Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = { {5, 2, 4}, {2, 7, 3}, {9, 5, 8} };
int[][] test2 = { {1} };
int[][] test3 = { {6, 7, 8}, {7, 5, 3, 1}, {2} };
String seq1 = "ABBAACBA";
```

Implementieren Sie folgende Methoden:

- `int[][] generate(int[][] input)` erzeugt aus `input` ein neues zweidimensionales Array und retourniert dieses. Dabei werden die einzelnen Zeilen von `input` kopiert, aber von der zweiten Zeile weg immer ein Element weniger. Bei der letzten Zeile wird daher nur mehr das erste Element kopiert. Danach werden die Zeilen (außer die letzte) in umgekehrter Reihenfolge mit ansteigender Größe (zwei Elemente von der vorletzten Zeile, drei Elemente von der vorvorletzten Zeile usw.) kopiert. Zuletzt wird daher wieder die erste Zeile mit allen Elementen kopiert.
  Vorbedingung(en): `input.length > 0, input.length == input[i].length` für alle gültigen `i`.
  Wird die Methode z.B. mit `test1` aufgerufen, entsteht folgendes Array:

5	2	4
2	7	
9		
2	7	
5	2	4

- `void reorder(int[][] input)` dreht den Inhalt aller Arrayeinträge innerhalb jeder Zeile von `input` um.
  Vorbedingung(en): `input.length > 0, input[i].length > 0` für alle gültigen `i`.
  Wird die Methode mit dem zuvor erzeugten Ergebnis aufgerufen, entsteht folgendes Array:

4	2	5
7	2	
9		
7	2	
4	2	5

- `boolean isPresentNTimes(String sequence, char marker, int count)` gibt `true` zurück, wenn das Zeichen `marker` genau `count`-Mal im String `sequence` vorkommt. Ansonsten wird `false` retourniert.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `sequence != null` und `count >= 0`

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = generate(test1)`	[[5, 2, 4], [2, 7], [9], [2, 7], [5, 2, 4]]
`result2 = generate(test2)`	[[1]]
`reorder(result1)`	[[4, 2, 5], [7, 2], [9], [7, 2], [4, 2, 5]]
`reorder(result2)`	[[1]]
`reorder(test3)`	[[8, 7, 6], [1, 3, 5, 7], [2]]
`isPresentNTimes(seq1, 'A', 4)`	true
`isPresentNTimes(seq1, 'A', 3)`	false
`isPresentNTimes(seq1, 'A', 5)`	false
`isPresentNTimes(seq1, 'Z', 0)`	true

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`generate`	Korrekte Anzahl an Zeilen	1
	Korrekte Erstellung des Arrays (beide Dimensionen)	3
	Korrekte Längen aller Zeilen	3
	Korrekter Inhalt in den Zeilen	3
	Korrekte Handhabung der Randfälle	1
`reorder`	Korrekte Schleifen	2
	Korrekter Tausch des Inhalts	2
	Korrekter Inhalt im Array	3
	Korrekte Handhabung der Randfälle	1
`isPresentNTimes`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall vorhanden	1
	Basisfall korrekt	1
	Fortschritt der Rekursion vorhanden	1
	Fortschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	2
	Korrekte Handhabung der Randfälle	1
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = { {1, 2, 0, -1, -2, 3}, {-1, 2, 3}, {0, 0}, {}, {4, 5, -1} };
int[][] test2 = { {1, 2, 3}, {4, 5, 2}, {-2, -3, 2, -1}, {3, 2, 1, 5}, {4, 5, 1, 4} };
int[][] test3 = { {1, -1, 2, -2, 3}, {1, 2, 3}, {-3, -1, -2} };
char[] age1 = {'d', 'u', '-', 'd', 'u', '-', 'd', 'a', '-', 'd', 'a'};
char[] age2 = {'m', 'a', 'm', 'a', '!', 'n', 'e', 'i', 'n'};
```

Implementieren Sie folgende Methoden:

- `int[][] generate(int[][] input)` gibt ein neues Array zurück, welches dieselbe Struktur wie `input` besitzt. Jede Zeile der Kopie enthält zuerst alle negativen Einträge der entsprechenden Zeile in `input`, und anschließend alle positiven Werte (inkl. 0). Die Reihenfolge innerhalb der negativen und positiven Werte wird nicht verändert.
  Vorbedingung(en): `input != null`.
  Wird die Methode z.B. mit `test3` aufgerufen, entsteht folgendes Array:

-1	-2	1	2	3
1	2	3		
-3	-1	-2		

- `void fill(int[][] target, int[] values, int[] times)` überschreibt die Werte in `target` zeilenweise mit den Werten in `values`, wobei `times` angibt, wie oft jeder Wert in `values` wiederholt werden soll, bevor der nächste verwendet wird. Die Arrays `values` und `times` dürfen dabei nicht verändert werden.
  Vorbedingung(en): `target != null`; `values.length == times.length`; `times[i] > 0` für alle gültigen i; die Summe der Werte in `times` entspricht der Anzahl der Einträge in `target`.
  Wird die Methode mit `target=test3`, `values=test2[0]` und `times=test2[1]` aufgerufen, entsteht folgendes Array:

1	1	1	1	2
2	2	2		
2	3	3		

- `String extractString(char[] sequence, int start, int end, char omit)` liefert einen String bestehend aus den Zeichen in `sequence` zwischen den Indizes `start` (inklusive) und `end` (exklusive) zurück, wobei alle Vorkommen des Zeichens `omit` ausgelassen werden.
  Diese Methode muss rekursiv implementiert werden. Es dürfen **keine** Methoden der Klasse String verwendet werden, Operatoren sind aber erlaubt!
  Vorbedingung(en): `sequence != null`, `start < end`, `start` und `end-1` sind gültige Indizes von `sequence`

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = generate(test1)`	[[-1, -2, 1, 2, 0, 3], [-1, 2, 3], [0, 0], [], [-1, 4, 5]]
`result2 = generate(test2)`	[[1, 2, 3], [4, 5, 2], [-2, -3, -1, 2], [3, 2, 1, 5], [4, 5, 1, 4]]
`fill(test1, test2[3], test2[4])`	[[3, 3, 3, 3, 2, 2], [2, 2, 2], [1, 5], [], [5, 5, 5]]
`fill(result1, test2[2], test2[4])`	[[-2, -2, -2, -2, -3, -3], [-3, -3, -3], [2, -1], [], [-1, -1, -1]]
`fill(test3, test2[2], test2[3])`	[[-2, -2, -2, -3, -3], [2, -1, -1], [-1, -1, -1]]
`extractString(age1, 0, age1.length, '-')`	dududada
`extractString(age1, 1, 7, 'u')`	-d-d
`extractString(age2, 0, 5, 'a')`	mm!
`extractString(age2, 5, age2.length, 'n')`	ei

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`generate`	Korrekte Anzahl an Zeilen	1
	Korrekte Erstellung des Arrays (beide Dimensionen)	3
	Korrekte Längen aller Zeilen	3
	Korrekter Inhalt in den Zeilen	3
	Korrekte Handhabung der Randfälle	1
`fill`	Korrekte Reihenfolge der Werte	2
	Korrekte Wiederholung der Werte	3
	Korrekter Inhalt in allen Arrays	2
	Korrekte Handhabung der Randfälle	1
`extractString`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall vorhanden	1
	Basisfall korrekt	1
	Fortschritt der Rekursion vorhanden	1
	Fortschritt der Rekursion korrekt	1
	Korrekte Rückgabe	2
	Korrekte Handhabung der Randfälle	1
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `isEmpty`, `length`, `substring`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] data0 = {{3, 0}, {0, 1}, {2, 2}};
int[][] data1 = {{0, 1, 0, 0, 1, 0}, {}, {2, 2, 2, 2, 0, 1}};
int[] target1 = {0, 0, 0};
int[] target2 = {9, 9, 9, 9};
```

Implementieren Sie folgende Methoden:

- **`int[][] labelPath(int n, int[][] points)`** erstellt ein neues quadratisches Array der Länge `n` und gibt dieses zurück. Jede Zeile in `points` beschreibt einen Punkt in einem zweidimensionalen Array. Die Spalte 0 steht dabei immer für den Zeilenindex und die Spalte 1 für den Spaltenindex eines Punktes. Das Rückgabearray enthält an jedem Punkt in `points` den Wert -1. An allen anderen Punkten enthält das Rückgabearray den Wert `n`.
  Vorbedingung(en): `n >= 1`, `points != null`, `0 <= points[i][j] < n` für alle gültigen Indizes `i,j`.
  Wird die Methode z.B. mit `n=4` und `points=data0` aufgerufen, entsteht folgendes Array:

4	-1	4	4
4	4	4	4
4	4	-1	4
-1	4	4	4

- **`void findMatches(int[][] data, int[] pattern, int[] target)`** bestimmt für jede Zeile `i` in `data`, wie oft die Folge der Werte in `pattern` vorkommt. Die jeweilige Anzahl wird in `target` am Index `i` abgelegt.
  Vorbedingung(en): `pattern.length > 0`, `target.length >= data.length`, `data != null`, `data[i] != null` für alle gültigen Indizes `i`.
  Wird die Methode mit `data=data1`, `pattern={0, 1}` und `target=target1` aufgerufen, entsteht folgendes Array:

2	0	1

- **`String insertMiddle(String input, String seps)`** fügt Zeichen aus `seps` in `input` ein. In der Mitte der Rückgabe befindet sich das erste Zeichen von `seps`. In der Mitte beider Hälften befindet sich dann das zweite Zeichen von `seps`, usw. Diese Vorgehensweise wird wiederholt, solange noch Zeichen zu vergeben sind und `input` in kleinere Teile geteilt werden kann. Ansonsten wird `input` zurückgegeben. Falls die Mitte nicht eindeutig bestimmt ist, wird so geteilt, dass die linke Hälfte die kürzere ist. Die Buchstaben aus `seps` werden bei der Bestimmung der Mitte nicht einbezogen.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `input != null`, `seps != null`

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`labelPath(3, new int[][] {})`	[[3, 3, 3], [3, 3, 3], [3, 3, 3]]
`labelPath(4, data0)`	[[4, -1, 4, 4], [4, 4, 4, 4], [4, 4, -1, 4], [-1, 4, 4, 4]]
`findMatches(data0, data0[1], target1)`	[0, 1, 0]
`findMatches(data1, data0[1], target1)`	[2, 0, 1]
`findMatches(data1, data0[2], target2)`	[0, 0, 3, 9]
`insertMiddle("XY", "abc")`	XaY
`insertMiddle("01234", "abc")`	0b1a2b3c4
`insertMiddle("01234567890123", "./-")`	0-12/34-56.7-89/01-23

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`labelPath`	Korrekte Erstellung des Arrays (beide Dimensionen)	1
	Einträge `n` korrekt	2
	Einträge `-1` korrekt	3
	Array enthält nur `-1` oder `n`	1
	Korrekte Handhabung von `n=1`, Mehrfachvorkommen in `points`	1
`findMatches`	Korrekte Schleifen	3
	Einzelne Vergleiche mit `pattern` korrekt	1
	Gefundene Werte korrekt in `target` eingetragen	2
	Korrekte Reihenfolge in `target`	1
	Nicht betroffene Indizes in `target` bleiben unverändert	1
	Korrekte Handhabung von zu großem / nicht vorhandenem `pattern`	1
`insertMiddle`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall vorhanden	1
	Basisfall korrekt	2
	Forschritt der Rekursion vorhanden	1
	Forschritt der Rekursion korrekt	2
	Korrekte Aufteilung des Strings	1
	Korrekte Rückgabe	2
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{5, 2, 4}, {8, 5, 4}, {9, 6, 8, 7}};
int[][] test2 = {{0, 1, 2}, {0, 1, 2}, {0, 1, 2}};
int[][] test3 = {{6}, {2, 4}, {2, 4}, {2, 4}, {4, 2}};
int[] seq1 = {4, 3, 2, 1, 10, 5, 5, 5};
```

Implementieren Sie folgende Methoden:

- `int[][] rearrange(int[][] inputArray)` erzeugt aus `inputArray` ein neues zweidimensionales Array und retourniert dieses. Das neue Array hat in jeder Zeile genau 4 Spalten und seine Werte werden zeilenweise mit den Werten aus `inputArray` befüllt. Dabei werden die Werte immer abwechselnd von links oder rechts beginnend in eine Zeile geschrieben. In der ersten Zeile wird links gestartet. Wenn das neue Array nicht vollständig befüllt werden kann (d.h. wenn die Anzahl der Elemente in `inputArray` nicht durch 4 teilbar ist), werden die übrigen Array-Einträge auf 0 gesetzt.
  Hinweis: Es müssen alle Elemente aus `inputArray` in das neue Array übernommen werden, die korrekte Zeilenanzahl ergibt sich somit aus der Gesamtanzahl der Elemente in `inputArray`.
  Vorbedingung(en): `inputArray.length > 0`, `inputArray[i] != null` für alle gültigen Indizes `i`.
  Wird die Methode z.B. mit `test1` aufgerufen, entsteht folgendes Array:

5	2	4	8
6	9	4	5
8	7	0	0

- `void label(int[][] inputArray)` setzt alle Werte einer Zeile auf -9, wenn die darauf folgende Zeile dieselbe Länge hat und genau dieselben Werte in derselben Reihenfolge aufweist.
  Vorbedingung(en): `inputArray.length > 0`.
  Wird die Methode z.B. mit `test3` aufgerufen, entsteht folgendes Array:

6	
-9	-9
-9	-9
2	4
4	2

- `int findMaxOppositeSum(int[] sequence, int start, int end)` berechnet die Summe jeweils gegenüberliegender Elemente im Array `sequence` zwischen den Indizes `start` und `end` (also die Summe der Elemente an der Stelle `start` und `end`, die Summe der Elemente an der Stelle `start+1` und `end-1` usw.) und gibt deren Maximum zurück.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `sequence.length() > 1`, alle Werte in `sequence` sind `>=0`, `start < end`, die Anzahl der Elemente im Intervall [`start`,`end`] ist eine gerade Zahl, `start` und `end` sind gültige Indizes von `sequence`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = rearrange(test1)`	[[5, 2, 4, 8], [6, 9, 4, 5], [8, 7, 0, 0]]
`result2 = rearrange(test3)`	[[6, 2, 4, 2], [4, 4, 2, 4], [2, 0, 0, 0]]
`rearrange(new int[][]{{}})`	[]
`label(test1)`	[[5, 2, 4], [8, 6, 4], [9, 6, 8, 7]]
`label(test2)`	[[-9, -9, -9], [-9, -9, -9], [0, 1, 2]]
`label(test3)`	[[6], [-9, -9], [-9, -9], [2, 4], [4, 2]]
`findMaxOppositeSum(seq1, 0, 7)`	11
`findMaxOppositeSum(seq1, 0, 5)`	13
`findMaxOppositeSum(seq1, 4, 7)`	15

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`rearrange`	Korrekte Anzahl an Zeilen und Spalten	2
	Korrekte Schleifen	2
	Korrekte Werte	1
	Korrekte Richtung	3
`label`	Korrekte Schleifen	3
	Korrekte Überprüfung auf Gleichheit der Zeilen	4
	Richtige Zeilen auf -9 gesetzt	2
	Korrekter Inhalt im Array	1
`findMaxOppositeSum`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall vorhanden	1
	Basisfall korrekt	1
	Forschritt der Rekursion vorhanden	1
	Forschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty` .
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{0, 2, 4}, {2, 0, 0}, {0, 0, 1}};
int[][] test2 = {{1, 2, 3}, {1, 2, 3, 4, 5}, {1, 2, 3}, {1, 2, 3, 4, 5}};
int[][] test3 = {{2}, {0, 7}, {6, 7, 8}, {6, 0}, {0, 0}};
String seq1 = "ABA";
```

Implementieren Sie folgende Methoden:

- **`int[][] removeLeadingZeros(int[][] inputArray)`** erzeugt aus `inputArray` ein neues zweidimensionales Array und retourniert dieses. Das neue Array übernimmt jede Zeile aus dem `inputArray` und entfernt führende Nullen.
  Vorbedingung(en): `inputArray.length > 0`.
  Wird die Methode z.B. mit `test1` aufgerufen, entsteht folgendes Array:

2	4	
2	0	0
1		

- **`void mask(int[][] inputArray, int[] rows, int[] cols)`** setzt alle Werte in `inputArray` auf 0, wenn deren Zeilenindex in `rows` **und** deren Spaltenindex in `cols` vorkommt.
  Vorbedingung(en): `inputArray.length > 0`, die Werte in `rows` und `cols` sind `>=0` und aufsteigend sortiert.
  Wird die Methode z.B. mit `inputArray=test2`, `rows={1,2,3}` und `cols={0,1,4}` aufgerufen, entsteht folgendes Array:

1	2	3		
0	0	3	4	0
0	0	3		
0	0	3	4	0

- **`String replicateCharacters(String sequence, String repSequence)`** gibt einen neuen String zurück, bei dem das i-te Zeichen im String `sequence` zusätzlich wiederholt wird, wenn der i-te Eintrag im String `repSequence` eine 1 ist.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `sequence != null`, `repSequence != null`
  `sequence.length() == repSequence.length()`, alle Zeichen in `repSequence` sind Ziffern im Bereich 0 bis 1.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = removeLeadingZeros(test1)`	[[2, 4], [2, 0, 0], [1]]
`result2 = removeLeadingZeros(test3)`	[[2], [7], [6, 7, 8], [6, 0], []]
`mask(test2,new int[]{1,2,3},new int[]{0,1,4})`	[[1, 2, 3], [0, 0, 3, 4, 0], [0, 0, 3], [0, 0, 3, 4, 0]]
`mask(test3,new int[]{0,2,4},new int[]{0,1})`	[[0], [0, 7], [0, 0, 8], [6, 0], [0, 0]]
`mask(test1,new int[]{},new int[]{0,1})`	[[0, 2, 4], [2, 0, 0], [0, 0, 1]]
`replicateCharacters(seq1, "010")`	ABBA
`replicateCharacters("SAMBA", "10001")`	SSAMBAA

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`removeLeadingZeros`	Richtige Dimensionen	1
	Korrekte Schleifen	2
	Korrekte Längen aller Zeilen	3
	Korrekter Inhalt in den Zeilen	3
`mask`	Korrekte Schleifen	4
	Korrekte Elemente auf 0 gesetzt	5
`replicateCharacters`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall vorhanden	1
	Basisfall korrekt	1
	Forschritt der Rekursion vorhanden	1
	Forschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{5}, {5, 7, 9}, {8, 5}, {}};
int[][] test2 = {{1, 2}, {1, 2, 3}, {1, 2, 3, 4}};
int[][] test3 = {{}, {1, 2, 3, 4}, {1}};
int[] seq = {1, 2, -5, 3, -1, 6, -3, 3};
```

Implementieren Sie folgende Methoden:

- `int[][] getRectangular(int[][] input)` liefert ein neues Array zurück, das rechteckig ist (alle Zeilen haben dieselbe Länge). Die Länge jeder Zeile entspricht dabei der Länge der längsten Zeile von `input`, die Anzahl der Zeilen ist gleich wie bei `input`. Jede Zeile von `input` wird in die entsprechende Zeile des neuen Arrays kopiert, wobei kürzere Zeilen solange wiederholt werden, bis das Ende der Zeile erreicht ist. Leere Zeilen in `input` werden im neuen Array mit lauter Nullen befüllt.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` für alle gültigen `i`.
  Wird die Methode z.B. mit dem Parameter `test1` aufgerufen, entsteht folgendes Array:

5	5	5
5	7	9
8	5	8
0	0	0

- `void removeEntry(int[][] input, int col)` entfernt aus jeder Zeile von `input` das Element mit Spaltenindex `col`. Ist an einem Spaltenindex kein Element vorhanden, soll die Zeile nicht verändert werden.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` für alle gültigen `i`, `col >= 0`.
  Wird die Methode z.B. mit `test2` und `col = 2` aufgerufen, entsteht folgendes Array:

1	2	
1	2	
1	2	4

- `boolean isAlternating(int[] seq, int index)` überprüft, ob die Zahlenreihe `seq` ab der Stelle `index` (inklusive) alternierend ist. Bei einer alternierenden Zahlenreihe gibt es einen ständigen Vorzeichenwechsel, d.h., es gibt keine zwei aufeinander folgenden Zahlen mit demselben Vorzeichen. Ist die Zahlenreihe alternierend, soll `true` zurückgeliefert werden, andernfalls `false`.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `seq != null`, `seq` enthält keine 0, `index` beschreibt einen gültigen Index von `seq`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = getRectangular(test1)`	[[5, 5, 5], [5, 7, 9], [8, 5, 8], [0, 0, 0]]
`result2 = getRectangular(test2)`	[[1, 2, 1, 2], [1, 2, 3, 1], [1, 2, 3, 4]]
`result3 = getRectangular(test3)`	[[0, 0, 0, 0], [1, 2, 3, 4], [1, 1, 1, 1]]
`result4 = getRectangular(new int[][]{{}})`	[[]]
`removeEntry(test2, 2)`	[[1, 2], [1, 2], [1, 2, 4]]
`removeEntry(test3, 0)`	[[], [2, 3, 4], []]
`isAlternating(seq, 0)`	false
`isAlternating(seq, 1)`	true
`isAlternating(seq, 6)`	true
`isAlternating(seq, 7)`	true

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`getRectangular`	Korrekte Bestimmung der längsten Zeile	2
	Korrekte Dimensionen des neuen Arrays	1
	Befüllung der Zeilen: korrekter Ansatz	2
	Befüllung der Zeilen: korrektes Ergebnis	3
	Korrekte Behandlung leerer Zeilen	1
`removeEntry`	Korrekte Zeilenlängen abhängig von `col`	2
	Entfernen des Eintrags: korrekter Ansatz	3
	Entfernen des Eintrags: korrektes Ergebnis	4
`isAlternating`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall/Basisfälle vorhanden	1
	Basisfall/Basisfälle korrekt	1
	Fortschritt der Rekursion vorhanden	1
	Fortschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{5, 7, 5, 7}, {5}, {0, 1, 1, 0}};
int[][] test2 = {{5, 7, 9}, {5}, {8, 5}, {2}, {3}};
int[][] test3 = {{1, 2}, {1, 2, 3}, {}};
String seq = "1(234)67";
```

Implementieren Sie folgende Methoden:

- `int[][] addTriangle(int[][] input)` liefert ein neues Array zurück, das aus einer Kopie der Zeilen von `input` besteht. Zusätzlich enthält das neue Array oben n Zeilen, wobei n der Länge der obersten Zeile von `input` entspricht. Die oberste Zeile des neuen Arrays hat die Länge 1 mit Eintrag 1, die zweite Zeile die Länge 2 mit Einträgen von 2 usw., bis die Zeile der Länge n erreicht wird. Danach werden die Zeilen von `input` elementweise kopiert und unten eingefügt.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` für alle gültigen `i`, `input[0].length > 0`.
  Wird die Methode z.B. mit dem Parameter `test1` aufgerufen, entsteht folgendes Array:

1			
2	2		
3	3	3	
4	4	4	4
5	7	5	7
5			
0	1	1	0

- `void reverseBetween(int[][] input, int ind1, int ind2)` dreht die Reihenfolge der Zeilen in `input` von dem Zeilenindex `ind1` bis `ind2` (jeweils inklusive) um.
  Hinweis: Das gilt auch, wenn `ind1 > ind2`.
  Vorbedingung(en): `input != null`, `input.length > 0`, `ind1` und `ind2` sind gültige Zeilenindizes von `input`.
  Wird die Methode z.B. mit `test2`, `ind1 = 1` und `ind2 = 4` aufgerufen, entsteht folgendes Array:

5	7	9
3		
2		
8	5	
5		

- `String clean(String seq)` gibt einen neuen String zurück, der nur den Substring innerhalb der äußersten geschlossenen Klammer von `seq` enthält. Die Klammerzeichen '(' und ')' der äußersten geschlossenen Klammer werden dabei entfernt. Kommt in `seq` kein '(' vor einem ')' vor, wird ein leerer String zurückgeliefert.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `seq != null`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = addTriangle(test1)`	[[1], [2, 2], [3, 3, 3], [4, 4, 4, 4], [5, 7, 5, 7], [5], [0, 1, 1, 0]]
`result2 = addTriangle(test2)`	[[1], [2, 2], [3, 3, 3], [5, 7, 9], [5], [8, 5], [2], [3]]
`reverseBetween(test1, 2, 1)`	[[5, 7, 5, 7], [0, 1, 1, 0], [5]]
`reverseBetween(test2, 1, 4)`	[[5, 7, 9], [3], [2], [8, 5], [5]]
`reverseBetween(test3, 2, 2)`	[[1, 2], [1, 2, 3], []]
`reverseBetween(test3, 0, 2)`	[[], [1, 2, 3], [1, 2]]
`clean(seq)`	234
`clean("123(45))")`	45)
`clean("x)")`	
`clean(")x(")`	

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`addTriangle`	Korrekte Anzahl der Zeilen	2
	Erstellung Triangle: korrekter Ansatz	1
	Erstellung Triangle: korrektes Ergebnis	3
	Korrektes Kopieren der Zeilen	3
`reverseBetween`	Umdrehen der Reihenfolge: korrekter Ansatz	2
	Umdrehen der Reihenfolge: korrektes Ergebnis	4
	Korrekte Behandlung des Falles `ind1 > ind2`	2
	Korrekte Behandlung des Falles `ind1 == ind2`	1
`clean`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall/Basisfälle vorhanden	1
	Basisfall/Basisfälle korrekt	1
	Fortschritt der Rekursion vorhanden	1
	Fortschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] test1 = {{1}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2}};
int[][] test2 = {{1, 2, 3}, {0}, {1, 2}, {0}, {1}};
int[] seq = {2, 8, 8, 5, 2, 5, 7, 3};
```

Implementieren Sie folgende Methoden:

- **`int[][] repeat(int[][] input, int[] reps)`** liefert ein neues Array mit derselben Zeilenanzahl wie `input` zurück. Jede Zeile des neuen Arrays besteht aus den Einträgen der ursprünglichen Zeile in `input` und aus Wiederholungen dieser Zeile, wobei die Anzahl der Wiederholungen durch den jeweiligen Eintrag in `reps` bestimmt wird. Ein Eintrag von 0 in `reps` bedeutet, dass die Zeile nicht wiederholt wird und somit nur einmal elementweise in das neue Array kopiert wird. Bei einem negativen Eintrag in `reps` wird die Zeile rückwärts kopiert und auch rückwärts wiederholt, und zwar so oft, wie es dem Absolutbetrag des negativen Eintrags entspricht.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` und `input[i].length > 0` für alle gültigen `i`, `reps != null`, `reps.length == input.length`.
  Wird die Methode z.B. mit den Parametern `test1` und `reps = new int[]{1, -2, 1, 0}` aufgerufen, entsteht folgendes Array:

1	1

3	2	1	3	2	1	3	2	1

1	2	3	4	1	2	3	4

1	2

- **`void rasp(int[][] input)`** verändert das Array `input` auf folgende Weise: Das letzte Element der ersten Zeile (mit Index 0) wird entfernt und an das Ende der zweiten Zeile angehängt. Dieselbe Operation wird dann mit der dritten und vierten Zeile gemacht, und so weiter, bis das Ende von `input` erreicht wird. Bei einer ungeraden Zeilenanzahl wird die letzte Zeile nicht verändert.
  Vorbedingung(en): `input != null`, `input.length > 0`, `input[i] != null` und `input[i].length > 0` für alle gültigen `i`.
  Wird die Methode z.B. mit `test2` aufgerufen, entsteht folgendes Array:

1	2
0	3
1	
0	2
1	

- **`boolean hasNOrderedPairs(int[] seq, int n, int index)`** überprüft, ob in `seq` genau `n` Paare von direkt aufeinander folgenden Einträgen vorkommen, die richtig geordnet sind. Ein Paar gilt jeweils als richtig geordnet, wenn der erste Eintrag kleiner gleich dem zweiten Eintrag ist. Die Paare im Array dürfen sich auch überlappen (z.B. enthält das Array `{1, 2, 3}` zwei geordnete Paare, weil `1 <= 2` und `2 <= 3` ist). Die Zählung beginnt bei `index` (inklusive).
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `seq != null`, `seq.length > 0`, `n >= 0`, `index` beschreibt einen gültigen Index von `seq`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = repeat(test1, new int[]{1, -2, 1, 0})`	[[1, 1], [3, 2, 1, 3, 2, 1, 3, 2, 1], [1, 2, 3, 4, 1, 2, 3, 4], [1, 2]]
`result2 = repeat(test2, new int[]{1, 0, -3, 2, 0})`	[[1, 2, 3, 1, 2, 3], [0], [2, 1, 2, 1, 2, 1, 2, 1], [0, 0, 0], [1]]
`rasp(test1)`	[[], [1, 2, 3, 1], [1, 2, 3], [1, 2, 4]]
`rasp(test2)`	[[1, 2], [0, 3], [1], [0, 2], [1]]
`hasNOrderedPairs(seq, 4, 0)`	true
`hasNOrderedPairs(seq, 2, 2)`	true
`hasNOrderedPairs(seq, 1, 3)`	false
`hasNOrderedPairs(seq, 3, 3)`	false
`hasNOrderedPairs(seq, 0, 6)`	true

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`repeat`	Korrekte Länge der Zeilen	2
	Korrekte Wiederholung der Zeilen	3
	Korrekte umgedrehte Wiederholung der Zeilen	3
	Korrektes Kopieren der Zeilen bei `reps[i] == 0`	1
`rasp`	Korrekte Dimensionen	2
	Korrektes Löschen des letzten Elements	3
	Korrektes Einfügen in nächster Zeile	3
	Korrekte Behandlung bei ungerader Zeilenanzahl	1
`hasNOrderedPairs`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall/Basisfälle vorhanden	1
	Basisfall/Basisfälle korrekt	1
	Fortschritt der Rekursion vorhanden	1
	Fortschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30

Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder Klassenvariablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **weder** die Methode `clone` **noch** die Methode `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `length`, `substring`, `isEmpty`.
- Aus der Klasse `Math` dürfen alle Methoden verwendet werden. Sonst darf **keine** weitere Klasse verwendet werden.

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[] test1 = {3, 0, 6, -1, 1};
int[][] test2 = {{0}, {6, -5}, {0, 0}, {0, 1, 2, 0}};
int[][] test3 = {{1, 2, 7, 3, 0}, {-8}, {0, 2}, {1, 4, -2, 1}};
```

Implementieren Sie folgende Methoden:

- `boolean[][] create(int[] input)` liefert ein neues Array zurück, in dem alle Zeilen eine Mindestlänge von 3 Einträgen haben und Einträge an in `input` spezifizierten Positionen `true` und alle übrigen Einträge `false` sind. Jede Zeile hat maximal einen Eintrag, der `true` ist. In `input[i]` steht für die Zeile mit Index `i` der Spaltenindex dieses Eintrags. Ist `input[i]` kleiner 0, hat die Zeile keinen Eintrag `true` und alle 3 Einträge sind `false`. Die Zeilen des zurückgelieferten Arrays können länger als 3 sein und zwar dann, wenn `input[i]` größer als 2 ist. In diesem Fall hat die Zeile die Länge `input[i]+1`, sodass der letzte Eintrag `true` ist.
  Vorbedingung(en): `input != null` .
  Wird die Methode z.B. mit den Parametern `test1` aufgerufen, entsteht folgendes Array:

false	false	false	true
true	false	false	
false	false	false	false	false	false	true
false	false	false				
false	true	false				

- `void move(int[][] input)` ändert das angegebene Array `input` so, dass alle Einträge in den Zeilen um eine Position nach rechts (Richtung größerem Index) geschoben werden. An der ersten Position jeder Zeile rücken neue Einträge mit dem Wert 0 nach. Stand am Ende der Zeile der Wert 0, wird dieser nicht mehr verschoben, sondern vom neuen Wert überschrieben und die Zeile ändert ihre Länge nicht (in diesem Fall wird die Zeile nicht ersetzt, sondern deren Einträge geändert). Steht im ursprünglichen Array am Ende der Zeile ein Wert ungleich 0, wird auch dieser verschoben und die entsprechende Zeile wird durch eine um einen Eintrag längere Zeile ersetzt.
  Vorbedingung(en): `input != null` und für alle gültigen `i` gilt `input[i] != null` und `input[i].length > 0`.
  Wird die Methode z.B. mit `test2` aufgerufen, entsteht folgendes Array:

0			
0	6	-5	
0	0		
0	0	1	2

- `boolean oddOccurrences(String s, char ch)` liefert `true` genau dann, wenn die Anzahl der Vorkommnisse des Zeichens `ch` in `s` eine ungerade Zahl ist.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `s != null`.

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben. Erzeugen Sie diese Ausgaben nur in `main`, nicht in den implementierten Methoden.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = create(new int[]{3})`	[[false, false, false, true]]
`result2 = create(new int[]{-2, 0})`	[[false, false, false], [true, false, false]]
`result3 = create(new int[]{})`	[]
`move(test2)`	[[0], [0, 6, -5], [0, 0], [0, 0, 1, 2]]
`move(test3)`	[[0, 1, 2, 7, 3], [0, -8], [0, 0, 2], [0, 1, 4, -2, 1]]
`oddOccurrences("This is not a test!",'s')`	`true`
`oddOccurrences("This is not a test!",'T')`	`true`
`oddOccurrences("This is not a test!",'t')`	`true`
`oddOccurrences("This is not a test!",' ')`	`false`
`oddOccurrences("",'x')`	`false`

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	1
	Testfälle korrekt implementiert	2
`create`	Korrekte Anzahl der Zeilen	2
	Korrekte Länge jeder Zeile	4
	Richtige Einträge bei negativem Index	1
	Richtige Einträge in Zeile mit Länge 3	1
	Richtige Einträge in Zeile mit Überlänge 3	1
`move`	Korrektes Verschieben der Einträge	3
	Korrektes Kopieren bei erweiterten Zeilen	3
	Korrekte Länge aller Zeilen	2
	Korrekte erste Stelle in den Zeilen	1
`oddOccurrences`	Korrekter Methodenansatz (Rückgabe vorhanden)	1
	Basisfall/Basisfälle vorhanden	1
	Basisfall/Basisfälle korrekt	1
	Forschritt der Rekursion vorhanden	1
	Forschritt der Rekursion korrekt	1
	Korrekter Rückgabewert	4
Gesamt		30