**Further Object- Oriented Programming**

U08026: Week 9

Searching

David Lightfoot

Department of Computing and Communication Technologies

# Practical: Testing a Binary Search

# Exercise 1 - Setup

# Extract the supplied code files from the *.zip* file, and get them up and running as a project as usual. The files are as follows:

# • *Tester.java* contains a small amount of testing, to get you started.

# • *AscendingSequence.java*

# Take a look at the *AscendingSequence* class in particular. Note that the way it represents a sequence is to use an array *a*, of length *max*. The sequence is stored at positions 0 to *size*-1 of the array *a*.

# Various public methods are available for managing the sequence, which make it handy to set up sequences for testing purposes. The methods include:

# • *AscendingSequence(int max*) – constructor

# • *insert* – inserts a new element (can be a duplicate of one already there)

# • *clear* – makes the sequence empty

# • *insertLots* – a shorthand way of adding several values

# • *remove* – removes an element identified by its index – must be in range!

# There are also two methods for displaying the sequence, including:

# • *toString* – produces a string representation of the sequence

# • *containerArrayAsString* – returns a string representation of the entire array, not just the sequence.

# There is also a method to search for values in the sequence:

# • *search* – uses the binary search technique to search for a value.

# It is the *search* method that you will be testing (not any of the other methods). It attempts to find a value *val* in the sequence by using a binary search. It is wrong!

# public boolean search(int val) {

# int low = 0;

# int high = size-1;

# int middle = -1;

# boolean found = false;

# while (!found && low < high) {

# middle = low + (high-low)/2;

# if (a[middle] == val)

# found = true;

# else if (a[middle] < val)

# low = middle + 1;

# else // (a[middle] > val)

# high = middle - 1;

# }

# return found;

# }

# You are provided with a main method (in *Tester.java*) to get you started. It has a small amount of testing in it.

# Try running the provided main program as written. Does the search seem to work ok?

# Exercise 2

# For this part, you should edit the provided file *Tester.java*. Don’t forget to edit the *author*. You should provide six test cases for testing the search method, using white-box testing. For each of the six test cases:

# • Indicate the test case clearly (e.g. using a *System.out.println* statement)

# • Describe the nature of the test (for example: test that the middle element of the sequence can be found)

# • Provide a suitable sequence and an element to test for.

# • Annotate your main method with the expected outcome of your test and write messages (using *System.out.println*) to explain what you are doing.

# • Include white-box testing and consideration of what should be true at various points in the method (assertions).

# • Copy the contents of the output window into your *Tester.java* file as a /\*… \*/-type comment.

# Hints: Try a wide variety of sequences and test cases.

# Exercise 3

# Explain what error(s) you have found in the method *search*. Include a detailed explanation of why it is wrong.

# Exercise 4

# By editing the method *search* in the file *AscendingSequence.java*, try to correct the error(s) in that method.

# Annotate your method search with comments indicating the corrections you have made.

# Exercise 5

# Rerun your test cases on the corrected method and write a comment indicating the degree of success you have achieved.

# Copy the contents of the output window into your Tester.java file as a /\* … \*/ type comment, below the /\* … \*/ comment you have provided above.

# What you have to submit

# Please paste your *Tester.java* and corrected *AscendingSequence.java* files into a *Word* document (including the contents of the output).

# Zip your source files. Upload *Word* and *zipped* to Moodle.