CS61B Lecture #5: Arrays

structured container whose components are

fixed integer.

e of **length** simple containers of the same type, num-
m 0.

eld usually implicit in diagrams.)

nonymous, like other structured containers.

red to with pointers.

inted to by A,

A.length

component i is A[i] (i is the *index*)

feature: index can be *any integer expression*.

Recreation

of the coefficients of

$$(1 - 3x + 3x^2)^{743}(1 + 3x - 3x^2)^{744}$$

and collecting terms?

Example: Accumulate Values

up the elements of array A.

```
(int[] A) {

= 0; i < A.length; i += 1)
```

```
// New (1.5) syntax
for (int x : A)
   N += x;
```

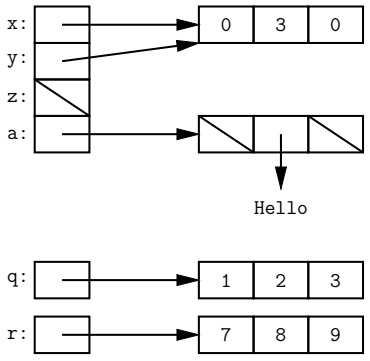
```
d-core: could have written

 i<A.length; N += A[i], i += 1)
just ;

don't: it's obscure.
```


A Few Samples

Java

```
;

[3];

';

{ 1, 2, 3 };
for declarations:
 8, 9 };
```

Results



(Aside) Java Shortcut

Can write just 'arraycopy' by including at the top of the

```
ic java.lang.System.arraycopy;
```

define the simple name arraycopy to be the equivalent
g.System.arraycopy in the current source file."

ame for out so that you can write

```
(...);
```

println(...);

claration like

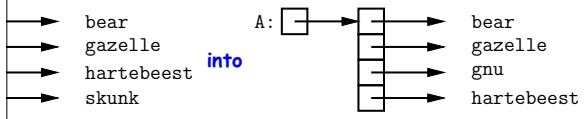
```
ic java.lang.Math.*;
```

all the (public) static definitions in java.lang.Math and
available in this source file by their simple names (the
he last dot)."

unctions like sin, sqrt, etc.

Example: Insert into an Array

t a call like insert(A, 2, "gnu") to convert (destruc-

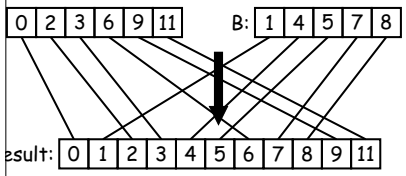



```
location K in ARR, moving items K, K+1, ... to locations
..  The last item in ARR is lost. */
rt (String[] arr, int k, String x) {
rr.length-1; i > k; i -= 1) // Why backwards?
[i-1];
 to this loop:
rraycopy(arr, k, arr, k+1, arr.length-k-1);*/
```

from *to* *# to copy*

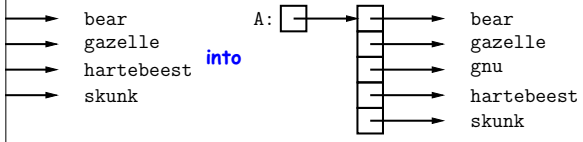
Example: Merging

n two sorted arrays of ints, A and B, produce their
l array containing all items from A and B.



Growing an Array

ose that we want to change the description above, so
t2 (A, 2, "gnu") does *not* shove "skunk" off the end,
ws" the array.




```
, r, where r.length = ARR.length+1; r[0..K-1]
ARR[0..K-1], r[k] = x, r[K+1..] same as ARR[K..]. */
insert2(String[] arr, int k, String x) {
t = new String[arr.length + 1];
 0, result, 0, k);
 k, result, k+1, arr.length-k);
```

d a different return type from `insert2`??

Example: Merging Program

n two sorted arrays of ints, A and B, produce their
l array containing all from A and B.
der to solve this recursively, it is useful to *generalize*
ction to allow merging *portions* of the arrays.

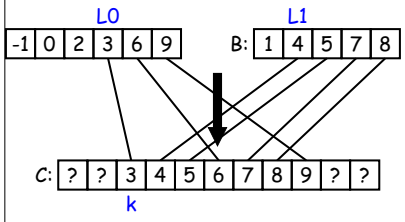
```
nd B are sorted, returns their merge. */
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0);


 A[L0..] and B[L1..] assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1) {
gth - L0 + B.length - L1; int[] C = new int[N];
ength) arraycopy(B, L1, C, 0, N);
= B.length) arraycopy(A, L0, C, 0, N);
] <= B[L1]) {
0]; arraycopy(mergeTo(A, L0+1, B, L1), 0, C, 1, N-1);

1]; arraycopy(mergeTo(A, L0, B, L1+1), 0, C, 1, N-1);
```

What is wrong with this implementation?

A Tail-Recursive Strategy

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){
```

d merges *part* of A with part of B into part of C. For
er a possible call `mergeTo(A, 3, B, 1, C, 2)`



A Tail-Recursive Solution

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){
ength && L1 >= B.length) {

) {
0];
geTo(A, ??, B, ??, C, ??)

1];
geTo(A, ??, B, ??, C, ??)
```


A Tail-Recursive Solution

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){


) {
0];
geTo(A, ??, B, ??, C, ??)

1];
geTo(A, ??, B, ??, C, ??)
```

A Tail-Recursive Solution

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){
ength && L1 >= B.length) {

 >= B.length || (L0 < A.length && A[L0] <= B[L1])) {
0];
geTo(A, L0 + 1, B, L1, C, k + 1);

1];
geTo(A, ??, B, ??, C, ??)
```


Iterative Solution

lon't use either of the previous approaches in languages
Array manipulation is most often iterative:

```
t[] merge(int[] A, int[] B) {
 int[A.length + B.length];
 0, B, 0, C, 0)
;
 0;


 B.length || (L0 < A.length && A[L0] <= B[L1])) {
= A[L0];


= B[L1];
```


Iterative Solution

't use either of the previous approaches in languages like C and
lation is most often iterative:

```
t[] merge(int[] A, int[] B) {
 int[A.length + B.length];
 0, B, 0, C, 0)
;
 0;

.length || L1 < B.length) {
 B.length || (L0 < A.length && A[L0] <= B[L1])) {
= A[L0];
 1; k += 1;

= B[L1];
 1; k += 1;
```


A Tail-Recursive Solution

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){
ength && L1 >= B.length) {

 >= B.length || (L0 < A.length && A[L0] <= B[L1])) {
0];
geTo(A, ??, B, ??, C, ??)

1];
geTo(A, ??, B, ??, C, ??)
```


A Tail-Recursive Solution

```
t[] merge(int[] A, int[] B) {
o(A, 0, B, 0, new int[A.length+B.length], 0);


] and B[L1..] into C[K..], assuming A and B sorted. */
geTo(int[] A, int L0, int[] B, int L1, int[] C, int k){
ength && L1 >= B.length) {

 >= B.length || (L0 < A.length && A[L0] <= B[L1])) {
0];
geTo(A, L0 + 1, B, L1, C, k + 1);

1];
geTo(A, L0, B, L1 + 1, C, k + 1);
```


Iterative Solution

't use either of the previous approaches in languages like C and
lation is most often iterative:

```
t[] merge(int[] A, int[] B) {
 int[A.length + B.length];
 0, B, 0, C, 0)
;
 0;

.length || L1 < B.length) {
 B.length || (L0 < A.length && A[L0] <= B[L1])) {
= A[L0];


= B[L1];
```

Iterative Solution II

or loop:

```
t[] merge(int[] A, int[] B) {
 int[A.length + B.length];


); k < C.length; k += 1) {
 B.length || (L0 < A.length && A[L0] <= B[L1])) {
= A[L0]; L0 += 1;

= B[L1]; L1 += 1;
```

er int k = 0):

$th \ \wedge \ 0 \leq L1 < B.length \wedge \ C.length = A.length + B.length \ \wedge \ k = L0 + L1$

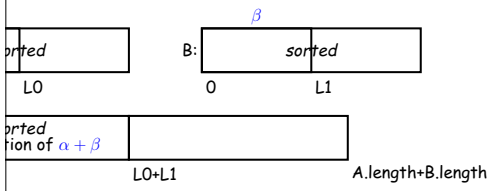
rmutation of A[0:L0] + B[0:L1]

re sorted.

Alternative Solution: Removing k

riant that k=L0+L1 simplifies things:

```
t[] merge(int[] A, int[] B) {
 int[A.length + B.length];
L0 = L1 = 0;
1 < C.length) {
 B.length || (L0 < A.length && A[L0] < B[L1])) {
+ L1] = A[L0]; L0 += 1;

+ L1] = B[L1]; L1 += 1;
```



Multidimensional Arrays

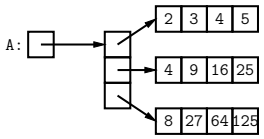
r higher-dimensional layouts, such as

$$A = \begin{array}{|c|c|c|c|} \hline 2 & 3 & 4 & 5 \\ \hline 4 & 9 & 16 & 25 \\ \hline 8 & 27 & 64 & 125 \\ \hline \end{array}$$?

Multidimensional Arrays in Java

itive in Java, but we can build them as arrays of arrays:

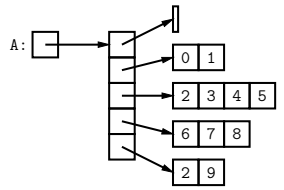
```
ew int[3][];
t[] {2, 3, 4, 5};
t[] {4, 9, 16, 25};
t[] {8, 27, 64, 125};


[] { {2, 3, 4, 5},
     {4, 9, 16, 25},
     { 8, 27, 64, 125} };

 {2, 3, 4, 5},
 {4, 9, 16, 25},
 {8, 27, 64, 125} };

ew A[3][4];
0; i < 3; i += 1)
j = 0; j < 4; j += 1)
j] = (int) Math.pow(j + 2, i + 1);
```



Exotic Multidimensional Arrays

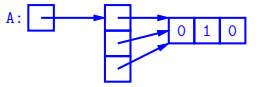
ment of an array is independent, there is no single "width" in gen-

```
= new int[5][];
 int[] {};
 int[] {0, 1};
 int[] {2, 3, 4, 5};
 int[] {6, 7, 8};
 int[] {9};
```



print?

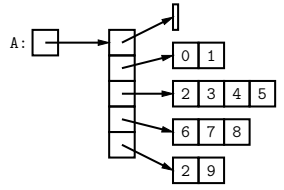
```
RO = new int[3][];
ZERO[1] = ZERO[2] =
t[] {0, 0, 0};
 = 1;
.println(ZERO[2][1]);
```



Exotic Multidimensional Arrays

ment of an array is independent, there is no single "width" in gen-

```
= new int[5][];
 int[] {};
 int[] {0, 1};
 int[] {2, 3, 4, 5};
 int[] {6, 7, 8};
 int[] {9};
```



print?

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