

Modul Datenstrukturen, Algorithmen und Programmierung 1

Einfache Zusatzaufgaben

- Die Aufgaben stammen aus den Klausuren der vergangenen Jahre.
- Die Aufgaben sind unterschiedlich schwer und unterschiedlich umfangreich. Dieses wurde in den Klausuren durch die Zahl der zugeordneten Punkte berücksichtigt.
- Alle Aufgaben lassen sich mit den Kenntnissen der Vorlesungsinhalte bis einschließlich Kapitel 5 bearbeiten.
- Die Aufgaben dienen in verschiedener Art dem Training für die Klausur:
 - In Programmierung ungeübte Studierende können weitere Erfahrungen sammeln.
 - In Programmierung bereits geübte Studierende können sich an die Form der Klausuraufgaben gewöhnen.
 - Bei allen Studierenden wird das algorithmische Denken trainiert, das auch zum Lösen der Aufgaben zu fortgeschrittenen Themen benötigt wird.
- Zu diesen Aufgaben werden keine Beispiellösungen veröffentlicht.
- Die Zusammenstellung ist ausgehend von den vorliegenden Klausuren erfolgt. Möglicherweise sind einzelne Aufgaben – oder Varianten davon – bereits in die Übungs- oder Praktikumsaufgaben übernommen worden und tauchen jetzt doppelt auf.
- Die Aufgaben der hier präsentierten Form haben in den Klausuren etwa 5 bis 15 Prozent des Gesamtumfangs ausgemacht.
- Die Aufgaben 1 bis 4 erfordern als ersten Schritt das vervollständigen einfacher Methoden zu einfachen Klassen, die im zweiten Schritt geeignet genutzt werden müssen.
- Die Aufgaben 5 bis 26 fordern Methoden, die auf einem Feld arbeiten.
- Die Aufgaben 27 bis 37 fordern rekursive Methoden.

Aufgabe 1

Gegeben ist die Klasse Tuple:

```
public class Tuple {
    private String text;
    private int value;
    public Tuple( String t, int v ) { text = t; value = v; }
    public String getText() { return text; }
    public int getValue() { return value; }
    public void setValue( int v ) { value = v; }
}
```

a) Vervollständigen Sie die Klasse Group:

- Die Methode Tuple delete(int i) entfernt am Index i das Objekt aus dem Feld map und gibt es zurück. Falls der Index außerhalb der Grenzen des Felds map liegt oder am Index i der Wert null steht, soll null zurückgegeben werden.
- Die Methode Group copy() gibt ein neu erzeugtes Objekt der Klasse Group zurück, das exakt dem ausführenden Objekt entspricht und jeweils **Kopien** der in map vorhandenen Objekte der Klasse Tuple an genau den gleichen Indizes enthält.
- Die Methode boolean insert(Tuple p) fügt ein Objekt der Klasse Tuple, dass als Argument an p übergeben wird, am kleinsten Index im Feld map ein, an dem der Wert null steht, und gibt true zurück. Wird als Argument null übergeben oder gibt es im Feld map keinen Wert null, wird false zurückgegeben und map bleibt unverändert.

```
public class Group
{
    private Tuple[] map;

    public Group( int n ) {
        map = new Tuple[n];
    }

    public int size() {
        return map.length;
    }

    public Tuple delete( int i )
    {

```

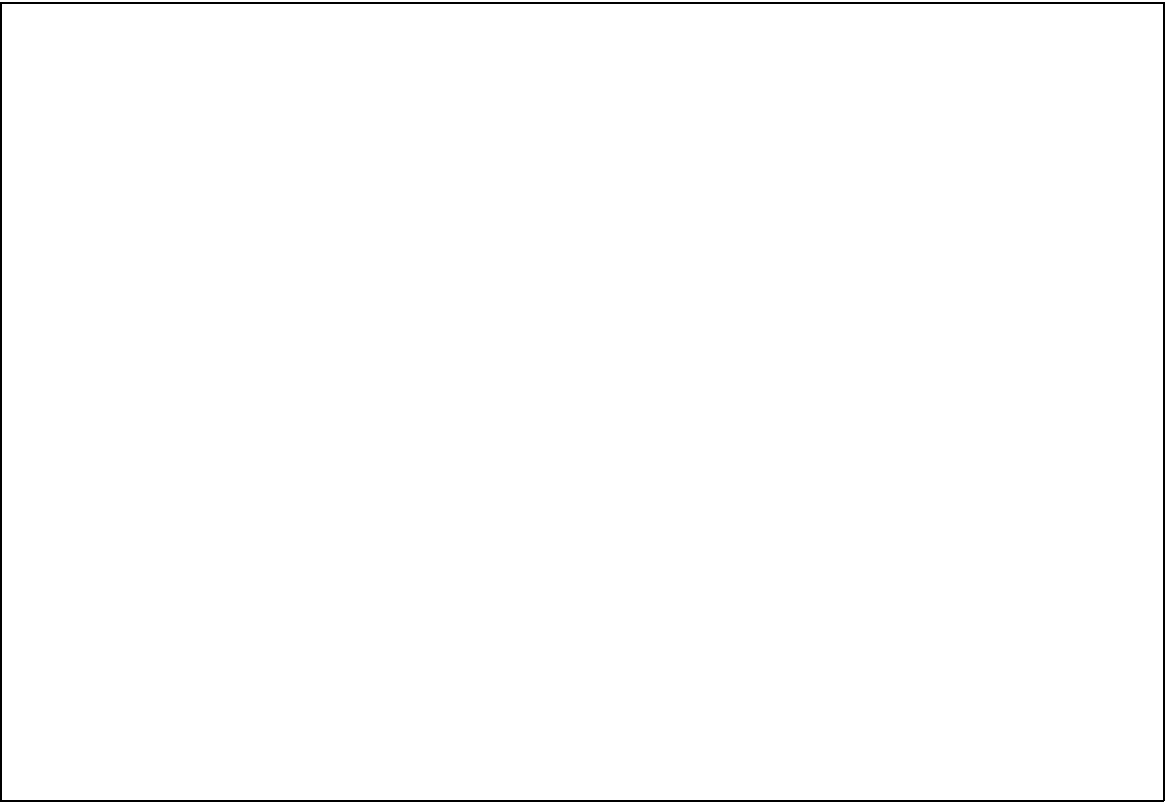
```

    }
}
```

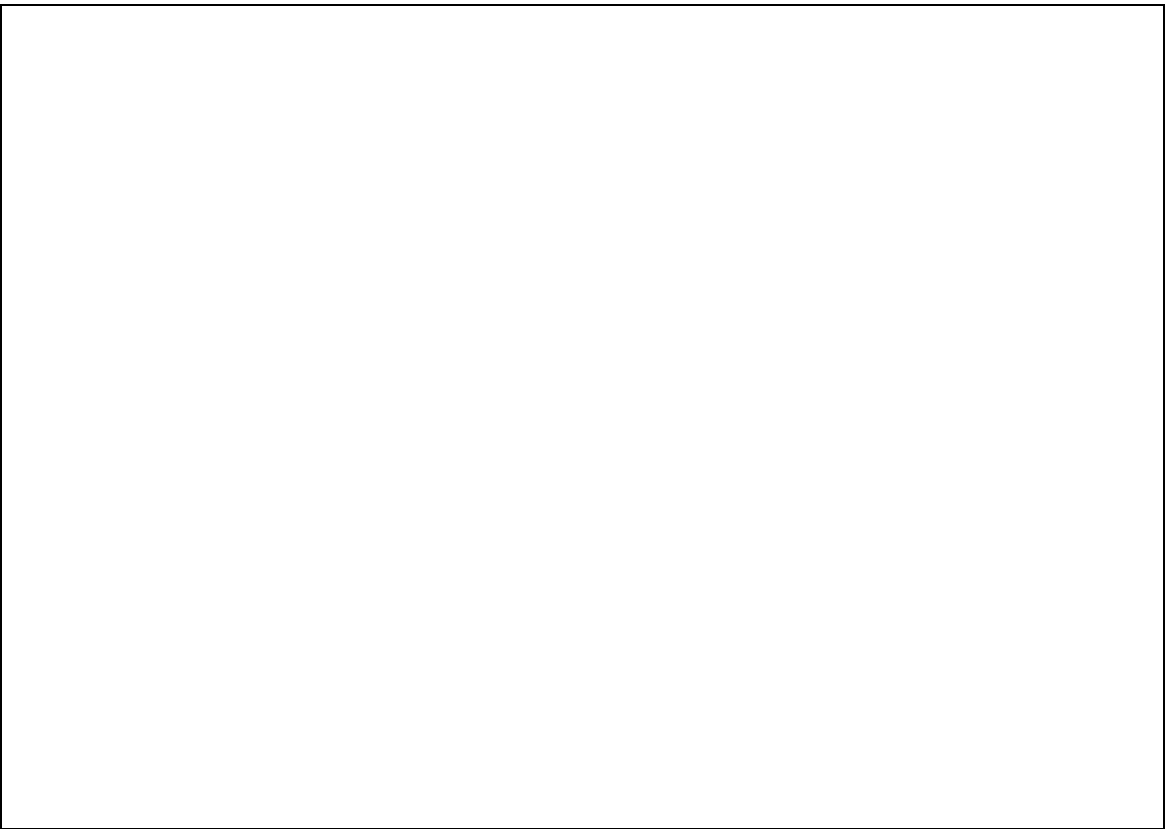
(Fortsetzung auf der nächsten Seite)

(Fortsetzung Aufgabe 2 a)

```
public Group copy()  
{
```



```
public boolean insert( Tuple p )  
{
```



```
}
```

b) Vervollständigen Sie **in der Klasse Use** die folgenden beiden Methoden, die die aus Aufgabenteil a) bekannten Klassen Tuple und Group *benutzen*.

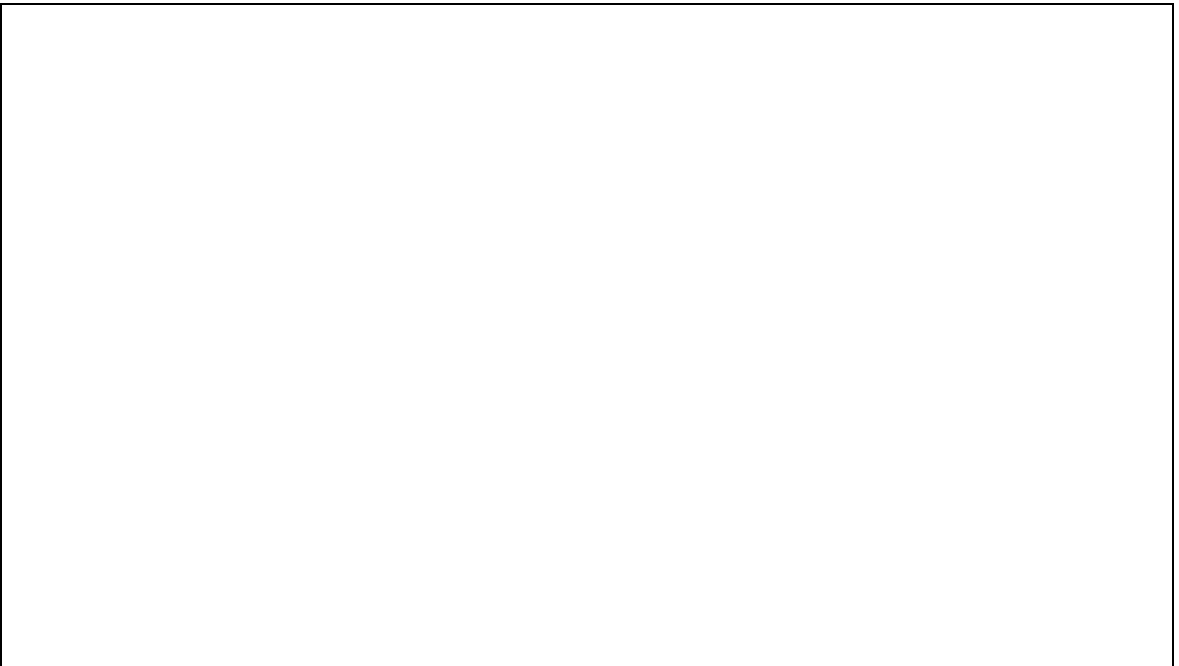
- Die Methode Group extract(Group g, int v) erzeugt ein Objekt der Klasse Group, das alle Objekte der Klasse Tuple aus dem Feld map des ausführenden Objekts enthält, deren Attribut value kleiner als der Wert v ist. g soll dabei unverändert bleiben.
- Die Methode int maxValue(Group source) soll den größten Wert zurückgeben, den die Methode getValue für alle Tuple-Objekte aus source liefert. source soll dabei unverändert bleiben.

```
public class Use
```

```
{
```

```
    public static Group extract( Group g, int v )
```

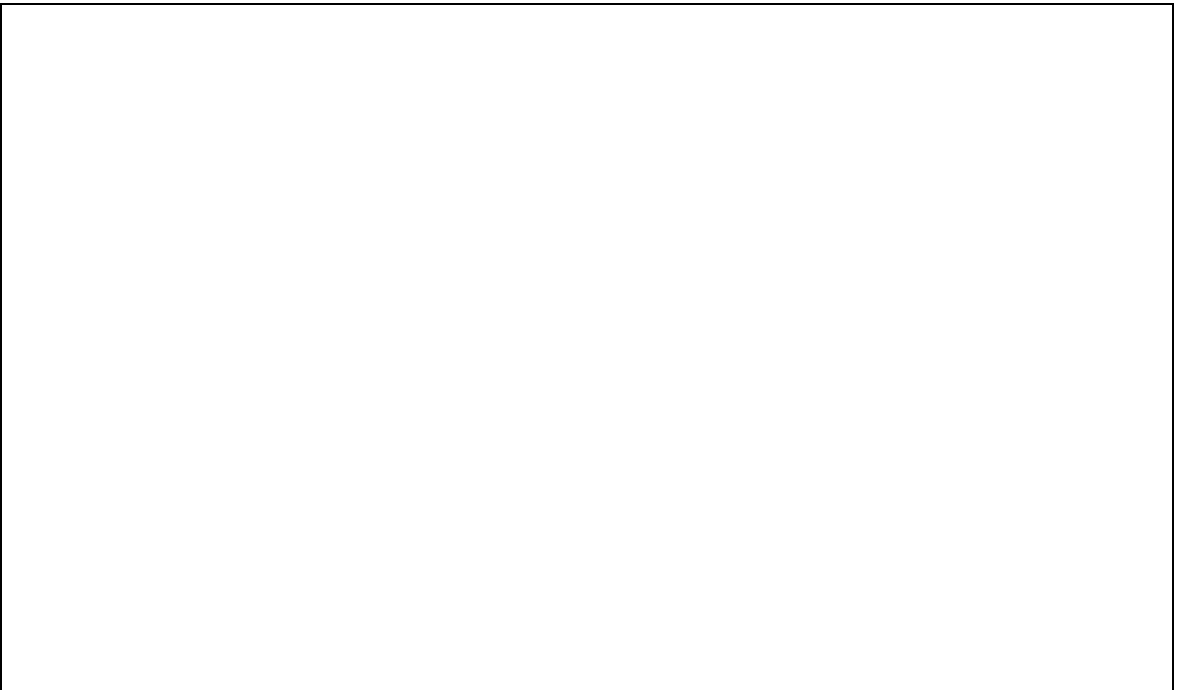
```
    {
```



```
    }
```

```
    public static int maxValue( Group source )
```

```
    {
```



```
    }
```

```
}
```

Aufgabe 2

- a) Vervollständigen Sie die Klasse `Ints` um die folgenden zusätzlichen Methoden:
- `int size()` soll die Anzahl der Elemente des Feldes `numbers` zurückgeben.
 - `int get(int index)` soll den Wert am Index `index` im Feld `numbers` zurückgeben. Besitzt `index` einen ungültigen Wert, wird eine `RuntimeException` geworfen.
 - `boolean only(int value)` soll `true` zurückgeben, falls alle Elemente des Feldes `numbers` den Wert `value` besitzen. Sonst soll `false` zurückgegeben werden. Hat das Feld kein Element, soll ebenfalls `true` zurückgegeben werden.

```
public class Ints
```

```
{
```

```
    private int[] numbers;
```

```
    public Ints( int[] array )
```

```
    {
```

```
        numbers = array;
```

```
    }
```

```
    public int size()
```

```
    {
```

```
    }
```

```
    public int get( int index )
```

```
    {
```

```
        if (  )
```

```
        throw new RuntimeException();
```

```
    }
```

```
    public boolean only( int value )
```

```
    {
```

```
        for( int candidate : numbers )
```

```
        return  ;
```

```
    }
```

b) Vervollständigen Sie in der Klasse `Use` zwei Methoden.

- **boolean** `contains(int value)` gibt **true** zurück, falls der Wert `value` mindestens einmal in den in `structure` abgelegten Werten vorkommt. Sonst wird **false** zurückgegeben.
- **boolean** `allZero()` gibt **true** zurück, falls in `structure` alle Elemente den Wert `0` besitzen. Sonst wird **false** zurückgegeben. Enthält `structure` keine Werte, soll ebenfalls **true** zurückgegeben werden.

```
public class Use
```

```
{
```

```
    private Ints[] structure;
```

```
    // ... (Konstruktor und weitere Methoden sind nicht von Interesse)
```

```
    public boolean contains( int value )
```

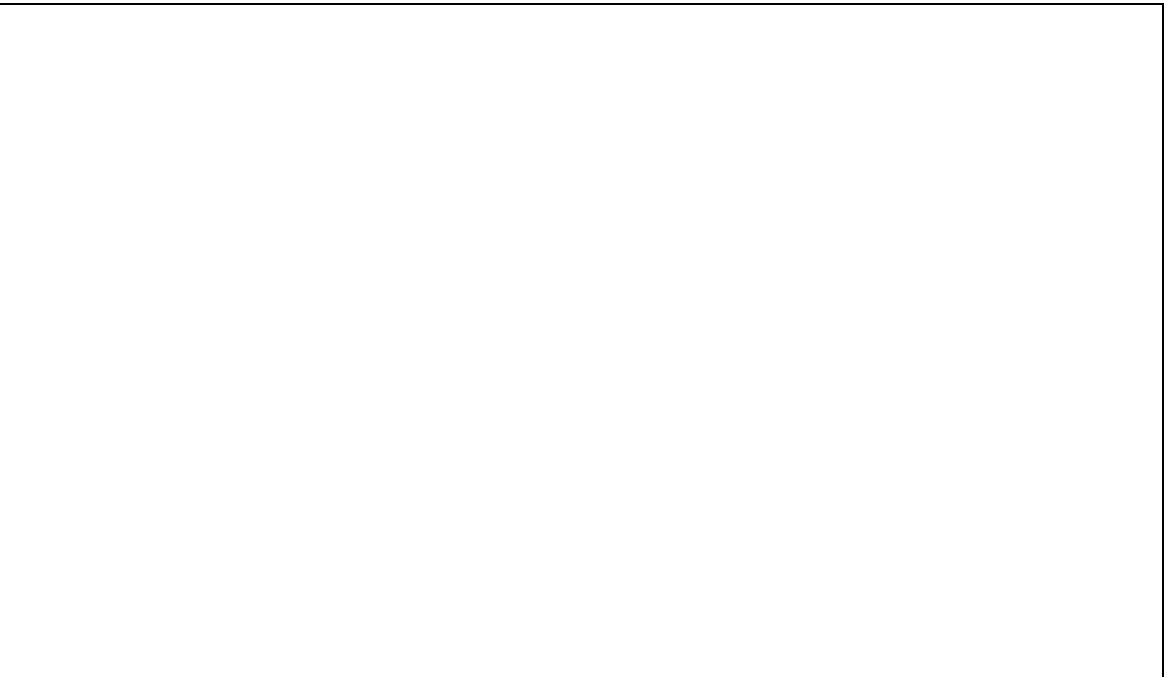
```
    {
```



```
    }
```

```
    public boolean allZero()
```

```
    {
```



```
    }
```

```
}
```

Aufgabe 3

- a) Vervollständigen Sie in der Klasse `DataValues` zwei Methoden:
- `int quantityZero()` soll die Häufigkeit des Auftretens des Wertes `0` im Feldes `numbers` bestimmen und zurückgeben.
 - `boolean oneInBetween(int bottom, int top)` soll nur dann `true` zurückgeben, falls es **genau ein** Element in `numbers` gibt, dessen Wert größer als `bottom` und kleiner als `top` ist. Ist `bottom` nicht kleiner als `top`, wird eine `RuntimeException` geworfen.

```
public class DataValues
```

```
{
```

```
    private int[] numbers;
```

```
    public DataValues( int[] array )
```

```
    {
```

```
        numbers = array;
```

```
    }
```

```
    public int size()
```

```
    {
```

```
        return numbers.length;
```

```
    }
```

```
    public int quantityZero()
```

```
    {
```

```
        int count = 0;
```

```
        for( int candidate : numbers )
```

```
        {
```

```
        }
```

```
        return count;
```

```
    }
```

```
    public boolean oneInBetween( int bottom, int top )
```

```
    {
```

```
        boolean result = false;
```

```
        if ( bottom < top )
```

```
        {
```

```
            for( int candidate : numbers )
```

```
            {
```

```
            }
```

```
            return result;
```

```
        }
```

```
        throw new RuntimeException();
```

```
    }
```

b) Vervollständigen Sie in der Klasse `Use` zwei Methoden:

- **boolean** `containsSingle(int value)` gibt **true** zurück, falls der Wert `value` **genau einmal** unter allen in `structure` abgelegten Werten vorkommt. Sonst wird **false** zurückgegeben.
- **boolean** `allZero()` gibt **true** zurück, falls in `structure` kein anderer Wert als 0 abgelegt ist. Sonst wird **false** zurückgegeben.
Enthält `structure` keine Werte, soll ebenfalls **true** zurückgegeben werden.

```
public class Use
```

```
{
    private DataValues[] structure;           // DataValues aus Aufgabenteil a)

    // ... (Konstruktor und weitere Methoden sind nicht von Interesse)

    public boolean containsSingle( int value )
    {
        int count = 0;

        for (  )
        {

        }

        return count == 1;
    }

    public boolean allZero()
    {
        boolean result = true;

        for (  )
        {

        }

        return result;
    }
}
```


Aufgabe 4

Vervollständigen Sie die Methode `int maxDiff(int[] arr)`. Die Methode `maxDiff` gibt den größten Abstand zurück, der zwischen zwei direkt aufeinander folgenden Elementen des Feldes `arr` auftritt. Hat das Feld weniger als zwei Elemente, wird `0` zurückgegeben. Das als Argument übergebene Feld darf dabei nicht verändert werden.

Beispiel: Für das Feld

| | | | | | | |
|---|---|----|---|----|----|----|
| 1 | 5 | -4 | 9 | 21 | 20 | 16 |
|---|---|----|---|----|----|----|

 liefert `maxDiff` den Wert 13 (aus -4 ... 9)

```
public static int maxDiff( int[] arr )  
{
```

```
}
```

Aufgabe 5

Vervollständigen Sie die Methode `boolean allTriples(int[] arr)`.

Die Methode `allTriples` soll `true` zurückgeben, falls jeder Wert im Feld `arr` genau dreimal vorkommt. Sonst soll `false` zurückgegeben werden. Das als Argument übergebene Feld darf dabei nicht verändert werden.

```
public static boolean allTriples( int[] arr )
```

```
{
```

```
}
```

Aufgabe 6

Vervollständigen Sie die Methode `int[] subset(int[] arr, int from)`.

Die Methode `subset` soll ein neues Feld anlegen und zurückgeben. Dieses Feld soll genau nur die Inhalte aus `arr`, deren Indizes größer oder gleich `from` sind, in der durch `arr` gegebenen Reihenfolge enthalten.

Der Wert von `from` muss ein zulässiger Index von `arr` sein.

Ist diese Bedingungen nicht erfüllt, soll ein Feld ohne Elemente zurückgegeben werden.

```
public static int[] subset( int[] arr, int from )
{
    if (  )
    {
        
    }
    else
    {
         ;
    }
}
```

Aufgabe 7

Vervollständigen Sie die Methode `boolean containsTriple(int[] arr)`.

Die Methode `containsTriple` soll `true` zurückgeben, falls es mindestens einen Wert im Feld `arr` gibt, der genau dreimal vorkommt. Sonst soll `false` zurückgegeben werden. Das als Argument übergebene Feld darf dabei nicht verändert werden.

```
public static boolean containsTriple( int[] arr )
```

```
{
```

```
}
```

Aufgabe 8

Vervollständigen Sie die Methode `int positiveSequences(int[] arr)`.

Die Methode `positiveSequences` soll in dem Feld `arr` die Anzahl der Teilfolgen bestimmen, die aus unmittelbar aufeinander folgenden positiven Werten bestehen.

- Eine solche Teilfolge beginnt immer mit einem positiven Wert.
- Eine solche Teilfolge endet bei `0`, einem negativen Wert oder dem Ende des Feldes.
- Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int positiveSequences( int[] arr )  
{  
    int quantity = 0;
```

```
        return quantity;  
    }
```

Aufgabe 9

Vervollständigen Sie die Methode `boolean semiFibs(int[] arr)`.

Die Methode `semiFibs` soll `true` zurückgeben, falls für alle Paare von jeweils zwei unmittelbar aufeinander folgenden Indizes gilt, dass die Summe ihrer Werte kleiner oder gleich dem Wert des ihnen unmittelbar folgenden Wertes ist.

Hat das Feld weniger als drei Elemente, soll `true` zurückgegeben werden.

Sonst soll `false` zurückgegeben werden.

```
public static boolean semiFibs( int[] arr )
{
    for (  )
    {
        
    }
     ;
}
```

Aufgabe 10

Vervollständigen Sie die Methode `int[] deleted(int[] arr, int value)`.

Die Methode `deleted` soll ein neues Feld zurückgeben, in dem alle Vorkommen des Wertes `value` fehlen und alle anderen Werte des Feldes `arr` in unveränderter Reihenfolge auftreten.

Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int[] deleted( int[] arr, int value )
```

```
{
```

```
}
```

Aufgabe 11

Vervollständigen Sie die Methode `int[] head(int[] arr, int to)`.

Die Methode `head` soll ein neues Feld anlegen und zurückgeben. Dieses Feld soll genau nur die Inhalte aus `arr`, deren Indizes kleiner als oder gleich `to` sind, in der durch `arr` gegebenen Reihenfolge enthalten.

Der Wert von `to` muss ein zulässiger Index von `arr` sein.

Ist diese Bedingung nicht erfüllt, soll ein Feld zurückgegeben werden, das keine Elemente enthält.

```
public static int[] head( int[] arr, int to )
{
    if (  )
    {
        
    }
    else
    {
         ;
    }
}
```


Aufgabe 12

Vervollständigen Sie die Methode `boolean semiSorted(int[] arr)`.

Die Methode `semiSorted` soll `true` zurückgeben, falls das Feld `arr` weniger als zwei Elemente hat oder für jedes Element mit einem ungeraden Index größer als `0` gilt, dass sein Wert größer als der des unmittelbar vorangehenden Elements ist.

Sonst soll `false` zurückgegeben werden.

```
public static boolean semiSorted( int[] arr )
{
    for (  )
    {
        
    }
     ;
}
```

Aufgabe 13

Vervollständigen Sie die Methode `int[] odd(int[] arr)`.

Die Methode `odd` soll ein neues Feld anlegen und zurückgeben. Dieses Feld soll genau nur die Inhalte mit ungeraden Werten aus `arr` in der durch `arr` gegebenen Reihenfolge enthalten.

```
public static int[] odd( int[] arr )  
{
```

```
    int[] result;
```

```
    return result;
```

```
}
```

Aufgabe 14

Vervollständigen Sie die Methode `boolean almostUnique(int[] arr)`.

Die Methode `almostUnique` soll `true` zurückgeben, falls im Feld `arr` jeder Wert höchstens zweimal vorkommt. Sonst soll `false` zurückgegeben werden.

```
public static boolean almostUnique( int[] arr )
```

```
{
```

```
}
```

Aufgabe 15

Vervollständigen Sie die Methode `int[] tail(int[] arr, int position)`.

Die Methode `tail` soll ein neues Feld zurückgeben, das nur diejenigen Werte des Feldes `arr` in unveränderter Reihenfolge enthält, die in diesem ab dem Index `position` auftreten.

- Der Inhalt des Parameters `arr` darf nicht geändert werden.
- Liegt `position` nicht im Bereich der gültigen Indizes, wird eine `RuntimeException` geworfen.

```
public static int[] tail( int[] arr, int position )
{
    if (  )
    {
        
    }
    else
    {
        throw new RuntimeException();
    }
}
```

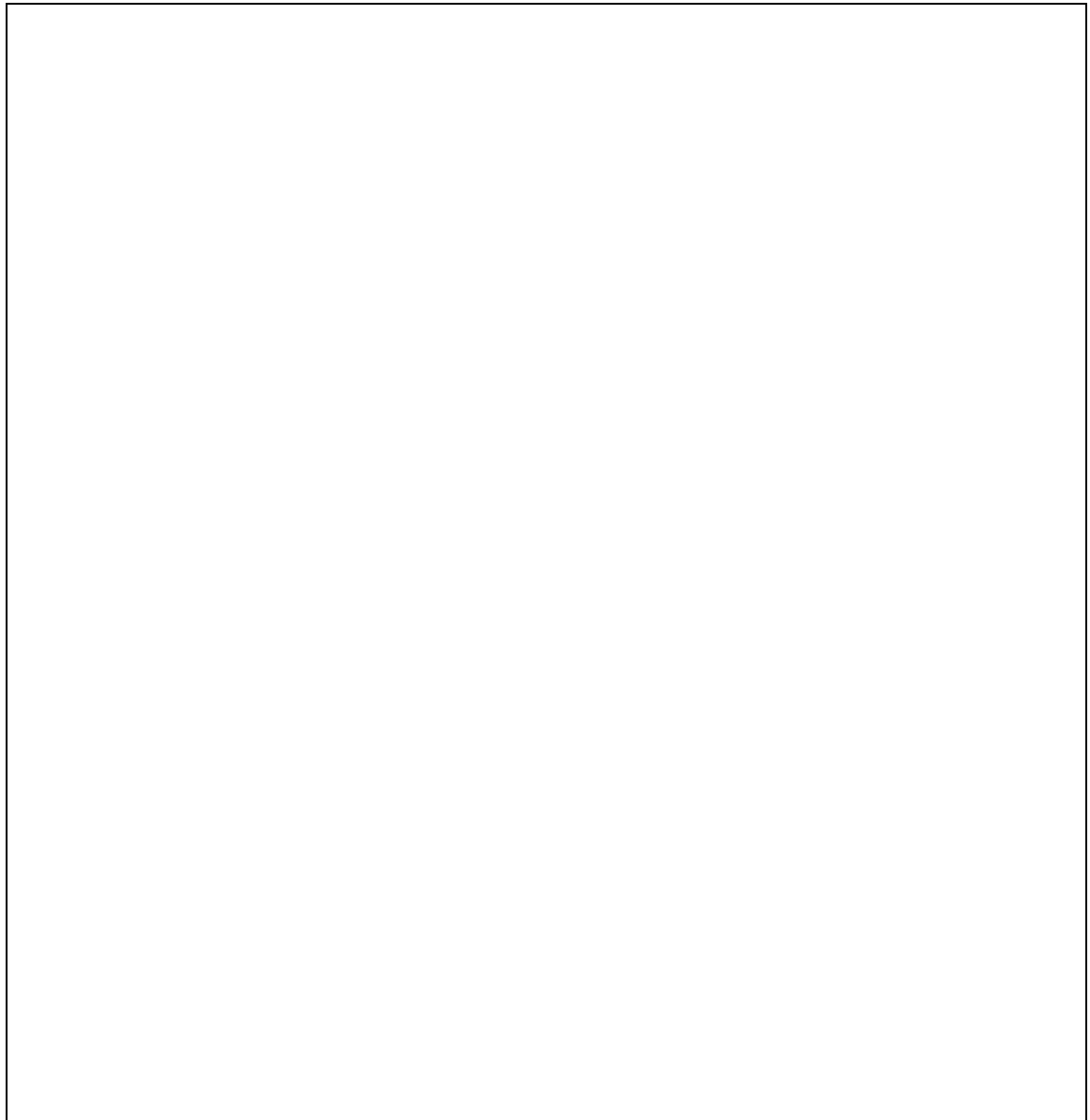
Aufgabe 16

Vervollständigen Sie die Methode `int longestSequence(int[] arr)`.

Die Methode `longestSequence` soll in dem Feld `arr` die Länge der **längsten** Teilfolge bestimmen, die aus unmittelbar aufeinander folgenden positiven Werten besteht.

- Eine solche Teilfolge beginnt immer mit einem positiven Wert.
- Eine solche Teilfolge endet bei `0`, einem negativen Wert oder dem Ende des Feldes.
- Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int longestSequence( int[] arr )  
{  
    int length = 0;
```



```
        return length;  
    }
```

Aufgabe 17

Vervollständigen Sie die Methode `int twice(int[] arr)`.

Die Methode `twice` soll die Anzahl der Werte zurückgeben, die im Feld `arr` genau zweimal vorkommen.

```
public static int twice( int[] arr )  
{  
    int count = 0;
```

```
    return count;  
}
```

Aufgabe 18

Vervollständigen Sie die Methode `int[] negatives(int[] arr)`.

Die Methode `negatives` soll ein neues Feld anlegen und zurückgeben. Dieses Feld soll genau nur die Inhalte mit negativen Werten aus `arr` in der durch `arr` gegebenen Reihenfolge enthalten.

```
public static int[] negatives( int[] arr )  
{
```

```
    int[] result;
```

```
    return result;
```

```
}
```

Aufgabe 19

Vervollständigen Sie die Methode `int countValues(int[] arr)`.

Die Methode `countValues` soll die Anzahl der unterschiedlichen Werte im Feld `arr` zurückgeben.

Das Feld `arr` soll unverändert bleiben.

```
public static int countValues( int[] arr )  
{
```

```
    int count = 0;
```

```
    return count;  
}
```


Aufgabe 20

Vervollständigen Sie die Methode `int[] odds(int[] arr)`.

Die Methode `odds` soll ein neu erzeugtes Feld zurückgeben, das genau nur die im Feld `arr` an ungeraden Indizes stehenden Inhalte in umgekehrter Reihenfolge enthält.

Das Feld `arr` soll unverändert bleiben.

```
public static int[] odds( int[] arr )
```

```
{
```

```
}
```

Aufgabe 21

Vervollständigen Sie die Methode `int compare(int[] arr1, int[] arr2)`.

Die Methode `compare` soll:

- den Wert `-1` zurückgeben, wenn die Felder `arr1` und `arr2` die gleiche Länge haben und für jeden Index `i` gilt: `arr1[i] < arr2[i]`,
- den Wert `1` zurückgeben, wenn die Felder `arr1` und `arr2` die gleiche Länge haben und für jeden Index `i` gilt: `arr1[i] > arr2[i]`,
- sonst den Wert `0` zurückgeben.

Die Felder `arr[1]` und `arr[2]` sollen unverändert bleiben.

```
public static int compare( int[] arr1, int[] arr2 )
{

    int greater = 0;
    int smaller = 0;

    if ( arr1.length == arr2.length )
    {
        for ( int i=0; i<arr1.length; i++ )
        {
            if ( arr1[i] < arr2[i] )
            {
                greater++;
            }
            else if ( arr1[i] > arr2[i] )
            {
                smaller++;
            }
        }
    }

}
```

```
}
```

Aufgabe 22

Vervollständigen Sie die Methode `int longestAscendingSequence(int[] arr)`.

Die Methode `longestAscendingSequence` soll die Länge der längsten Teilfolge mit aufsteigenden Werten im Feld `arr` zurückgeben. Die Länge einer solchen Teilfolge ergibt sich aus der Zahl der aufeinander folgenden Elemente, bei denen der Wert am größeren Index größer ist als alle davor stehenden Werte dieser Teilfolge.

Enthält das Feld keine Werte, soll `0` zurückgegeben werden.

Das Feld `arr` soll unverändert bleiben.

```
public int longestAscendingSequence( int[] arr )
{
    if ( arr.length < 2 )
    {
        return arr.length;
    }
    else
    {
        
    }
}
```

Aufgabe 23

Vervollständigen Sie die Methode `int embeddedSequence(int[] arr)`.

Die Methode `embeddedSequence` soll die Länge der längsten Teilfolge von Werten im Feld `arr` angeben, die mit dem gleichen Wert beginnt und endet. Gibt es keine solche Teilfolge, soll `0` zurückgegeben werden.

```
public static int embeddedSequence( int[] arr )
```

```
{
```

```
}
```

Aufgabe 24

Vervollständigen Sie die Methode `int hasTriple(int[] arr, int n)`, die rekursiv arbeiten soll. Die Methode `hasTriple` soll `true` zurückgeben, wenn es im Feld `arr` drei unmittelbar aufeinander folgende Indizes gibt, an denen der gleiche Wert abgelegt ist. Sonst soll `false` zurückgegeben werden. Wird für `n` ein Wert außerhalb der Grenzen des Felds `arr` übergeben, soll `false` zurückgegeben werden.

Die Implementierung darf **keine** Schleifen enthalten.

```
public static boolean hasTriple( int[] arr, int n )  
{
```

```
    if (  )
```

```
    {  
        return false;  
    }
```

```
    else
```

```
    {
```

```
    }
```

```
}
```

Aufgabe 25

Vervollständigen Sie die Methode `int getIndex(int[] arr, int v, int n)`, die *rekursiv* arbeiten soll. Die Methode `getIndex` soll den kleinsten Index zurückgeben, an dem der Wert `v` in `arr` im Bereich der Indizes `0..n` vorkommt. Kommt `v` nicht vor oder wird für `n` ein Wert außerhalb der Grenzen des Felds `arr` übergeben, soll `-1` zurückgegeben werden.

Die Implementierung darf **keine** Schleifen enthalten.

```
public static int getIndex( int[] arr, int v, int n )
{
```

```
if (
```

```
{
    return -1;
}
```

else

}

Aufgabe 26

Vervollständigen Sie die *rekursive* Methode `int at(int[] arr, int limit, int val)`.

Die Methode `at` soll den größten Index im Bereich der Indizes `0, ..., limit` zurückgeben, an dem im Feld `arr` der Wert `val` steht.

- Wird ein unzulässiges Argument für `limit` übergeben oder kommt `val` nicht in `arr` vor, so wird `-1` zurückgegeben.
- Die Methode `at` soll *rekursiv* arbeiten. Die Implementierung darf keine Schleifen enthalten.
- Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int at( int[] arr, int limit, int val )
{
    if (  )
    {
        
    }
    else
    {
        return  ;
    }
}
```

Aufgabe 27

Vervollständigen Sie die *rekursive* Methode `boolean nonZero(int[] arr, int limit)`. Die Methode `nonZero` soll `true` zurückgeben, wenn im Feld `arr` im Bereich der Indizes `0, ..., limit` der Wert `0` nicht vorkommt.

- Wird ein unzulässiges Argument für `limit` übergeben oder kommt `0` in `arr` vor, so wird `false` zurückgegeben.
- Die Methode `nonZero` soll *rekursiv* arbeiten. Die Implementierung darf keine Schleifen enthalten.
- Der Inhalt des Parameters `arr` darf nicht verändert werden.

```
public static boolean nonZero( int[] arr, int limit )
{
    if (  )
    {
        
    }
    else
    {
         ;
    }
}
```


Aufgabe 28

Vervollständigen Sie die *rekursive* Methode

```
boolean allGreater( int[] arr1, int[] arr2, int p )
```

Die Methode `allGreater` soll `true` zurückgeben, falls für das kürzere der beiden Felder für jeden gültigen Index `i` gilt, dass der Wert von `arr1[i]` größer ist als der Wert von `arr2[i]`. Sonst soll `false` zurückgegeben werden. Hat eines der Felder die Länge `0`, soll `true` zurückgegeben werden. Der erste Aufruf von `allGreater` soll mit dem Argument `0` für `p` erfolgen.

Die Implementierung darf **keine** Schleifen enthalten.

```
public static boolean allGreater( int[] arr1, int[] arr2, int i )
```

```
{
```

```
    if (
```

```
)
```

```
    {
```

```
        return true;
```

```
    }
```

```
    else
```

```
    {
```

```
        if (
```

```
)
```

```
        {
```

```
        }
```

```
    else
```

```
    {
```

```
    }
```

```
    }
```

```
}
```

Aufgabe 29

Vervollständigen Sie die **rekursive** Methode `int greater(int[] arr, int limit)`. Die Methode `greater` soll die Anzahl der Werte des Feldes `arr` im Bereich der Indizes `0 ... limit` zurückgeben, die größer als der Wert des vorangehenden Index sind, sofern dieser existiert.

- Wird ein unzulässiges Argument für `limit` übergeben, wird eine `RuntimeException` geworfen.
- Die Methode `greater` soll **rekursiv** arbeiten. Die Implementierung darf **keine** Schleifen enthalten.
- Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int greater( int[] arr, int limit )
{
    if (  )
    {
        
    }
    else
    {
        throw new RuntimeException();
    }
}
```

Aufgabe 30

c)[2 Punkte] Vervollständigen Sie die *rekursive* Methode `boolean firstRep(int[] arr, int limit)`. Die Methode `firstRep` soll `true` zurückgeben, wenn im Feld `arr` der Wert des Elements mit dem Index `0` mindestens noch ein zweites Mal im Bereich der Indizes `1, ..., limit` vorkommt.

- Die Methode `firstRep` soll `false` zurückgeben, wenn ein unzulässiges Argument für `limit` übergeben wird oder der Wert am Index `0` kein weiteres Mal in `arr` vorkommt.
- Die Methode `firstRep` soll *rekursiv* arbeiten. Die Implementierung darf keine Schleifen enthalten.
- Der Inhalt des Parameters `arr` darf nicht verändert werden.

```
public static boolean firstRep( int[] arr, int limit )
{
    if (  )
    {
        
    }
    else
    {
         ;
    }
}
```

Aufgabe 31

c)[3 Punkte] Vervollständigen Sie die Methode `boolean multiple(int[] arr, int x)` und die *rekursive* Methode `boolean check(int[] arr, int x, int from, int to)`

- Die Methode `multiple` soll das Ergebnis `true` zurückgeben wenn
 - das Feld `arr` eine gerade Anzahl von Elementen besitzt und
 - im Feld `arr` der Wert von `x` mindestens je einmal in der oberen Hälfte und der unteren Hälfte des Feldes vorkommt.Sonst soll `false` zurückgegeben werden.
- Die Methode `check` soll *rekursiv* arbeiten. Die Implementierung darf keine Schleifen enthalten. Die Methode `check` darf den Inhalt des Parameters `arr` nicht verändern.

```
public static boolean multiple( int[] arr, int x )
{
    if (  )
    {
        return check(  )
            && check(  );
    }
    else
    {
        return false;
    }
}
```

```
public static boolean check( int[] arr, int x, int from, int to )
{

}
```

Aufgabe 32

Vervollständigen Sie die **rekursive** Methode `int number(int[] arr, int limit, int val)`. Die Methode `number` soll die Anzahl der Werte des Feldes `arr` im Bereich der Indizes `0... limit` zurückgeben, die gleich dem Wert `val` sind.

- Wird ein unzulässiges Argument für `limit` übergeben, wird eine `RuntimeException` geworfen.
- Die Methode `number` soll **rekursiv** arbeiten. Die Implementierung darf **keine** Schleifen enthalten.
- Der Inhalt des Parameters `arr` darf nicht geändert werden.

```
public static int number( int[] arr, int limit, int val )
{
    if (  )
    {
        
    }
    else
    {
        throw new RuntimeException();
    }
}
```

Aufgabe 33

Vervollständigen Sie die Methode `boolean check(int[] arr, int x)` und die *rekursive* Methode `boolean search(int[] arr, int x, int from, int to)`

- Die Methode `check` soll das Ergebnis `true` zurückgeben, wenn
 - das Feld `arr` eine gerade Anzahl von Elementen besitzt und
 - im Feld `arr` der Wert von `x` mindestens je einmal in der ersten Hälfte und der zweiten Hälfte des Feldes vorkommt.Sonst soll `false` zurückgegeben werden.
- Die Methode `search` soll *rekursiv* arbeiten. Die Implementierung darf keine Schleifen enthalten. Die Methode `search` darf den Inhalt des Parameters `arr` nicht verändern.

```
public static boolean check( int[] arr, int x )
{
    if (  )
    {
        return search(  )
            && search(  );
    }
    else
    {
        return false;
    }
}
```

```
public static boolean search( int[] arr, int x, int from, int to )
{
    
}
```

Aufgabe 34

Vervollständigen Sie die Methode `int positiveMinimum(int[] arr, int limit)`, die *rekursiv* arbeiten soll. Die Methode `positiveMinimum` soll den kleinsten positiven Wert zurückgeben, der im Feld `arr` im Bereich der Indizes `0 ... limit` vorhanden ist. Gibt es keinen positiven Wert, soll `-1` zurückgegeben werden. Wird ein unzulässiges Argument für `limit` übergeben, soll eine Ausnahme der Klasse `IllegalArgumentException` geworfen werden.

Die Implementierung darf *keine* Schleifen enthalten.

```
public static int positiveMinimum( int[] arr, int limit)
{
    if ( limit < 0 || limit >= arr.length )
    {
        throw new IllegalArgumentException();
    }
    else
    {
        
    }
}
```