CS100J   03 April 2007

Sorting: insertion- selection- quick- sort
Rectangular arrays and ragged arrays. Secs. 9.1 – 9.3

Do exercises on pp. 311-312 to get familiar with concepts and develop skill. Practice in DrJava! Test your methods!

Haikus (5-7-5) seen on Japanese computer monitors

Yesterday it worked.
Today it is not working.
Windows is like that.

A crash reduces
Your expensive computer
To a simple stone.

Three things are certain:
Death, taxes and lost data.
Guess which has occurred?

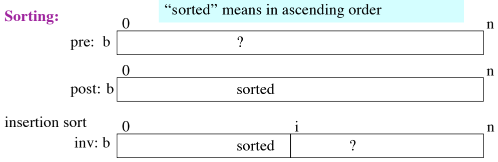
Serious error.
All shortcuts have disappeared.
Screen. Mind. Both are blank.

The Web site you seek
Cannot be located, but
Countless more exist.

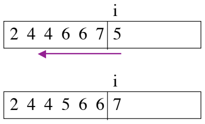
Chaos reigns within.
Reflect, repent, and reboot.
Order shall return.

Sorting:

"sorted" means in ascending order

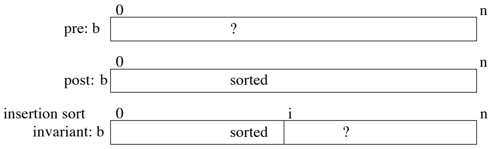


```
for (int i= 0;  i < n;  i= i+1) {
    Push b[i] down into its sorted
        position in b[0..i];
}
```

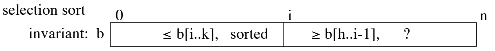


Iteration i makes up to i swaps. In worst case, number of swaps needed is 1 + 2 + 3 + … (n-1) = (n-1)*n / 2.

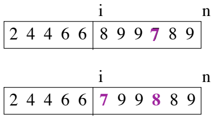
Called an "n-squared", or n^2, algorithm.



Add property to invariant: first segment contains smaller values.



```
for (int i= 0;  i < n;  i= i+1) {
    int j= index of min of b[i..n-1];
    Swap b[j] and b[i];
}
```



Also an "n-squared", or n^2, algorithm.

Quicksort

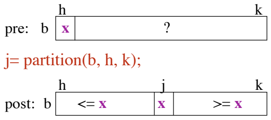
```
/** Sort b[h..k] */
public static void qsort(int[] b, int h, int k) {
    if (b[h..k] has fewer than 2 elements)
        return;

    int j= partition(b, h, k);
    // b[h..j–1] <= x < b[j+1..k]

    // Sort b[h..j–1]  and  b[j+1..k]
    qsort(b, h, j–1);
    qsort(b, j+1, k);

}
```

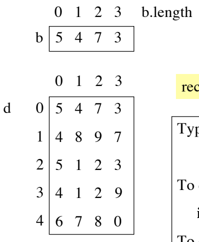
To sort array of size n. e.g. 2^{15}

Worst case: n^2 e.g. 2^{30}

Average case:
n log n.   e.g. 15 * 2^{15}

2^{15} = 32768





b.length

one-dimensional array

rectangular array: 5 rows and 4 columns

Type of d is **int**[][] ("**int** array array",
"an array of **int** arrays")

To declare variable d: number of rows

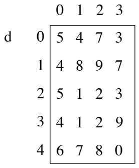
int d[][].

To create a new array and assign it to d:

d= **new int**[5][4];

To reference element at row r column c:

d[r][c]   number of cols



Type of d is **int**[][] ("**int** array array",
"an array of **int** arrays")

To declare variable d: number of rows

int d[][].

To create a new array and assign it to d:

d= **new int**[5][4];

To reference element at row r column c:

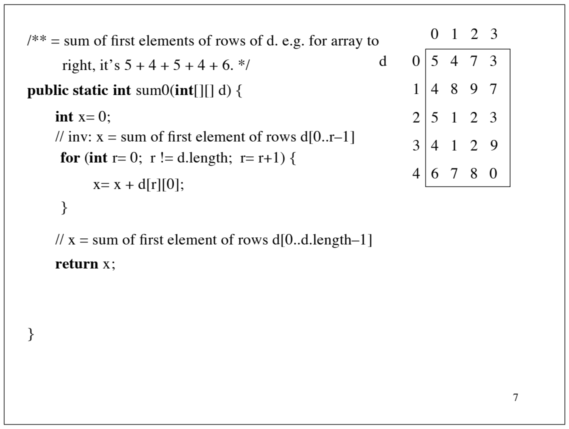
d[r][c]   number of cols

Number of rows: d.length
Number of columns in row r: d[r].length

"Length of one array in array of arrays"

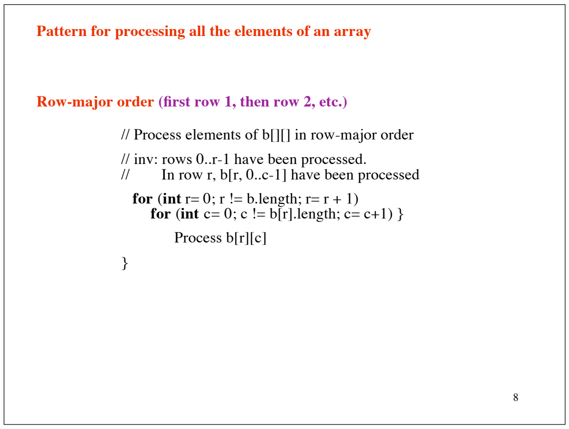
Using an array initializer:

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



```
/** = sum of first elements of rows of d. e.g. for array to
      right, it's 5 + 4 + 5 + 4 + 6. */
public static int sum0(int[][] d) {
    int x= 0;
    // inv: x = sum of first element of rows d[0..r–1]
     for (int r= 0;  r != d.length;  r= r+1) {
          x= x + d[r][0];
     }
    // x = sum of first element of rows d[0..d.length–1]
    return x;


}
```

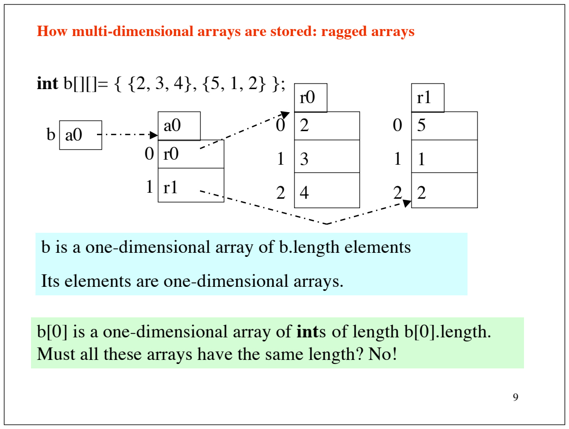
d	0	1	2	3
0	5	4	7	3
1	4	8	9	7
2	5	1	2	3
3	4	1	2	9
4	6	7	8	0



Pattern for processing all the elements of an array

Row-major order (first row 1, then row 2, etc.)

```
// Process elements of b[][] in row-major order
// inv: rows 0..r-1 have been processed.
//        In row r, b[r, 0..c-1] have been processed
  for (int r= 0; r != b.length; r= r + 1)
      for (int c= 0; c != b[r].length; c= c+1) }
           Process b[r][c]
}
```



How multi-dimensional arrays are stored: ragged arrays

int b[][]= { {2, 3, 4}, {5, 1, 2} };

b a0

a0: 0 r0, 1 r1

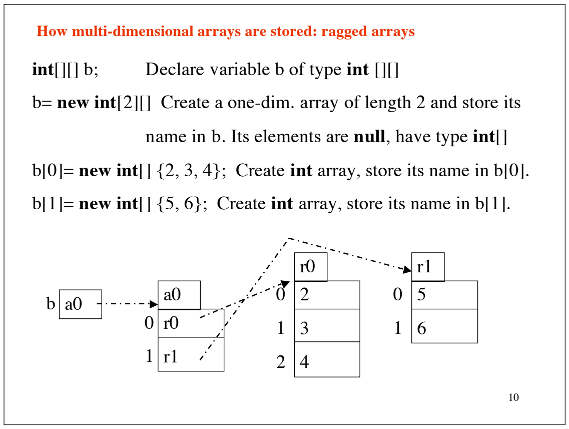
r0: 0 2, 1 3, 2 4

r1: 0 5, 1 1, 2 2

b is a one-dimensional array of b.length elements

Its elements are one-dimensional arrays.

b[0] is a one-dimensional array of **int**s of length b[0].length. Must all these arrays have the same length? No!



How multi-dimensional arrays are stored: ragged arrays

int[][] b;   Declare variable b of type **int** [][]

b= **new int**[2][] Create a one-dim. array of length 2 and store its name in b. Its elements are **null**, have type **int**[]

b[0]= **new int**[] {2, 3, 4}; Create **int** array, store its name in b[0].

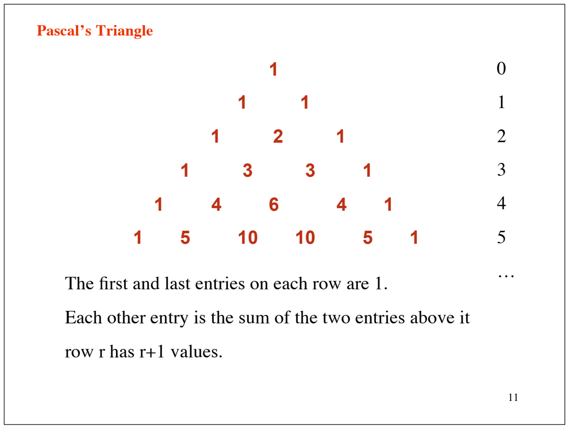
b[1]= **new int**[] {5, 6}; Create **int** array, store its name in b[1].

b a0

a0: 0 r0, 1 r1

r0: 0 2, 1 3, 2 4

r1: 0 5, 1 6



Pascal's Triangle

```
                1                 0
             1     1              1
          1     2     1           2
       1     3     3     1        3
    1     4     6     4     1     4
 1     5    10    10     5     1  5
                                  …
```

The first and last entries on each row are 1.

Each other entry is the sum of the two entries above it

row r has r+1 values.