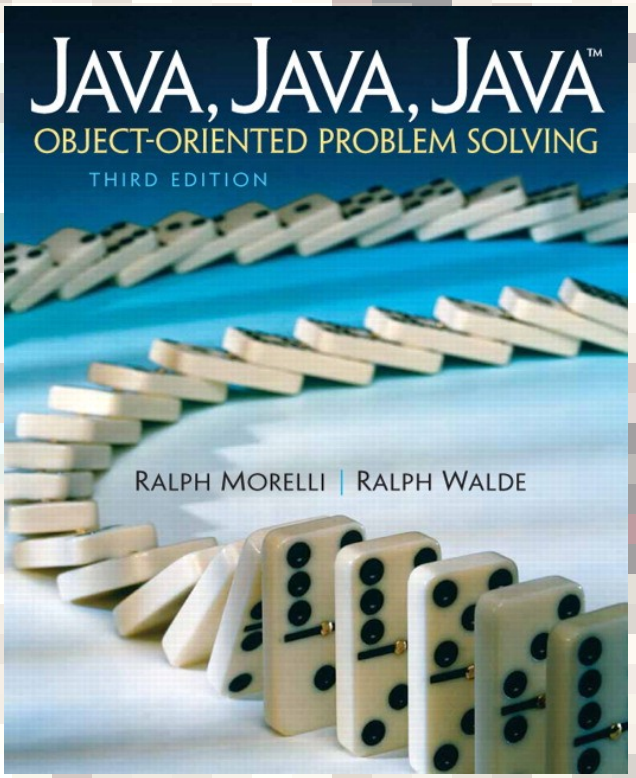


presentation slides for

JAVA, JAVA, JAVA

Object-Oriented Problem Solving

Third Edition

Ralph Morelli | Ralph Walde
Trinity College
Hartford, CT

published by Prentice Hall

Java, Java, Java

Object Oriented Problem Solving

Lecture 06: Arrays Review - Linear and Binary Searching Algorithms

Objectives

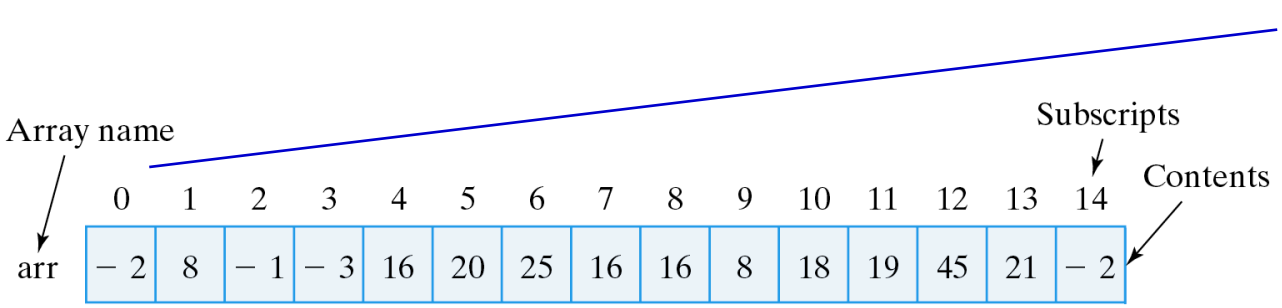
- Revise of array data structures.
- Be able to solve problems that require collections of data.
- Be familiar with sequential and binary search algorithms.

Outline

- Revised of 1D array and 2D arrays
- Two-Dimensional and Multidimensional Arrays
- Array Searching Algorithms: Linear and Binary
- From the Java Library: Vector

One-Dimensional Arrays

- An *array* is a named collection of *contiguous* storage locations holding data of the *same type*.
- For an *n*-element array named *arr*, the elements are named *arr[0], arr[1], arr[2], ...,arr[n-1]*.
- The following array contains 15 int elements.

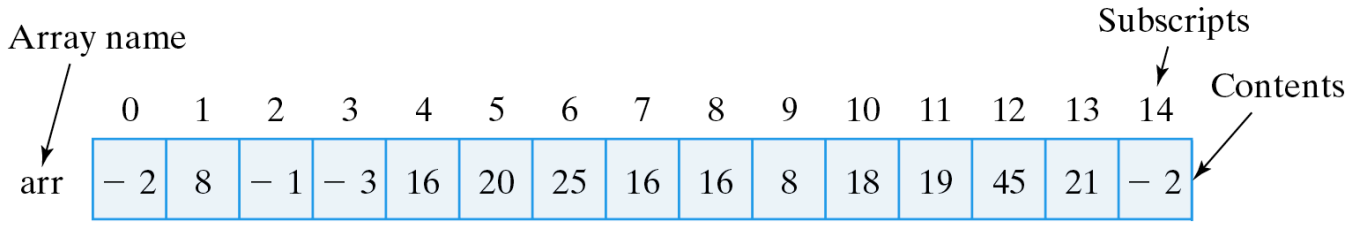


Arrays are *zero indexed*.

- Array syntax: *arrayname* [*subscript*]

  where *arrayname* is the array name and *subscript* is an integer giving the element's relative position.

Referring to Array Elements



- Valid References: Suppose *j* is 5 and *k* is 7.

```
arr[4]          // Refers to 16
arr[j]          // Is arr[5] which refers to 20
arr[j + k]      //  Is arr[5+7] which is arr[12] which refers to 45
arr[k % j]      //  Is arr[7%5] which is arr[2] which refers to -1
```

- Invalid References:

```
arr[5.0]        // 5.0 is a float and can't be an array subscript
arr['5']        // '5' in Unicode would be out of bounds, arr[53]
arr["5"]        // "5" is a string not an integer
arr[-1]         // Arrays cannot have negative subscripts
arr[15]         // The last element of arr has subscript 14
arr[j*k]        // Since j*k equals 35
```

Are Arrays Objects?

- Arrays are (mostly) treated as objects:
  - Instantiated with the new operator.
  - Have instance variables (e.g., length).
  - Array variables are *reference variables*.
  - As a parameter, a reference to the array is passed rather than copies of the array's elements.
- But…
  - Arrays don't fit into the Object hierarchy.
  - Arrays don't inherit properties from Object.

Some Array Terminology

- An *empty array* contains zero variables.
- The variables are called *components*.
- The *length* of the array is the number of components it has.
- Each component of an array has the same *component type*.
- A *one-dimensional array* has components that are called the array's elements. Their type is the array's *element type*.
- An array's elements may be of any type, including primitive and reference types.

Declaring and Creating an Array

- Creating a one-dimensional array: Indicate both the array's *element type* and its *length*.
- Declare the array's name and create the array itself.

```
int arr[];              // Declare a name for the array
arr = new int[15];      // Create the array itself
```

- Combine two steps into one:

```
int arr[] = new int[15];
```

The array's name is *arr*.

The array contains 15 int variables.

- 15 variables: arr[0], arr[1], .., arr[14] (*zero indexed*)

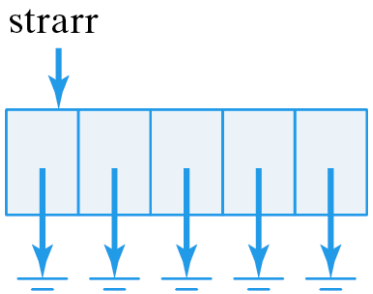
Creating an Array of Strings

(a)


```
String strarr[]; // Null array variable
```

Declare array variable.

(b)

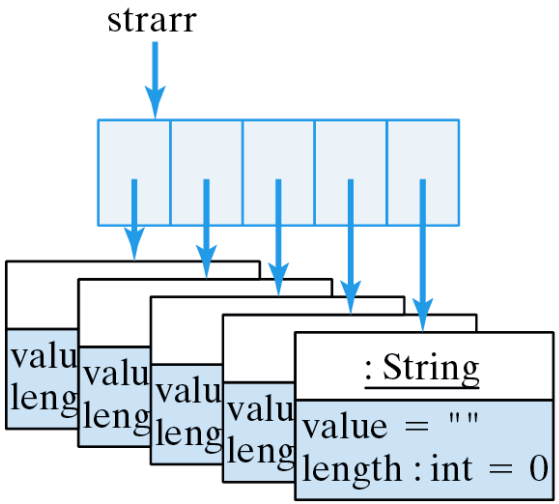



```
// Creates an array of null String
// references.

strarr = new String [5];
```

Instantiate the array.

(c)



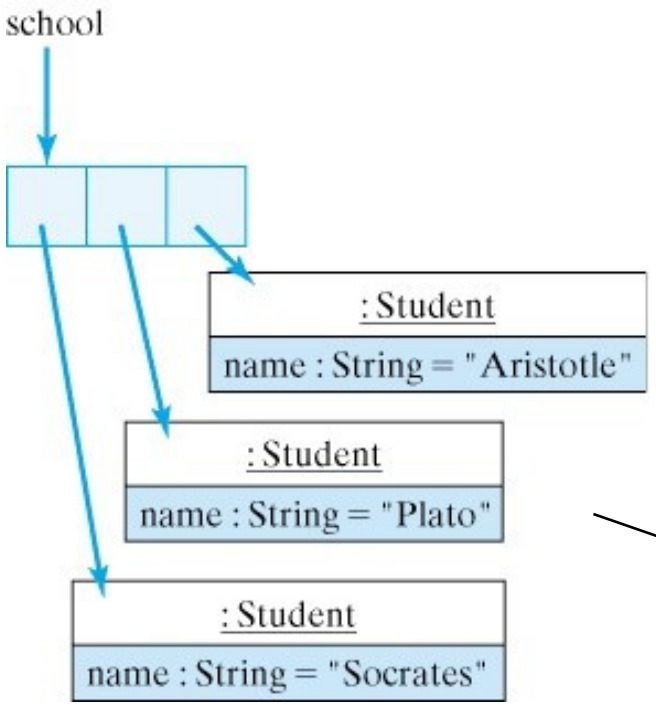

```
// Creates 5 Strings and
/  assigns them to the array

for (int k = 0; k < strarr.length; k++ )
    strarr[k] = new String();
```

Store 5 Strings in it.

Creating an Array of Students

```
Student arrStudents[] = new Student[3];     // Create an array of 3 Students
arrStudent[0] = new Student("Socrates");    // Create the first Student
arrStudent[1] = new Student("Plato");       // Create the second Student
arrStudent[2] = new Student("Aristotle");   // Create the third Student
```



- **Debugging Tip:** Creating a new array does not also create the objects that are stored in the array. They must be instantiated separately.

There are four objects here. One array and 3 Students.

Initializing Arrays

- Array elements are initialized to default values:
  - Integer and real types are initialized to 0.
  - Reference types (objects) are initialized to null.
- Arrays can be assigned initial values when they are created:

```
int arr[] = { -2,8,-1,-3,16,20,25,16,16,8,18,19,45,21,-2 } ;

String strings[] = { "hello", "world", "goodbye", "love" } ;
```

- **Java Language Rule:** When an array initialization expression is used, don't use the keyword new to create the array.

Assigning and Using Array Values

- Subscripted array variables are used like other variables:

```
arr[0] = 5;
arr[5] = 10;
arr[2] = 3;
strings[0] = "who";
strings[1] = "what";
strings[2] = strings[3] = "where";
```

- A loop to assign the first 15 squares, 1, 4, 9 …, to the array arr:

```
for (int k = 0; k < arr.length; k++)
        arr[k] = (k+1) * (k+1);
```

- A loop to print the values of arr:

```
for (int k = 0; k < arr.length; k++)
        System.out.println(arr[k]);
```

Note: *length* is an instance variable, not a method.

Example: Print an Array

- Print an array of int and an array of double:

```
public class PrintArrays  {
   static final int ARRSIZE = 10;                  // The array's size

   static int intArr[] = new int[ARRSIZE];         // Create the int array
   static double realArr[] = { 1.1, 2.2, 3.3, 4.4,
         5.5, 6.6, 7.7, 8.8, 9.9, 10.10 };         // And a double array

   public static void main(String args[]) {
      System.out.println("Ints \t Reals");
      for (int k = 0; k < intArr.length; k++)
          System.out.println( intArr[k] + " \t " + realArr[k]);
    } // main()
} // PrintArrays
```

These must be static ...

… in order to refer to them in static main()

Uninitialized int array has default values of 0.

Program Output

Ints	Reals
0	1.1
0	2.2
0	3.3
0	4.4
0	5.5
0	6.6
0	7.7
0	8.8
0	9.9
0	10.10

Generating Random Numbers (cont)

- An expression of the form

  (int)(Math.random() * N)

  will generate random integer values in the range 0 to *N-1*.
- *N* is called the *scaling factor*.
- To generate values in the range 0 to 5, use:

  (int)(Math.random() * 6);
- To simulate a die roll we must *shift* the values into the range 1 to 6:

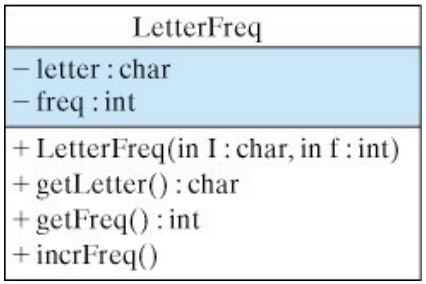
  int die = 1 + (int)(Math.random() * 6);

Example: Counting Letter Frequencies

- Design a class that can be used to store the frequencies of letters of the alphabet.

```
public class LetterFreq {
    private char letter;     //A character being counted
    private int freq;        //The frequency of letter

    public LetterFreq(char ch, int fre) {
        letter = ch;
        freq = fre;
    }
    public char getLetter() {
        return  letter;
    }
    public int getFreq() {
        return  freq;
    }
    public void incrFreq() {
        freq++;
    }
} //LetterFreq
```



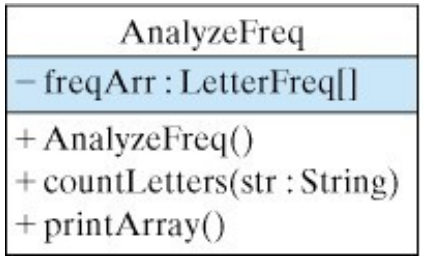
A Class to Count Frequencies

- A class that counts letters in a document.

```
public class AnalyzeFreq {
  private LetterFreq[] freqArr; // An array of frequencies

  public AnalyzeFreq() {
    freqArr = new LetterFreq[26];
    for (int k = 0; k < 26; k++) {
      freqArr[k] = new LetterFreq((char)('A' + k), 0);
    } //for
  }
  public void countLetters(String str) {
    char let; //For use in the loop.
    str = str.toUpperCase();
    for (int k = 0; k < str.length(); k++) {
      let = str.charAt(k);
      if ((let >= 'A') && (let <= 'Z')) {
        freqArr[let - 'A'].incrFreq();
      } // if
    } // for
  } // countLetters()

  public void printArray() {
    for (int k = 0; k < 26; k++) {
      System.out.print("letter: " + freqArr[k].getLetter());
      System.out.println(" freq: " + freqArr[k].getFreq());
    } //for
  } // printArray()
} //AnalyzeFreq
```



Note how it uses an array of LetterFreq objects to store letters and their frequencies.

Two-Dimensional Arrays

- ***Two-dimensional array***: an array whose ***components*** are themselves 1D arrays.
- Example: Compiling daily rainfall data. A ***one-dimensional*** array makes it hard to calculate average monthly rainfall:

```
double rainfall[] = new double[365];
```

- A ***two-dimensional array*** is an array of arrays. The first is the 12 months, indexed from 0 to 11. Each month array is an array of 31 days, indexed from 0 to 30.

```
double rainfall[][] = new double[12][31];
```

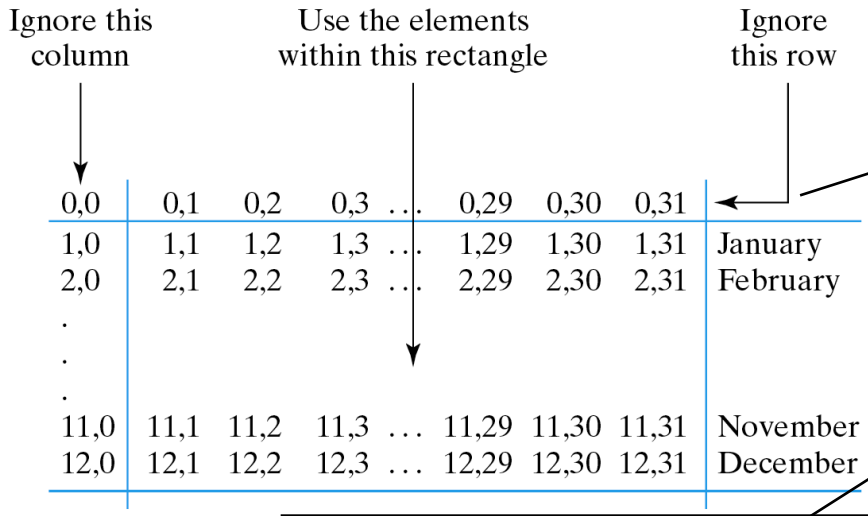
Month index

Day index

A More Appropriate 2-D Representation

- What is rainfall[0][4] ? Avoid zero indexing by creating an extra row and column and ignoring the 0 indexes.

```
double rainfall[][] = new double[13][32];
```



Don't use the 0 indexes.

January 5 is now at rainfall[1][5]

```
rainfall[1][5] = 1.15;          // Rainfall for January 5
System.out.println(rainfall[4][1]); // April 1st
rainfall[13][32] = 0.15 ;    // No such element
rainfall[11][32] = 1.3;      // No such column
rainfall[13][30] = 0.74;     // No such row
```

Calculate Average Daily Rainfall

```
/**
  * Computes average daily rainfall
  * @param rain is a 2D-array of rainfalls
  * @return The sum of rain[x][y] / 356
  * Pre:  rain is non null
  * Post: The sum of rain / 365 is calculated
  * Note that the loops are unit indexed
  */
public double avgDailyRain(double rain[][]) {
     double total = 0;
     for (int month = 1; month < rain.length; month++)
         for (int day = 1; day < rain[month].length; day++)
             total += rain[month][day];
     return total/365;
} // avgDailyRain()
```

A 2-D array parameter

Nested for loops iterate 12 x 31 times

Method call uses the array's name.

```
System.out.println("Daily Avg: " + avgRainForMonth(rainfall));
```

Calculate Average Rain for a Month (cont)

- Pass just part of a 2-D array -- e.g., a month.

```
/**
  * Computes average rainfall for a given month
  * @param monthRain is a 1D-array of rainfalls
  * @param nDays is the number of days in monthRain
  * @return The sum of monthRain / nDays
  * Pre:  1 <= nDays <= 31
  * Post: The sum of monthRain / nDays is calculated
  */
public double avgRainForMonth(double monthRain[], int nDays) {
     double total = 0;
     for (int day = 1; day < monthRain.length; day++)
         total = total + monthRain[day];
     return total/nDays;
} // avgRainForMonth()
```

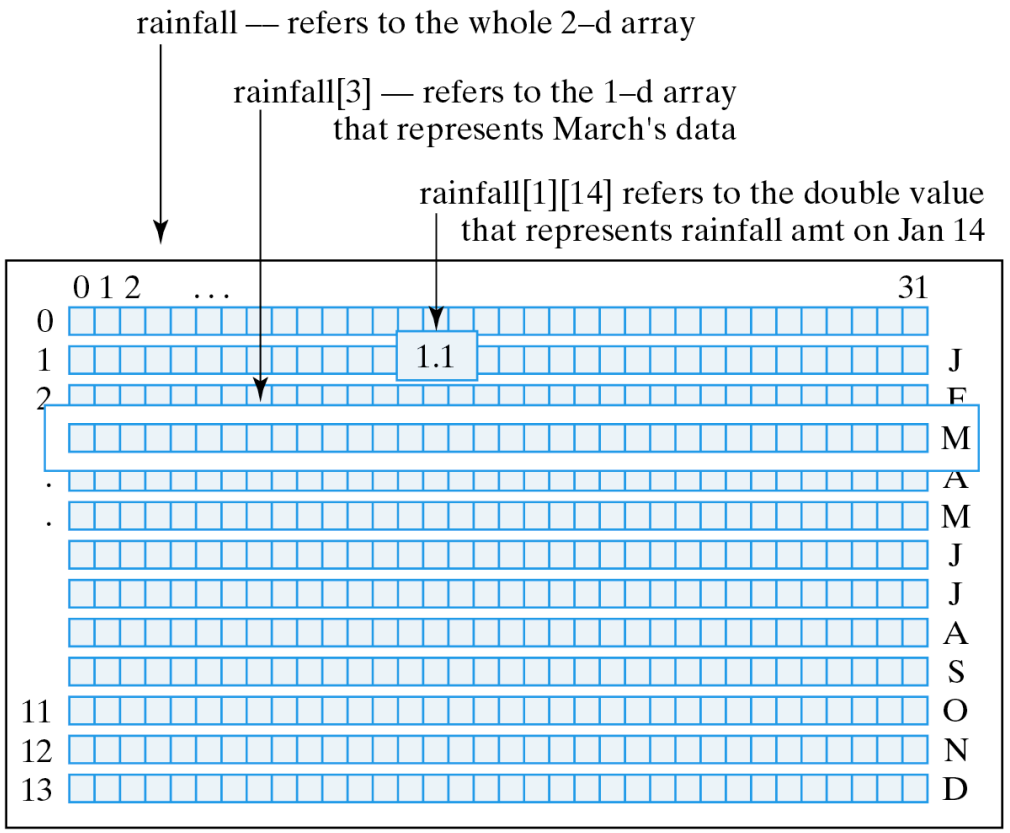
Pass the *array* for the given month.

We're passing a reference to a 1-D array.

```
System.out.println("March Avg: " + avgRainForMonth(rainfall[3], 31));
```

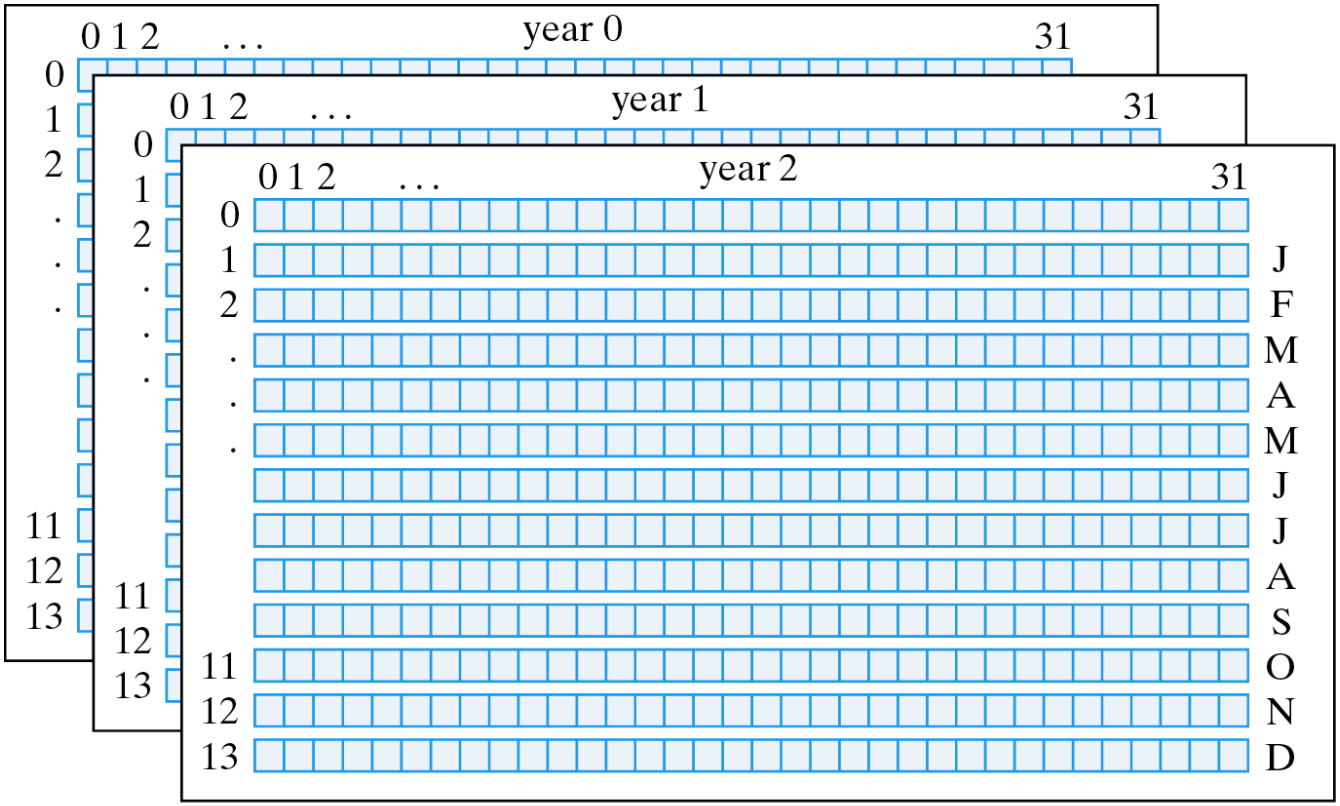
Array Arguments and Parameters

- The argument in a method call must match the data type in the method definition. This applies to all parameters, including array parameters.



Multidimensional Arrays

- A 3-dimensional array can be used to record rainfall over a ten year period.



A 3-D Rainfall Array

- Declaring a 3-D Array:

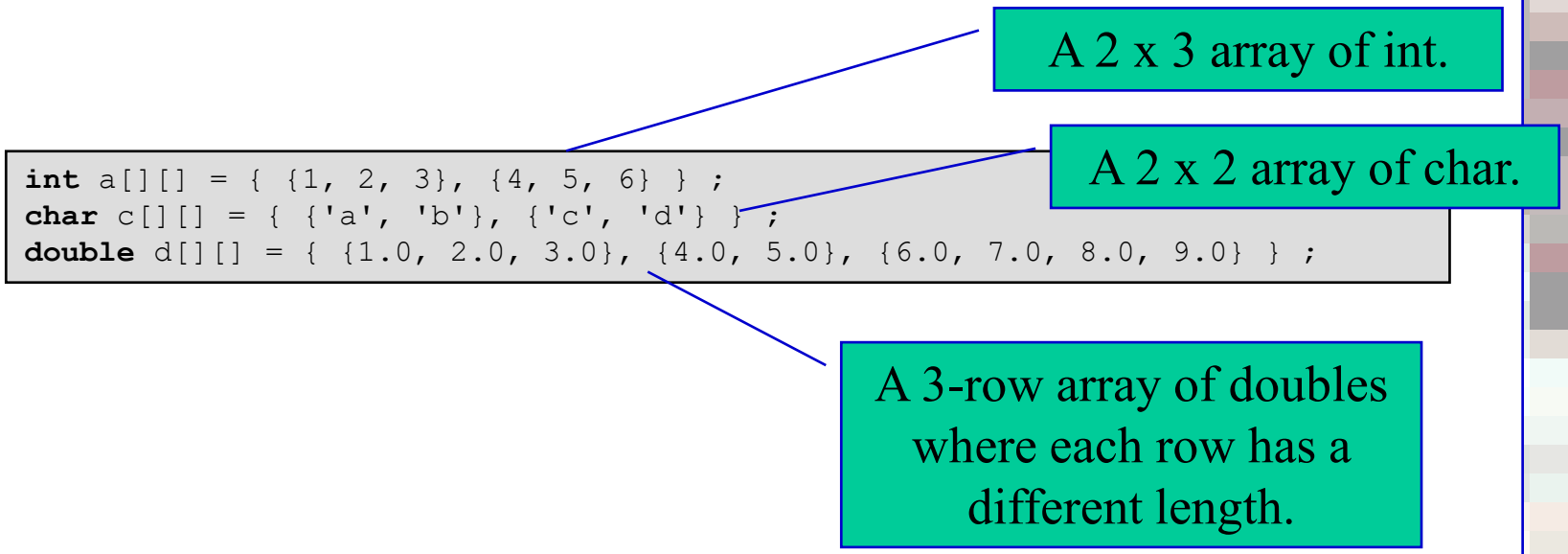
```
final int NYEARS = 10;
final int NMONTHS = 13;
final int NDAYS = 32;
double rainfail[][][] = new double[NYEARS][NMONTHS][NDAYS];
```

- Initializing a 3-D Array:

```
for (int year = 0; year < rainfall.length; year++)
      for (int month = 0; month < rainfall[year].length; month++)
          for (int day = 0; day < rainfall[year][month].length; day++)
              rainfall[year][month][day] = 0.0;
```

Multidimensional Array Initializers

- For small arrays, an initializer expression can be used to assign initial values:

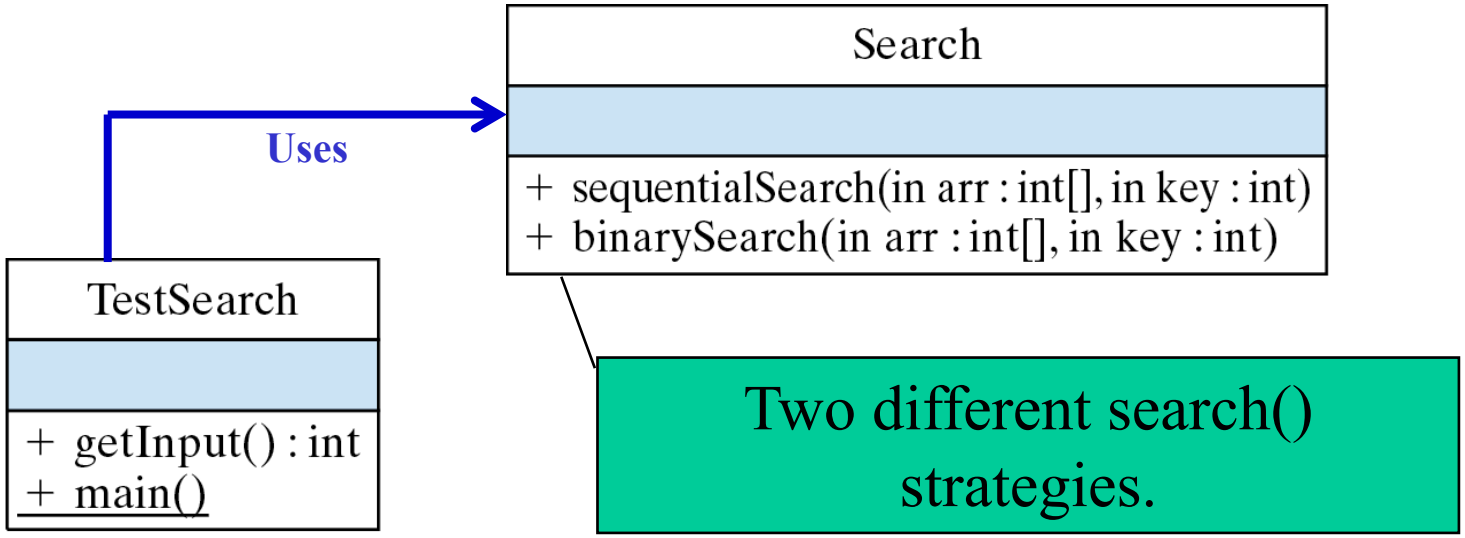



```
int a[][] = { {1, 2, 3}, {4, 5, 6} } ;
char c[][] = { {'a', 'b'}, {'c', 'd'} } ;
double d[][] = { {1.0, 2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0, 8.0, 9.0} } ;
```

- Each dimension of a multidimensional array can have a different length.

Part 2: Array Searching Algorithms

- **Searching**: Determine whether a given value exists in a data structure or a storage media
- Two searching methods on arrays: **linear** (sequential) and **binary** search



Array Algorithm: Sequential Search

- **Problem:** Search an array for a ***key*** value. If the array is not sorted, we have to search ***sequentially***.

```
/**
  * Performs a sequential search of an integer array
  * @param arr is the array of integers
  * @param key is the element being searched for
  * @return the key's index is returned if the key is
  *  found otherwise -1 is returned
  * Pre:  arr is not null
  * Post: either -1 or the key's index is returned
  */
public int sequentialSearch(int arr[], int key) {
     for (int k = 0; k < arr.length; k++)
         if (arr[k] == key)
             return k;
     return -1;      // Failure
} // sequentialSearch()
```

Return as soon as the key is found.

Search fails if you get to the end of the array.

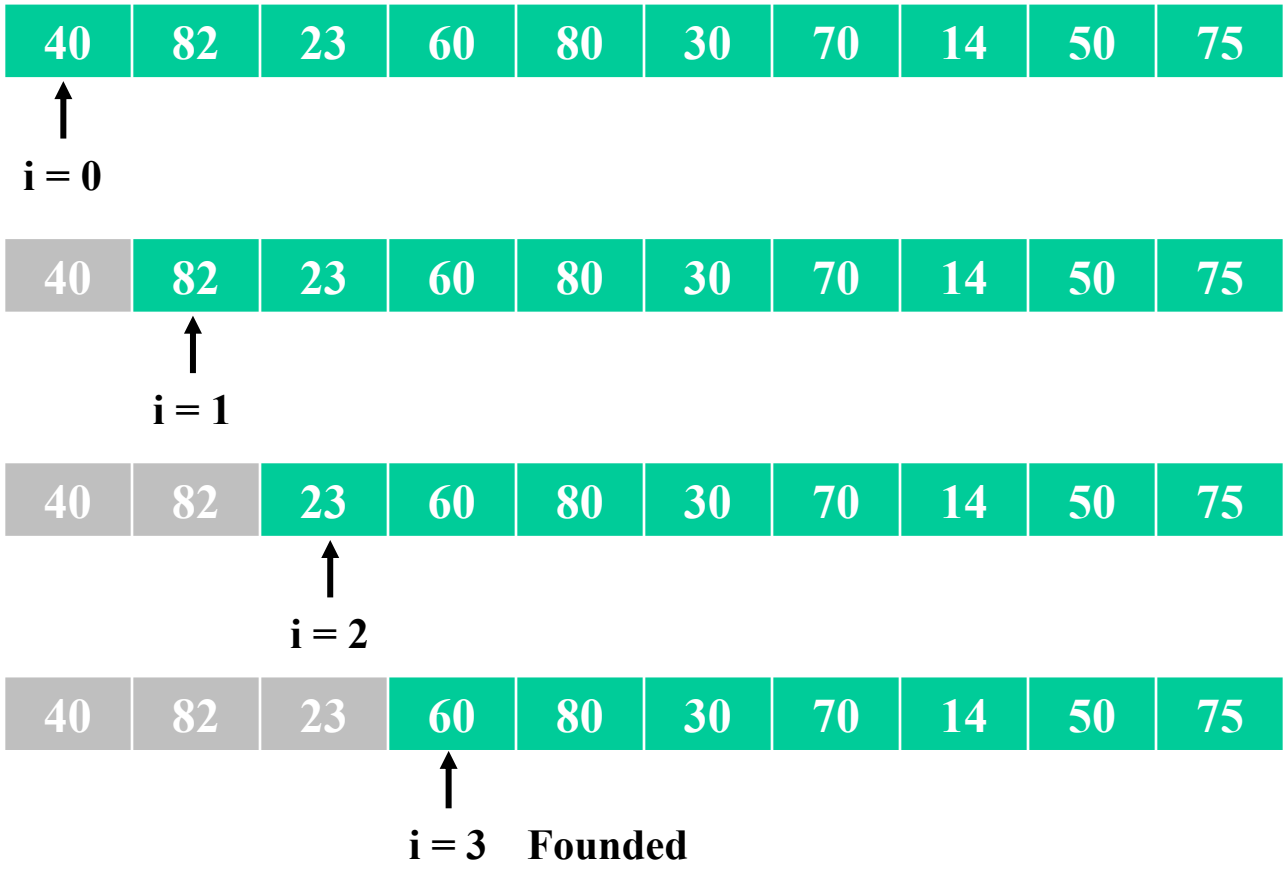
Linear (Sequential) Search

- The linear (or sequential) search algorithm on an array is:
  - Sequentially scan the array, comparing each array item with the searched value.
  - If a match is found; return the index of the matched element; otherwise return –1.
- Note: linear search can be applied to both sorted and unsorted arrays.

```
public static int linearSearch(Object[] array, Object key) {
    for(int k = 0; k < array.length; k++)
      if(array[k].equals(key))
              return k;
    return -1;
 }
```

Linear (Sequential) Search Illustrations

- Linear search of 60 in the below array



Running Time Analysis of Linear Search

- Given an array of n elements
- Best case: The search element is the first element in the array => search 1 time
- Worst case: The search element is the last element or not in the array => search n time
- Average case:
  - Average search time = $\frac{(1 + 2 + 3 + \ldots + n)}{n} = \frac{n+1}{2}$

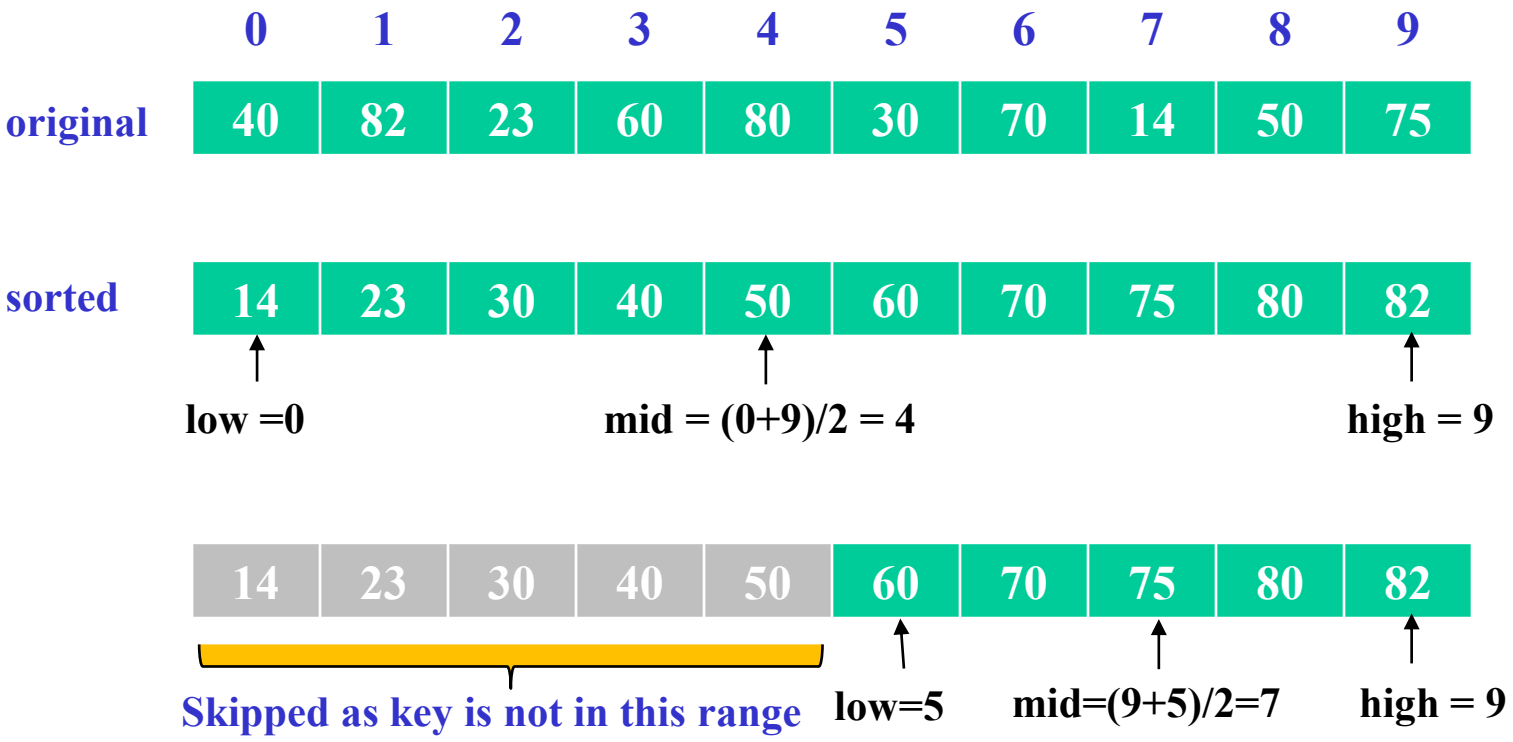
    >= $\frac{n}{2}$ time => O(n) time

Array Algorithm: Binary Search

- ***Binary search*** uses a ***divide-and-conquer*** strategy on a ***sorted*** array ($a[0] \leq a[1] \leq \cdots \leq a[n-1]$).
- Binary search pseudocode:
  - If the search value equals to the value of the middle element of the array or array segment, return the middle index
  - Else if the search value is greater than the value in the middle of the array or array segment:
    - Skip the left half of the array or array segment
    - Repeat binary search on the second half of the array or segment
  - Else,
    - Skip the right half of the array or array segment
    - Repeat binary search on the left half of the array or segment

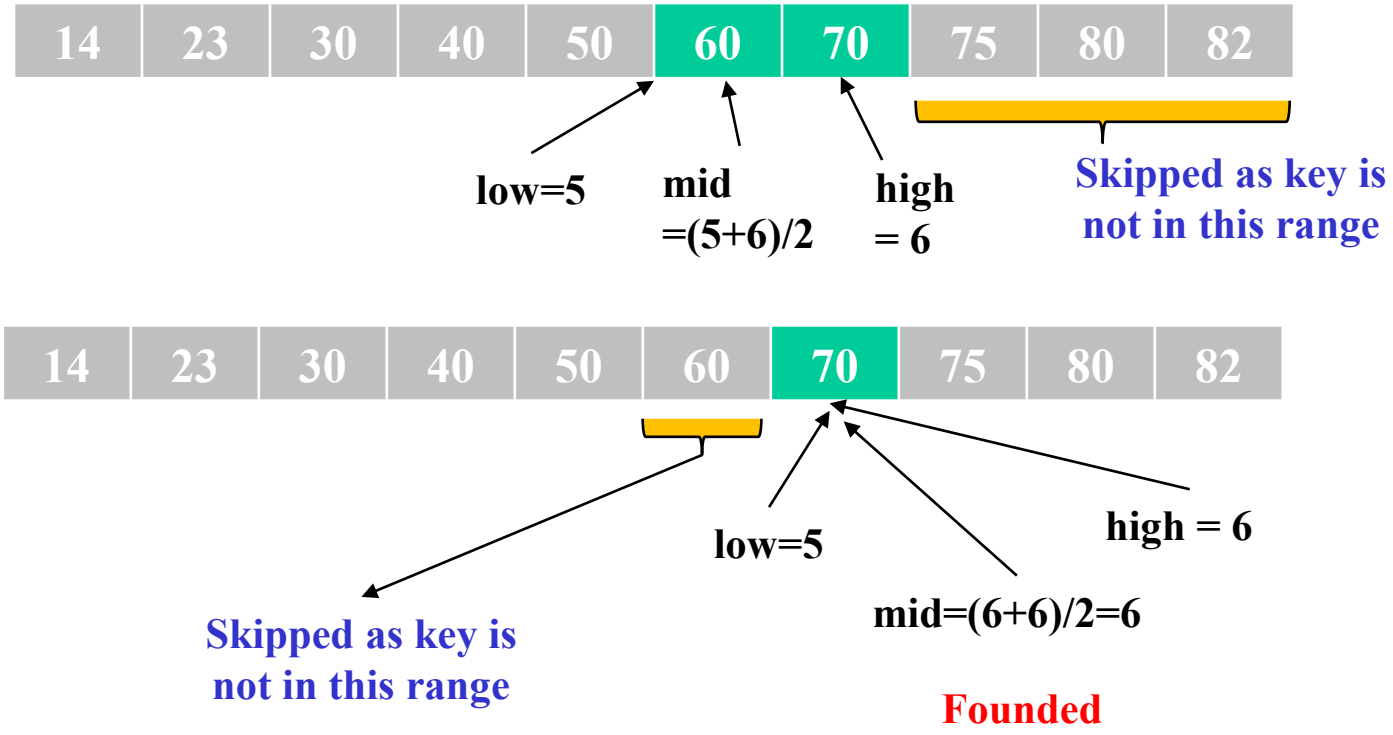
Binary Search Illustrations

- Searching for a key = 70 in an array of 10 elements

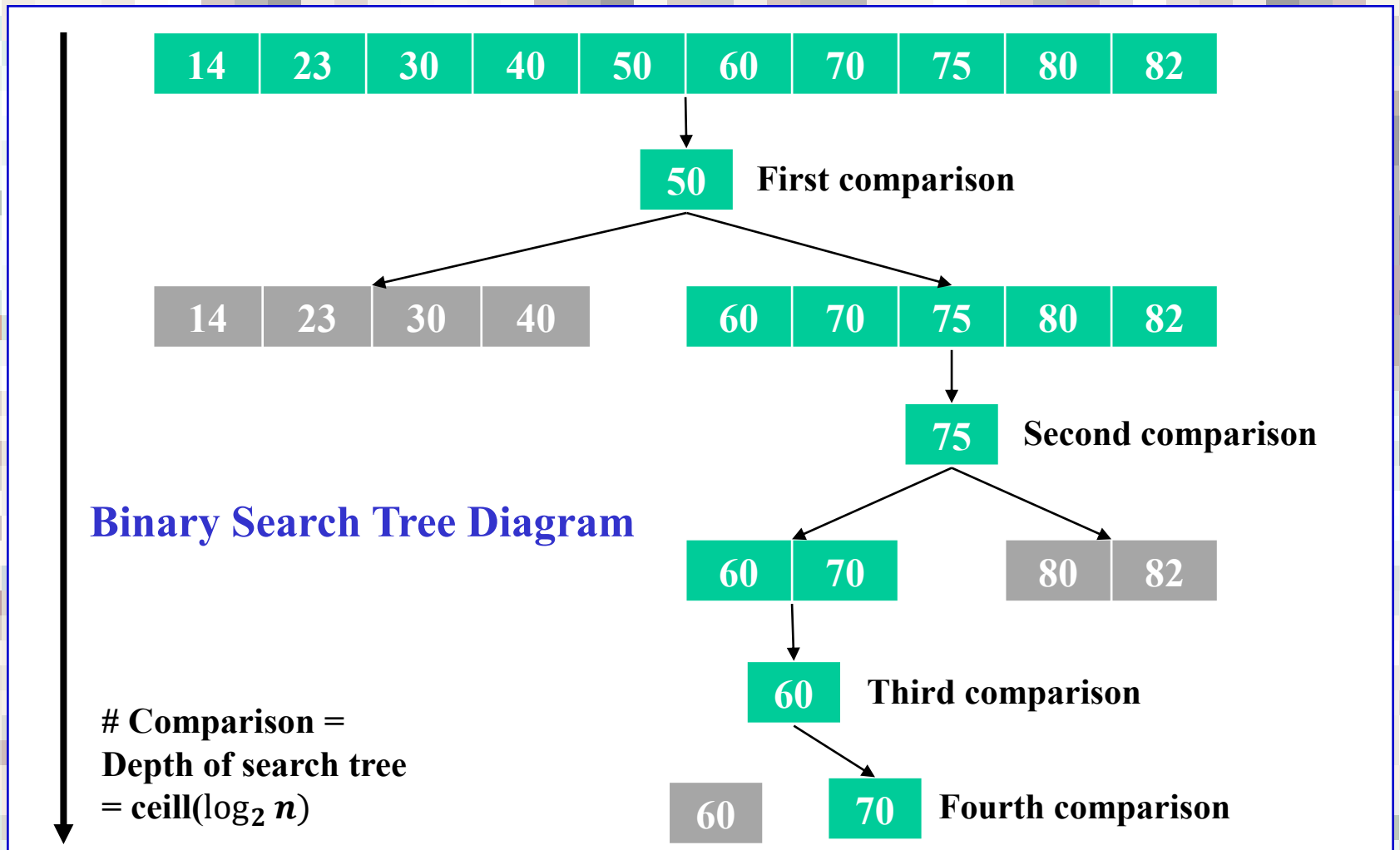


Binary Search Illustrations (cont.)

- Searching for a key = 70 in an array of 10 elements (cont.)



Binary Search Illustrations (cont.)



Binary Search Illustrations (cont.)

14	23	30	40	50	60	70	75	80	82

Tracing table the binary search of 70 on the above array

Key	Iteration	Low	High	Mid
70	0	0	9	4
70	1	5	9	7
70	2	5	6	5
70	3	6	6	6

The binarySearch() Method

- Algorithm Design: *low* and *high* point to first and last elements of the *subarray*, and *mid* gives its current midpoint.

```
/**
 * Pre: arr is an array of int in ascending order
 * Post: -1 or arr[k] where arr[k] == key
 */
public int binarySearch(int arr[], int key) {
        int low = 0;                    // Initialize low and high bounds
        int high = arr.length - 1;
        while (low <= high) {           // While not done
            int mid = (low + high) / 2;
            if (arr[mid] == key)
                return mid;             // Success
            else if (arr[mid] < key)
                low = mid + 1;          // Search right half
            else
                high = mid - 1;         // Search left half
        }  // while
        return -1;     // Post condition: low > high implies search
failed
} // binarySearch()
```

If low becomes greater than high, the key is not in the array.

Calculate a new midpoint.

Update low and high to cut the array in half.

Efficiency of Binary Search

- The binary search algorithm is extremely fast compared to a linear (sequential) algorithm that check all array elements in order
  - Among a binary search, first we can skip half of the array, and skip a quarter of the array, then an eighth of the array, and so on until found or not
- The binary search algorithm has a worst case running time of $O(\log n)$ compared to O(n) of linear (sequential) search
  - Given an array of 1,024 elements, binary search will need to compare about 10 elements with the search value, while the linear search requires an average of 500
  - An array of one billion elements takes <= 30 comparisons

Part 3: java.util.Vector Library

- The java.util.Vector class implements an array of objects that can grow as needed *(dynamic)*.
- Regular arrays are limited to their initial size. They cannot grow or shrink. (*static*)

Vector
+ Vector() + Vector(in size : int) + addElement(in o : Object) + elementAt(in index : int) : Object + insertElementAt(in o : Object, in x : int) + indexOf(in o : Object) : int + lastIndexOf(in o : Object) : int + removeElementAt(in index : int) + size( ) : int

Vector Illustration

- Use a Vector to store items when you don't know in advance how many items there are.

```
import java.util.Vector;

public class VectorDemo {

    public static void printVector( Vector v) {
        for (int k=0; k < v.size(); k++)
            System.out.println(v.elementAt(k).toString());
    } // printVector()

    public static void main(String args[]) {
        Vector vector = new Vector();              // An empty vector
        int bound = (int)(Math.random() * 20);
        for (int k = 0; k < bound; k++ )           // Insert a random
            vector.addElement(new Integer(k));     //  number of Integers
        printVector(vector);                       // Print the elements
    } // main()
} // VectorDemo
```

Technical Terms

- array
- array initializer
- array length
- binary search
- data structure
- element
- element type
- insertion sort
- multidimensional array
- one-dimensional array
- polymorphic sort method
- selection sort
- sequential search
- sorting
- subscript
- two-dimensional array

Summary Of Important Points

- An *array* is a named collection of *contiguous* storage locations holding data of the *same type*.
- An array's values may be initialized by assigning values to each array location. An *initializer* expression may be included as part of the array declaration.
- For multidimensional arrays, each dimension of the array can have its own length variable.

Summary Of Important Points (cont)

- *Array Parameters:* When an array is passed as a parameter, a reference to the array is passed rather than the entire array itself.
- Swapping two elements of an array, or any two locations in memory, requires the use of a temporary variable.
- Sequential search and binary search are examples of array searching algorithms. Binary search requires that the array be sorted.