SCHOOL OF COMPUTER SCIENCE

PRIFYSGOL BANGOR UNIVERSITY

Object Oriented Programming in Java

Laboratory 7 Searching and Sorting

Dave Perkins

Introduction

This laboratory session covers a variety of search and sort algorithms, including:

- Linear Search
- Binary Search
- Selection Sort
- Bubble Sort

This lab contains no assessed lab work.

Exercise 1: Linear Search

Modify Horstmann's `LinearSearcher` class so that it contains a method `searchforAll()` which returns a list of the indexes of *all the occurrences* of the target value.

For example, given the array below and a search key of 4 your search algorithm will return a two element list holding the values 2 and 9.

```
list = {0, 5, 4, 7, 2, 8, 9, 0, 3, 4}
```

You must decide how best to implement the list and also what should be returned if the search key is not in the list.

Develop a range of test cases and use these to write a test driver for your new method.

Exercise 2: Recursive Linear Search

Write a recursive version of the linear search algorithm and add this to your modified `LinearSearcher` class. The heading of the method is provided below:

```
public int recursiveSearch(int key, int index)
```

If the search key is found the index of the key is returned, otherwise -1 is returned. As always test your method against a set of carefully selected test cases.

Hint. Each call to the method should check just one position in the array.

Exercise 3: Implement Binary Search for Strings

Using material from Lecture 11 on searching, write a Java method to implement the binary search algorithm for an array of strings.

To test this method you might want to write a method to generate an array of random strings – you could store this method in **`ArrayUtil`** (see Appendix 1)|.

NB. For binary search to work the elements in the array must *already* be in sorted order. This means that you need to be able to sort an array of strings. How to do this? Have a look in the class **`Arrays`** and see if there is a sort method there to help you.

Exercise 4: Modifying Binary Search

The binary search algorithm presented by Horstmann returns the value -1 if no match is found. Modify the algorithm so that if the target or key value is not found the method returns

(-*insertionpoint*) - 1

Note that the *insertionpoint* is defined as the position in the array at which the key value would have been inserted. This guarantees that values >=0 are only returned by the search method if the key is present in the array.

This implementation corresponds to the version of **`binarySearch`** in **`Arrays`**.

Why is it necessary to subtract 1 from the insertion point? Place a comment in the program text to clarify this matter.

Exercise 5: Selection Sort

Modify the selection sort algorithm so that the order of sorting is *descending* rather than *ascending*. Also implement the algorithm so that it sorts a list of strings in descending lexicographic order.

Use Horstmann's class **`SelectionSorter`** as your starting point in this exercise but this class will obviously need to be modified.

Exercise 6: Investigating Bubble Sort

Bubble Sort is a simple, but not very efficient, sort algorithm.

Using Horstmann'sSelectionSorter class as a guide implement a class **BubbleSorter** and a test class **BubbleSortTester**. Once you have this search algorithm working profile the performance of Bubble Sort by completing the following table:

n	**Time(ms)**
10,000	
20,000	
30,000	
40,000	
50,000	
60,000	

Note that the first column **n** signifies the number of elements in the array to be sorted, whilst the second column measures the time taken for the sort in milliseconds.

Use Horstmann's **StopWatch** class to measure the time. (Study Sections 13.1 and 13.2 carefully)

Exercise 7: Sorting Bank Accounts (Challenge)

Modify the class **BankAccount** so that an appropriate sort method in the utility class **Arrays** can be used to put an array of **BankAccount** objects into sorted order.

Hint. The solution involves implementing an interface.

Appendix 1: Array Utility Class

```
import java.util.Random;

/**
      The class ArrayUtil contains utility methods for array
      manipulation.
*/

public class ArrayUtil
{
   /**
      Creates an array filled with random values.
      @param length the length of the array
      @param n the number of possible random values
      @return an array filled with length numbers between
      0 and n - 1
   */

   public static int[] randomIntArray(int length, int n)
   {
      int[] a = new int[length];
      for (int i = 0; i <a.length; i++)
         a[i] = generator.nextInt(n);

      return a;
   }

   private static Random generator = new Random();
}
```

NB. If you copy and paste this code from the PDF file you may get some formatting codes mixed in with your Java code. This will cause problems for the compiler.