Recursion may require extra parameters

Consider writing a recursive procedure to sort an array $b$ — elsewhere in this JavaHyperText we show two such recursive procedures, quicksort and mergesort. We can’t just use

```java
/** Sort $b$. */
public static void sort(int[] b)
```

because there is no way to write a recursive call with an argument that is “smaller” than $b$. Instead, we generally add two extra parameters and write

```java
/** Sort $b[h..k]$. */
public static void sort(int[] b, int h, int k)
```

We could, if we wanted, have both procedures, as shown to the right, with the one-parameter `sort` calling the recursive `sort`. This is standard practice.

Making recursive methods on strings more efficient

In introducing recursion, we wrote function `ct`, on the right, to calculate the number of times a character $c$ appears in a string $s$. This method is far too inefficient, taking time proportional to the square of the length of $s$. This is because the substring operation takes time proportional to the length of the substring—a new String object has to be created and the substring copied into it.

To get a more efficient recursive function, write a function `ct` with three parameters, as shown below, and change the two-parameter function `ct` to call the three-parameter one. Recursive function `ct(c, s, h)` does not use the expensive substring operation, and the time it takes is proportional to the length of $s$.

The three-parameter function `ct` is harder to read and understand than the original one, but the increased efficiency is worth it.

Since we are talking about efficiency, we also add that an iterative implementation of `ct` would be even better because there would be no recursive calls at all. We wrote `ct` recursively only to help introduce recursion, and the iterative version is preferred.

An exercise for you

Function `isPal`, to the right below, determines whether its parameter $s$ is a palindrome —whether it reads the same backward and forward. Examples of palindromes are “” (the empty string), “$n$”, and “noon”.

Function `isPal` is inefficient because it uses the substring operation. Rewrite it using extra parameters, as we did above.

```java
public static boolean isPal(String s) {
    int n= s.length()-1;
    return s.charAt(0) == s.charAt(n) && isPal(s.substring(1, n));
}
```