More Arrays

Sorting and Nested Arrays

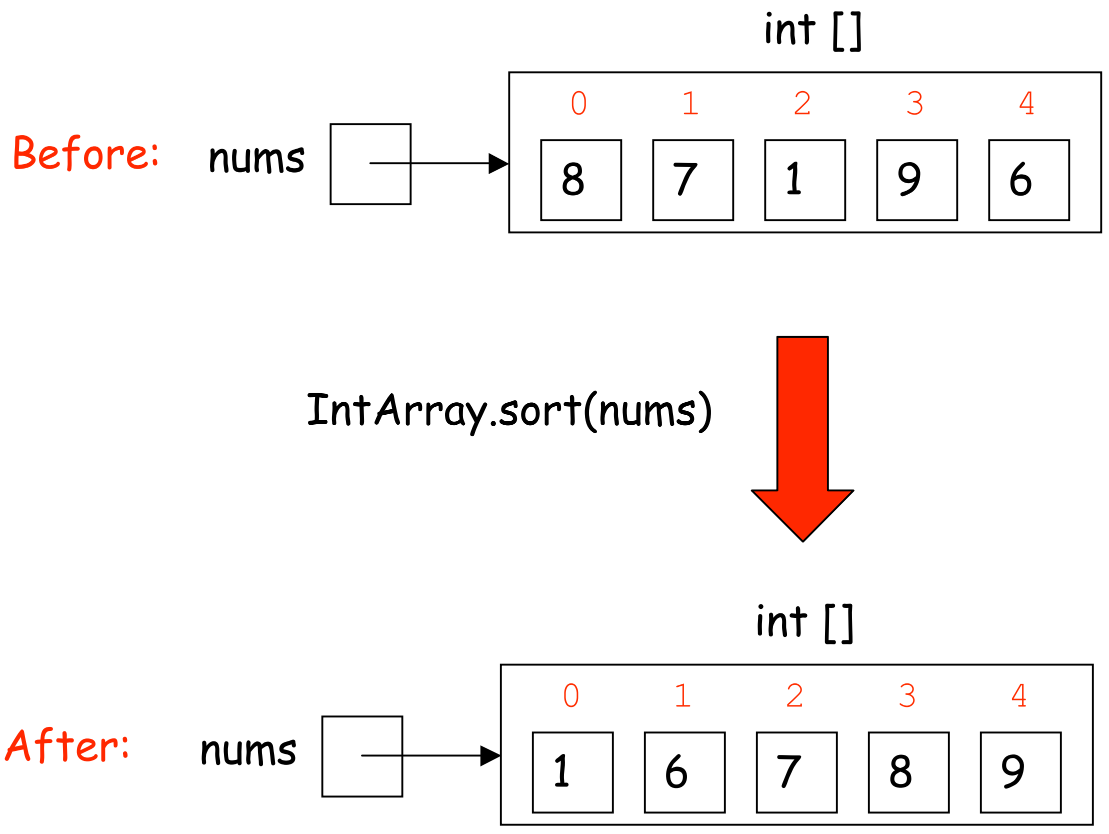
Tuesday, November 13, 2007

CS111 Computer Programming

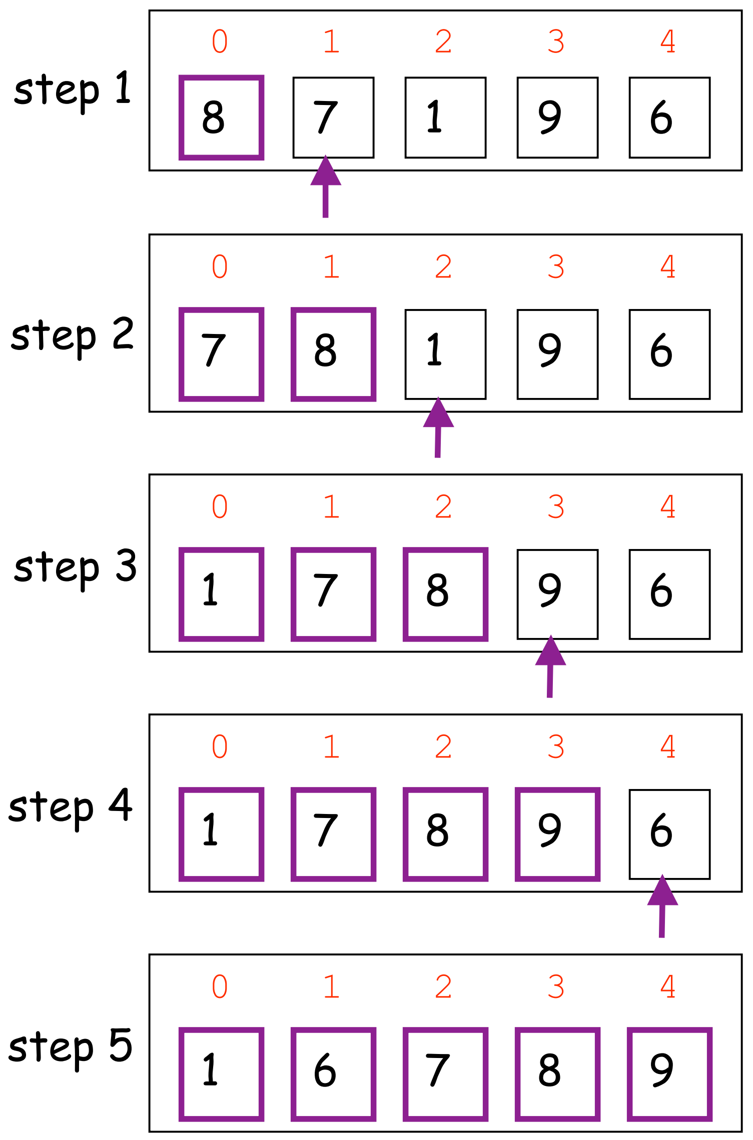
Department of Computer Science
Wellesley College

Destructive Array Sorting

```
public static void sort (int [] a);
Modify the array a so that all the elements
of a are sorted from low to high.
```



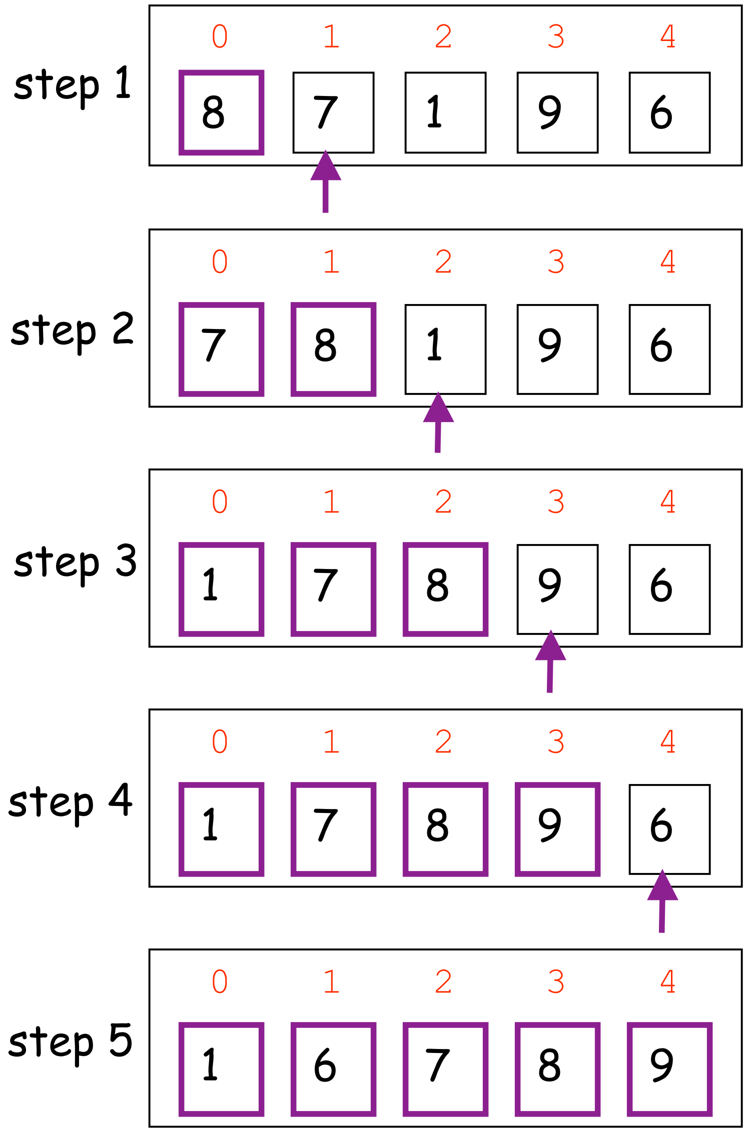
One Sorting Algorithm: Insertion Sort



Thinking about Insertion Sort

Let a[i..j] stand for the slots of array a between indices i and j (inclusive).

What can you say about a[0..(i-1)] at step i of the algorithm?



Insertion Sort: Loop Invariant

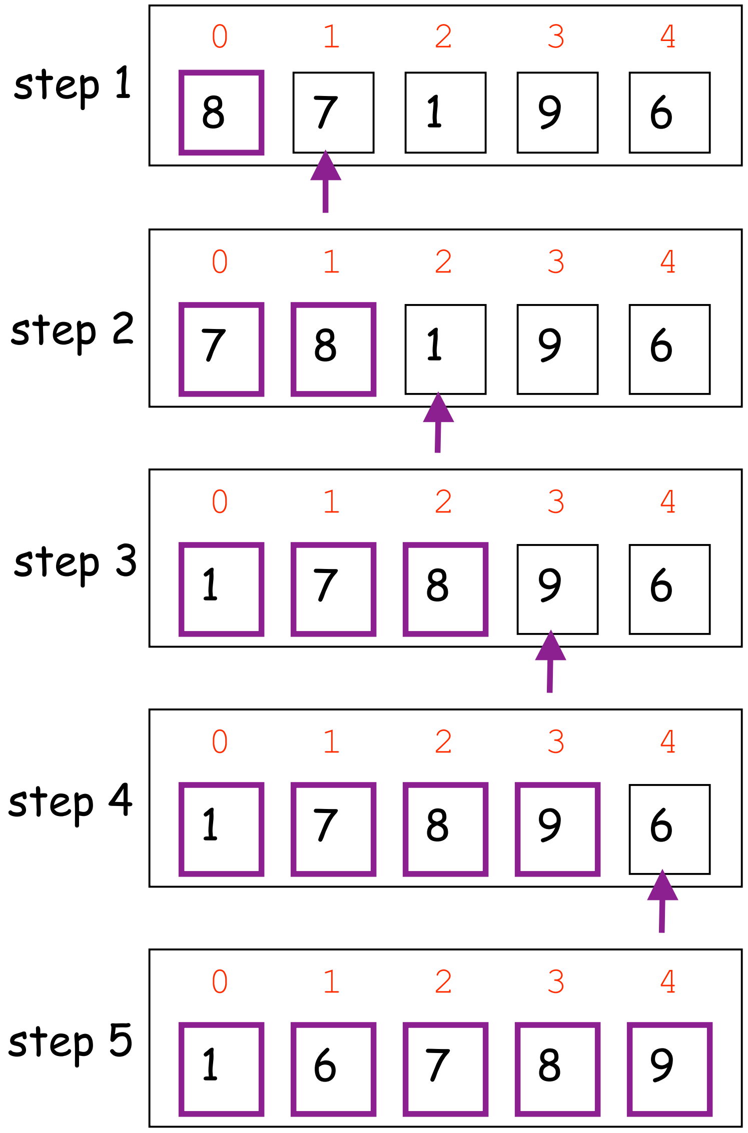
Insertion sort is based on the following **loop invariant** (something that is true at the beginning of every step of an iteration):

> At the beginning of the ith step to insertion sort, a[0..(i-1)] is sorted.

This is true for the first step (i=1) because a[0..0] is sorted (it has only one element).

To make the invariant true for the (i+1)th step, the ith step moves the value in a[i] leftward until a[0..i] is sorted.

If there are n elements in a, the array mustbe sorted by the beginning of the nth step, since a[0..(n-1)] is sorted.



Insertion Sort in Java

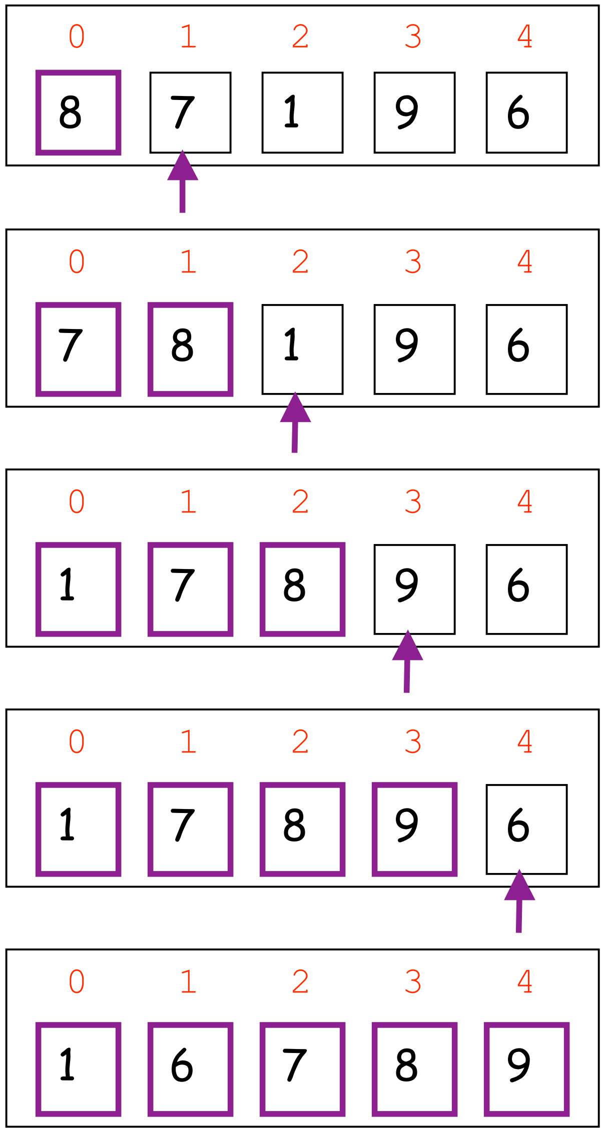
Suppose we have been given a helper method that performs the leftward insertion:

```
public static void insert (int [] a, int i);
```

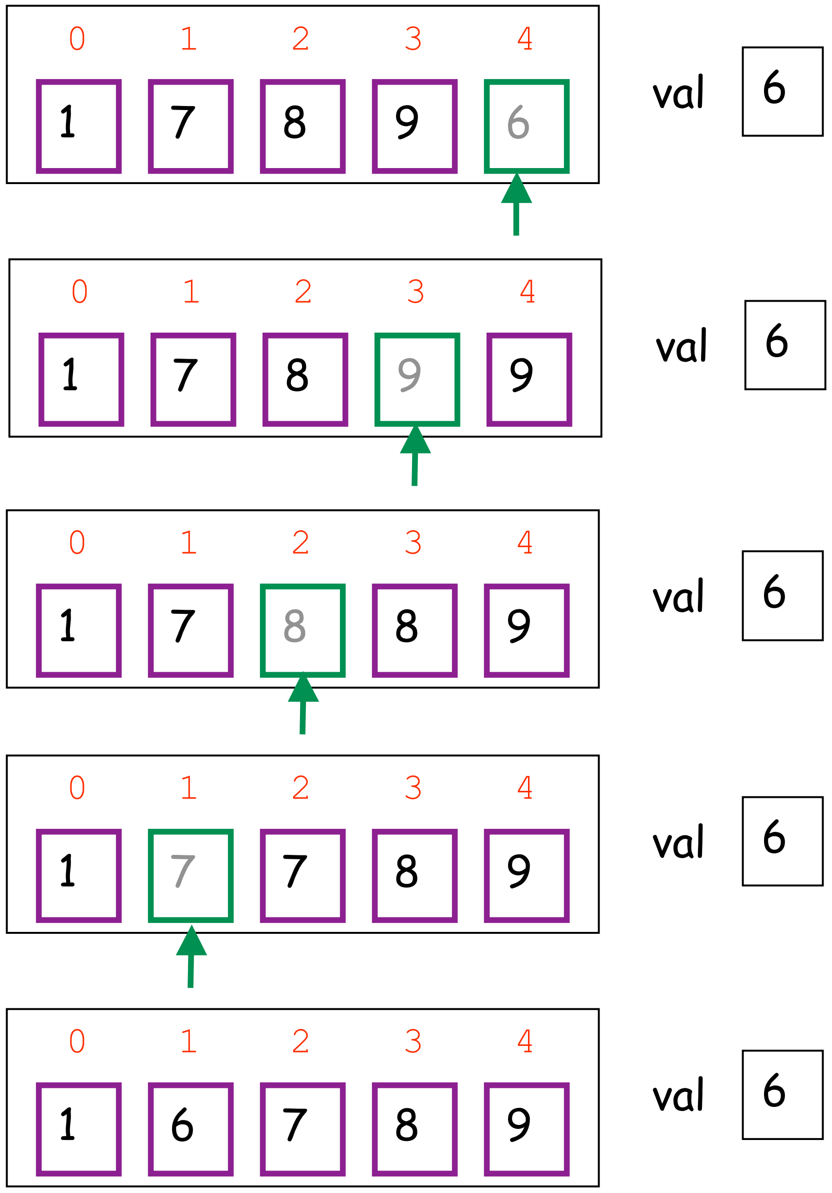
Assume a[0..(i-1)] is sorted. Move the value in a[i] leftward until a[0..i] is sorted.

Write sort() using the insertion sort algorithm:

```
public static void sort (int [] a) {




}
```



How To Insert?



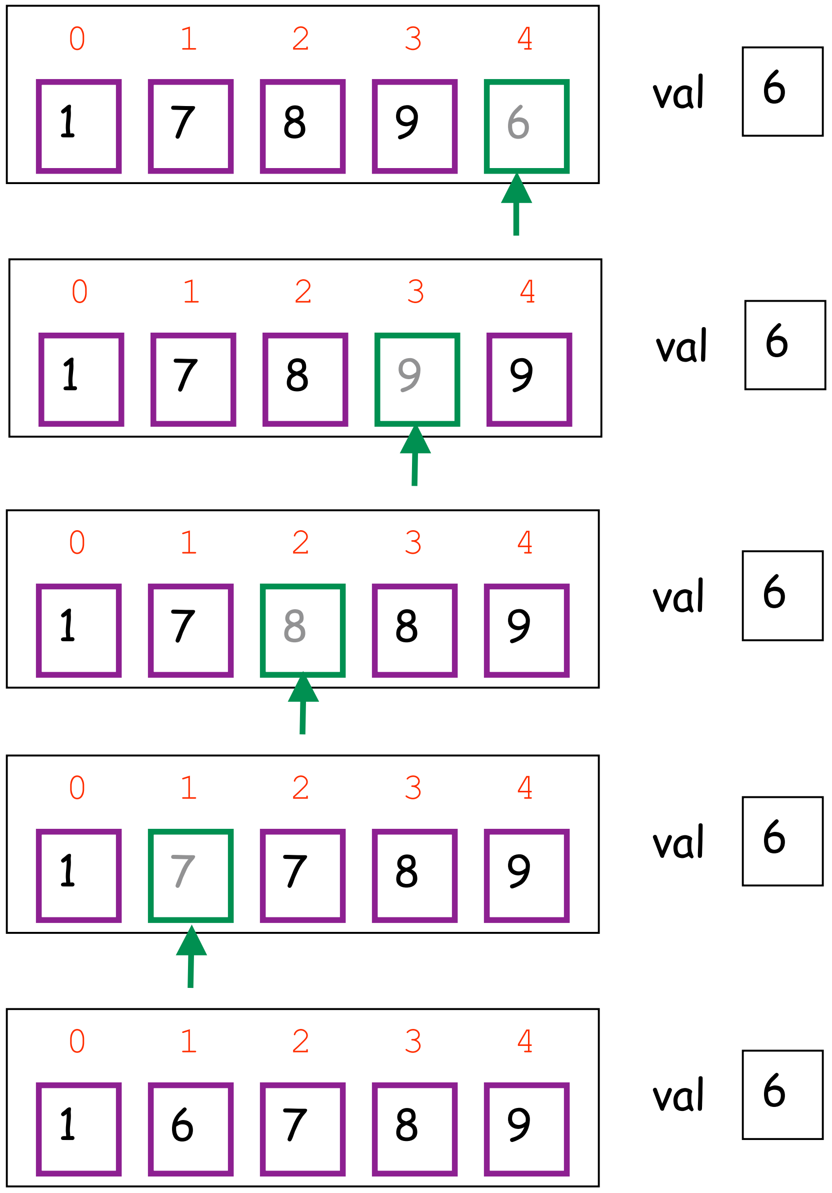
Leftward Insertion

To insert a[i] leftward, save it in a variable val and bubble down the "hole" left behind (shown in green) to the spot where val would be in correct sorted order.

Finally, fill the hole with val.

In Java:

```
public static void insert (int [] a, int i) {
    int val = a[i]; // remember value in a[i]







}
```



Insertion Sort via Nested Loops

```
public static void sort (int [ ] a) {
    for (int i = 1; i < a.length; i++) {
        // Loop invariant: a[0..(i-1)] is sorted
        // Insert a[i] into a[0..(i-1)] so that a[0..i] is sorted.
        int val = a[i]; // remember value in a[i] before we overwrite it.
        int j = i; // initialize insertion index j.
        while ((j > 0) && (a[j-1] > val)) { // order of tests is crucial!
            a[j] = a[j-1]; // shift value in a[j-1] to right = shift hole in a[j] to left.
            j--;
        }
        // At this point, either j = 0 or ((j >=1) && (a[j-1] <= val))
        a[j] = val; // Insert val where it belongs in sorted order.
    }
    // At this point, a[0..(a.length-1)], so whole array is sorted.
}
```

Sorting Discussion

- Insertion sort can be modified to work on other kinds of elements. For example, to sort arrays of strings, change

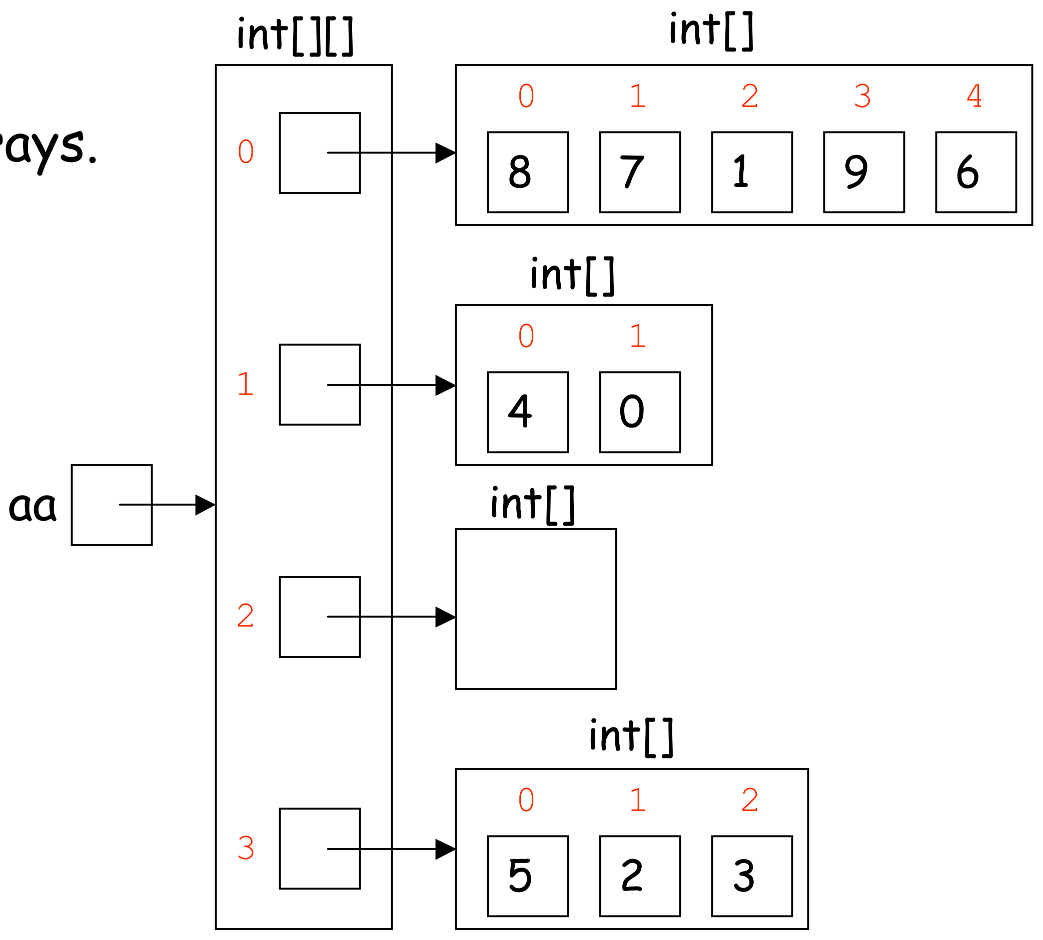
  `(a[j-1] > val)` to `(a[j-1].compare(val) > 0)`

- There are many other algorithms for sorting arrays:
  - You will study one of them (selection sort) this week in lab.
  - Several more sorting algorithms are presented in CS230.

Arrays of Arrays

Arrays can have elements of any type, including other arrays.

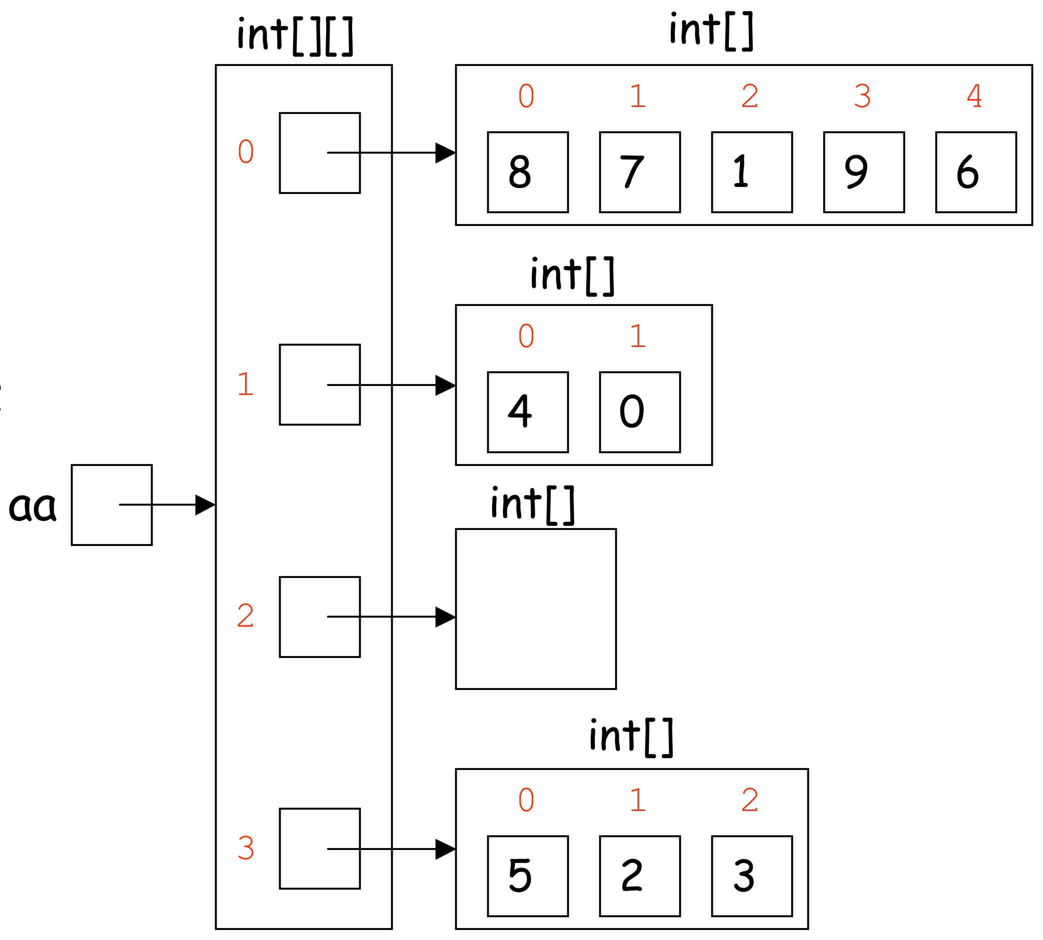
Here is an array of integer arrays.



Array Diagrams Illustrate Sharing

What do the following statements do to the array diagram?

```
aa[0][1] = aa[1][0];

aa[2] = aa[0];

aa[0][2] = aa[2][0];

aa[3][1] = aa[1][0] + aa[2][2];
```



Creating Arrays of Arrays

How can we create aa?

Way 1:

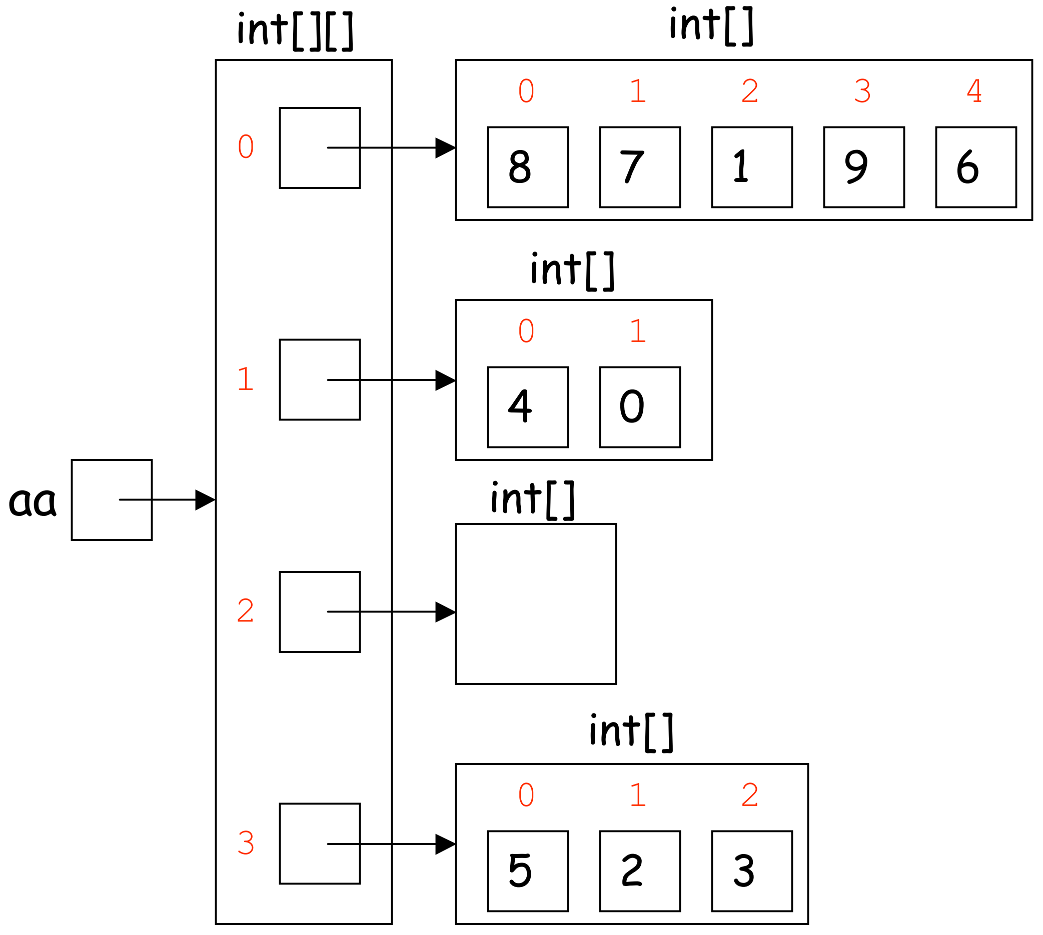
```
int [] [] aa = new int [] []
                  {{8,7,1,9,6},
                   {4,0},{},{5,2,3}};
```

Way 2:

```
int [] [] aa = new int [4] [];
aa[0] = new int [] {8,7,1,9,6};
aa[1] = new int [] {4,0};
aa[2] = new int [] {};
aa[3] = new int [] {5.2.3};
```

Way 3:

```
int [] [] aa = new int [4] [];
aa[0] = new int [5];
aa[0][0] = 8;
aa[0][1] = 7;
aa[0][2] = 1;
...
```



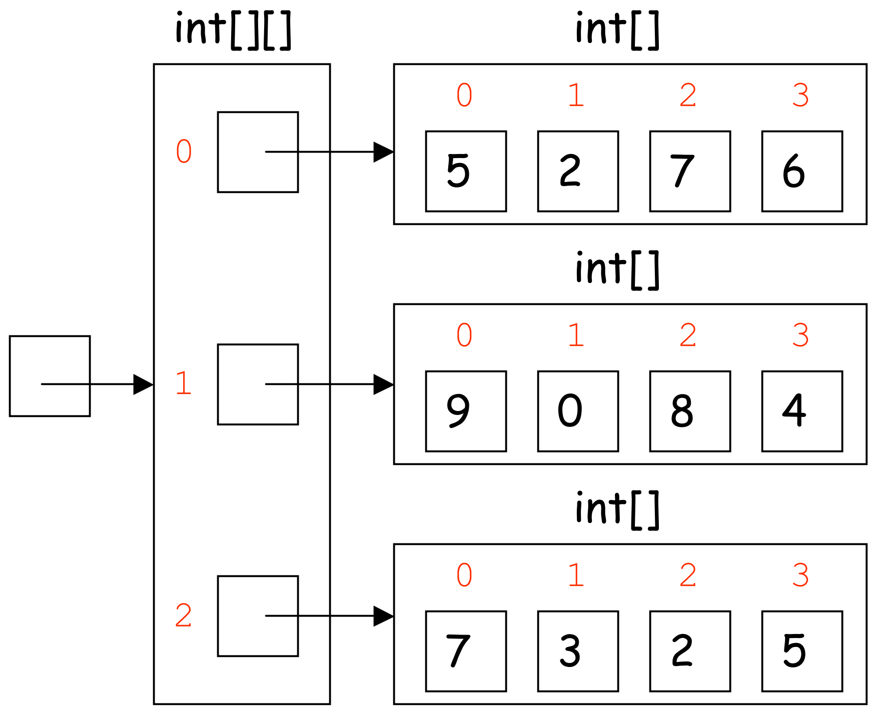
Summing Nested Arrays

```
// Sum all the integers in an array of array of ints
public static int sumArrayofArrays (int [] [] aai) {















}
```

Regular 2D Array of Arrays

A two-dimensional (2D) array is **regular** if each sub-array has the same length.

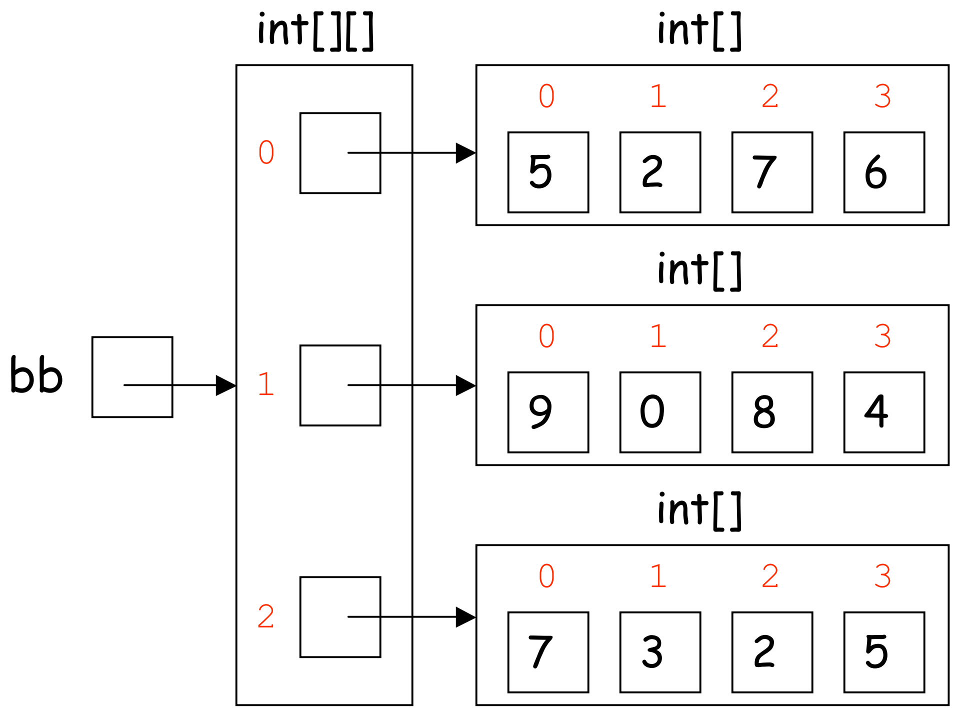
Certain operations make sense on regular 2D arrays that don't make sense on irregular ones -- e.g. summing the columns of the array.



```
sumColumns(bb);
// returns the 1D array {21, 5, 17, 15}
```

sumColumns()

```
public static int[] sumColumns (int[][] aai) {
   // Assume aai is a regular 2D array

}
```



int[][]
int[]
bb
0
1
2
3
5
2
7
6
9
0
8
4
7
3
2
5