Thinking Recursively

Understand and check `isDivisible(13, 2)` by tracing one level.

```java
/** True iff X is divisible by
* some number >=K and < X,
* given K > 1. */
boolean isDivisible (int x, int k) {
  if (k >= x)
    return false;
  else if (x % k == 0)
    return true;
  else
    return isDivisible (x, k+1);
}
```

Lesson: Comments aid understanding. Make them count!

• Call assigns `x=13, k=2`
• Body has form ‘if (k >= x) S₁ else S₂’.
• Since 2 < 13, we evaluate the first else.
• Check if 13 mod 2 = 0; it’s not.
• Left with `isDivisible(13,3)`.
• Rather than tracing it, instead use the comment:
  • Since 13 is not divisible by any integer in the range 3..12 (and 3 > 1), `isDivisible(13,3)` must be false, and we’re done!
• Sounds like that last step begs the question. Why doesn’t it?

Iteration

• `isDivisible` is tail recursive, and so creates an iterative process.
• Traditional “Algol family” production languages have special syntax for iteration. Four equivalent versions of `isDivisible`:

```java
if (k >= x)
  return false;
else if (x % k == 0)
  return true;
else
  return isDivisible (x, k+1);
```

```java
while (k < x) { // ! (k >= x)
  if (x % k == 0)
    return true;
  k = k+1;
// or k += 1, or k++ (yuch).
}
return false;
```

```java
int k1 = k;
while (k1 < x) {
  if (x % k1 == 0)
    return true;
  k1 += 1;
}
return false;
```

```java
for (int k1 = k; k1 < x; k1 += 1) {
  if (x % k1 == 0)
    return true;
}
```

More Iteration: Sort an Array

Problem. Print out the command-line arguments in order:

```bash
% java sort the quick brown fox jumped over the lazy dog
brown dog fox jumped lazy over quick the
```

Plan.

```java
class sort {
  public static void main (String[] words) {
    sort (words, 0, words.length-1);
    print (words);
  }

  /** Sort items A[L..U], with all others unchanged. */
  static void sort (String[] A, int L, int U) {
    // TOMORROW
  }

  /** Print A on one line, separated by blanks. */
  static void print (String[] A) {
    // TOMORROW
  }
}
```

• Call assigns `x=13, k=2`
• Body has form ‘if (k >= x) S₁ else S₂’.
• Since 2 < 13, we evaluate the first else.
• Check if 13 mod 2 = 0; it’s not.
• Left with `isDivisible(13,3)`.
• Rather than tracing it, instead use the comment:
  • Since 13 is not divisible by any integer in the range 3..12 (and 3 > 1), `isDivisible(13,3)` must be false, and we’re done!
• Sounds like that last step begs the question. Why doesn’t it?
**Selection Sort**

/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k = indexOfLargest (A, L, U);
        sort (A, L, U-1);  // Sort items L to U-1 of A
    }
}

Iterative version:

    while (L < U) {
        int k = indexOfLargest (A, L, U);
        U -= 1;
    }

And we're done! Well, OK, not quite.

---

**Really Find Largest**

/** Value k, I0<=k<=I1, such that V[k] is largest element among
 * V[I0], ... V[I1]. Requires I0<=I1. */
static int indexOfLargest (String[] V, int i0, int i1) {
    if (i0 >= i1)
        return i1;
    else /* if (i0 < i1) */ {
        int k = indexOfLargest (V, i0+1, i1);
        return (V[i0].compareTo (V[k]) > 0) ? i0 : k;
        // or if (V[i0].compareTo (V[k]) > 0) return i0; else return k;
    }
}

Iterative:

    int i, k;
    k = i1;  // Deepest iteration
    for (i = i1-1; i >= i0; i -= 1)
        k = (V[i].compareTo (V[k]) > 0) ? i : k;
    return k;

---

**Finally, Printing**

/** Print A on one line, separated by blanks. */
static void print (String[] A) {
    for (int i = 0; i < A.length; i += 1)
        System.out.print (A[i] + " ");
    System.out.println ();
}

/* Looking ahead: There's a brand-new syntax for the for loop here (as of J2SE 5): */
for (String s : A)
    System.out.print (s + " ");
/* Use it if you like, but let's not stress over it yet! */