2-ARRAYS

Topics

- Introduction to Arrays
- Array variations
- Arrays in Java
- Operations on Arrays
- Array as an Abstract Data Type
- Case Study: Subarrays
- OOP: Factory Methods
- 2 Dimensional Arrays
- Array Implementation in Java
  - OOP: Dividing Programs into Classes
- Ordered Arrays
  - Insertion
  - Deletion
  - Searching
- Ordered vs. Unordered Arrays
- ArrayList and Vector

Introduction to Arrays

- An array is an indexed sequence of components
  - Typically, the array occupies sequential storage locations
  - The length of the array is determined when the array is created, and cannot be changed
  - Each component of the array has a fixed, unique index
    - Indices range from a lower bound to an upper bound
  - Any component of the array can be inspected or updated by using its index
    - This is an efficient operation: O(1) = constant time

Array variations - I

- The array indices may be integers (C, Java) or other discrete data types (Pascal, Ada)

- The lower bound may be zero (C, Java), one (Fortran), or chosen by the programmer (Pascal, Ada)

- In most languages, arrays are homogeneous (all components must have the same type); in some (Lisp, Prolog) the components may be heterogeneous (of varying types)

Array variations - II

- In an object-oriented language, arrays may be objects (Java, Ruby) or not objects (C++)

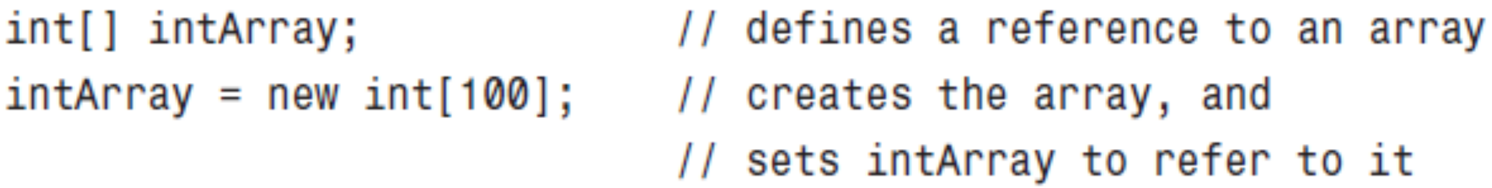
- Arrays may carry additional information about themselves, such as type and length (Java), or may consist *only* of their components (C, C++)
  - We will use the terms reflective and non-reflective, respectively, to refer to these two types of arrays
  - This is not standard terminology, but it is consistent with other uses of the terms

Arrays in Java - I

- Array indices are integers
  - Java's integral types are byte, char, short, int, and long
- An array of length n has bounds 0 and n-1
- Arrays are homogeneous
  - However, an array of an object type may contain objects of any subtype of that object
    - For example, an array of Animal may contain objects of type Cat and objects of type Dog
    - An array of Object may contain any type of object (but cannot contain primitives)

Arrays in Java - II

- Arrays are objects
  - Arrays are allocated by new, manipulated by reference, and garbage collected
  - However, the usual bracket notation intArray[i] is provided as syntactic sugar

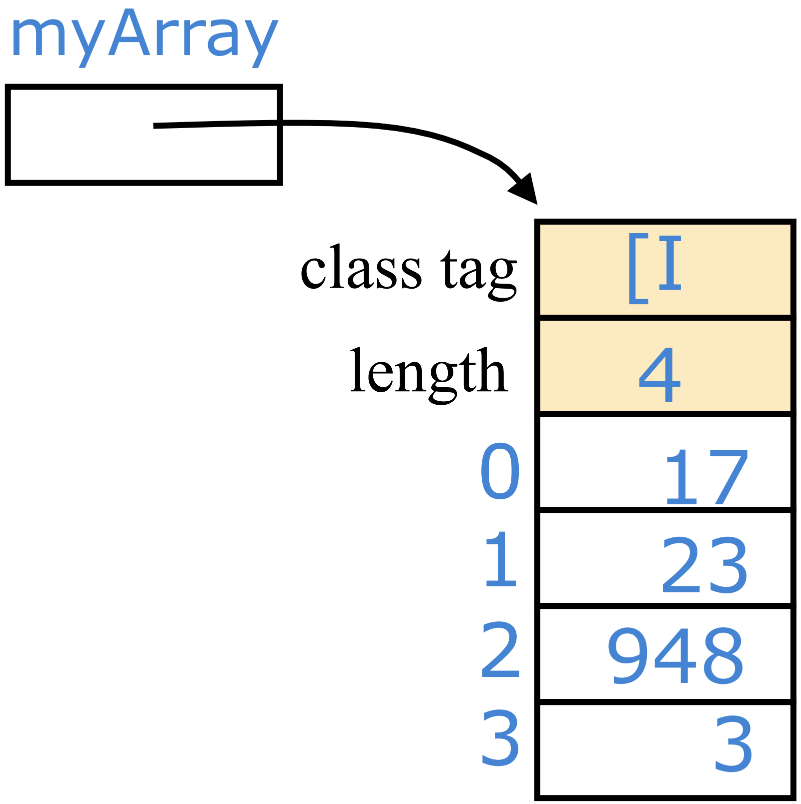


```
int[] intArray;              // defines a reference to an array
intArray = new int[100];     // creates the array, and
                             // sets intArray to refer to it
```

- Arrays are reflective
  - intArray.length is the length of array intArray
  - intArray.getClass() is the type of array intArray
    - An array of integers has type [I
    - An array of Strings has type [Ljava.lang.String;

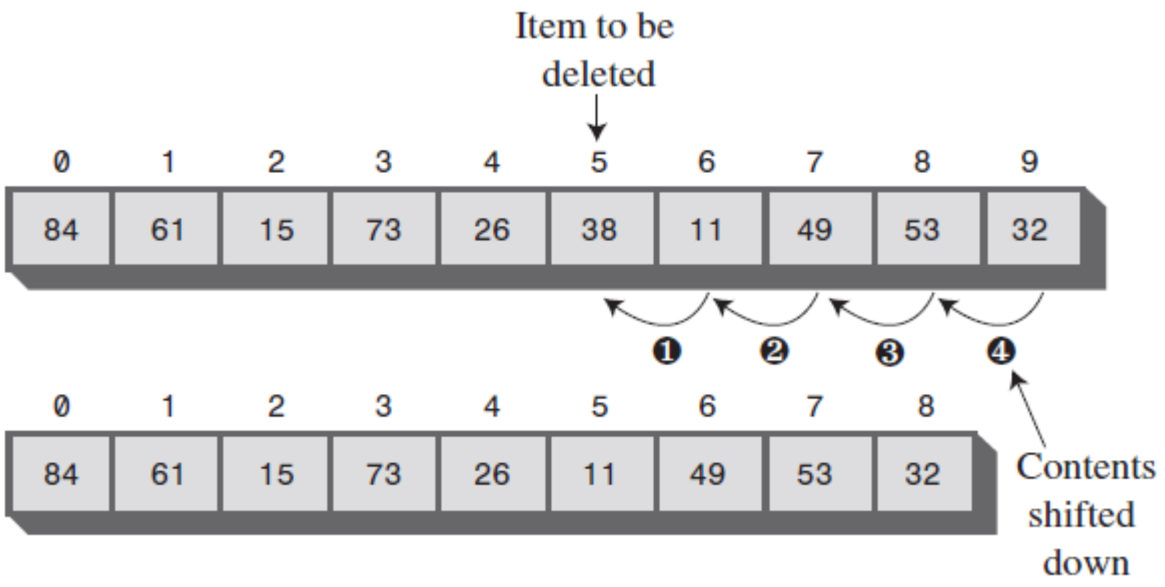
Arrays in Java - III

- Here's one way to visualize an array in Java:



Operations on Arrays – I

- Array Workshop applet
  - C:\> appletviewer Array.html
- Insertion
- Searching
- Deletion



Operations on Arrays – II

□ Duplicates issues: (Searching, Insertion and Deletion)

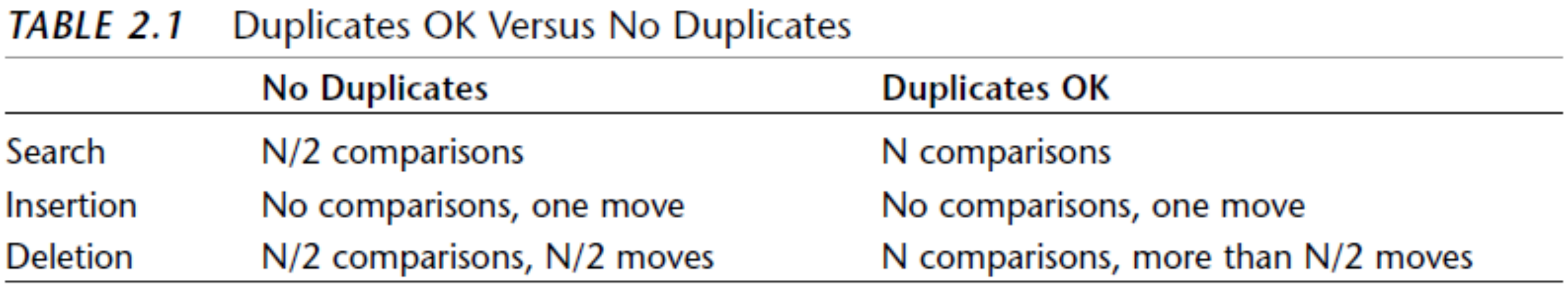


TABLE 2.1 Duplicates OK Versus No Duplicates

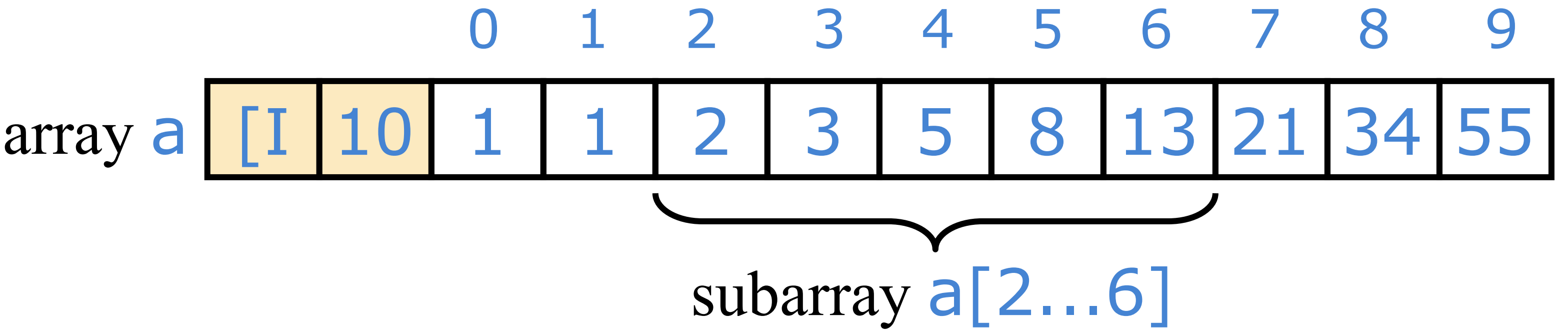
	No Duplicates	**Duplicates OK**
Search	N/2 comparisons	N comparisons
Insertion	No comparisons, one move	No comparisons, one move
Deletion	N/2 comparisons, N/2 moves	N comparisons, more than N/2 moves

Array as an Abstract Data Type (ADT)

- An Abstract Data Type (ADT) is:
  - a set of *values*
  - a set of *operations,* which can be applied uniformly to all these values
  - We abstract away implementation details.
- An array is an Abstract Data Type
  - The array type has a set of *values*
    - The values are all the possible arrays
  - The array type has a set of *operations* that can be applied uniformly to each of these values
    - Insert
    - Find
    - Delete
  - It's *abstract* because the implementation is hidden: all access is via the defined operations

Case Study: Subarrays

- A subarray is a consecutive portion of an array



- Java provides *no language support* for subarrays
- To use a subarray, *you* must keep track of (1) the array itself, (2) the lower bound, and (3) the upper bound
- Typically, these will be three parameters to a method that does something with the subarray

A Subarray class, - I

- Suppose you want to create a "live" subarray class, so that changes to the array affect the subarray, and vice versa
  - And suppose you want the subarray to use zero-based indexing, as usual
  - As noted earlier, to use a subarray, you must keep track of (1) the array itself, (2) the lower bound, and (3) the upper bound
- This suggests the following design:

```
class Subarray<V> {
    private V[ ] array; // a reference to the "real" array
    private int lowerBound, upperBound;
    // Constructor, some methods...
}
```

- Advantages:
  - There's just one object (the subarray) to pass around, rather than three values
  - You can use methods to handle the index transformations
- Disadvantages:
  - The subarray must hold Objects, not primitives
  - You lose the nice array syntax

A Subarray class, - II

□
```
public class Subarray<V> {
    private V[] array;
    private int lowerBound;
    private int upperBound;

    public Subarray(V[] array, int lowerBound, int upperBound) {
        this.array = array;
        this.lowerBound = lowerBound;
        this.upperBound = upperBound;
    }
    public V get(int index) {
        return array[lowerBound + index];
    }
    public void set(int index, V value) {
        array[lowerBound + index] = value;
    }
    public int length() {
        return upperBound - lowerBound + 1;
    }
}
```

Testing the Subarray class

```
public static void main(String[] args) {
    String[] array = new String[] {"zero", "one", "two", "three", "four" };
    Subarray<String> sub = new Subarray<String>(array, 1, 3);

    for (int i = 0; i < sub.length(); i++) {
        sub.set(i, i + "");
    }
    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i]);
    }
}
```

```
zero
0
1
2
four
```

Questions

- We never used upperBound; should we delete it?
- No, we should put tests in both set and get to possibly throw an exception
- Java has an ArrayIndexOutOfBoundsException; we should use that instead of creating a new kind of exception

- What if we create a subarray with illegal indices, for example, new Subarray<String>(array, 10, 5) ?
- Java has both an ArrayIndexOutOfBoundsException and a NegativeArraySizeException; should we use one or both of those?
- It would be *okay* to throw these exceptions, but that will happen *after* the constructor creates the object
  - It might be better to use a factory method

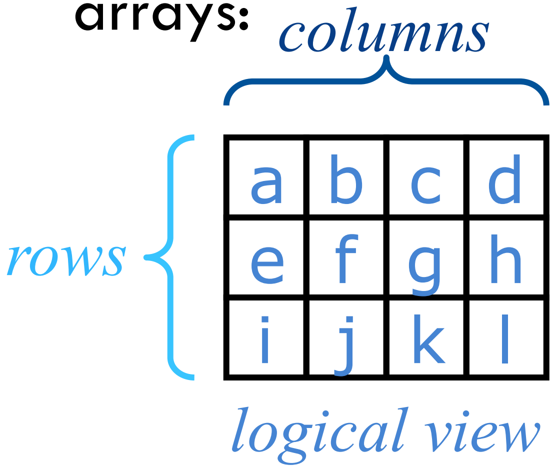
OOP: Factory methods

- A factory method is a method used in place of a constructor
  - All constructors for the object are made private
  - The factory method is static
  - The factory method uses the constructor after testing for possible errors
- Example:
  - `private Subarray(V[] array, int lowerBound, int upperBound) {...}`

  -
    ```
    public static <V> Subarray<V> newInstance(V[] array,
               int lowerBound, int upperBound) {
        // test if lowerBound >= 0, lowerBound <= upperBound,
        // and upperBound < array.length, and throw some exception
        // if any of these tests fail
        return new Subarray<V>(array, lowerBound, upperBound);
    }
    ```
- Note: The extra occurrence of the type parameter <V> in the factory method is because the method is static, and there is no instance object of the class. So, the type T will be inferred from the target type.
  - String [] array = new String[10];
  - Subarray<String> sub = Subarray.instance(array, 2, 6);

Two-dimensional arrays - I

- □ Some languages (Fortran, Pascal) support two-dimensional (2D) arrays:



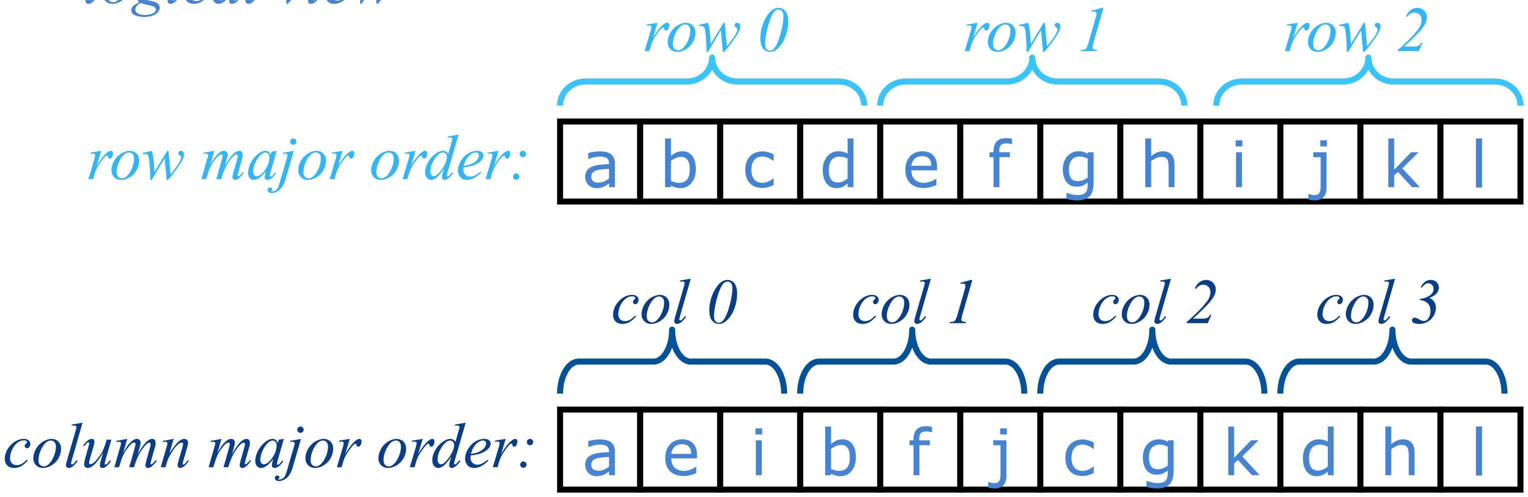
columns

rows

a	b	c	d
e	f	g	h
i	j	k	l

logical view

- □ A two-dimensional array may be stored in one-dimensional computer memory in either of two ways:



row 0 row 1 row 2

row major order: a b c d e f g h i j k l

col 0 col 1 col 2 col 3

column major order: a e i b f j c g k d h l

Two-dimensional arrays - II

□ In a 2D array, we generally consider the first index to be the row, and the second to be the column: a [*row*] [*col*]

columns

		0	*1*	*2*	*3*	*4*
rows	*0*	*0,0*	*0,1*	*0,2*	*0,3*	*0,4*
	1	*1,0*	*1,1*	*1,2*	*1,3*	*1,4*
	2	*2,0*	*2,1*	*2,2*	*2,3*	*2,4*
	3	*3,0*	*3,1*	*3,2*	*3,3*	*3,4*

□ In most languages we don't need to know the implementation--we work with this *abstraction*

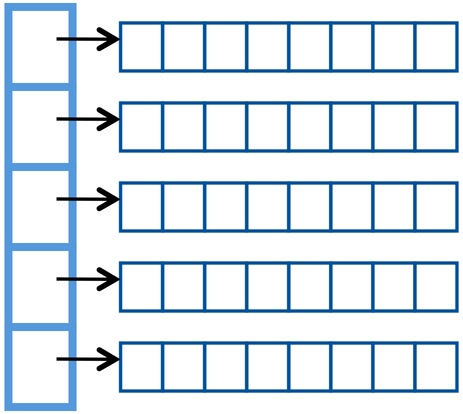
□ In C and C++, we *do* need to know the implementation

2D arrays in Java

- Java doesn't have "real" 2D arrays, but array elements can themselves be arrays:
  - int x[][] denotes an array x of *array* components, each of which is an array of *integer* components
- We can define the array on the right like this:

  ```
  x = new int[5][8];
  ```

  and treat it as a regular 2D array



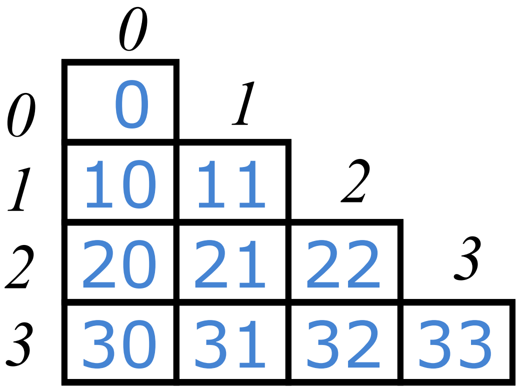
- This is an *array of 5 arrays*
  - Each subarray is an array of 8 ints
- However, we can do fancier things than this with arrays in Java

Ragged arrays

```
int ragged[][] = new int[4][];

for (int i = 0; i < 4; i++) {
    ragged[i] = new int[i + 1];
}

for (int i = 0; i < 4; i++) {
    for (int j = 0; j < ragged[i].length; j++) {
        ragged[i][j] = 10 * i + j;
    }
}
```



	0	1	2	3
0	0			
1	10	11		
2	20	21	22	
3	30	31	32	33

Java Code Implementation - Array

- Implementing an array and operations like searching and deletion.
- (see array.java listing 2.1, page 41)
- Program is not well structured !!

OOP: Dividing Program into Classes – I

- Two classes:
  - Data storage structure itself, and
  - The program that uses this data structure

OOP: Dividing Program into Classes – II

- One example is lowarray.java in Lisiting 2.2 page 44 (LowArray and LowArrayApp classes)
- Advantages:
  - Array is hidden from the outside world inside the class; it' s private, only LowArray class methods can access it.
  - Three LowArray methods: setElem(), getElem(), and a constructor, which creates an empty array of a specified size.
- Disadvantages:
  - Methods setElem() and getElem() operate on a low conceptual level, performing exactly the same tasks as the [] operator,
  - the main() method in the LowArrayApp class, ends up having to carry out low-level operations

OOP: Dividing Program into Classes – III

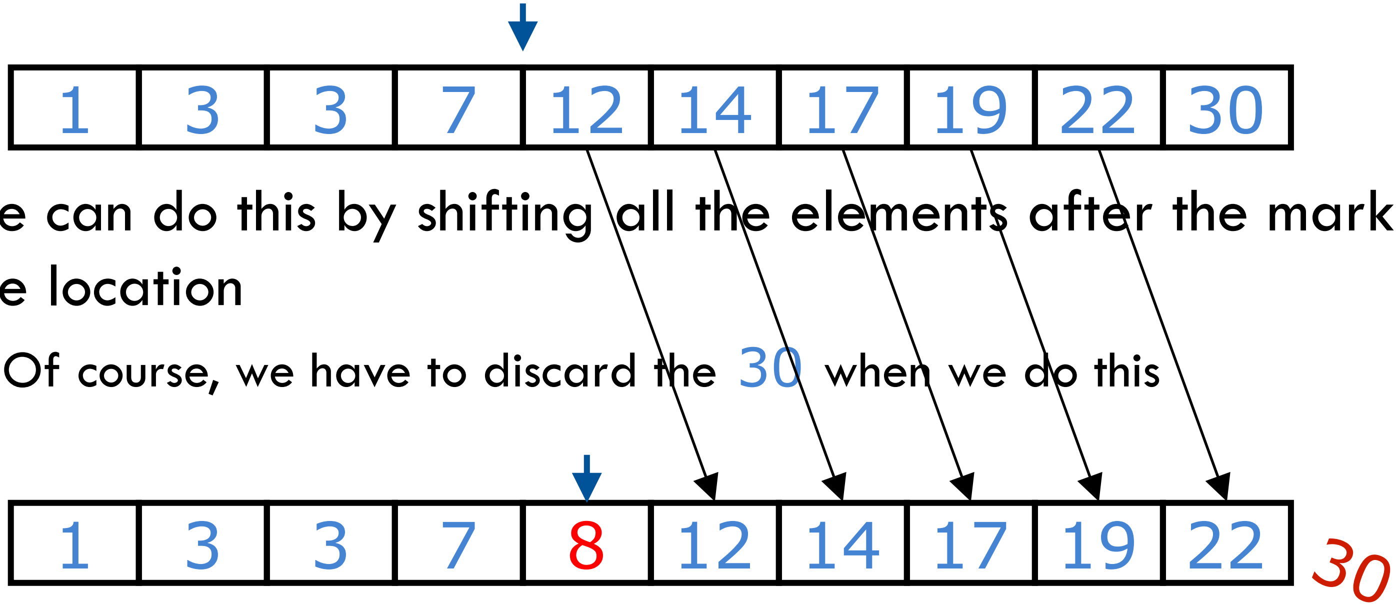
- Improved example is higharray.java in Lisiting 2.2 page 44 (HighArray and HighArrayApp classes)
- Advantages:
  - The setElem() and getElem() methods are replaced by insert(), find(), and delete(). Don't require an index number as an argument.
  - The class user (the HighArrayApp class) no longer needs to think about index numbers.

Ordered Arrays

- Ordered Workshop applet
  - (C:\>appletviewer ordered.html)
- Insertion (in correct location. Why? Speed up searching)
- Deletion
- Searching
  - Linear search
  - Binary Search

Inserting an element into an array

- □ Suppose we want to insert the value 8 into this sorted array (while keeping the array sorted)



1	3	3	7	12	14	17	19	22	30

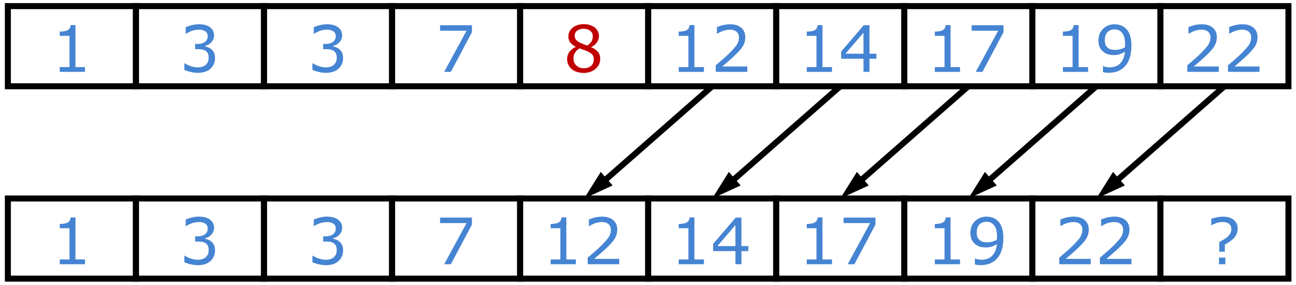
- □ We can do this by shifting all the elements after the mark right by one location
  - Of course, we have to discard the 30 when we do this

1	3	3	7	8	12	14	17	19	22

- Moving all those elements makes this a *slow* operation (linear in the size of the array)

Deleting an element from an array

- Deleting an element is similar--again, we have to move all the elements after it.



1	3	3	7	8	12	14	17	19	22
1	3	3	7	12	14	17	19	22	?

- Deletion is a slow operation; we don’t want to do it very often
- Deletion leaves a “vacant” location at the end
  - How do we mark it vacant?
    - Every bit pattern represents a valid integer
    - We must keep a count of how many valid elements are in the array

Ordered Arrays – Linear Search

- Linear Search
  - Starts sequentially from the beginning of array.
  - Either finds the element or quits when a larger value is found. Why?

Ordered Arrays – Binary Search – (1)

- Binary Search (ordArray.java, Listing 2.4, p. 59)
  - Like "Guess a number" game: either smaller, larger or equal (select 33, say)
  - Needs fewer steps that Linear Search. (7 guesses instead of 33)

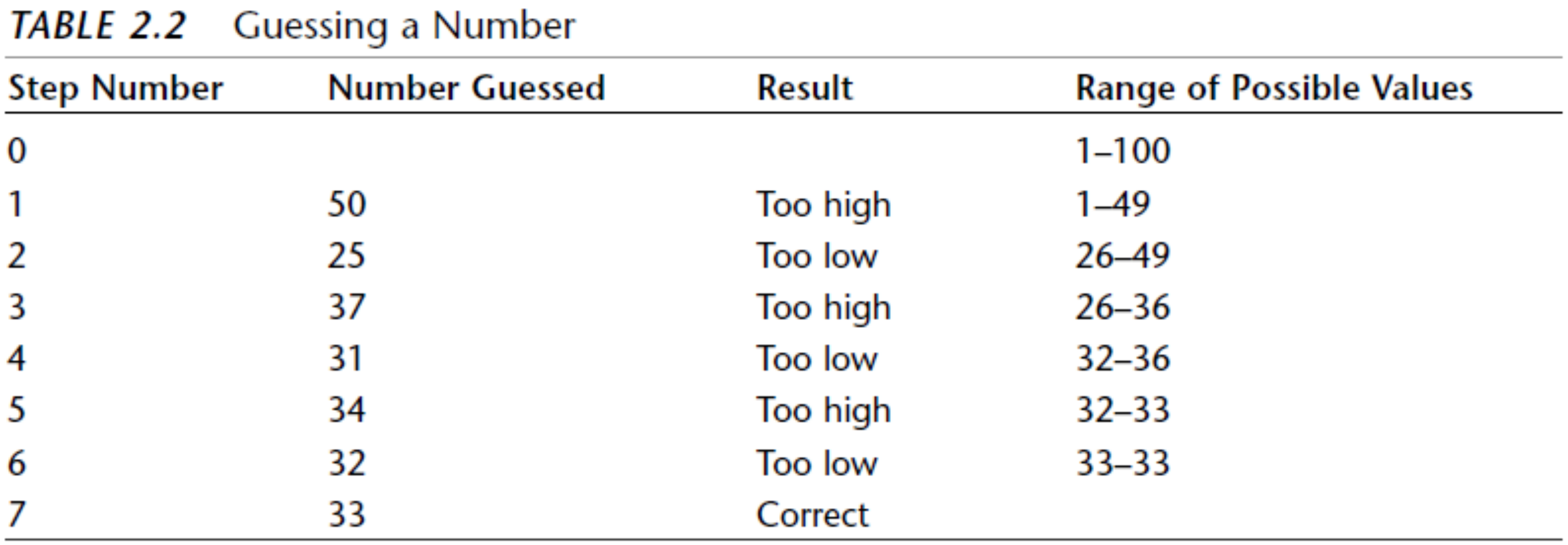


TABLE 2.2 Guessing a Number

Step Number	Number Guessed	Result	Range of Possible Values
0			1–100
1	50	Too high	1–49
2	25	Too low	26–49
3	37	Too high	26–36
4	31	Too low	32–36
5	34	Too high	32–33
6	32	Too low	33–33
7	33	Correct	

Ordered Arrays – Binary Search (2)

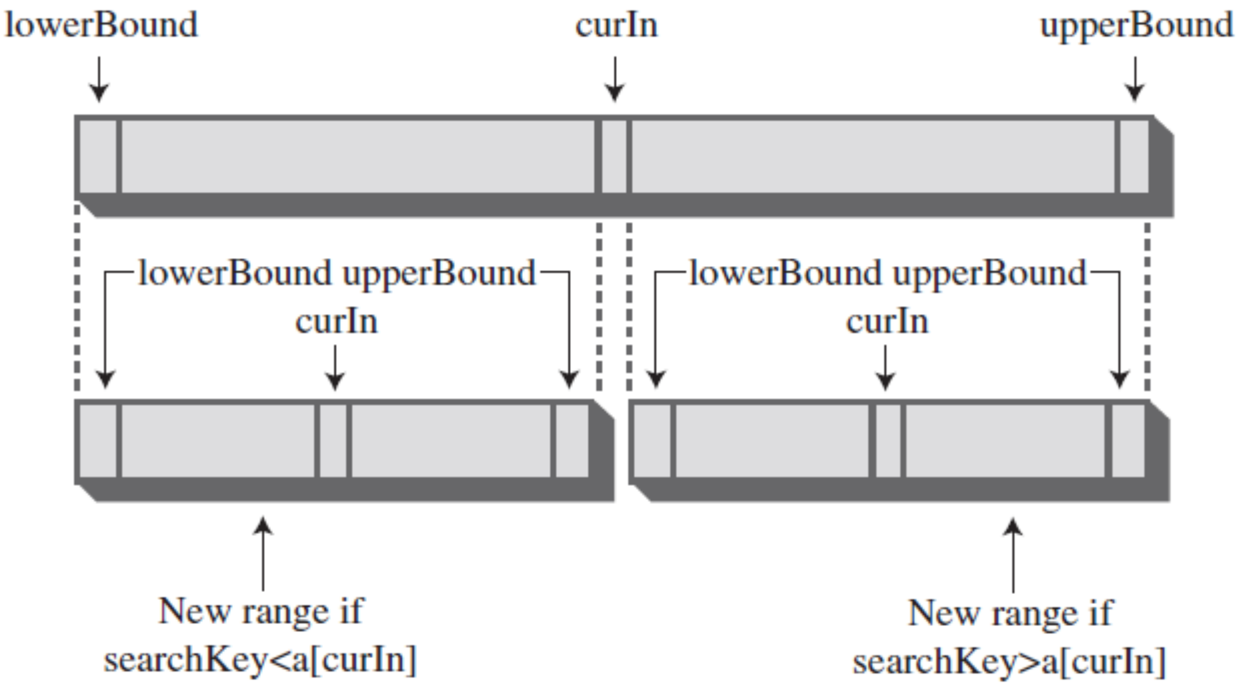


FIGURE 2.8 Dividing the range in a binary search.

Ordered Arrays – Binary Search (3)

```
public int find(long searchKey)
{
   int lowerBound = 0;
   int upperBound = nElems-1;
   int curIn;

   while(true)
   {
   curIn =
   (lowerBound + upperBound ) / 2;

      if(a[curIn]==searchKey)
         return curIn; // found it
```

```
else if(lowerBound > upperBound)
   return nElems; //can't find it
else // divide range
{
   if(a[curIn] < searchKey)
      // it's in upper half
      lowerBound = curIn + 1;
   else
      // it's in lower half
      upperBound = curIn - 1;
}  // end else divide range
}  // end while
}  // end find()
```

Ordered vs. Unorderd Arrays

- Searching in ordered arrays is much faster than in unordered arrays.
- Insertion in ordered arrays is slower than in unordered arrays, as it needs to move elements to give space for the inserted element.
- Deletion is slow in both ordered and unordered arrays.

- Ordered arrays are useful in situations in which "Searching" is frequent, such as database of company employees. Hiring and laying off of employees is relatively infrequent.
- However, a retail store inventory, is not a good candidate for Ordered arrays, due to frequent insertions and deletions.

ArrayLists and Vectors

- An ArrayList is a type of List (a sequence of values) that can be used like an array, but lacks the special array syntax
  - Instead of: a[i] = a[j];
  - You would say: a.set(i, a.get(j));
- The name reveals the implementation: it is a list implemented (behind the scenes) with an array

- The advantage of an ArrayList is that it expands as elements are added
- The disadvantage of an ArrayList is that it lacks the special [ ] syntax

- Vector is an older class, but very similar to ArrayList

Conclusions

- Arrays are not identical in all languages

- Arrays have the following advantages:
  - Accessing an element by its index is very fast (constant time)

- Arrays have the following disadvantages:
  - All elements must be of the same type
  - The array size is fixed and can never be changed
  - Insertion into arrays and deletion from arrays is very slow

The End