| | |
|---|---|
| **Begonnen am** | Donnerstag, 13. Januar 2022, 18:20 |
| **Status** | Beendet |
| **Beendet am** | Donnerstag, 13. Januar 2022, 19:19 |
| **Verbrauchte Zeit** | 59 Minuten 18 Sekunden |
| **Bewertung** | **15,00** von 20,00 (**75**%) |

Erreichte Punkte 0,00 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];`<br>`for (int i = 0; i < data.length - 1; i += 2) {`<br>`  data[i] = i + 1;`<br>`  data[i + 1] = i + 1;`<br>`}`<br>`// data enthält nur ungerade Werte >= 1` | ✖ |
| ○ ☑ | ◉ ✖ | `int[] data = new int[n];`<br>`for (int i = data.length - 1; i > 0; i--) {`<br>`  data[i] = i;`<br>`}`<br>`// data enthält nur Werte aus dem Intervall [0, n - 1]` | ✖ |
| ○ ✖ | ◉ ☑ | `int[] data = new int[n];`<br>`int i = 0;`<br>`while (i < data.length) {`<br>`  data[i] = i;`<br>`  ++i;`<br>`}`<br>`// data enthält nur Werte aus dem Intervall [1, n - 1]` | ✔ |
| ○ ✖ | ◉ ☑ | `int[] data = new int[n];`<br>`for (int i = 0; i < data.length - 1; i++) {`<br>`  data[i] = i + 1;`<br>`}`<br>`// data enthält nur Werte aus dem Intervall [0, n - 2]` | ✔ |

```
int[] data = new int[n];
for (int i = 0; i < data.length - 1; i += 2) {
    data[i] = i + 1;
    data[i + 1] = i + 1;
}
// data enthält nur ungerade Werte >= 1
```
: Falsch

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

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

```
int[] data = new int[n];
for (int i = 0; i < data.length - 1; i++) {
    data[i] = i + 1;
}
// data enthält nur Werte aus dem Intervall [0, n - 2]
```
: Falsch

Erreichte Punkte 1,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static int[][] truncate(int[][] input, int size) {
    int[][] result = new int[input.length][];
    for (int row = 0; row < input.length; row++) {
        int[] newRow = new int[size];
        for (int col = 0; col < Math.min(size, input[row].length); col++) {
            newRow[col] = input[row][col];
        }
        result[row] = newRow;
    }
    return result;
}
```

Gehen Sie davon aus, dass als Vorbedingung `input != null`, `input[i] != null` für alle gültigen i und `size >= 0` gilt. Beurteilen Sie die Richtigkeit der folgenden Aussagen!

| Richtig | Falsch | | |
|---|---|---|---|
| ◉ | ○ | Hat `size` den Wert 0, dann enthält das zurückgegebene Array keine numerischen Werte. | ✔ |
| ○ | ◉ | Ist in `input` der Wert 0 nicht vorhanden, dann kann er auch nie in `result` vorhanden sein. | ✘ |
| ◉ | ○ | Ohne die Vorbedingungen können Ausnahmen (Exceptions) auftreten. | ✔ |
| ○ | ◉ | Die Methode kann auch eine `null`-Referenz zurückliefern. | ✔ |

Hat `size` den Wert 0, dann enthält das zurückgegebene Array keine numerischen Werte.: Richtig
Ist in `input` der Wert 0 nicht vorhanden, dann kann er auch nie in `result` vorhanden sein.: Falsch
Ohne die Vorbedingungen können Ausnahmen (Exceptions) auftreten.: Richtig
Die Methode kann auch eine `null`-Referenz zurückliefern.: Falsch

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)`<br>`int sum = 0;`<br>`for (int i = 0; i < n * n; i++) {`<br>`  for (int j = 0; j < 100; j++) {`<br>`  sum += i + j;`<br>`  }`<br>`}` | ✔ |
| ○ ✗ | ◉ ☑ | `// Die Laufzeit liegt in O(n^2)`<br>`int sum = 0;`<br>`for (int i = 0; i < 10; i++) {`<br>`  for (int j = 0; j < 10; j++) {`<br>`  sum += n;`<br>`  }`<br>`}` | ✔ |
| ○ ✗ | ◉ ☑ | `// Die Laufzeit liegt in O(n)`<br>`int sum = 0;`<br>`for (int i = 0; i < n * n; i++) {`<br>`  for (int j = 0; j < n / n; j++) {`<br>`  sum += i + j;`<br>`  }`<br>`}` | ✔ |
| ◉ ☑ | ○ ✗ | `// Die Laufzeit liegt in O(n)`<br>`int sum = 0;`<br>`for (int i = 0; i < 10; i++) {`<br>`  for (int j = 0; j < n; j++) {`<br>`  sum += i + j;`<br>`  }`<br>`}` | ✔ |

```
// Die Laufzeit liegt in O(n)
int sum = 0;
for (int i = 0; i < n * n; i++) {
    for (int j = 0; j < 100; j++) {
        sum += i + j;
    }
}
```
: Falsch

```
// Die Laufzeit liegt in O(n^2)
int sum = 0;
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        sum += n;
    }
}
```
: Falsch

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

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < n; j++) {
        sum += i + j;
    }
}
```

: Richtig

Erreichte Punkte 1,00 von 2,00

Die Methode `div(int a, int b)` gibt zurück, wie oft b in a enthalten ist.

Beispiele für korrektes Verhalten:

```
div(0, 4) liefert 0 zurück
div(7, 4) liefert 1 zurück
div(28, 4) liefert 7 zurück
```

Die Vorbedingung lautet: a >= 0, b > 0.

Sie haben unterschiedliche rekursive Implementierungen gegeben. Beurteilen Sie die Richtigkeit dieser Implementierungen bezüglich Rückgabe und Termination!

| Richtig | Falsch | | |
|---|---|---|---|
| ◉☑ | ○✖ | `private static int div(int a, int b) { if (a >= b) { return 1 + div(a - b, b); } else { return 0; } }` | ✔ |
| ○✖ | ◉☑ | `private static int div(int a, int b) { if (a < b) { return 0; } else { return 1 + div(b - a, a); } }` | ✔ |
| ○✖ | ◉☑ | `private static int div(int a, int b) { if (a <= b) { return 1; } else { return 1 + div(a - b, b); } }` | ✔ |
| ◉✖ | ○☑ | `private static int div(int a, int b) { if (a <= b) { return 0; } else { return 1 + div(a - b, b); } }` | ✖ |

```
private static int div(int a, int b) {
    if (a >= b) {
        return 1 + div(a - b, b);
    } else {
        return 0;
    }
}
```
: Richtig

```
private static int div(int a, int b) {
    if (a < b) {
        return 0;
    } else {
        return 1 + div(b - a, a);
    }
}
```
: Falsch

```
        return 1;
    } else {
        return 1 + div(a - b, b);
    }
}
```

: Falsch

```
private static int div(int a, int b) {
    if (a <= b) {
        return 0;
    } else {
        return 1 + div(a - b, b);
    }
}
```

: Falsch

Frage **5**

Richtig

Erreichte Punkte 2,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static boolean check(int n, int k) {
    if (n <= 9) {
        return n == k;
    } else if (n % 10 == k) {
        return true;
    } else {
        return check(n / 10, k);
    }
}
```

Die Vorbedingung lautet: n >= 0, k >= 0 && k <= 9.

Beurteilen Sie die Richtigkeit der folgenden Aussagen!

| Richtig | Falsch | | |
|---|---|---|---|
| ○ ✗ | ◉ ☑ | Die Methode liefert `false` zurück, falls die Ziffer `k` in der Zahl `n` mehrmals vorkommt. | ✔ |
| ◉ ☑ | ○ ✗ | Es handelt sich hier um eine endrekursive Methode. | ✔ |
| ◉ ☑ | ○ ✗ | Es wird hier eine lineare Rekursion verwendet. | ✔ |
| ◉ ☑ | ○ ✗ | Die Methode besitzt zwei Basisfälle. | ✔ |

Die Methode liefert `false` zurück, falls die Ziffer `k` in der Zahl `n` mehrmals vorkommt.: Falsch
Es handelt sich hier um eine endrekursive Methode.: Richtig
Es wird hier eine lineare Rekursion verwendet.: Richtig
Die Methode besitzt zwei Basisfälle.: Richtig

Erreichte Punkte 1,00 von 2,00

Sie haben folgende Methode gegeben:

```
private static void generate(String input, int n) {
    if (n <= 0) {
        System.out.println(input);
    } else {
        generate(input + "A", n - 1);
        generate(input + "B", n - 1);
    }
}
```

Vorbedingung: n >= 0.

Beurteilen Sie die Richtigkeit der folgenden Aussagen.

| Richtig | Falsch | | |
|---|---|---|---|
| ○ ✗ | ◉ ☑ | Das Ergebnis wird am Return-Pfad generiert. | ✓ |
| ◉ ✗ | ○ ☑ | Die Methode wird nie einen leeren String ausgeben. | ✗ |
| ◉ ☑ | ○ ✗ | Der String `AA` kann als erste Ausgabe bei der rekursiven Abarbeitung erzeugt werden. | ✓ |
| ◉ ☑ | ○ ✗ | Es wird hier eine verzweigte Rekursion verwendet. | ✓ |

Das Ergebnis wird am Return-Pfad generiert.: Falsch
Die Methode wird nie einen leeren String ausgeben.: Falsch
Der String `AA` kann als erste Ausgabe bei der rekursiven Abarbeitung erzeugt werden.: Richtig
Es wird hier eine verzweigte Rekursion verwendet.: 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 search(int[] data, int key) {
    int i = 0;
    boolean present = false;
    while (!present && i < data.length) {
        if (data[i] == key) {
            present = true;
        }
        i++;
    }
    if (present) {
        return i - 1;
    } else {
        return NOT_FOUND;
    }
}
```

Beurteilen Sie die Richtigkeit der folgenden Aussagen zu dieser Variante!

| Richtig | Falsch | | |
|---|---|---|---|
| ◉☑ | ○✖ | Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann wird die erste gefundene Position zurückgeliefert. | ✔ |
| ◉☑ | ○✖ | Die Laufzeit im Worst-Case liegt in $O(n)$, wobei $n$ der Länge von `data` entspricht. | ✔ |
| ◉☑ | ○✖ | Die Methode terminiert korrekt für alle möglichen `data`-Arrays mit einer Länge >= 0. | ✔ |
| ○✖ | ◉☑ | Befindet sich der gesuchte Wert `key` an der ersten Stelle im Array, dann wird -1 zurückgeliefert. | ✔ |

Kommt der gesuchte Wert `key` in `data` mehrmals vor, dann wird die erste gefundene Position zurückgeliefert.: Richtig
Die Laufzeit im Worst-Case liegt in $O(n)$, wobei $n$ der Länge von `data` entspricht.
: Richtig
Die Methode terminiert korrekt für alle möglichen `data`-Arrays mit einer Länge >= 0.: Richtig
Befindet sich der gesuchte Wert `key` an der ersten Stelle im Array, dann wird -1 zurückgeliefert.: Falsch

Erreichte Punkte 2,00 von 2,00

Gegeben sei der aus den Vorlesungsfolien bekannte Sortieralgorithmus Insertionsort.

```
private static void insertionSort(int[] data) {
    for (int i = 1; i < data.length; i++) {
        for (int j = i; j > 0 && data[j] < data[j - 1]; j--) {
            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 mögliche Belegung von `data` nach irgendeinem Durchlauf der inneren Schleife in Zeile A auftreten kann. Beispiel:

- Anfangsbelegung von data: [1, 8, 5, 6, 4, 2]
- Mögliche Belegung: [1, 5, 6, 8, 4, 2] (nach 3 Durchläufen der inneren Schleife)
- Nicht möglich: [1, 5, 8, 6, 2, 4]

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

| Richtig | Falsch | | |
|---|---|---|---|
| ◉☑ | ○✖ | • Ausgangsbelegung: [6, 5, 4, 1, 2, 3]<br>• Mögliche Belegung: [1, 2, 4, 5, 6, 3] | ✔ |
| ◉☑ | ○✖ | • Ausgangsbelegung: [4, 5, 7, 2, 1, 6]<br>• Mögliche Belegung: [1, 2, 4, 5, 7, 6] | ✔ |
| ◉☑ | ○✖ | • Ausgangsbelegung: [4, 3, 7, 6, 2, 1]<br>• Mögliche Belegung: [2, 3, 4, 6, 7, 1] | ✔ |
| ○✖ | ◉☑ | • Ausgangsbelegung: [7, 8, 4, 3, 2, 5]<br>• Mögliche Belegung: [3, 4, 7, 2, 8, 5] | ✔ |

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

: Richtig

- Ausgangsbelegung: [4, 5, 7, 2, 1, 6]
- Mögliche Belegung: [1, 2, 4, 5, 7, 6]

: Richtig

- Ausgangsbelegung: [4, 3, 7, 6, 2, 1]
- Mögliche Belegung: [2, 3, 4, 6, 7, 1]

: Richtig

- Ausgangsbelegung: [7, 8, 4, 3, 2, 5]
- Mögliche Belegung: [3, 4, 7, 2, 8, 5]

: Falsch

Erreichte Punkte 2,00 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 = i; j < data.length; j++)` | ✔ |
| ○✖ | ◉☑ | Zeile C: `if (data[j] > data[min])` | ✔ |
| ◉☑ | ○✖ | Zeile A: `for (int i = 0; i < data.length; i++)` | ✔ |
| ◉☑ | ○✖ | Zeile B: `for (int j = data.length - 1; j > i; j--)` | ✔ |

Zeile B: `for (int j = i; j < data.length; j++)` : Richtig
Zeile C: `if (data[j] > data[min])` : Falsch
Zeile A: `for (int i = 0; i < data.length; i++)` : Richtig
Zeile B: `for (int j = data.length - 1; j > i; j--)` : Richtig

Erreichte Punkte 2,00 von 2,00

Sie haben ein Array `data` gegeben, zu dem Sie folgende Informationen haben:

- Das Array beinhaltet $n-1$ Werte (ist also nicht gleich `null`).
- Es gibt keine Duplikate.
- Die Werte liegen im Intervall $[1, n]$, d. h. einer der Werte in diesem Intervall fehlt.
- Es gilt immer: $n \geq 1$.

Es soll ein Algorithmus verwendet werden, der überprüft, welches Element fehlt.

Ihnen steht folgende Implementierung (Methode `find1`) zur Verfügung:

```
private static int find1(int[] data) {
    boolean found;
    for (int i = 1; i <= data.length + 1; i++) {
        found = false;
        for (int datum : data) {
            if (datum == i) {
                found = true;
                break;
            }
        }
        if (!found) {
            return i;
        }
    }
    return -1;
}
```

Daneben gibt es noch eine weitere Implementierung (Methode `find2`):

```
private static int find2(int[] data) {
    int[] help = data.clone();
    sort(help);
    for (int i = 0; i < help.length; i++) {
        if (help[i] != i + 1) {
            return i + 1;
        }
    }
    return -1;
}
```

Bei der Methode `find2` 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 `find2` liegt im Worst-Case in $O(n \log n)$. | ✔ |
| ○✖ | ◉☑ | Beide Methoden liefern -1 zurück, falls das Array `data` leer ist. | ✔ |
| ◉☑ | ○✖ | Die Laufzeit von `find1` liegt im Worst-Case in $O(n^2)$. | ✔ |
| ○✖ | ◉☑ | Beide Methoden verändern den Inhalt des Arrays `data`. | ✔ |

Die Laufzeit von `find2` liegt im Worst-Case in $O(n \log n)$.
: Richtig
Beide Methoden liefern -1 zurück, falls das Array `data` leer ist.: Falsch
Die Laufzeit von `find1` liegt im Worst-Case in $O(n^2)$.
: Richtig
Beide Methoden verändern den Inhalt des Arrays `data`.: Falsch

◄ 1. TUWEL-Test (03.12., A)

Direkt zu:

2. TUWEL-Test (13.01. B) ►