1 Flatten

Write a method flatten that takes in a 2-D array x and returns a 1-D array that contains all of the arrays in x concatenated together.

For example, flatten({{1, 2, 3}, {}, {7, 8}}) should return {1, 2, 3, 7, 8}.

(Summer 2016 MT1)

```java
public static int[][] flatten(int[][] x) {
    int totalLength = 0;
    for (int i = 0; i < x.length; i++) {
        totalLength += x[i].length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
            a[aIndex] = x[i][j];
            aIndex++;
        }
    }
    return a;
}
```
2 Skippify

Suppose we have the following IntList class, as defined in lecture and lab, with an added skippify function.

Suppose that we define two IntLists as follows.

```
1 IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
2 IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

Fill in the method skippify such that the result of calling skippify on A and B are as below:
- After calling A.skippify(), A: (1, 3, 6, 10)
- After calling B.skippify(), B: (9, 7, 4)

(Spring ’17, MT1)

```
1 public class IntList {
2     public int first;
3     public IntList rest;
4
5     @Override
6     public boolean equals(Object o) { ... }
7     public static IntList list(int... args) { ... }
8
9     public void skippify() {
10         IntList p = this;
11         int n = 1;
12         while (p != null) {
13             IntList next = p.rest;
14             for (int i = 0; i < n; i += 1) {
15                 if (next == null) {
16                     break;
17                 }
18                 next = next.rest;
19             }
20             p.rest = next;
21             p = p.rest;
22             n++;
23         }
24     }
25     ...
26 }
```
Fill in the blanks below to correctly implement ilsans and dilsans.

(Spring '18, MT1)

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

    /** Non-destructively creates a copy of x that contains no y. */
    public static IntList ilsans(IntList x, int y) {
        if (x == null) {
            return null;
        }
        if (x.first == y) {
            return ilsans(x.rest, y);
        }
        return new IntList(x.first, ilsans(x.rest, y));
    }

    /** Destructively creates a copy of x that contains no y, 
     * without using the keyword "new". */
    public static IntList dilsans(IntList x, int y) {
        if (x == null) {
            return null;
        }
        x.rest = dilsans(x.rest, y);
        if (x.first == y) {
            return x.rest;
        }
        return x;
    }
}
```