CIS2344 Algorithms Processes and Data
Logbook

Contents

Week 1 / 2

Task 1

```java
package searcher;

/**
 * @author Adam Birch
 * @version October 2018

 */

public class CleverSearcher extends Searcher {

    CleverSearcher(int[] array, int k) {
        super(array, k);
    }


    @Override
    public int findElement() throws IndexingError {
        /**
         * This is the main search to be implemented
         * @return kth largest element
         * @exception IndexingError on bad input
         * @see IndexingError
         */
        int[] array = getArray();
        int k = getIndex();
        if (k <= 0 || k > array.length) { // Test that k is a valid search.
            throw new IndexingError();
        }
        int[] smallArray = new int[k];
                 // Create a small array of size k - the k'th largest is element 0
        for (int i = 0; i < array.length; i++) {
            int value = array[i];
            if (value > smallArray[0]){
                if (smallArray.length > 1){
                                    // Only loop if there are many values in the Array.
                    int x = 0;
                    while (x < smallArray.length - 1 && value > smallArray[x+1]){
                        smallArray[x] = smallArray[x+1];
                        x++;
                    }
                    smallArray[x] = value;
                 // Adds the new value to the Array once it is in the correct place.
                }
                else {
                    smallArray[0] = value;
                }
            }
        }
        return smallArray[0];
    }
}
```

When findElement() is called it will create a small array to hold the kth largest element – and the ones higher than it. It loops through all of the elements of the initial array, and sorts it into the smallArray (if it is larger than the current element in position 0).

Task 2

This is the created CleverSearcherTest

```
package searcher;

/**
 * @author Adam Birch
 * @version September 2018
 */

class CleverSearcherTest extends SearcherTest {

    protected Searcher createSearcher(int[] array, int index) throws IndexingError {
        return new CleverSearcher(array,index);
    }

}
```

This is the extended SearcherTest

```
package searcher;

import arrayGenerator.ArrayGenerator;
import arrayGenerator.CleverRandomListingGenerator;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

/**
 * @author Hugh Osborne
 * @version September 2018
 */

abstract class SearcherTest {

    /**
     * Create a searcher of the right type
     */
    abstract protected Searcher createSearcher(int[] array, int index) throws IndexingError;

    /**
     * Test that the searcher finds the correct value.  The test uses a random listing
                                                              generator to create
     * a random listing of the requited size.  Because of the properties of random listings,
                                                                        the kth largest
     * element of a random listing of size n must be n-k.
     * @param arraySize the size of the random listing to be generated (the "n" value)
     * @param index the index (the "k" value)
     * @throws IndexingError if k is out of bounds for n
     */
    private void testSearcher(int arraySize,int index) throws IndexingError {
        ArrayGenerator generator = new CleverRandomListingGenerator(arraySize);
        Searcher search = createSearcher(generator.getArray(), index);
        assertEquals(arraySize - index, search.findElement());
    }

    private void testNegativeSize(int arraySize) {

        Throwable e = assertThrows(IndexOutOfBoundsException.class, () -> {
            ArrayGenerator generator = new CleverRandomListingGenerator(arraySize);
        });
            assertEquals("Array size must be larger than 1.", e.getMessage());

    }

    private void testLargerIndex(int arraySize, int index) throws IndexingError {
        ArrayGenerator generator = new CleverRandomListingGenerator(arraySize);
        Searcher search = createSearcher(generator.getArray(), index);
        Throwable e = assertThrows(IndexingError.class, () -> {search.findElement();});
```

```java
        assertEquals("Index out of bounds", e.getMessage());

    }

    @Test
    void test2ndIn10() throws IndexingError {
        testSearcher(10,2);
    }

    @Test
    void test5thIn10() throws IndexingError {
        testSearcher(10,5);
    }
    @Test
    void test3rdIn100() throws IndexingError {
        testSearcher(100,3);
    }

    @Test
    void test16thIn100() throws IndexingError {
        testSearcher(100,16);
    }

    @Test
    void test8thIn1000() throws IndexingError {
        testSearcher(1000,8);
    }

    @Test
    void test107thIn1000() throws IndexingError {
        testSearcher(1000,107);
    }

    @Test
    void test1stIn10000() throws IndexingError {
        testSearcher(10000,1);
    }

    @Test
    void test1003rdn10000() throws IndexingError {
        testSearcher(10000,1003);
    }

    @Test
    void test11thIn100000() throws IndexingError {
        testSearcher(100000,11);
    }

    @Test
    void test4thIn1000000() throws IndexingError {
        testSearcher(1000000,4);
    }

        @Test // Test that a negative input throws an IndexOutOfBoundsException.
    /**
     * @throws IndexOutOfBoundsException
     */
    void testminus1(){
        testNegativeSize(-1);
    }

    @Test // Test that a size of 0 throws an IndexOutOfBoundsException.
    /**
     * @throws IndexOutOfBoundsException
     */
    void testZero(){
        testNegativeSize(0);
    }

    @Test // Test that a larger index throws the IndexingException
    /**
     * @throws IndexingError
     */
    void test10in1() throws IndexingError {
        testLargerIndex(1,10);
    }

    @Test // Test the smallest value in the array.
    void test100in100() throws IndexingError {
        testSearcher(100,100);
    }

}
```

I created 2 new types of test, each with its own test method. One to test for the outcome of a negative or zero size for the array, and the other for the outcome for a search when the index is larger than the array. Both of these test the type and message of a thrown exception – IndexOutOfBoundsException and IndexingError respectively.

I also extended the tests to include a 100 in 100 search – this was to test the borderline results, where the test was most likely to fail.

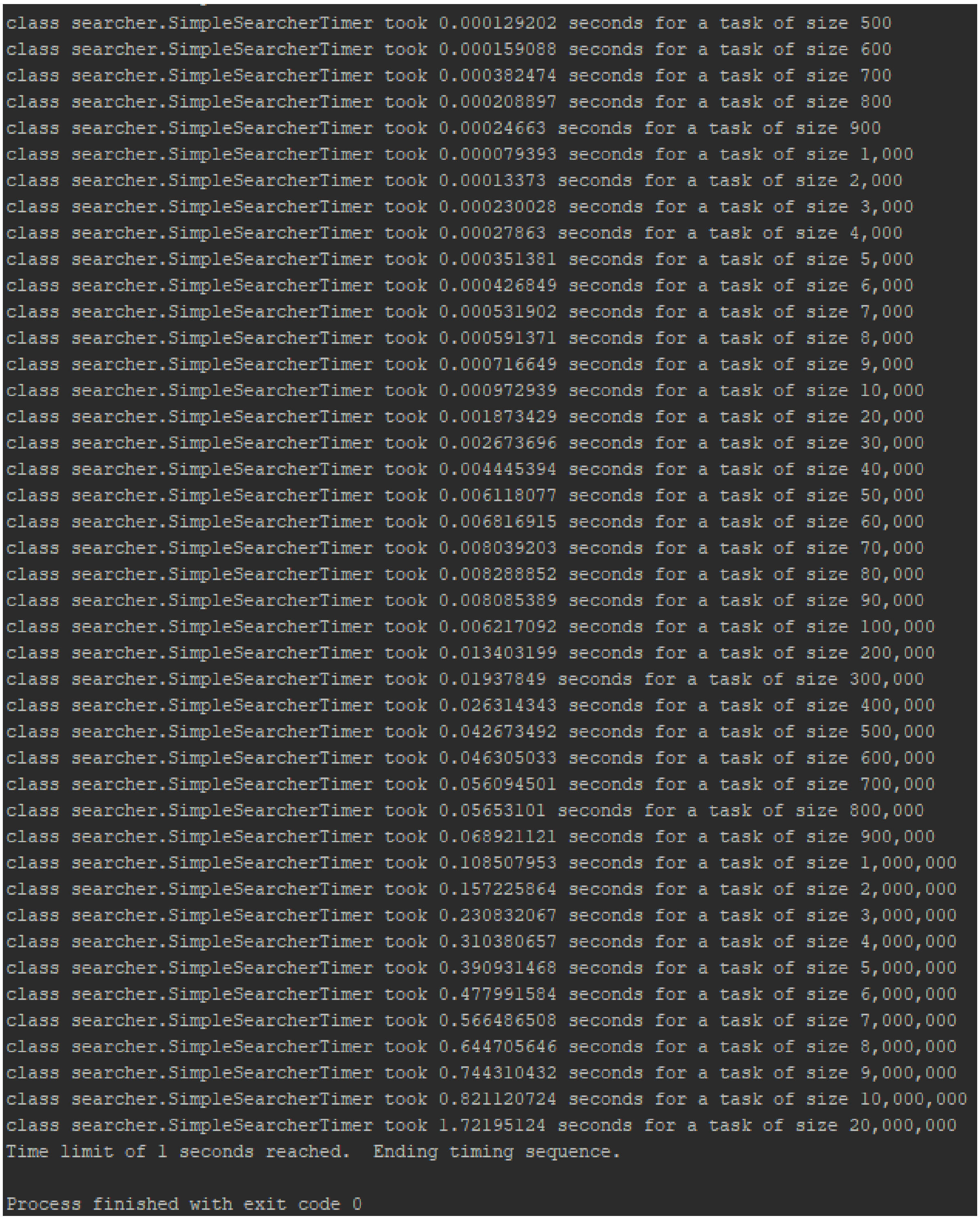
Task 3

```java
package searcher;

/*
  A timer method that records the execution time for the Clever Searcher

  @author Adam Birch
 * @version October 2018
 */

import arrayGenerator.ArrayGenerator;
import arrayGenerator.CleverRandomListingGenerator;
import timer.Timer;

public class CleverSearcherTimer extends CleverSearcher implements Timer {

    // All timings will be done with an index of 5
    private final static int K = 5;

    private CleverSearcherTimer(int[] array) {
        super(array, K);
    }

    @Override
    public void timedMethod() {
        try {
            findElement();
        } catch (IndexingError indexingError) {
            // simply ignore indexing errors here
            // with K at 5, and a minimum task size (array size) of 10, indexing errors
                                                                       should not occur
            // duirng timing
        }
    }

    @Override
    public long getMaximumRuntime() {
        return 1;
    }

    /**
     * Minimum task size (array size) is set to ten, to avoid indexing errors (index is
                                                                             always five)
     * @return minimum task size of ten
     */
    @Override
    public int getMinimumTaskSize() {
        return 10;
    }

    @Override
    public int getMaximumTaskSize() {
        return 100000000;
    }

    @Override
    public Timer getTimer(int size) {
        ArrayGenerator generator = new CleverRandomListingGenerator(size);
        return new CleverSearcherTimer(generator.getArray());
    }

    public static void main(String[] args) throws IndexingError {
        CleverSearcherTimer timer = new CleverSearcherTimer(null);
        timer.timingSequence();
    }
}
```

My CleverSearcher implementation is much faster than the SimpleSearcher. The CleverSearcher runs out of tests before terminating, while the SimpleSearcher runs out of time.

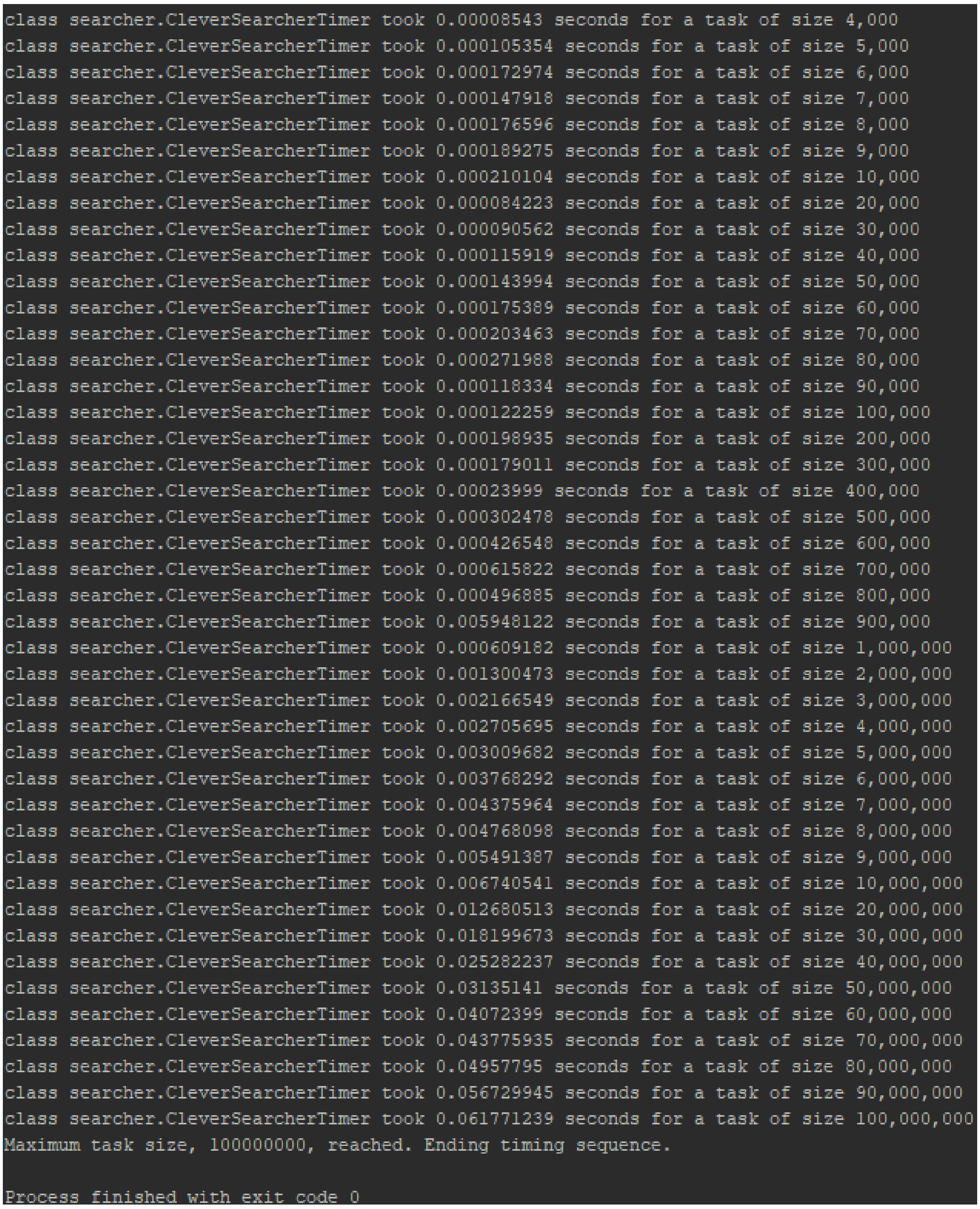
Here are console outputs to give exact comparisons:

SimpleSearcher



```
class searcher.SimpleSearcherTimer took 0.000129202 seconds for a task of size 500
class searcher.SimpleSearcherTimer took 0.000159088 seconds for a task of size 600
class searcher.SimpleSearcherTimer took 0.000382474 seconds for a task of size 700
class searcher.SimpleSearcherTimer took 0.000208897 seconds for a task of size 800
class searcher.SimpleSearcherTimer took 0.00024663 seconds for a task of size 900
class searcher.SimpleSearcherTimer took 0.000079393 seconds for a task of size 1,000
class searcher.SimpleSearcherTimer took 0.00013373 seconds for a task of size 2,000
class searcher.SimpleSearcherTimer took 0.000230028 seconds for a task of size 3,000
class searcher.SimpleSearcherTimer took 0.00027863 seconds for a task of size 4,000
class searcher.SimpleSearcherTimer took 0.000351381 seconds for a task of size 5,000
class searcher.SimpleSearcherTimer took 0.000426849 seconds for a task of size 6,000
class searcher.SimpleSearcherTimer took 0.000531902 seconds for a task of size 7,000
class searcher.SimpleSearcherTimer took 0.000591371 seconds for a task of size 8,000
class searcher.SimpleSearcherTimer took 0.000716649 seconds for a task of size 9,000
class searcher.SimpleSearcherTimer took 0.000972939 seconds for a task of size 10,000
class searcher.SimpleSearcherTimer took 0.001873429 seconds for a task of size 20,000
class searcher.SimpleSearcherTimer took 0.002673696 seconds for a task of size 30,000
class searcher.SimpleSearcherTimer took 0.004445394 seconds for a task of size 40,000
class searcher.SimpleSearcherTimer took 0.006118077 seconds for a task of size 50,000
class searcher.SimpleSearcherTimer took 0.006816915 seconds for a task of size 60,000
class searcher.SimpleSearcherTimer took 0.008039203 seconds for a task of size 70,000
class searcher.SimpleSearcherTimer took 0.008288852 seconds for a task of size 80,000
class searcher.SimpleSearcherTimer took 0.008085389 seconds for a task of size 90,000
class searcher.SimpleSearcherTimer took 0.006217092 seconds for a task of size 100,000
class searcher.SimpleSearcherTimer took 0.013403199 seconds for a task of size 200,000
class searcher.SimpleSearcherTimer took 0.01937849 seconds for a task of size 300,000
class searcher.SimpleSearcherTimer took 0.026314343 seconds for a task of size 400,000
class searcher.SimpleSearcherTimer took 0.042673492 seconds for a task of size 500,000
class searcher.SimpleSearcherTimer took 0.046305033 seconds for a task of size 600,000
class searcher.SimpleSearcherTimer took 0.056094501 seconds for a task of size 700,000
class searcher.SimpleSearcherTimer took 0.05653101 seconds for a task of size 800,000
class searcher.SimpleSearcherTimer took 0.068921121 seconds for a task of size 900,000
class searcher.SimpleSearcherTimer took 0.108507953 seconds for a task of size 1,000,000
class searcher.SimpleSearcherTimer took 0.157225864 seconds for a task of size 2,000,000
class searcher.SimpleSearcherTimer took 0.230832067 seconds for a task of size 3,000,000
class searcher.SimpleSearcherTimer took 0.310380657 seconds for a task of size 4,000,000
class searcher.SimpleSearcherTimer took 0.390931468 seconds for a task of size 5,000,000
class searcher.SimpleSearcherTimer took 0.477991584 seconds for a task of size 6,000,000
class searcher.SimpleSearcherTimer took 0.566486508 seconds for a task of size 7,000,000
class searcher.SimpleSearcherTimer took 0.644705646 seconds for a task of size 8,000,000
class searcher.SimpleSearcherTimer took 0.744310432 seconds for a task of size 9,000,000
class searcher.SimpleSearcherTimer took 0.821120724 seconds for a task of size 10,000,000
class searcher.SimpleSearcherTimer took 1.72195124 seconds for a task of size 20,000,000
Time limit of 1 seconds reached.  Ending timing sequence.

Process finished with exit code 0
```

CleverSearcher



```
class searcher.CleverSearcherTimer took 0.00008543 seconds for a task of size 4,000
class searcher.CleverSearcherTimer took 0.000105354 seconds for a task of size 5,000
class searcher.CleverSearcherTimer took 0.000172974 seconds for a task of size 6,000
class searcher.CleverSearcherTimer took 0.000147918 seconds for a task of size 7,000
class searcher.CleverSearcherTimer took 0.000176596 seconds for a task of size 8,000
class searcher.CleverSearcherTimer took 0.000189275 seconds for a task of size 9,000
class searcher.CleverSearcherTimer took 0.000210104 seconds for a task of size 10,000
class searcher.CleverSearcherTimer took 0.000084223 seconds for a task of size 20,000
class searcher.CleverSearcherTimer took 0.000090562 seconds for a task of size 30,000
class searcher.CleverSearcherTimer took 0.000115919 seconds for a task of size 40,000
class searcher.CleverSearcherTimer took 0.000143994 seconds for a task of size 50,000
class searcher.CleverSearcherTimer took 0.000175389 seconds for a task of size 60,000
class searcher.CleverSearcherTimer took 0.000203463 seconds for a task of size 70,000
class searcher.CleverSearcherTimer took 0.000271988 seconds for a task of size 80,000
class searcher.CleverSearcherTimer took 0.000118334 seconds for a task of size 90,000
class searcher.CleverSearcherTimer took 0.000122259 seconds for a task of size 100,000
class searcher.CleverSearcherTimer took 0.000198935 seconds for a task of size 200,000
class searcher.CleverSearcherTimer took 0.000179011 seconds for a task of size 300,000
class searcher.CleverSearcherTimer took 0.00023999 seconds for a task of size 400,000
class searcher.CleverSearcherTimer took 0.000302478 seconds for a task of size 500,000
class searcher.CleverSearcherTimer took 0.000426548 seconds for a task of size 600,000
class searcher.CleverSearcherTimer took 0.000615822 seconds for a task of size 700,000
class searcher.CleverSearcherTimer took 0.000496885 seconds for a task of size 800,000
class searcher.CleverSearcherTimer took 0.005948122 seconds for a task of size 900,000
class searcher.CleverSearcherTimer took 0.000609182 seconds for a task of size 1,000,000
class searcher.CleverSearcherTimer took 0.001300473 seconds for a task of size 2,000,000
class searcher.CleverSearcherTimer took 0.002166549 seconds for a task of size 3,000,000
class searcher.CleverSearcherTimer took 0.002705695 seconds for a task of size 4,000,000
class searcher.CleverSearcherTimer took 0.003009682 seconds for a task of size 5,000,000
class searcher.CleverSearcherTimer took 0.003768292 seconds for a task of size 6,000,000
class searcher.CleverSearcherTimer took 0.004375964 seconds for a task of size 7,000,000
class searcher.CleverSearcherTimer took 0.004768098 seconds for a task of size 8,000,000
class searcher.CleverSearcherTimer took 0.005491387 seconds for a task of size 9,000,000
class searcher.CleverSearcherTimer took 0.006740541 seconds for a task of size 10,000,000
class searcher.CleverSearcherTimer took 0.012680513 seconds for a task of size 20,000,000
class searcher.CleverSearcherTimer took 0.018199673 seconds for a task of size 30,000,000
class searcher.CleverSearcherTimer took 0.025282237 seconds for a task of size 40,000,000
class searcher.CleverSearcherTimer took 0.03135141 seconds for a task of size 50,000,000
class searcher.CleverSearcherTimer took 0.04072399 seconds for a task of size 60,000,000
class searcher.CleverSearcherTimer took 0.043775935 seconds for a task of size 70,000,000
class searcher.CleverSearcherTimer took 0.04957795 seconds for a task of size 80,000,000
class searcher.CleverSearcherTimer took 0.056729945 seconds for a task of size 90,000,000
class searcher.CleverSearcherTimer took 0.061771239 seconds for a task of size 100,000,000
Maximum task size, 100000000, reached. Ending timing sequence.

Process finished with exit code 0
```

The highest successful value on SimpleSearcher – 20,000 – took 1.72 seconds to sort, while the CleverSearcher required 0.012 seconds for the same calculation.

Note. There is a possibility of inaccuracies within the test as I cannot guarantee what my machine was doing in the background. Background processes would alter the results – however the general trend would remain the same.

Self-Assessment

Rating - 4

I would give myself a 4/5 for this work as it does exactly what is specified including suitable annotations, however it may not be the most efficient solution to the problem. A more efficient solution would include elements of both solutions, as finding the smallest in the initial array – such that k = array size – would be faster to sort than creating a new array. It would also require less memory as there would only be the one array of data, rather than two of the same.

Week 3

Task 1

```java
import java.util.ArrayList;

/**
 * @author Adam Birch
 * @version October 2018

 */


public class Swap<E> {

    public ArrayList<E> Switch (ArrayList <E> array, int index1, int index2) throws
                                                                IndexOutOfBoundsException{
        try {
            E temp1 = array.get(index1);
            E temp2 = array.get(index2);
            // Find the element at the two indexes
            array.set(index1, temp2);
            array.set(index2, temp1);
            //Swap the elements
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Exception");
            throw new IndexOutOfBoundsException("Index exceeds the bounds of the array. ");
        }

        return array;
    }
}
```

This is the generic function to swap the values.

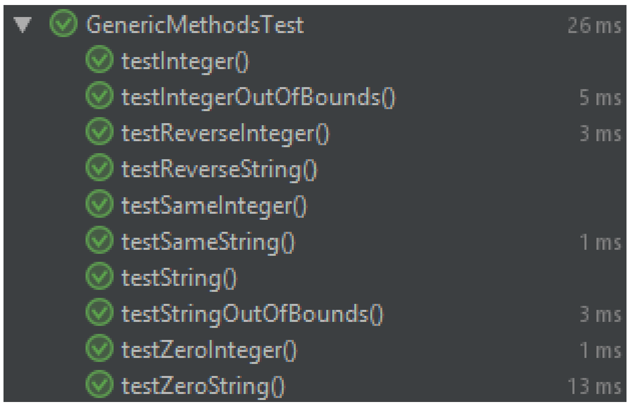
```java
import java.util.ArrayList;

/**
 * @author Adam Birch
 * @version October 2018
 */

public class GenericMethods {

    public ArrayList<String> strArray(Integer end, int index1, int index2) {
        //Creating a String array and using the Swap
        if(end >0){
            ArrayList<String> list = new ArrayList<>(end);
            for (int i = 0; i < end; i++){
                list.add(Integer.toString(i));
            }
            Swap<String> swap = new Swap<>();
            list = swap.Switch(list, index1, index2);
            return list;
        }
        else {
            throw new IndexOutOfBoundsException("Array size must be higher than 0.");
        }
    }

    public ArrayList<Integer> intArray(Integer end, int index1, int index2) {
        //Creating an Integer array and using the Swap
        if(end >0){
            ArrayList<Integer> list = new ArrayList<>(end);
            for (int i = 0; i < end; i++) {
                list.add(i);
            }
            Swap<Integer> swap = new Swap<>();
            list = swap.Switch(list, index1, index2);
            return list;
        }
        else {
            throw new IndexOutOfBoundsException("Array size must be higher than 0.");
        }
    }
}
```

Here is a GenericMethods class that calls the swap and supports integers and strings. This is not necessary for the swap function to work but is used to create an instance of the swap for testing.

```java
/*
 * @author Adam_Birch
 *                   u1761249
 */

import org.junit.jupiter.api.Test;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;

class GenericMethodsTest {


    void TestValidString(Integer end, int index1, int index2) {
        GenericMethods test = new GenericMethods();
        ArrayList expected = new ArrayList<>(end);
        for (int i = 0; i < end; i++){
            if(i==index1){expected.add(index2);}
            else if (i == index2){expected.add(index1);}
            else {expected.add(i);}
        }
        ArrayList data = test.strArray(end,index1,index2);
        for (int i = 0; i < 10; i++){
        assertEquals(expected.get(i).toString(),data.get(i).toString());
        }
    }


    void TestValidInteger(Integer end, int index1, int index2) {
        GenericMethods test = new GenericMethods();
        ArrayList expected = new ArrayList<>(end);
        for (int i = 0; i < end; i++){
            expected.add(i);
        }
        Object temp1 = expected.get(index1);
        Object temp2 = expected.get(index2);
        expected.set(index1,temp2);
        expected.set(index2,temp1);
        assertEquals(expected, test.intArray(end,index1,index2));
    }

    void TestStringException(Integer end, int index1, int index2){
        GenericMethods test = new GenericMethods();
        Throwable e = assertThrows(IndexOutOfBoundsException.class, () ->
            {test.strArray(end,index1,index2);});
        assertEquals("Index exceeds the bounds of the array. ", e.getMessage());
    }
    void TestIntegerException(Integer end, int index1, int index2){
        GenericMethods test = new GenericMethods();
        Throwable e = assertThrows(IndexOutOfBoundsException.class, () ->
            {test.intArray(end,index1,index2);});
        assertEquals("Index exceeds the bounds of the array. ", e.getMessage());
    }
    void TestStringSizeException(Integer end, int index1, int index2){
        GenericMethods test = new GenericMethods();
        Throwable e = assertThrows(IndexOutOfBoundsException.class, () ->
            {test.strArray(end,index1,index2);});
        assertEquals("Array size must be higher than 0.", e.getMessage());
    }
    void TestIntegerSizeException(Integer end, int index1, int index2){
        GenericMethods test = new GenericMethods();
        Throwable e = assertThrows(IndexOutOfBoundsException.class, () ->
            {test.intArray(end,index1,index2);});
        assertEquals("Array size must be higher than 0.", e.getMessage());
    }

    //Test that the program functions correctly
    @Test
    void testString(){
        TestValidString(10,2,7);
    }

    @Test
    void testInteger(){
        TestValidInteger(10,2,7);
    }

    //Test that the function works when the numbers are in reverse order

    @Test
    void testReverseString(){
        TestValidString(10,7,2);
    }

    @Test
    void testReverseInteger(){
        TestValidInteger(10,7,2);
    }

    //Test that the program makes no change and throws no error

    @Test
    void testSameString(){
        TestValidString(10,5,5);
    }

    @Test
    void testSameInteger(){
        TestValidInteger(10,5,5);
    }

    //Test that the correct exception is thrown

    @Test
    /**
     * @throws IndexOutOfBoundsException
     */
    void testStringOutOfBounds(){
        TestStringException(10,2,70);
    }

    @Test
    /**
     * @throws IndexOutOfBoundsException
     */
    void testIntegerOutOfBounds(){
        TestIntegerException(10,-2,7);
    }

    //Test that an exception is thrown if the input is 0.

    @Test
    /**
     * @throws IndexOutOfBoundsException
     */
    void testZeroString(){
        TestStringSizeException(0,0,0);
    }
    @Test
    /**
     * @throws IndexOutOfBoundsException
     */
    void testZeroInteger(){
        TestIntegerSizeException(0,0,0);
    }

}
```

This is a test class that was used to test the implementation. I used integer and string values to test the swap class as these are the most likely values to be swapped within a program. As the implementation uses generics it should also work with objects and any other non-primitive type.

The following screenshot is the test results for swapping Integers and Strings.



```
GenericMethodsTest              26 ms
    testInteger()
    testIntegerOutOfBounds()     5 ms
    testReverseInteger()         3 ms
    testReverseString()
    testSameInteger()
    testSameString()             1 ms
    testString()
    testStringOutOfBounds()      3 ms
    testZeroInteger()            1 ms
    testZeroString()            13 ms
```

Re-writing week 1&2 as Generics

I successfully modified the code such that the Simple Searcher and all required components run generic methods, however the Clever Searcher returns exceptions due to casting the type to allow the program to compile.

Modifying the Search algorithms to use Generics required the entire folder to be altered to use generics. This was problematic as I don't fully understand each java class however I was able to alter the required classes to make the search compile and run correctly.

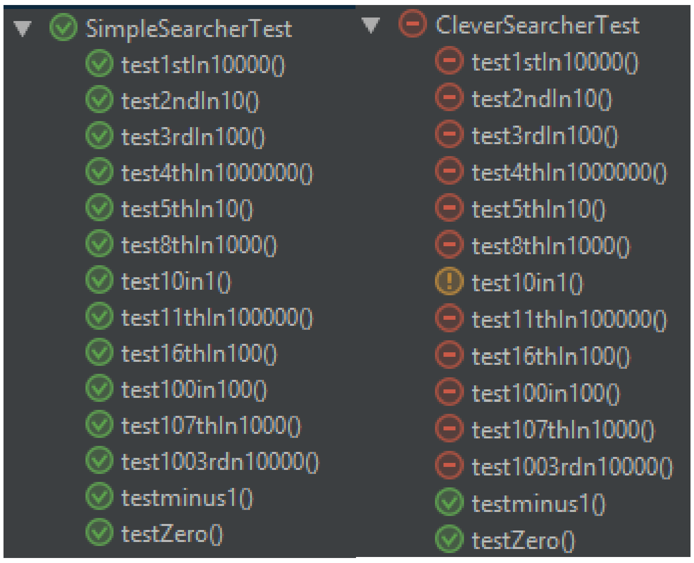
This is the generic Simple Searcher.

```java
package searcher;

/*
  Implements the find (kth) element method by sorting and indexing.
 */

import java.util.Arrays;

public class SimpleSearcher<T> extends Searcher {

    SimpleSearcher(T[] array, int k) {
        super(array, k);
    }

    /**
     * Find the kth largest element in an array of ints using the "obvious"
     * solution from the lecture
     *
     * <ul>
     *     <li> Sort the array</li>
     *     <li> Return the entry k spaces from the end</li>
     * </ul>
     *
     * @return kth largest element of array
     */
    public T findElement() throws IndexingError {
        T[] array = (T[]) getArray();
        int k = getIndex();
        if (k <= 0 || k > array.length) {
            throw new IndexingError();
        }
        Arrays.sort(array); // sort the whole array
        return array[array.length - k]; // desired element is kth from the end
    } // end of obvious solution method
}
```

This is the generic Clever Searcher.

```
package searcher;

/**
 * @author Adam Birch
 * @version October 2018

 */

public class CleverSearcher<T extends Comparable<? super T>> extends Searcher {

    CleverSearcher(Object[] array, int k) {
        super(array, k);
    }


    @Override
    public T findElement() throws IndexingError {
        /**
         * This is the main search to be implemented
         * @return kth largest element
         * @exception IndexingError on bad input
         * @see IndexingError
         */
        T[] array = (T[]) getArray();
        int k = getIndex();
        if (k <= 0 || k > array.length) { // Test that k is a valid search.
            throw new IndexingError();
        }
        T[] smallArray = (T[])new Object[k];  // Create a small array of size k - the
                                                     k'th largest is element 0
        for (Integer i = 0; i < array.length; i++) {
            T value = array[i];
            T temp = smallArray[0];
            if (value.compareTo(temp) > 0){
                if (smallArray.length > 1){ // Only loop if there are many values in the Array.
                    int x = 0;
                    while (x < smallArray.length - 1 && value.compareTo(smallArray[x+1]) > 0){
                        smallArray[x] = smallArray[x+1];
                        x++;
                    }
                    smallArray[x] = value; // Adds the new value to the Array once it is in
                                                                          the correct place.

                }
                else {
                    smallArray[0] = value;
                }
            }
        }
        return smallArray[0];
    }
}
```

This is the successful running for the Simple Searcher (left) and the error for the Clever Searcher(right):



✓ SimpleSearcherTest	⊖ CleverSearcherTest
✓ test1stIn10000()	⊖ test1stIn10000()
✓ test2ndIn10()	⊖ test2ndIn10()
✓ test3rdIn100()	⊖ test3rdIn100()
✓ test4thIn1000000()	⊖ test4thIn1000000()
✓ test5thIn10()	⊖ test5thIn10()
✓ test8thIn1000()	⊖ test8thIn1000()
✓ test10in1()	! test10in1()
✓ test11thIn100000()	⊖ test11thIn100000()
✓ test16thIn100()	⊖ test16thIn100()
✓ test100in100()	⊖ test100in100()
✓ test107thIn1000()	⊖ test107thIn1000()
✓ test1003rdn10000()	⊖ test1003rdn10000()
✓ testminus1()	✓ testminus1()
✓ testZero()	✓ testZero()

The failed tests were because:

"java.lang.ClassCastException: class [Ljava.lang.Object; cannot be cast to class [Ljava.lang.Comparable; ([Ljava.lang.Object; and [Ljava.lang.Comparable; are in module java.base of loader 'bootstrap')"

Self-Assessment

Rating - 5

I feel like I understand the utilisation and implementation of Generics, even though I was unable to make the rewritten Clever Search algorithm function correctly. I was able to implement fully automated testing within the logbook exercises and have documented my code and testing to a reasonable level.

Week 5

Task 3.4

I ran tests for the Bubble, Selection and Quick sorts and recorded the time taken for an array size of: 1, 12, 1000, 2000, 10000, and 50000.

I ran this three times and used the averages to plot a graph. They are as follows (Time is recorded in milliseconds):

Test 1

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0	1	1	6	46
SelectionSort	0	0	2	4	78	1910
BubbbleSort	0	0	3	21	368	9522

Test 2

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	1	1	2	10	41
SelectionSort	0	0	2	5	84	1910
BubbbleSort	1	0	6	15	365	9651

Test 3

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	1	2	3	12	63
SelectionSort	1	0	2	6	80	1919
BubbbleSort	0	0	4	13	358	9725

Average

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0.6666667	1.333333	2	9.333333	50
SelectionSort	0.333333	0	2	5	80.66667	1913
BubbbleSort	0.333333	0	4.333333	16.33333	363.6667	9632.667

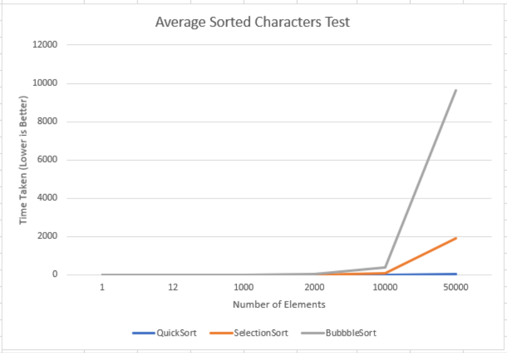


Figure 1 - Average Sorted Characters Test

Test 1

Sort Type	1	12	1000	2000	10000	50000
QuickSort	1	1	3	7	133	4322
SelectionSort	1	1	1	11	244	6510
BubbbleSort	1	0	6	21	543	15037

Test 2

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0	4	8	126	4350
SelectionSort	0	0	3	10	248	6527
BubbbleSort	0	0	5	19	546	14891

Test 3

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	1	4	8	133	4350
SelectionSort	0	1	4	12	246	6551
BubbbleSort	0	0	4	13	358	9725

Average

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0.333333	0.666667	3.666667	7.666667	130.6667	4340.667
SelectionSort	0.333333	0.666667	2.666667	11	246	6529.333
BubbbleSort	0.333333	0	5	17.66667	482.3333	13217.67

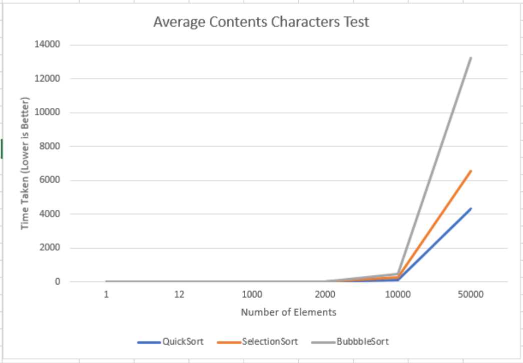


Figure 2 - Average Contents Characters Test

Test 1

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0	2	12	676	10750
SelectionSort	0	0	3	13	350	11114
BubbbleSort	0	0	4	15	687	11904

Test 2

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0	3	14	349	11326
SelectionSort	0	0	3	13	347	10962
BubbbleSort	0	1	4	14	347	11914

Test 3

Sort Type	1	12	1000	2000	10000	50000
QuickSort	1	0	2	10	339	11147
SelectionSort	0	0	2	11	362	10832
BubbbleSort	0	0	3	17	389	11949

Average

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0.333333	0	2.333333	12	454.6667	11074.33
SelectionSort	0	0	2.666667	12.33333	353	10969.33
BubbbleSort	0	0.333333	3.666667	15.33333	474.3333	11922.33

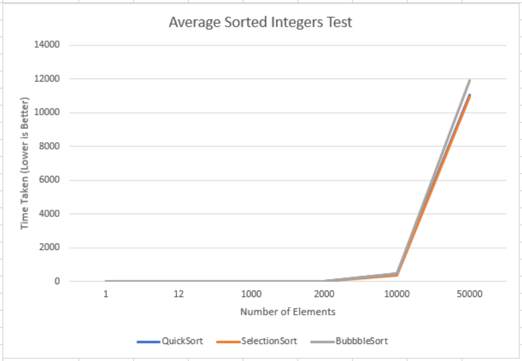
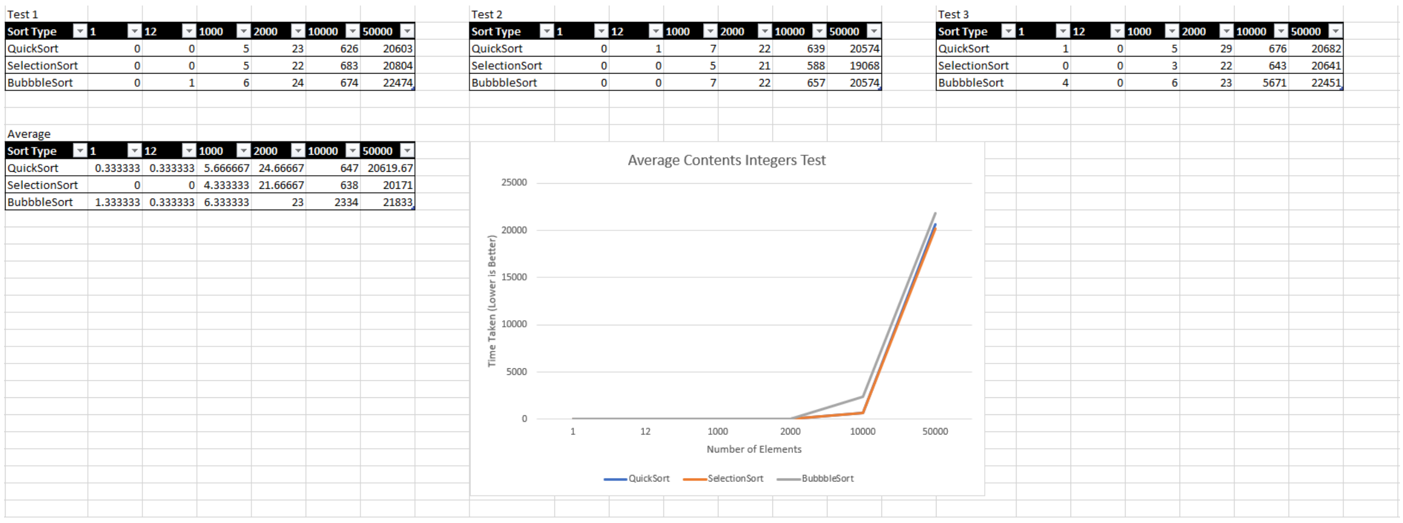


Figure 3 - Average Sorted Integers Test



Test 1

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	0	5	23	626	20603
SelectionSort	0	0	5	22	683	20804
BubbbleSort	0	1	6	24	674	22474

Test 2

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0	1	7	22	639	20574
SelectionSort	0	0	5	21	588	19068
BubbbleSort	0	0	7	22	657	20574

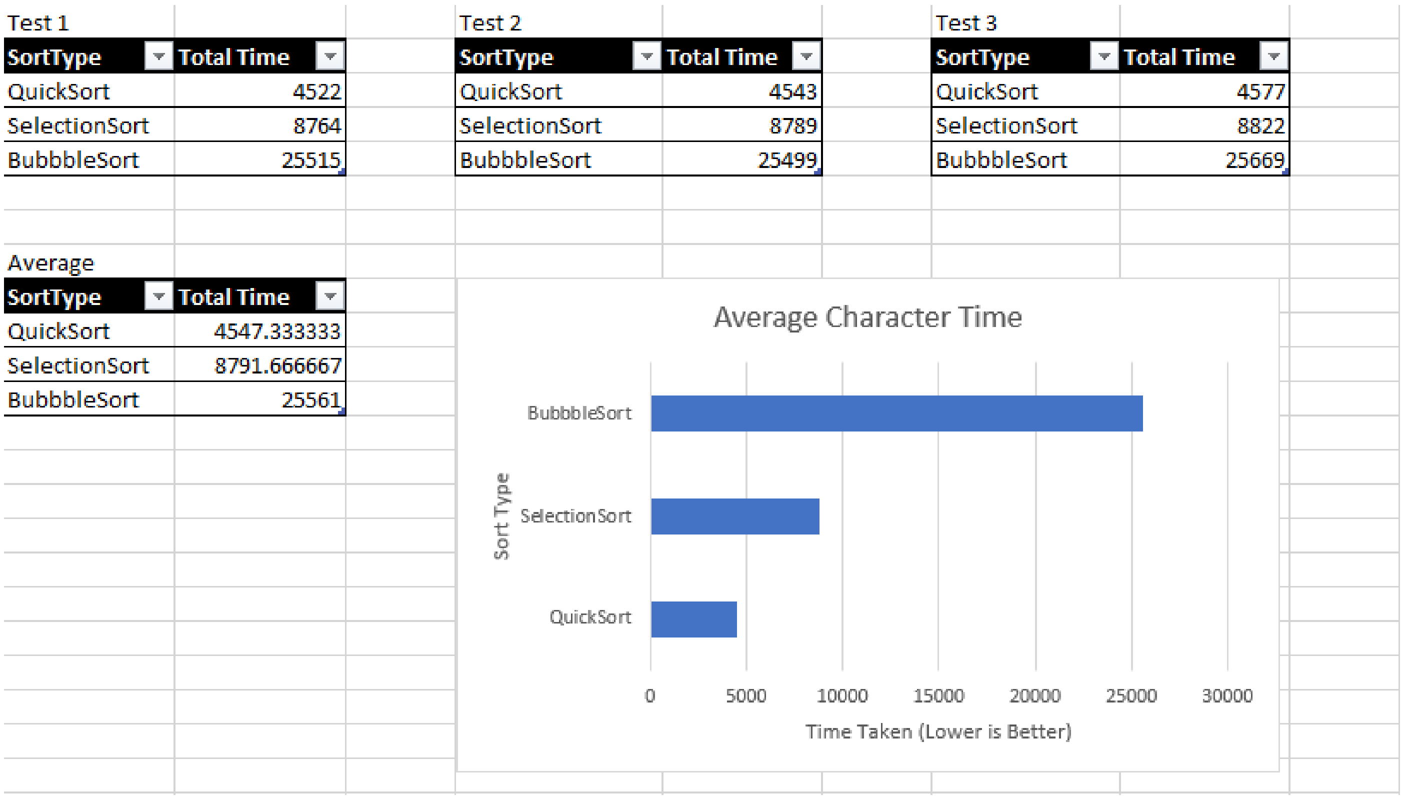
Test 3

Sort Type	1	12	1000	2000	10000	50000
QuickSort	1	0	5	29	676	20682
SelectionSort	0	0	3	22	643	20641
BubbbleSort	4	0	6	23	5671	22451

Average

Sort Type	1	12	1000	2000	10000	50000
QuickSort	0.333333	0.333333	5.666667	24.66667	647	20619.67
SelectionSort	0	0	4.333333	21.66667	638	20171
BubbbleSort	1.333333	0.333333	6.333333	23	2334	21833

Figure 4 - Average Contents Integers Test



Test 1

SortType	Total Time
QuickSort	4522
SelectionSort	8764
BubbbleSort	25515

Test 2

SortType	Total Time
QuickSort	4543
SelectionSort	8789
BubbbleSort	25499

Test 3

SortType	Total Time
QuickSort	4577
SelectionSort	8822
BubbbleSort	25669

Average

SortType	Total Time
QuickSort	4547.333333
SelectionSort	8791.666667
BubbbleSort	25561

Figure 5 - Average Character Time

Test 1

SortType	Total Time
QuickSort	32994
SelectionSort	32397
BubbbleSort	35489

Test 2

SortType	Total Time
QuickSort	31387
SelectionSort	31007
BubbbleSort	33540

Test 3

SortType	Total Time
QuickSort	32892
SelectionSort	32519
BubbbleSort	35513

Average

SortType	Total Time
QuickSort	32424.33333
SelectionSort	31974.33333
BubbbleSort	34847.33333

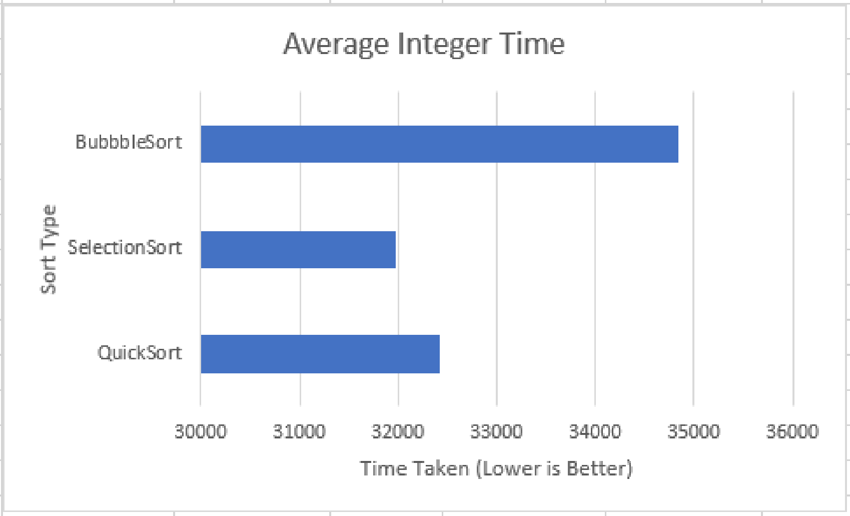


Figure 6 - Average Integer Time

From these results, I can conclude that Bubble Sort is always the slowest regardless of data types. I can also conclude that the data type does make a difference to processing time and makes Selection Sort faster than Quick Sort for Integers. The code for the Selection Sort and the Quick Sort can be found at:

https://stackoverflow.com/questions/12128066/selection-sort-with-generics
and
https://github.com/chrisloy/java-algorithms/blob/master/src/uk/co/chrisloy/sandpit/sort/QuickSort.java

respectively.

Selection Sort may be faster for Integers than Quick Sort as there are less possibilities (There are 10 numerical digits used while there are 26 characters used). This difference may make a large impact on the efficiency of the solutions.

Big O Notation is an algebraic expression for the efficiency of a solution. O_n means the code will take n cycles for n amount of data, O_n^2 will take n^2 for n amount of data, etc.

The Average case Big O Notations for Bubble, Selection, and Quick Sorts are O_n^2, O_n^2, and O(n log(n)) respectively.

This is calculated by the Bubble and Selection Sorts both having nested loops within, where the number of loops varies due to the number of elements in the array(n).

Self-Assessment

Rating - 5

I would give myself 5/5 for this weeks exercises as I successfully implemented additional sorting algorithms to the code, and I have successfully tested and documented however I am unsure as to how the Selection Sort is better than the Quick Sort for sorting a large quantity of integers. It behaved as I would expect for characters but not for Integers. Due to this I feel like I am unable to award myself 5 as I cannot explain the results effectively. I have attempted to draw upon previous teaching of Big O Notation to express the efficiency of the sorting algorithms. I implemented larger data within my testing data sets, though I couldn't use larger than I did due to running time on my machine. I have shown the difference between each sort in a controlled environment using standard scientific practices to ensure data is more consistent – this being repeating the test and using the average results.