Aufgabe 1: „Rekursion und Induktion“

(a) Gegeben sei die Methode `BigInteger lfBig(int n)` zur Berechnung der eingeschränkten Linksfakultät:

$$!n := \begin{cases} n!(n-1) - (n-1)!(n-2) & \text{falls } 1 < n < 32767 \\ 1 & \text{falls } n = 1 \\ 0 & \text{sonst} \end{cases}$$

```java
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
import static java.math.BigInteger.ONE;

public class LeftFactorial {

  BigInteger sub(BigInteger a, BigInteger b) {
    return a.subtract(b);
  }

  BigInteger mul(BigInteger a, BigInteger b) {
    return a.multiply(b);
  }

  BigInteger mul(int a, BigInteger b) {
    return mul(BigInteger.valueOf(a), b);
  }

  // returns the left factorial !n
  BigInteger lfBig(int n) {
    if (n <= 0 || n >= Short.MAX_VALUE) {
      return ZERO;
    } else if (n == 1) {
      return ONE;
    } else {
      return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
    }
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

Implementieren Sie unter Verwendung des Konzeptes der *dynamischen Programmierung* die Methode `BigInteger dp(int n)`, die jede $!n$ auch bei mehrfachem Aufrufen mit dem gleichen Parameter höchstens einmal rekursiv berechnet. Sie dürfen der Klasse `LeftFactorial` genau ein Attribut beliebigen Datentyps hinzufügen und die in `lfBig(int)` verwendeten Methoden und Konstanten ebenfalls nutzen.

Wir führen ein Attribut mit dem Namen `store` ein und erzeugen ein Feld vom Typ `BigInteger` mit der Länge $n+1$. Die Länge des Feld $n+1$ hat den Vorteil, dass nicht ständig $n-1$ verwendet werden muss, um den gewünschten Wert zu erhalten.

In der untenstehenden Implementation gibt es zwei Methoden mit dem Namen `dp`. Die untenstehende Methode ist nur eine Hüllmethode, mit der nach außen hin die Berechnung gestartet und das `store`-Feld neu gesetzt wird. So ist es möglich `dp()` mehrmals hintereinander mit verschiedenen Werten aufzurufen (siehe `main()`-Methode).

```java
BigInteger[] store;

BigInteger dp(int n, BigInteger[] store) {
  if (n > 1 && store[n] != null) {
    return store[n];
  }
  if (n <= 0 || n >= Short.MAX_VALUE) {
    return ZERO;
  } else if (n == 1) {
    return ONE;
  } else {
    BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2, store)));
    store[n] = result;
    return result;
  }
}

BigInteger dp(int n) {
  store = new BigInteger[n + 1];
  return dp(n, store);
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

(b) Betrachten Sie nun die Methode `lfLong(int)` zur Berechnung der vorangehend definierten Linksfakultät ohne obere Schranke. Nehmen Sie im Folgenden an, dass der Datentyp `long` unbeschränkt ist und daher kein Überlauf auftritt.

```java
long lfLong(int n) {
  if (n <= 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`

Beweisen Sie *formal* mittels *vollständiger Induktion*:

$$\forall n \geq 0 : \text{lfLong}(n) \equiv \sum_{k=0}^{n-1} k!$$

Induktionsanfang — Beweise, dass $A(1)$ eine wahre Aussage ist. —

$$n = 1 \Rightarrow \texttt{lfLong(1)} = 1 = \sum_{k=0}^{n-1} k! = 0! = 1$$

$$
\begin{aligned}
n = 2 \Rightarrow & \text{lfLong}(2) \\
&= (n+1) \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! \\
&= \texttt{2 * lfLong(1) - 1 * lfLong(0)} \\
&= 2 \\
&= \sum_{k=0}^{1} k! \\
&= 1! + 0! \\
&= 1 + 1 \\
&= 2
\end{aligned}
$$

Induktionsvoraussetzung — Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$.

$$
\texttt{lfLong(n)} = \sum_{k=0}^{n-1} k!
$$

Induktionsschritt — Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

$$
\begin{aligned}
&A(n+1) \\
&\qquad = \texttt{lfLong(n + 1)} \\
&\qquad = \texttt{(n + 1) * lfLong(n) - n * lfLong(n - 1)} \\
&\qquad = (n+1)\sum_{k=0}^{n-1} k! - n \sum_{k=0}^{(n-1)-1} k! && \text{Formel eingesetzt} \\
&\qquad = (n+1)\sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{subtrahiert} \\
&\qquad = n\sum_{k=0}^{n-1} k! + \sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{ausmultipliziert mit } (n+1) \\
&\qquad = \sum_{k=0}^{n-1} k! + n\sum_{k=0}^{n-1} k! - n \sum_{k=0}^{n-2} k! && \text{Reihenfolge der Terme ändern} \\
&\qquad = \sum_{k=0}^{n-1} k! + n\left((n-1)! + \sum_{k=0}^{n-2} k!\right) - n \sum_{k=0}^{n-2} k! && (n-1)! \text{ aus Summenzeichen entfernen} \\
&\qquad = \sum_{k=0}^{n-1} k! + n\left((n-1)! + \sum_{k=0}^{n-2} k! - \sum_{k=0}^{n-2} k!\right) && \text{Distributivgesetz } ac - bc = (a-b)c \\
&\qquad = \sum_{k=0}^{n-1} k! + n(n-1)! && +\Sigma - \Sigma = 0 \\
&\qquad = \sum_{k=0}^{n-1} k! + n! && \text{Fakultät erhöht} \\
&\qquad = \sum_{k=0}^{n} k! && \text{Element zum Summenzeichen hinzugefügt} \\
&\qquad = \sum_{k=0}^{(n+1)-1} k! && \text{mit } (n+1) \text{ an der Stelle von } n
\end{aligned}
$$

Komplette Klasse LeftFactorial

```java
import java.math.BigInteger;
import static java.math.BigInteger.ZERO;
import static java.math.BigInteger.ONE;

public class LeftFactorial {

  BigInteger sub(BigInteger a, BigInteger b) {
    return a.subtract(b);
  }

  BigInteger mul(BigInteger a, BigInteger b) {
    return a.multiply(b);
  }
```

```java

  BigInteger mul(int a, BigInteger b) {
    return mul(BigInteger.valueOf(a), b);
  }

  // returns the left factorial !n
  BigInteger lfBig(int n) {
    if (n <= 0 || n >= Short.MAX_VALUE) {
      return ZERO;
    } else if (n == 1) {
      return ONE;
    } else {
      return sub(mul(n, lfBig(n - 1)), mul(n - 1, lfBig(n - 2)));
    }
  }

  BigInteger[] store;

  BigInteger dp(int n, BigInteger[] store) {
    if (n > 1 && store[n] != null) {
      return store[n];
    }
    if (n <= 0 || n >= Short.MAX_VALUE) {
      return ZERO;
    } else if (n == 1) {
      return ONE;
    } else {
      BigInteger result = sub(mul(n, dp(n - 1, store)), mul(n - 1, dp(n - 2, store)));
      store[n] = result;
      return result;
    }
  }

  BigInteger dp(int n) {
    store = new BigInteger[n + 1];
    return dp(n, store);
  }

  long lfLong(int n) {
    if (n <= 0) {
      return 0;
    } else if (n == 1) {
      return 1;
    } else {
      return n * lfLong(n - 1) - (n - 1) * lfLong(n - 2);
    }
  }

  public static void main(String[] args) {
    LeftFactorial lf = new LeftFactorial();

    for (int i = 0; i < 15; i++) {
      System.out.println(lf.lfBig(i));
    }

    for (int i = 0; i < 15; i++) {
      System.out.println(lf.dp(i));
    }

    for (int i = 0; i < 15; i++) {
```

```java
        System.out.println(lf.lfLong(i));
      }
    }
  }
```

Code-Beispiel auf Github ansehen:
`src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/fruehjahr/LeftFactorial.java`