

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

```
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

```
{
  { 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.)

```
1  /** Fill the lower-left triangle of S with elements of LL and the
2   * upper-right triangle of S with elements of UR (from left-to
3   * right, top-to-bottom in each case). Assumes that S is square and
4   * LL and UR have at least sufficient elements. */
5  public static void fillGrid(int[] LL, int[] UR, int[][] S) {
6      int N = S.length;
7      int kL, kR;
8      kL = kR = 0;
9
10     for (int i = 0; i < N; i += 1) {
11
12         -----
13
14         -----
15
16         -----
17
18         -----
19
20         -----
21
22         -----
23
24         -----
25
26         -----
27
28         -----
29     }
30 }
```

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.

```

1  public class IntList {
2      public int first;
3      public IntList rest;
4      public IntList (int f, IntList r) {
5          this.first = f;
6          this.rest = r;
7      }
8
9      public static void evenOdd(IntList lst) {
10
11         if (_____ ) {
12             return;
13         }
14
15         _____
16
17         _____
18
19         while (_____ ) {
20
21             _____
22
23             _____
24
25             _____
26
27             _____
28         }
29
30         _____
31     }
32 }

```

3 Partition

Implement `partition`, which takes in an `IntList lst` and an integer `k`, and *destructively* partitions `lst` into `k` `IntList`s such that each list has the following properties:

1. It is the **same** length as the other lists. If this is not possible, i.e. `lst` cannot be equally partitioned, then the later lists should be **one** element smaller. For example, partitioning an `IntList` of length 25 with `k = 3` would result in partitioned lists of lengths 9, 8, and 8.
2. Its ordering is consistent with the ordering of `lst`, i.e. items in earlier in `lst` must **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 `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 `reverse`, which destructively reverses the ordering of a given `IntList` and returns a pointer to the reversed `IntList`. You may not create any `IntList` instances. You may not need all the lines.

Hint: You may find the `%` operator helpful.

```

1 public static IntList[] partition(IntList lst, int k) {
2     IntList[] array = new IntList[k];
3     int index = 0;
4     IntList L = _____
5     while (L != null) {
6
7         _____
8
9         _____
10
11        _____
12
13        _____
14
15        _____
16
17        _____
18
19        _____
20    }
21    return array;
22 }
```