Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder globalen Variablen 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 **nicht** die Methoden `clone` und `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`

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, 2, 3}, {1, 2, 3}, {1, 2, 3}};
int[][] test3 = {{6}, {6, 7}, {6, 7}, {6, 7}, {7, 6}};
int[] seq1 = {1, 2, 4, 4, 5, 5, 5, 10};
```

Implementieren Sie folgende Methoden:

- **`int[][] reshape(int[][] inputArray)`** erzeugt aus `inputArray` ein neues zweidimensionales Array und retourniert dieses. Das neue Array hat in jeder Zeile genau 2 Spalten und seine Werte werden zeilweneweise 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` ungerade ist), wird der überschüssige Array-Eintrag 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
2	4
7	3
5	9
8	0

- **`void mask(int[][] inputArray)`** setzt alle Werte einer Zeile auf -1, wenn die darauf folgende Zeile die selbe Länge hat und genau die selben Werte in der selben Reihenfolge aufweist.
  Vorbedingung(en): `inputArray.length > 0`.
  Wird die Methode z.B. mit `test3` aufgerufen, entsteht folgendes Array:

6	
-1	-1
-1	-1
6	7
7	6

- **`int getMaxOppositeSum(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.

Aufruf	**Ausgabe in main auf der Konsole**
`result1 = reshape(test1)`	[[5, 2], [2, 4], [7, 3], [5, 9], [8, 0]]
`result2 = reshape(test3)`	[[6, 6], [6, 7], [7, 6], [7, 7], [6, 0]]
`reshape(new int[][]{{}})`	[]
`mask(test1)`	[[5, 2, 4], [2, 7, 3], [9, 5, 8]]
`mask(test2)`	[[-1, -1, -1], [-1, -1, -1], [1, 2, 3]]
`mask(test3)`	[[6], [-1, -1], [-1, -1], [6, 7], [7, 6]]
`getMaxOppositeSum(seq1, 0, 7)`	11
`getMaxOppositeSum(seq1, 0, 5)`	8
`getMaxOppositeSum(seq1, 4, 7)`	15

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	/ 1
	Testfälle korrekt implementiert	/ 2
`reshape`	Korrekte Anzahl an Zeilen und Spalten	/ 2
	Korrekte Schleifen	/ 2
	Korrekte Werte	/ 1
	Korrekte Richtung	/ 3
`mask`	Korrekte Schleifen	/ 3
	Korrekte Überprüfung auf Gleichheit der Zeilen	/ 4
	Richtige Zeilen auf -1 gesetzt	/ 2
	Korrekter Inhalt im Array	/ 1
`getMaxOppositeSum`	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