1 Fill Grid

Given two one-dimensional arrays \( LL \) and \( UR \), fill in the program on the next page to insert the elements of \( LL \) into the lower-left triangle of a square two-dimensional array \( S \) and \( UR \) into the upper-right triangle of \( S \), without modifying elements along the main diagonal of \( S \). You can assume \( LL \) and \( UR \) both contain at least enough elements to fill their respective triangles. (Spring 2020 MT1)

For example, consider

```c
int[] LL = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0 };
int[] UR = { 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 };
int[][] S = {
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 },
    { 0, 0, 0, 0, 0 }
};
```

After calling `fillGrid(LL, UR, S)`, \( S \) should contain

```c
{
    { 0, 11, 12, 13, 14 },
    { 1, 0, 15, 16, 17 },
    { 2, 3, 0, 18, 19 },
    { 4, 5, 6, 0, 20 },
    { 7, 8, 9, 10, 0 }
}
```

(The last two elements of \( LL \) are excess and therefore ignored.)
/** Fill the lower-left triangle of S with elements of LL and the
 * upper-right triangle of S with elements of UR (from left-to
 * right, top-to-bottom in each case). Assumes that S is square and
 * LL and UR have at least sufficient elements. */

public static void fillGrid(int[] LL, int[] UR, int[][] S) {
    int N = S.length;
    int kL, kR;
    kL = kR = 0;
    for (int i = 0; i < N; i += 1) {
        // code here
    }
}
2 Even Odd

Implement the method `evenOdd` by *destructively* changing the ordering of a given `IntList` so that even indexed links *precede* odd indexed links.

For instance, if `lst` is defined as `IntList.list(0, 3, 1, 4, 2, 5), evenOdd(lst)` would modify `lst` to be `IntList.list(0, 1, 2, 3, 4, 5).

You may not need all the lines.

**Hint:** Make sure your solution works for lists of odd and even lengths.

```java
public class IntList {
    public int first;
    public IntList rest;
    public IntList (int f, IntList r) {
        this.first = f;
        this.rest = r;
    }

    public static void evenOdd(IntList lst) {
        if (__________________________________________) {
            return;
        }

        __________________________________________________________
        __________________________________________________________
        __________________________________________________________
        __________________________________________________________

        while (__________________________________________________) {
            __________________________________________________________
            __________________________________________________________
            __________________________________________________________
            __________________________________________________________
        }
    }
}
```

3 Partition

Implement `partition`, which takes in an `IntList lst` and an integer `k`, and *destructively* partitions `lst` into `k` `IntLists` such that each list has the following
properties: Firstly, it is the same length as the other lists. If this is not possible, i.e. \textit{lst} cannot be equally partitioned, then the later lists should be one element smaller. For example, partitioning an \texttt{IntList} of length 25 with \( k = 3 \) would result in partitioned lists of lengths 9, 8, and 8. Secondly, its ordering is consistent with the ordering of \textit{lst}, i.e. items in earlier in \textit{lst} must \texttt{precede} items that are later.

These lists should be put in an array of length \( k \), and this array should be returned. For instance, if \textit{lst} contains the elements 5, 4, 3, 2, 1, and \( k = 2 \), then a possible partition (note that there are many possible partitions), is putting elements 5, 3, 2 at index 0, and elements 4, 1 at index 1.

You may assume you have the access to the method \texttt{reverse}, which destructively reverses the ordering of a given \texttt{IntList} and returns a pointer to the reversed \texttt{IntList}. You may not create any \texttt{IntList} instances. You may not need all the lines.

\textbf{Hint:} You may find the \% operator helpful.

```java
class IntList {
    // Implementation
}

public static IntList[] partition(IntList lst, int k) {
    IntList[] array = new IntList[k];
    int index = 0;
    IntList L = ________________________________
    while (L != null) {
        ________________________________
        ________________________________
        ________________________________
        ________________________________
        ________________________________
        ________________________________
        ________________________________
    }
    return array;
}
```