**Logbook**

**Bradley Pratt - Computer Games Programming U1664020314**

**Algorithms Processes and Data**

**Week 1-2 :**

**package** intArrays;

**import** java.util.Arrays;

**public** **class** CleverRandomListing **extends** RandomListing {

**public** CleverRandomListing (**int** size) {

**super**(size);

}

/\*\*

\* The purpose of this method is to rebuild an array in a completely random order

\* Passes an array in from the SortedListing class.

\*/

**protected** **void** randomise() {

**for** (**int** index = 0; index < getArray().length; index++) {

**int** randomArray = getRandomIndex(); // Uses the getRandomIndex method to randomise the array index

**int** newInt = getArray()[randomArray];

getArray()[randomArray] = getArray()[index]; // Changes the grabbed array to randomise its index

getArray()[index] = newInt; // Builds the array using its new index

}

}

**public** **static** **void** main(String[] args) {

RandomListing count = **new** CleverRandomListing(50); // create a new list, as long as the specified length.

System.***out***.println(Arrays.*toString*(count.getArray())); // prints the array to the console

}

}

The tests for this class showed that is more efficient than the standard sorting class, with a testMillionSize taking 96423 milliseconds. In SimpleRandomTesting testMillionSize took 261148 milliseconds in my last test; proving the above shown method is more efficient.

**Week 3-4**

/\*\*

\* Swaps the specified elements within the array

\* **@param** array the array which is passed into the method

\* **@param** index1 the index which needs to be swapped with index2

\* **@param** index2

\*/

**public** **static** <T> **void** swap(T[] array, **int** index1, **int** index2) {

T objectOne = array[index1];

T objectTwo = array[index2];

array[index1] = objectTwo; //Uses the defined first position and places "objectTwo" there

array[index2] = objectOne; //Uses the defined second position and places "objectOne" there

}

/\*\*

\* The purpose of max is to find the largest element in between index1 and index2.

\* **@param** array is the array that is passed in

\* **@param** index1 is the first index, which elements before it may be ignored

\* **@param** index2 is the second index location, which elements after it may be ignored

\* **@return** returns the largest element

\*/

**public** **static** <T> String max(String[] array, **int** index1, **int** index2) {

**int** index = 0;

**int** elementLength = array[0].length();

System.***out***.println();

**for** (**int** i = 0; i < array.length; i++) {

**if** (i >= index1 && i <= index2) {

**if** (array[i].length() > elementLength) {

index = i;

elementLength = array[i].length();

}

}

}

**return** array[index];

}

**public** **static** void main(String[] args) {

String[] names = {"Hugh", "Andrew", "Ebrahim","Diane","Paula", "Simon"};

System.***out***.println(*max*(names,1,4));

}

}

The tests class for the swap method shows that the elements are successfully swapped, using the array 1,2,3,4,5 and adding index1 as 1, and index2 as 2 showed that the array became 1,3,2,4,5 as expected.

**Week 5**

(10,000) = 180

(20,000) = 890

(30,000) = 2150

(40,000) = 4000

(50,000) = 6500

I ran the test three times and above are the results I got, the formulas show the average result found between the three tests. A function couldn’t be found since there doesn’t seem to be a running trend that would allow you to predict the next result with reasonably accuracy.