CS61B Lecture 6

Monday, February 3, 2020

Sorting

Here's a common, or at least, plausible problem: we want to print out the command-line arguments in lexicographic order.

```
% java sort the quick brown fox
brown fox quick the
```

And here's the plan:

```
public class sort {

    /** Sort and print WORDS lexicographically */
    public static void main(String[] words){
        sort(words,0,words.length-1);
        print(words);
    }

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

    /** Print the items of the array on one line */
    static void print(String[] A){
        // Do something
    }

}
```

How do we know if our code works?

Unit testing refers to the testing of individual units (methods, classes) within a program, rather than the whole program. In this class, we mainly use the JUnit tool for unit testing. An example is `AGTestYear.java` in Lab 1.

Integration testing refers to the testing of entire (integrated) set of modules — the whole program. In this course, we'll look at various ways to run the program against prepared inputs and checking the output.

Regression testing refers to testing with the specific goal of checking that fixes, enhancements, or other changes have not introduced faults (regressions).

Test-Driven Development

Idea: write tests first.

Implement unit at a time, run tests, fix and refactor until it works. We're not really going to push it in this course, but it is useful and has quite a following.

Testing `sort` program

This is pretty easy: just give a bunch of arrays to sort and then
make sure they each get sorted properly. We just have to make sure we cover the necessary cases:

- **Corner cases.** For example, empty array, one-element, all elements the same.
- **Representative "middle" cases**. For example, elements reversed, elements in order, one pair of elements reversed, ...

Simple JUnit

The JUnit package provides some handy tools for unit testing. The Java annotation `@Test` on a method tells the JUnit machinery to call that method. (An annotation in Java, like a decorator in Python, provides information about a method, class, etc., that can be examined within Java itself.) A collection of methods with names beginning with assert then allow your test cases to check conditions and report failures.

Our `sort` method

Below is a very quick overview of what our `sort` method will look like:

```
/** Sort items A[L..U], with all others unchanged. */
static void sort(String[] A, int L, int U) {
    if (L < U) {
        int k = /*( Index s.t. A[k] is largest in A[L],...,A[U])*/;
        /*{ swap A[k] with A[U] }*/;
        /*{ Sort items L to U-1 of A. }*/;
    }
}
```

And here's the completed program:

```
/** 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);
        String tmp = A[k]; A[k] = A[U]; A[U] = tmp;
        sort(A, L, U-1); // Sort items L to U-1 of A
    }
}
```

And here's an iterative version:

```
while (L < U) {
    int k = indexOfLargest(A, L, U);
    String tmp = A[k]; A[k] = A[U]; A[U] = tmp;
    U -= 1;
}
```

Now that this is done, we need to write the `indexOfLargest` method.

```java
/** Index 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 {
        int k = indexOfLargest(V, i0 + 1, i1);
        return (V[i0].compareTo(V[k]) > 0) ? i0 : k;
        // if (V[i0].compareTo(V[k]) > 0) return i0
        // else return k;
    }
}
```

And here is an iterative version, which is **not** tail-recursive.

```java
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
```

Printing

And finally, we need to provide a special method that can print arrays neatly.

```java
/** 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();
    }
}
```

Java also provides a simple, specialized syntax for looping through an entire array:

```java
for (String s : A) {
    System.out.print(s + " ");
}
```