CS61B Lecture #6: Arrays

Readings for Monday: Chapters 2, 4 of Head First Java (5 also useful, but its really review).

Upcoming readings: Chapters 7, 8 of Head First Java.

Public Service Announcement. HKN has free drop-in tutoring M-F 11AM–5PM in 290 Cory and 345 Soda. Visit hkn.eecs.berkeley.edu to find out about all of our student services:

- Weekly Tutoring Schedule
- Midterm Review Sessions
- Exam Archive
- Course Surveys
- Course Guide

A Few Samples

Java

```java
int[] x, y, z;
String[] a;
x = new int[3];
y = x;
a = new String[3];
x[1] = 2;
y[1] = 3;
a[1] = "Hello";
```

Results

```
x: 0 3 0
y: 
z: [ ]
a: [ ]
   Hello
```

Example: Accumulate Values

Problem: Sum up the elements of array A.

```java
static int sum (int[] A) {
    int N;
    N = 0; // New (1.5) syntax
    for (int i = 0; i < A.length; i += 1) for (int x : A)
        N += A[i];
    return N;
}
```

// For the hard-core: could have written

```java
int N, i;
for (i=0, N=0; i<A.length; N += A[i], i += 1) // or just 
    }
```

// But please don’t: it’s obscure.
**Example: Insert into an Array**

**Problem:** Want a call like `insert (A, 2, "gnu")` to convert (destructively)

```
A: bear
gazelle
hartebeest
skunk
```

```
A: bear
gazelle
gnu
hartebeest
```

// ** Insert X at location K in ARR, moving items
* K, K+1, ... to locations K+1, K+2, ...
* The last item in ARR is lost. */
static void insert (String[] arr, int k, String x) {
    for (int i = arr.length-1; i > k; i -= 1) // Why backwards?
        arr[i] = arr[i-1];
    // Alternative to this loop:// System.arraycopy (arr, k,
    // from arr, k+1, arr.length-k-1)
    arr[k] = x;
}

**Useful tip:** Can write just `arraycopy` by including at top of file:
```
import static java.lang.System.*;
```

**Growing an Array**

**Problem:** Suppose that we want to change the description above, so that `A = insert2 (A, 2, "gnu")` does not shove "skunk" off the end, but instead "grows" the array.

```
A: bear
gazelle
hartebeest
skunk
```

```
A: bear
gazelle
gnu
hartebeest
skunk
```

// ** Return array, r, where r.length = ARR.length+1; r[0..K-1]
* the same as ARR[0..K-1], r[k] = x, r[K+1..] same as ARR[K..]. */
static String[] insert2 (String[] arr, int k, String x) {
    String[] result = new String[arr.length + 1];
    arraycopy (arr, 0, result, 0, k);
    arraycopy (arr, k, result, k+1, arr.length-k);
    result[k] = x;
    return result;
}

- Why do we need a different return type from `insert`??

**Example: Merging**

**Problem:** Given two sorted arrays of ints, `A` and `B`, produce their merge: a sorted array containing all items from `A` and `B`.

```
A: 0 2 3 6 9 11
B: 1 4 5 7 8
```

result: 0 1 2 3 4 5 6 7 8 9 11

**Example: Merging Program**

**Problem:** Given two sorted arrays of ints, `A` and `B`, produce their merge: a sorted array containing all from `A` and `B`.

```
/** Assuming A and B are sorted, returns their merge. */
public static int[] merge(int[] A, int[] B) {
    return merge(A, 0, A.length-1, B, 0, B.length-1);
}
/** The merge of A[L0..U0] and B[L1..U1] assuming A and B sorted. */
static int[] merge(int[] A, int L0, int U0, int[] B, int L1, int U1) {
    int N = U0 - L0 + U1 - L1 + 2;
    int[] C = new int[N];
    if (U0 < L0) arraycopy (B, L1, C, 0, N);
    else if (U1 < L1) arraycopy (A, L0, C, 0, N);
    else if (A[L0] <= B[L1]) {
        C[0] = A[L0];
        arraycopy (merge(A, L0+1, U0, B, L1, U1), 0, C, 1, N-1);
    } else {
        C[0] = B[L1];
        arraycopy (merge(A, L0, U0, B, L1+1, U1), 0, C, 1, N-1);
    }
    return C;
}
```

What is wrong with this implementation?
A Tail-Recursive Strategy

```java
public static int[] merge(int[] A, int[] B) {
    return merge(A, 0, A.length-1, B, 0, B.length-1,
                 new int[A.length+B.length], 0);
}
/** Merge A[L0..U0] and B[L1..U1] into C[K...], assuming A and B sorted. */
static int[] merge(int[] A, int L0, int U0, int[] B, int L1, int U1, int[] C, int k){
    if (U0 < L0)
        arraycopy(B, L1, C, k, U1-L1+1);
    else if (U1 < L1)
        arraycopy(A, L0, C, k, U0-L0+1);
    else if (A[L0] <= B[L1]) {
        C[k] = A[L0];
        merge(A, L0+1, U0, B, L1, U1, C, k+1);
    } else {
        C[k] = B[L1];
        merge(A, L0, U0, B, L1+1, U1, C, k+1);
    }
    return C;
}
```

This last method merges part of A with part of B into part of C. For example, consider a possible call `merge(A, 2, 4, B, 1, 4, C, 3)`

![Diagram showing merge process]

Iterative Solution

In general, we don’t use either of the previous approaches in languages like C and Java. Array manipulation is most often iterative:

```java
public static int[] merge(int[] A, int[] B) {
    int[] C = new int[A.length + B.length];
    int L0, L1;
    L0 = L1 = 0;
    for (int k = 0; k < C.length; k += 1) {
        if (L0 >= A.length) {
            C[k] = B[L1]; L1 += 1;
        } else if (L1 >= B.length) {
            C[k] = A[L0]; L0 += 1;
        } else if (A[L0] <= B[L1]) {
            C[k] = A[L0];
            merge(A, L0+1, U0, B, L1, U1, C, k+1);
        } else {
            C[k] = B[L1];
            merge(A, L0, U0, B, L1+1, U1, C, k+1);
        }
    }
    return C;
}
```

Iterative Solution II

```java
public static int[] merge(int[] A, int[] B) {
    int[] C = new int[A.length + B.length];
    int L0, L1;
    L0 = L1 = 0;
    for (int k = 0; k < C.length; k += 1) {
        if (L0 >= A.length) {
            C[k] = B[L1]; L1 += 1;
        } else if (L1 >= B.length) {
            C[k] = A[L0]; L0 += 1;
        } else if (A[L0] <= B[L1]) {
            C[k] = A[L0];
            merge(A, L0+1, U0, B, L1, U1, C, k+1);
        } else {
            C[k] = B[L1]; L1 += 1;
        }
    }
    return C;
}
```
Multidimensional Arrays

- What about two- or higher-dimensional layouts, such as

\[
A = \begin{bmatrix}
  2 & 3 & 4 & 5 \\
  4 & 9 & 16 & 25 \\
  8 & 27 & 64 & 125
\end{bmatrix}
\]

- Not primitive in Java, but we can build them as arrays of arrays:

```java
int[][] A = new int[3][];
A[0] = new int[] {2, 3, 4, 5};
A[2] = new int[] {8, 27, 64, 125};
```

// or
```
int[][] A = { {2, 3, 4, 5}, {4, 9, 16, 25}, {8, 27, 64, 125} };
```

Exotic Multidimensional Arrays

- Since every element of an array is independent, there is no single "width" in general:

```
int[][] A = new int[5][];
A[0] = new int[] {};
A[1] = new int[] {0, 1};
A[3] = new int[] {6, 7, 8};
```

- What does this print?

```
int[][] ZERO = new int[3][];
ZERO[0] = ZERO[1] = ZERO[2] = new int[] {0, 0, 0};
ZERO[0][1] = 1;
System.out.println(ZERO[2][1]);
```

\[
A = \begin{bmatrix}
  0 & 1 \\
  2 & 3 & 4 & 5 \\
  6 & 7 & 8 \\
  2 & 9
\end{bmatrix}
\]