JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris July 2001

3 LINEAR SEARCH

3.1 LEARNING OUTCOMES

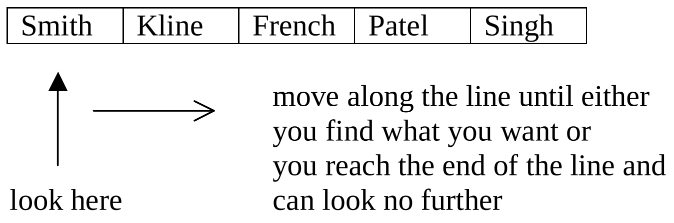
By the end of this lesson the student should be able to

- describe how a linear search algorithm works
- implement and test a linear search algorithm

3.2 INTRODUCTION

We search through a collection of objects, looking for one object in particular. A search always has one of two outcomes: either success - we have found what we are looking for, or failure - we have not found what we are looking for.

An algorithm is a sequence of statements to accomplish some task. We shall describe the linear search algorithm. The search is described as linear because we are, in affect, looking along a line of objects until either we find the one we are looking for, or we reach the end of the line.


Smith
Kline
French
Patel
Singh
look here
move along the line until either
you find what you want or
you reach the end of the line and
can look no further

3.3 LINEAR SEARCH

We shall create the following array

array

13	19	11	17	15
0	1	2	3	4

To test the search algorithm we shall

(a) look for the *first* item in the array - 13
(b) look for the *last* item in the array - 15
(c) look for an item not in the array - 23

The first and last items in an array represent its boundaries and therefore should always feature in any testing we do.

Here is the Java code fragment to create the array.

```
final int capacity = 5;  // indexed 0..4
Integer[] array = new Integer[capacity];


array[0] = new Integer(13); array[1] = new Integer(19);
array[2] = new Integer(11); array[3] = new Integer(17);
array[4] = new Integer(15);
```

We fill the array with *Integer* objects rather than *int* values because we want our search algorithm to work with any kind of object.

We display the contents of the array so that we can *see* that it contains exactly what we think it contains.

```
     System.out.println("The array contains");
     String string = "[ ";
     for (int i = 0; i < array.length; i++) {
       string += array[i];
       if (i + 1 < array.length)
         string += ", ";
     }
     string += " ]";
     System.out.println(string);
```

There are two things to notice here.

First, the number of elements in the array is represented by *array.length*. *length* is not a method (it has no brackets); rather it is a public data field belonging to the array.

Second, we are assuming that the objects have a *toString()* method.

Now onto the linear search algorithm itself.

```
  public static String search(Object[] array, Object toFind)
  {
    for (int i = 0; i < array.length; i++) {
      if (array[i].equals(toFind))
        return "success";
    }
    return "failure";
  }
```

We pass an array and the object we are trying to find into the parameters *array* and *toFind*.

We set up an index, *i*, to look at each element in the array in turn.

If in some location *i* in the array we find that its contents equal the object we are looking for, we return *success*.

If we reach the end of the array and have not yet found what we are looking for, it must be because it is not there; we return *failure*.

There are three things to notice here.

The linear search algorithm requires objects to have their own *equals()* method.

The algorithm is both *public* and *static*.

If the array contains duplicate objects equal to the given object, it is the first object that is found.

Now for the testing.

```
     System.out.println("Searching for 13 ... " +
                        search(array, new Integer(13)));
     System.out.println("Searching for 15 ... " +
                        search(array, new Integer(15)));
     System.out.println("Searching for 23 ... " +
                        search(array, new Integer(23)));
```

We pass the array and the object we are searching for as argument values to the search algorithm.

```
     search(array, new Integer(13)));
```

We display the value, *success* or *failure*, returned by *search()*.

```
     System.out.println(search(array, new Integer(13)));
```

The results of the tests are shown below.

```
     Searching for 13 ... success
     Searching for 15 ... success
     Searching for 23 ... failure
```

Here is the complete program.

```
/* LinearSearch.java
   Terry Marris  20 July 2001
*/


public class LinearSearch {
  public static String search(Object[] array, Object toFind)
  {
    for (int i = 0; i < array.length; i++) {
      if (array[i].equals(toFind))
        return "success";
    }
    return "failure";
  }


  public static void main(String[] s)
  {
    final int capacity = 5;  // indexed 0..4
    Integer[] array = new Integer[capacity];

    array[0] = new Integer(13); array[1] = new Integer(19);
    array[2] = new Integer(11); array[3] = new Integer(17);
    array[4] = new Integer(15);

    System.out.println("The array contains");
    String string = "[ ";
    for (int i = 0; i < array.length; i++) {
      string += array[i];
      if (i + 1 < array.length)
        string += ", ";
    }
    string += " ]";
    System.out.println(string);

    System.out.println("Searching for 13 ... " +
                       search(array, new Integer(13)));
    System.out.println("Searching for 15 ... " +
                       search(array, new Integer(15)));
    System.out.println("Searching for 23 ... " +
                       search(array, new Integer(23)));
  }
}
```

```
The array contains[ 13, 19, 11, 17, 15 ]
Searching for 13 ... success
Searching for 15 ... success
Searching for 23 ... failure
```

3.4 REVIEW

3.5 FURTHER READING

In the next lesson we look at sorting.

3.6 EXERCISES

1 Explain what is meant by *searching*. State the two possible outcomes of any search operation.

2 Dry run the linear search algorithm two times

> **(a)** when *toFind* = 11
> **(b)** when *toFind* = 10

(a) is started for you.

array

13	19	11	17	15
0	1	2	3	4

↑

i = 0

array

13	19	11	17	15
0	1	2	3	4

array

13	19	11	17	15
0	1	2	3	4

toFind	i	i<array.length	toFind.equals(array[i])
11	0	true	false

return

2 Re-write the linear *search(Object [], Object)* algorithm described in §3.3 above so that it returns the index where the given object is located in the given array, or -1 if the array does not contain the object. Test your amendments.