✔

◉	○	`// Die Laufzeit liegt in O(n)`
		`int sum = 0;`
		`for (int i = 0; i < n; i++) {`
		`  for (int j = 0; j < n; j += n / 2) {`
		`  sum += i + j;`
		`  }`
		`}`
			✔

◉	○	`// Die Laufzeit liegt in O(n log n)`
		`int sum = 0;`
		`for (int i = 0; i < n; i++) {`
		`  int j = 1;`
		`  while (j < n) {`
		`  sum += 1;`
		`  j *= 2;`
		`  }`
		`}`
			✔

○	◉	`// Die Laufzeit liegt in O(n log n)`
		`int sum = 0;`
		`for (int i = 0; i < n; i++) {`
		`  for (int j = 0; j < n / 2; j += 2) {`
		`  sum += i + j;`
		`  }`
		`}`
			✔

```
// Die Laufzeit liegt in O(n^3)
int sum = 0;
for (int i = 0; i < n * n; i++) {
    for (int j = 0; j < n / 2; j++) {
        sum += i + j;
    }
}
```
: Richtig

```
// Die Laufzeit liegt in O(n)
int sum = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j += n / 2) {
        sum += i + j;
    }
}
```
: Richtig

```
for (int i = 0; i < n; i++) {
    int j = 1;
    while (j < n) {
        sum += 1;
        j *= 2;
    }
}
```

: Richtig

```
// Die Laufzeit liegt in O(n log n)
int sum = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n / 2; j += 2) {
        sum += i + j;
    }
}
```

: Falsch

Frage **7**

Richtig

Erreichte Punkte 2,00 von 2,00

Gegeben sei die folgende Variante des linearen Suchalgorithmus.

```
private static final int NOT_FOUND = -1;

private static int variant(int[] data, int key) {
    int last = data[data.length - 1];
    data[data.length - 1] = key;
    int i = 0;
    while (data[i] != key) {
        i++;
    }
    data[data.length - 1] = last;
    if (i >= data.length - 1 && key != data[data.length - 1]) {
        return NOT_FOUND;
    }
    return i;
}
```

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu dieser Variante!

Richtig	Falsch		
◉☑	○✖	Die Laufzeit liegt in $O(n)$, wobei n der Länge von `data` entspricht.	✔
○✖	◉☑	Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann wird die letzte gefundene Position zurückgeliefert.	✔
○✖	◉☑	Befindet sich der gesuchte Wert `key` an der ersten Stelle im Array, dann wird -1 zurückgeliefert.	✔
◉☑	○✖	Für alle möglichen `data`-Arrays mit einer Länge größer 0 wird ein ganzzahliger Wert zurückgeliefert.	✔

Die Laufzeit liegt in $O(n)$, wobei n der Länge von `data` entspricht.
: Richtig
Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann wird die letzte gefundene Position zurückgeliefert.: Falsch
Befindet sich der gesuchte Wert `key` an der ersten Stelle im Array, dann wird -1 zurückgeliefert.: Falsch
Für alle möglichen `data`-Arrays mit einer Länge größer 0 wird ein ganzzahliger Wert zurückgeliefert.: Richtig

Erreichte Punkte 2,00 von 2,00

Gegeben sei der aus den Vorlesungsfolien bekannte Sortieralgorithmus Bubblesort.

```
private static void bubbleSort(int[] data) {
    for (int i = 0; i < data.length - 1; i++) {
        for (int j = 0; j < data.length - i - 1; j++) {
            if (data[j] > data[j + 1]) {
                exchange(data, j, j + 1);
            }
        }
        // Zeile A
    }
}

private static void exchange(int[] a, int i, int j) {
    int swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Sie müssen überprüfen, ob für eine Anfangsbelegung von `data` die gegebene veränderte Belegung von `data` nach irgendeinem Durchlauf der inneren Schleife in Zeile A auftreten kann. Beispiel:

- Anfangsbelegung von data: [1, 3, 5, 6, 4, 2]
- Mögliche Belegung: [1, 3, 4, 2, 5, 6] (nach 2 Durchläufen der inneren Schleife)
- Nicht möglich: [1, 3, 5, 2, 4, 6]

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu den Belegungen für unterschiedliche Ausgangsbelegungen von `data`!

Richtig	Falsch		
◉☑	○✖	• Ausgangsbelegung: [6, 5, 4, 1, 3, 2] • Mögliche Belegung: [4, 1, 3, 2, 5, 6]	✔
◉☑	○✖	• Ausgangsbelegung: [1, 3, 5, 6, 4, 2] • Mögliche Belegung: [1, 3, 2, 4, 5, 6]	✔
◉☑	○✖	• Ausgangsbelegung: [1, 4, 5, 6, 3, 2] • Mögliche Belegung: [1, 3, 2, 4, 5, 6]	✔
○✖	◉☑	• Ausgangsbelegung: [1, 3, 5, 6, 4, 2] • Mögliche Belegung: [1, 3, 5, 2, 4, 6]	✔

- Ausgangsbelegung: [6, 5, 4, 1, 3, 2]
- Mögliche Belegung: [4, 1, 3, 2, 5, 6]

: Richtig

- Ausgangsbelegung: [1, 3, 5, 6, 4, 2]
- Mögliche Belegung: [1, 3, 2, 4, 5, 6]

: Richtig

- Ausgangsbelegung: [1, 4, 5, 6, 3, 2]
- Mögliche Belegung: [1, 3, 2, 4, 5, 6]

: Richtig

- Ausgangsbelegung: [1, 3, 5, 6, 4, 2]
- Mögliche Belegung: [1, 3, 5, 2, 4, 6]

: Falsch

Erreichte Punkte 0,00 von 2,00

Gegeben sei der aus den Vorlesungsfolien bekannte Sortieralgorithmus Selectionsort.

```
private static void selectionSort(int[] data) {
    for (int i = 0; i < data.length - 1; i++)     // Zeile A
    {
        int min = i;
        for (int j = i + 1; j < data.length; j++) // Zeile B
        {
            if (data[j] < data[min])              // Zeile C
            {
                min = j;
            }
        }
        exchange(data, i, min);                   // Zeile D
    }
}

private static void exchange(int[] data, int i, int j) {
    int swap = data[i];
    data[i] = data[j];
    data[j] = swap;
}
```

Beurteilen Sie für jede der unten angeführten Veränderungen, ob folgende Aussage richtig oder falsch ist: Der Algorithmus terminiert nach der Veränderung noch immer korrekt und sortiert alle möglichen `data`-Arrays mit einer Länge größer 0 noch immer aufsteigend.

Richtig	Falsch		
◉✖	○☑	Zeile B: `for (int j = 0; j < data.length; j++)`	✖
◉☑	○✖	Zeile D: `exchange(data, min, i);`	✔
○☑	◉✖	Zeile C: `if (data[j] <= data[min])`	✖
○✖	◉☑	Zeile A: `for (int i = 0; i <= data.length; i++)`	✔

Zeile B: `for (int j = 0; j < data.length; j++)` : Falsch
Zeile D: `exchange(data, min, i);` : Richtig
Zeile C: `if (data[j] <= data[min])` : Richtig
Zeile A: `for (int i = 0; i <= data.length; i++)` : Falsch

Erreichte Punkte 2,00 von 2,00

Es soll ein Algorithmus verwendet werden, der überprüft, ob ein Element in einem int-Array `data` der Größe n doppelt vorkommt (Duplikat erkennen).

Ihnen steht folgende Implementierung (Methode `check1`) zur Verfügung:

```
private static boolean check1(int[] data) {
    for (int i = 0; i < data.length; i++) {
        for (int j = i + 1; j < data.length; j++) {
            if (data[i] == data[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Daneben gibt es noch eine weitere Implementierung (Methode `check2`):

```
private static boolean check2(int[] data) {
    int[] help = data.clone();
    sort(help);
    for (int i = 0; i < help.length - 1; i++) {
        if (help[i] == help[i + 1]) {
            return true;
        }
    }
    return false;
}
```

Bei der Methode `check2` wird eine Methode `sort` verwendet, die einen Sortieralgorithmus implementiert, dessen Laufzeit im Worst-Case in $O(n \log n)$ liegt.

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu den beiden Methoden!

Richtig	Falsch		
◉☑	○✖	Die Laufzeit von `check2` liegt im Worst-Case in $O(n \log n)$.	✔
◉☑	○✖	Beide Methoden liefern `true` zurück, wenn zumindest ein Duplikat im Array enthalten ist.	✔
◉☑	○✖	Die Laufzeit von `check1` liegt im Worst-Case in $O(n^2)$.	✔
○✖	◉☑	Beide Methoden verändern den Inhalt des Arrays `data`.	✔

Die Laufzeit von `check2` liegt im Worst-Case in $O(n \log n)$.
: Richtig
Beide Methoden liefern `true` zurück, wenn zumindest ein Duplikat im Array enthalten ist.: Richtig
Die Laufzeit von `check1` liegt im Worst-Case in $O(n^2)$.
: Richtig
Beide Methoden verändern den Inhalt des Arrays `data`.: Falsch

◄ Tuwel-Test zur Vorbereitung (TUWEL-Test 1)

Direkt zu:

1. TUWEL-Test (26.11., A) ►