CS1150
Principles of Computer Science
Arrays (Part II)

Yanyan Zhuang

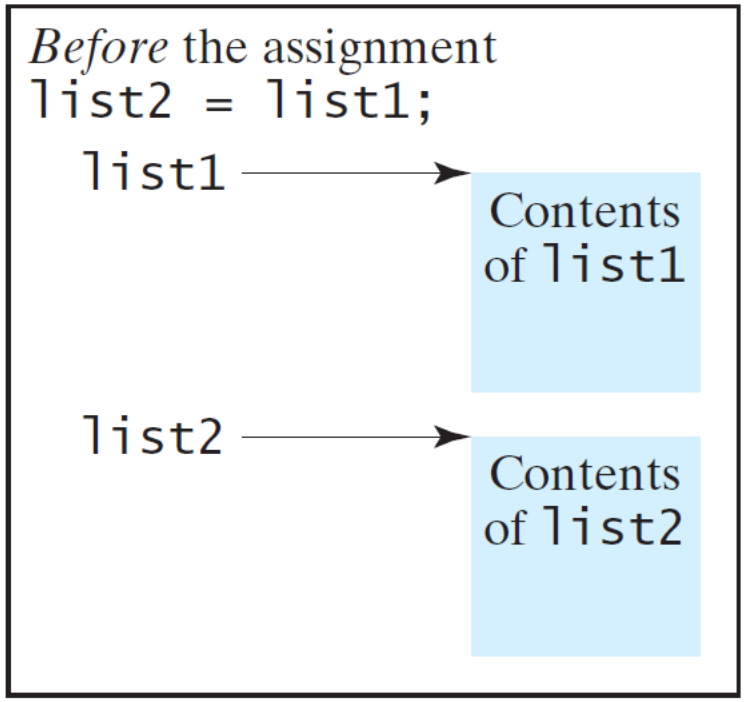
Department of Computer Science

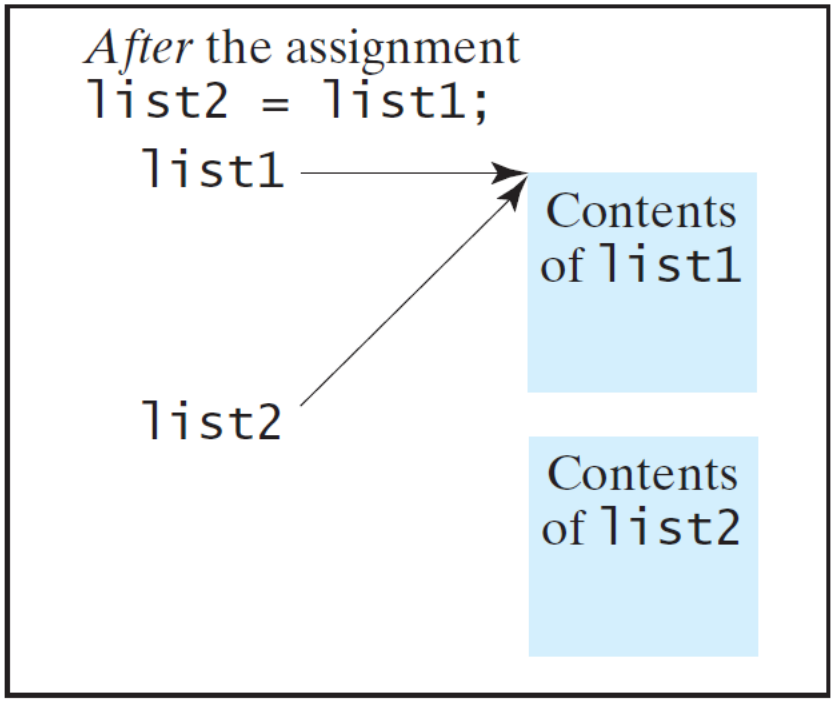
http://www.cs.uccs.edu/~yzhuang

Copying Arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

list2 = list1;





Copying Arrays

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];


for (int i = 0; i < sourceArrays.length; i++)
    targetArray[i] = sourceArray[i];
```

Passing Arrays to Methods

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Invoke the method

```
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Invoke the method

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

Anonymous Array

The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ..., literalk};
```

There is no explicit reference variable for the array.

Such array is called an *anonymous array*.

Pass By Value

Java uses *pass by value* to pass arguments to a method. There are important differences between passing a value of variables of primitive data types and passing arrays.

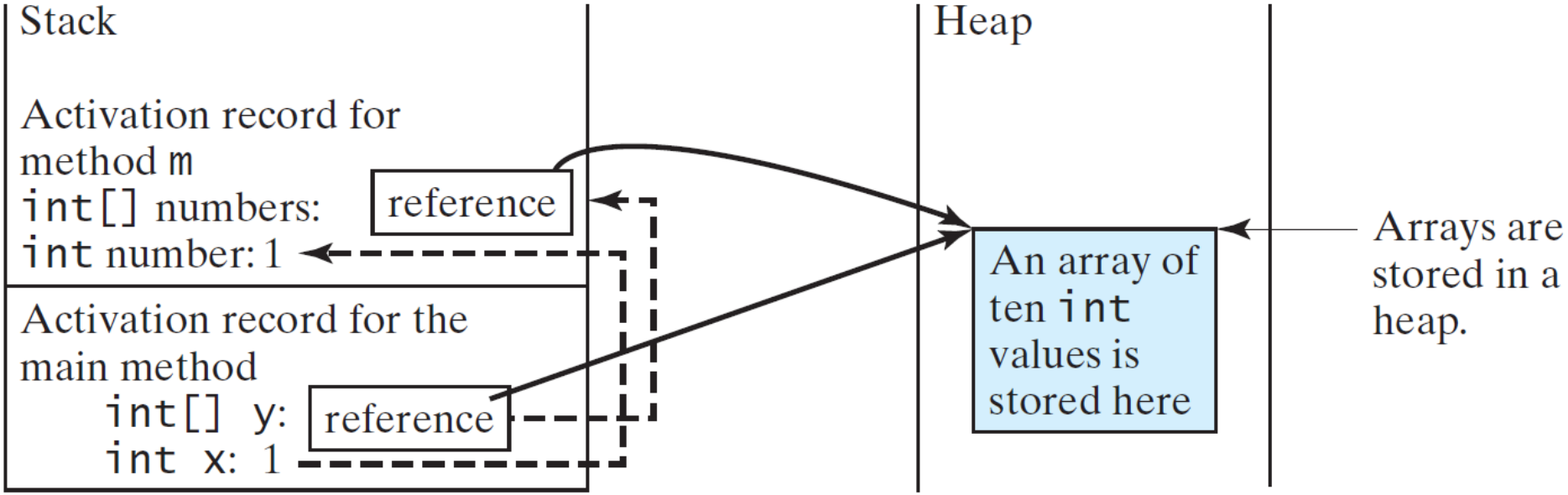
- For a parameter of a **primitive type** value, the actual value is passed. Changing the value of the local parameter inside the method *does not affect the value of the variable outside* the method.

- For a parameter of an **array type**, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body *will affect the original array* that was passed as the argument.

Simple Example

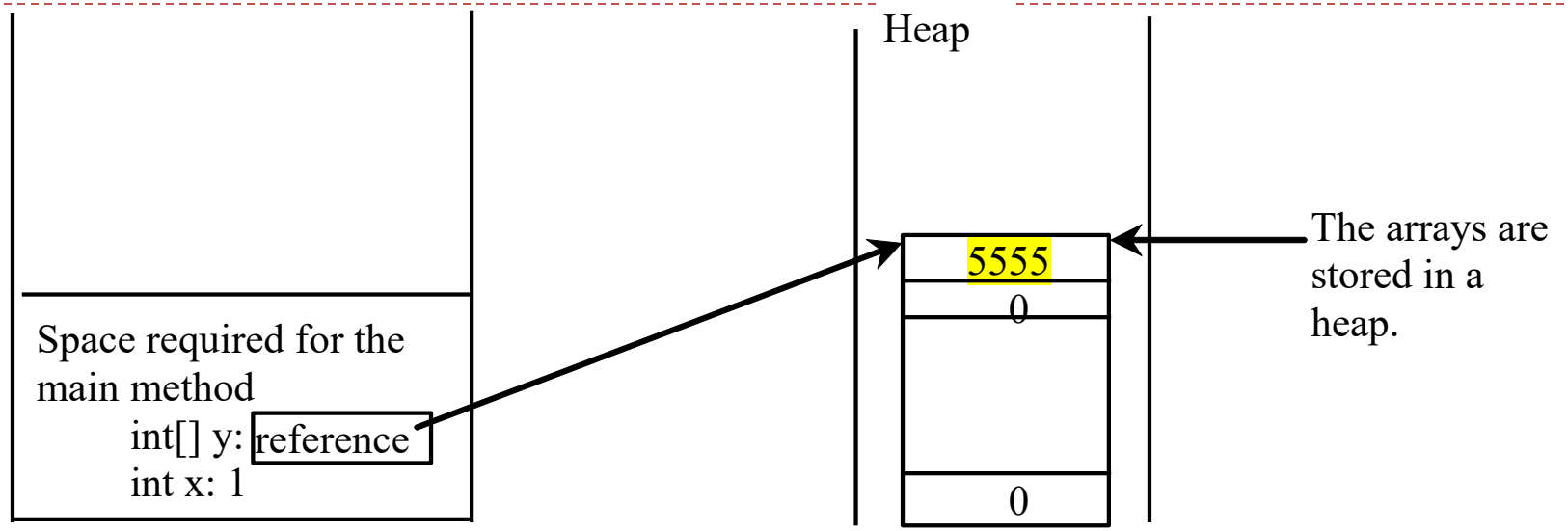
```
public class Test {
  public static void main(String[] args) {
    int x = 1; // x represents an int value
    int[] y = new int[10]; // y represents an array of int values

    m(x, y); // Invoke m with arguments x and y

    System.out.println("x is " + x);    // x is 1
    System.out.println("y[0] is " + y[0]);  // y[0] is 5555
  }

  public static void m(int number, int[] numbers) {
    number = 1001; // Assign a new value to number
    numbers[0] = 5555; // Assign a new value to numbers[0]
  }
}
```

Call Stack



When invoking m(x, y), the values of x and y are passed to number and numbers. Since y contains the reference value to the array, numbers now contains the same reference value to the same array.

Heap



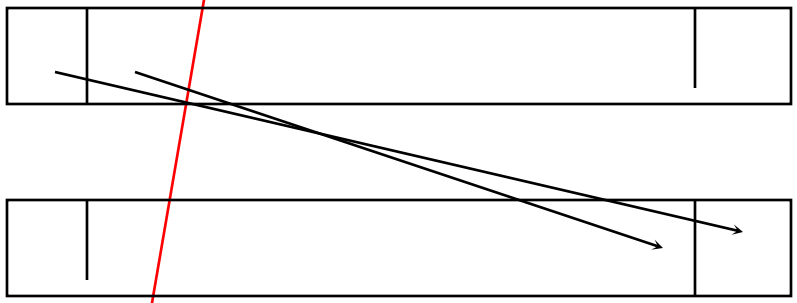
Java stores the array in an area of memory, called *heap*, which is used when **blocks of memory** are allocated and freed in an arbitrary order.

Passing Arrays as Arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables (example Array6.java)

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```



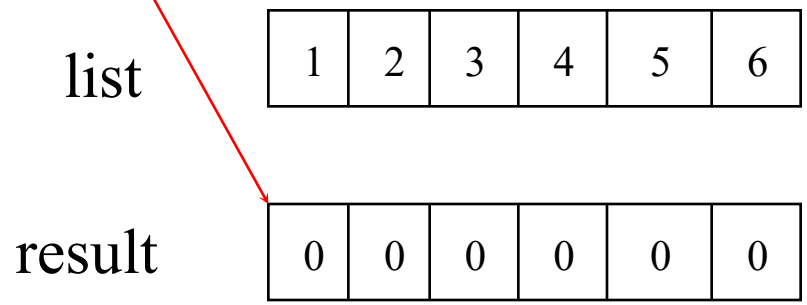

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Trace the reverse Method

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

Declare result and create array

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

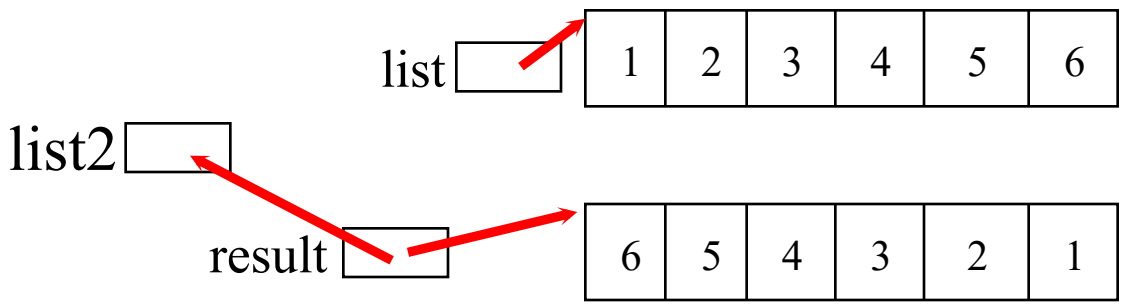


Trace the reverse Method, cont.

```
int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);
```

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1;
       i < list.length; i++, j--) {
    result[j] = list[i];
  }

  return result;
}
```

Return result



Main Method Is Just a Regular Method

You can call a regular method by passing actual parameters. Can you pass arguments to main? Of course, yes.

For example, the main method in class B is invoked by a method in A, as shown below:

```
public class A {
  public static void main(String[] args) {
    String[] strings = {"New York",
      "Boston", "Atlanta"};
    B.main(strings);
  }
}
```

```
class B {
  public static void main(String[] args) {
    for (int i = 0; i < args.length; i++)
      System.out.println(args[i]);
  }
}
```

Command-Line Parameters

```
class TestMain {
  public static void main(String[] args) {
  ...
  }
}

java TestMain arg0 arg1 arg2 ... argn
```

Processing Command-Line Parameters

In the main method, get the arguments from `args[0], args[1], ..., args[n]`, which corresponds to `arg0, arg1, ..., argn` in the command line.

Example: O07CommandLineArgs.java

Variable-Length Arguments

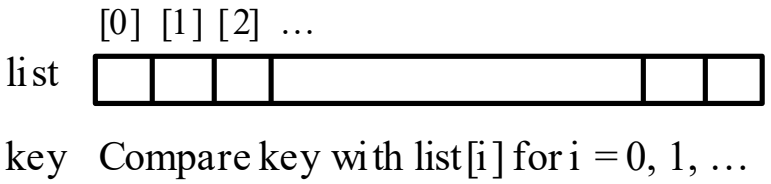
You can pass a variable number of arguments of the same type to a method

- double... means any number of double arguments can be sent
- All double arguments comes to the method as a double array
- We can have only one variable length argument, and it must be the last one
- Example: VarLengthArgs.java

Searching Arrays

- Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores.
- Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching.
- In this section, two commonly used approaches are discussed, *linear search* and *binary search*.

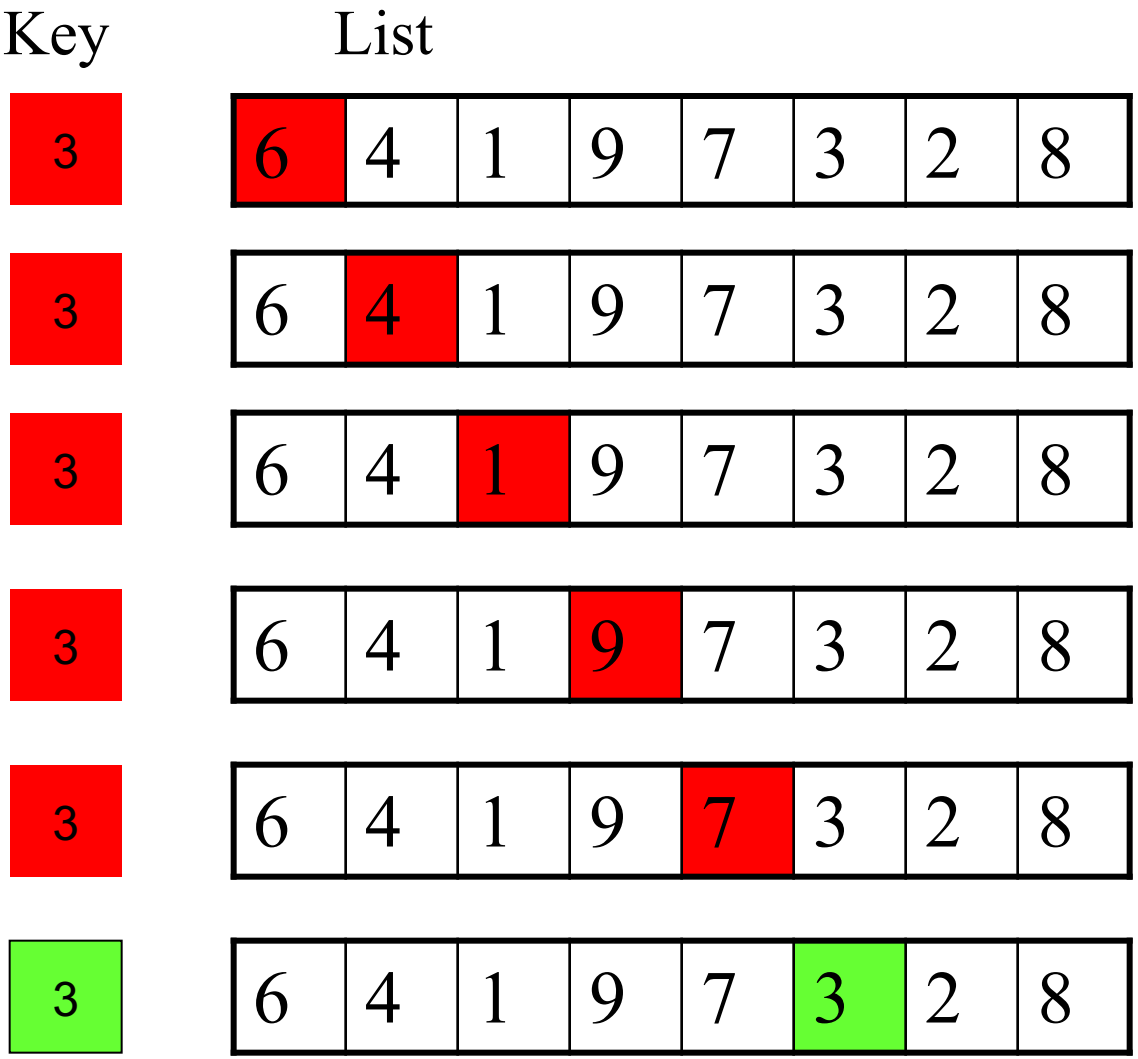
```
public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
      if (key == list[i])
        return i;
    return -1;
  }
}
```



Linear Search

- Compares the key element, key, *sequentially* with each element in the array list.
- Continues until the key matches an element in the list or the list is exhausted without a match.
- If a match is found, returns the index of the element in the array that matches the key. Otherwise returns -1.

Linear Search Animation



From Idea to Solution

```
/** The method for finding a key in the list */
public static int linearSearch(int[] list, int key) {
  for (int i = 0; i < list.length; i++)
    if (key == list[i])
      return i;
  return -1;
}
```

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Binary Search

For binary search to work, the elements in the array must **already be ordered**. Without loss of generality, assume that the array is in ascending order.

e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79

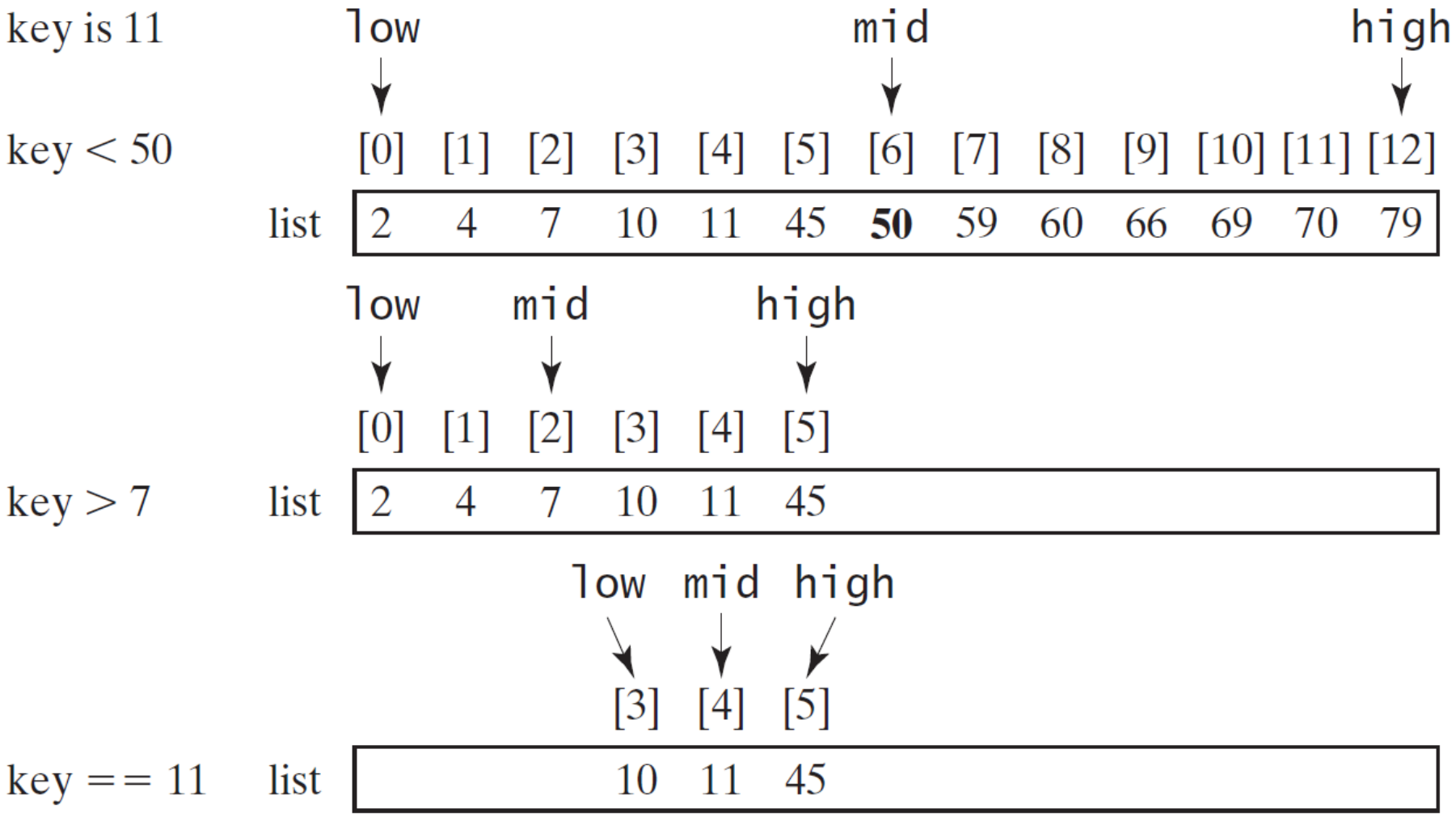
The binary search first compares the key with the element in the middle of the array.

Binary Search, cont.

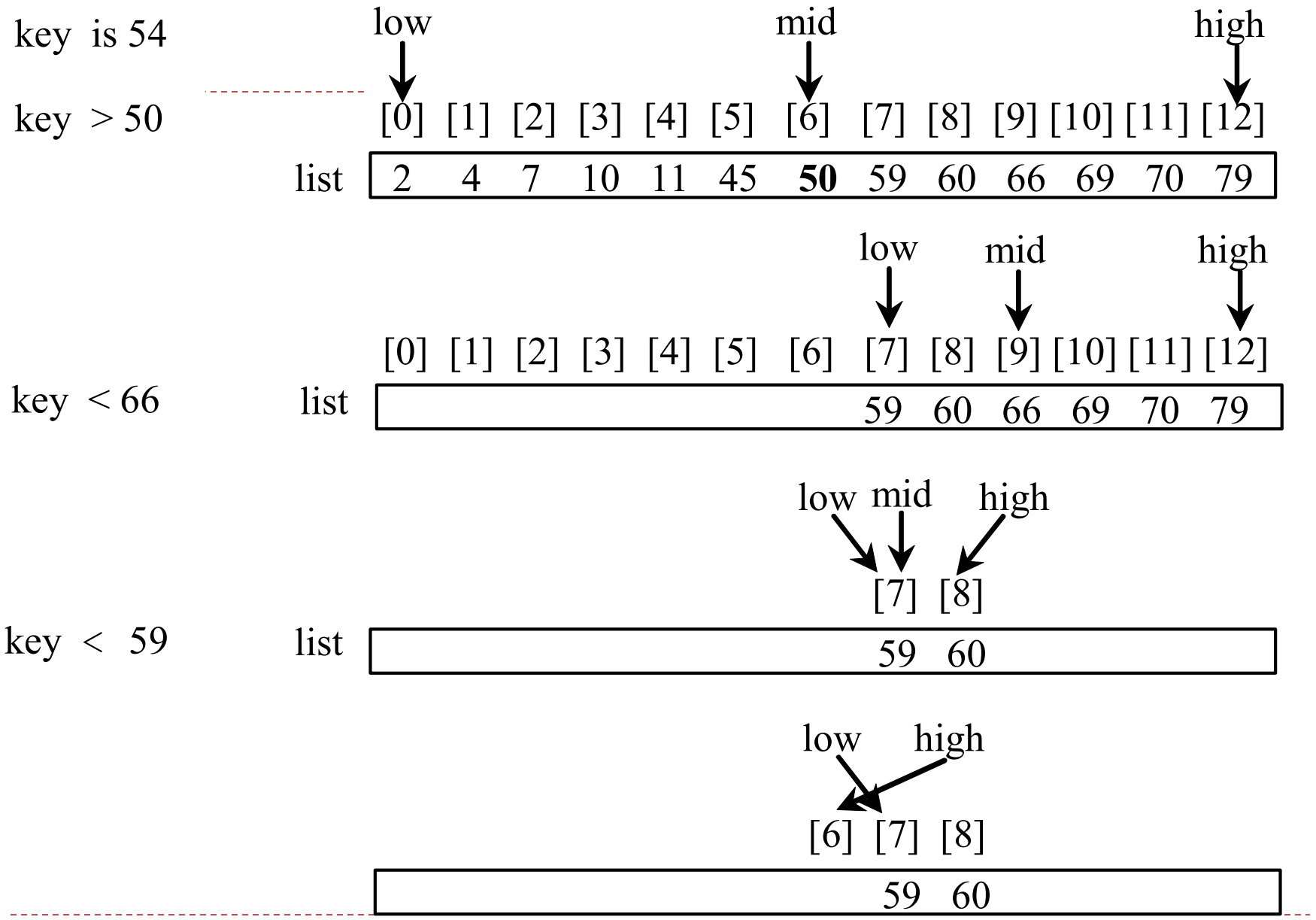
Consider the following three cases:

- If the key is **less than** the middle element, you only need to search the key in the first half of the array.
- If the key is **equal** to the middle element, the search ends with a match.
- If the key is **greater than** the middle element, you only need to search the key in the second half of the array.

Binary Search, cont.



Binary Search, cont.



From Idea to Soluton

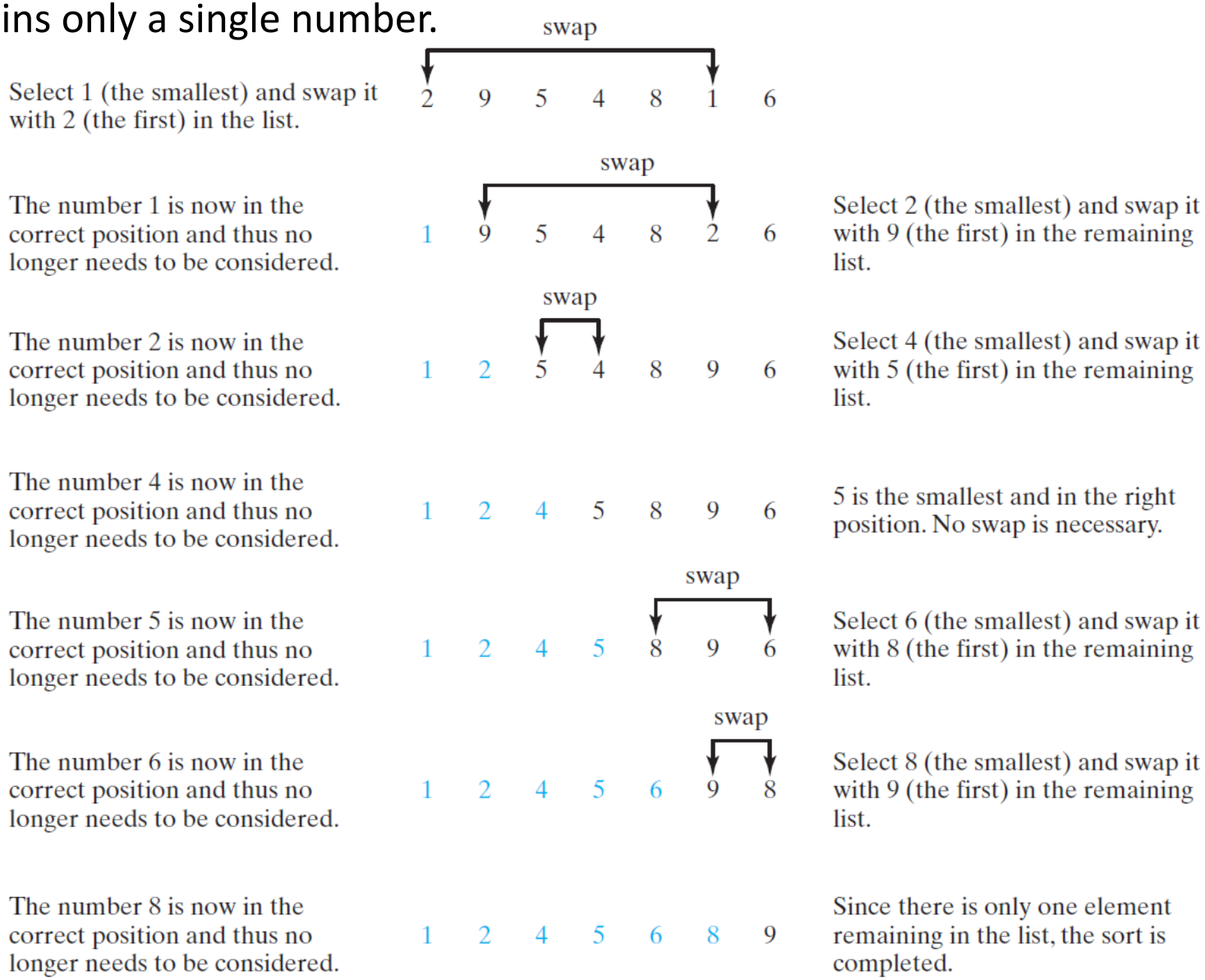
```
/** Use binary search to find the key in the list */
public static int binarySearch(int[] list, int key) {
  int low = 0;
  int high = list.length - 1;

  while (high >= low) {
    int mid = (low + high) / 2;
    if (key < list[mid])
      high = mid - 1;
    else if (key == list[mid])
      return mid;
    else
      low = mid + 1;
  }

  return -1;
}
```

Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithms: *selection sort*.

Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.



From Idea to Solution

```
for (int i = 0; i < list.length; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i+1..listSize-1]
}
```

```
list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]

list[0] list[1] list[2] list[3] ...                    list[10]
```

http://www.cs.armstrong.edu/liang/animation/web/SelectionSort.html

```
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

Expand

```
double currentMin = list[i];
int currentMinIndex = i;
for (int j = i+1; j < list.length; j++) {
  if (currentMin > list[j]) {
    currentMin = list[j];
    currentMinIndex = j;
  }
}
```

```
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i..listSize-1]
}
```

Expand

```
if (currentMinIndex != i) {
  list[currentMinIndex] = list[i];
  list[i] = currentMin;
}
```

Wrap it in a Method

```
/** The method for sorting the numbers */
public static void selectionSort(double[] list) {
  for (int i = 0; i < list.length; i++) {
    // Find the minimum in the list[i..list.length-1]
    double currentMin = list[i];
    int currentMinIndex = i;
    for (int j = i + 1; j < list.length; j++) {
      if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
      }
    }

    // Swap list[i] with list[currentMinIndex] if necessary;
    if (currentMinIndex != i) {
      list[currentMinIndex] = list[i];
      list[i] = currentMin;
    }
  }
}
```

Summary

- Creating/accessing arrays
- Common array operations
- Foreach loops
- Searching/sorting arrays