

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)

```
1 public static int[] flatten(int[][] x) {
2     int totalLength = 0;
3
4     for (int[] i : x) {
5         totalLength += i.length;
6     }
7
8     int[] a = new int[totalLength];
9     int aIndex = 0;
10
11    for (int[] i : x) {
12        System.arraycopy(x, 0, a, aIndex, i.length);
13
14        aIndex += i.length;
15
16
17
18
19
20    }
21
22    return a;
23 }
24 }
```

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
14            IntList next = p.rest;
15
16            for (int i = 0; i < n; i += 1) {
17
18                if (next == null) {
19
20                    break;
21                }
22
23                next = next.rest;
24            }
25
26            p.rest = next;
27
28            p = next;
29
30            n += 1;
31        }
32    }
33 }
```

3 Remove Duplicates

Fill in the blanks below to correctly implement `removeDuplicates`.
(Spring '17, MT1)

```

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      /**
10     * Given a sorted linked list of items - remove duplicates.
11     * For example given 1 -> 2 -> 2 -> 2 -> 3,
12     * Mutate it to become 1 -> 2 -> 3 (destructively)
13     */
14     public static void removeDuplicates(IntList p) {
15         if (p == null) {
16             return;
17         }
18         IntList current = p.next;
19         IntList previous = p;
20
21         while (current != null) {
22             if (previous.first == current.first) {
23                 previous.rest = current.rest; // previous stay the same
24             } else {
25                 previous = current;
26             }
27             current = current.rest;
28         }
29     }
30 }

```