//Assignment02

//Rocco Piccirillo

//SelectionSort

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** SelectionSort {

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting

//added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

*sort*(wordList);

*printArray*(wordList);

}

//created a method to take in the magicItems List

**public** **static** ArrayList<String> sort(ArrayList<String> A)

{

//created an int to store number of swaps

**int** numSwap = 0;

//loops over the arrayList

**for**(**int** i = 0; i < A.size()-1; i++)

{

numSwap++;

//sets smallPos as i or the initial index

**int** smallPos = i;

//loop over each string in the arrayList

**for**(**int** j = i+1; j < A.size(); j++)

{

numSwap++;

//compare to returns a positive num or negative num

//if j is less than smallPos, a negative number prints

//if j is greater than smallPos, a positive number prints

//so if the result is negative, it is less than 0 so we //swap

**if**(A.get(j).compareTo(A.get(smallPos)) < 0)

{

//the small position gets swapped with j

smallPos = j;

//the amount of swaps increases as well

numSwap++;

}

}

//the temp string is the smallest position

String temp = A.get(smallPos);

//the smallest position of magicItems gets set to i

A.set(smallPos, A.get(i));

//and then i gets set to temp

A.set(i, temp);

}

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

**return** A;

}

//this makes an easily accessible printing method for the wordList

**public** **static** **void** printArray(ArrayList<String> wordList)

{

**for**(**int** i = 0; i < wordList.size(); i++)

{

System.***out***.println(wordList.get(i));

}

}

}

//Assignment02

//Rocco Piccirillo

//InsertionSort

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** InsertionSort

{

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting //added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

*sort*(wordList);

*printArray*(wordList);

}

**public** **static** ArrayList<String> sort(ArrayList<String> A)

{

//created an int to store number of swaps

**int** numSwap = 0;

//loops over the arrayList

**for**(**int** i = 1; i < A.size(); i++)

{

//key is set to the current position of i

String key = A.get(i);

//j is the point before key

**int** j = i-1;

//makes sure that j is in index 0 or greater

//and that j is > key before initiating this loop

**while**(j >= 0 && A.get(j).compareTo(key) > 0 )

{

//this then sets the index value of the greater word to +1

//so that we will be kicked out of the while loop

A.set(j+1, A.get(j));

j--;

//increments the swap count

numSwap++;

}

//so since we changed, the key now moves to the next string

A.set(j+1, key);

}

System.***out***.println(numSwap + " comparisons performed.");

**return** A;

}

//this makes an easily accessible printing method for the wordList

**public** **static** **void** printArray(ArrayList<String> wordList)

{

**for**(**int** i = 0; i < wordList.size(); i++)

{

System.***out***.println(wordList.get(i));

}

}

}

//Assignment02

//Rocco Piccirillo

//MergeSort

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** MergeSort

{

**public** **static** **int** *compare* = 0;

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting //added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

wordList = *mergeSort*(wordList);

*printArray*(wordList);

System.***out***.print(*compare* + " is the number of comparisons");

}

**public** **static** ArrayList<String> mergeSort(ArrayList<String> A)

{

//if the size of the given array is less than 1

//we just return

**if**(A.size() <= 1)

{

**return** A;

}

//first we get the midpoint of the array

**int** midpoint = A.size() / 2;

//creating a subarray called left or the left half

ArrayList<String> left = **new** ArrayList<String>();

//and another sub array for the rightmost half

ArrayList<String> right = **new** ArrayList<String>();

//the last array which will be the product of

//the merged left and right array

ArrayList<String> result = **new** ArrayList<String>();

//we are going to traverse from 0 to midpoint

//and add until we reach there

**for**(**int** i = 0; i < midpoint; i++)

{

left.add(A.get(i));

}

//we are going to start at midpoint

//and add until we reach the end of the arrayList

**for**(**int** j = midpoint; j < A.size(); j++)

{

right.add((A.get(j)));

}

//we are going to call the method again to

//split even further until reduced to multiple

//arrays of 1 and 1 and 1

//the right is going through the same thing

left = *mergeSort*(left);

right = *mergeSort*(right);

//once finished divided into singled out elements

//we are going to merge

result = *merge*(left, right);

//System.out.println(numSwap + " number of comparisons.");

**return** result;

}

**public** **static** ArrayList<String> merge(ArrayList<String> left, ArrayList<String> right)

{

//this is the merged arrayList containing the elements

//of both the arrays

ArrayList<String> result = **new** ArrayList<String>();

//created indexes for both to start at

**int** indexL = 0;

**int** indexR = 0;

//while elements are still in the left or in the right

**while**(indexL < left.size() || indexR < right.size())

{

*compare*++;

//if the left AND the right still have elements

//they need to be merged in the right order

**if**(indexL < left.size() && indexR < right.size())

{

*compare*++;

//depending on which is larger we need to know

//which to add first

//if the left is smaller than the right

**if**(left.get(indexL).compareTo(right.get(indexR)) < 0)

{

*compare*++;

//we add left first if it is smaller

result.add(left.get(indexL));

//we need to also update the index

indexL++;

}

//if the right is bigger, we add the right after

**else**

//if(right.get(indexR).compareTo(left.get(indexL)) > 0)

{

*compare*++;

result.add(right.get(indexR));

indexR++;

}

}

//if there are still elements in the left

//but not the right

**else** **if** (indexL < left.size())

{

*compare*++;

result.add(left.get(indexL));

indexL++;

//or, if the left is the empty but the right isn't

//the right needs to be added

} **else** **if** (indexR < right.size())

{

*compare*++;

result.add(right.get(indexR));

indexR++;

}

}

**return** result;

}

//this makes an easily accessible printing method for the wordList

**public** **static** **void** printArray(ArrayList<String> wordList)

{

**for**(**int** i = 0; i < wordList.size(); i++)

{

System.***out***.println(wordList.get(i));

}

}

}

//Assignment02

//Rocco Piccirillo

//QuickSort

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** QuickSort

{

**public** **static** **int** *compare* = 0;

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting //added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

*QuickSort*(wordList, 0, wordList.size() - 1);

*printArray*(wordList);

System.***out***.print(*compare* + " is the number of comparisons.");

}

**private** **static** **void** QuickSort(ArrayList<String> array, **int** left, **int** right)

{

**int** index = *partition*(array, left, right);

//if left is smaller than the index value

**if**(left < index -1)

{

//then call the sort method for left

*QuickSort*(array, left, index - 1);

//if right is bigger than the index value

**if**(index < right)

{

//then we call the quickSort method from that right value

*QuickSort*(array, index, right);

}

}

}

**private** **static** **int** partition(ArrayList<String> array, **int** left, **int** right)

{

//this string is the midpoint of the array

String pivot = array.get((left + right) / 2);

//while the left value is less than the right value

**while**(left<= right)

{

*compare*++;

//while the left value is lower than the pivot point

//increment the left value

**while**(array.get(left).compareTo(pivot) < 0)

{

*compare*++;

left++;

}

//while the right value is greater than the pivot point

//decrement the right value

**while**(array.get(right).compareTo(pivot) > 0)

{

*compare*++;

right--;

}

//if the left is less than the right

**if**(left <= right)

{

*compare*++;

//the value will be swapped

//the value first gets stored in a temp variable

String temp = array.get(left);

//set left string equal to the right string

array.set(left, array.get(right));

//and lastly make temps new value, the right string

array.set(right, temp);

//after moving on we move forward with the left

//and move closer with the right

left++;

right--;

}

}

**return** left;

}

//this makes an easily accessible printing method for the wordList

**public** **static** **void** printArray(ArrayList<String> wordList)

{

**for**(**int** i = 0; i < wordList.size(); i++)

{

System.***out***.println(wordList.get(i));

}

}

}

//Assignment02

//Rocco Piccirillo

//LinearSearch

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** LinearSearch

{

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting //added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

//creates a new arrayList to store our random items

ArrayList<String> randList = **new** ArrayList<String>();

//while the size of the randList is less than 42

**while** (randList.size() < 42)

{

//a random Index of wordList will be selected

**int** randIndex = (**int**) (Math.*random*()\*(wordList.size()-1));

**if** (randList.indexOf(randIndex) == -1)

{

//the randList adds the random indexed number to the List

randList.add(wordList.get(randIndex));

}

}

//first we will be sorting the arrayList

//this is being done so that finding everything in the list won't be //incredibly skewed

*sort*(wordList);

//this list is being done so that we can sort the

//amount of comparisons being performed

ArrayList<Integer> comparisons = **new** ArrayList<Integer>();

//a basic counter being made for the while loop

**int** counter = 0;

//made so we can store the index of randList and

//end up incrementing it as we continue

**int** index = 0;

//this is so we know where we are in the magicItemsList

//this will be used for knowing how many comparisons have been performed

**int** pos = 0;

//while the counter has not yet reached 42 comparisons yet

**while**(counter < 42)

{

//if the current position of the wordList is equal to the

//current selected string from randList

**if**(wordList.get(pos).equals(randList.get(index)))

{

pos++;

//once found the position or number of comparisons will be //added

comparisons.add(pos);

//this is so we can keep track of what word is being //searched for

//and what position we found said word at

System.***out***.println(randList.get(index) + " found at " + pos);

//the position gets reset back to 0 for our next searched //word

pos = 0;

//the next word in the randList gets selected

index++;

//the counter also gets moved up as well so that

//once we get to 42 we exit

counter++;

//if not found, we move to the next word in the sorted wordList

} **else**

{

//if we don't find it at the given position,

//we increment to the next index

pos++;

}

}

//the average of comparisons is being printed here

System.***out***.print("The average is " + *average*(comparisons, comparisons.size()));

}

**public** **static** ArrayList<String> sort(ArrayList<String> A)

{

//loops over the arrayList

**for**(**int** i = 1; i < A.size(); i++)

{

//key is set to the current position of i

String key = A.get(i);

//j is the point before key

**int** j = i-1;

//makes sure that j is in index 0 or greater

//and that j is > key before initiating this loop

**while**(j >= 0 && A.get(j).compareTo(key) > 0 )

{

//this then sets the index value of the //greater word to +1

//so that we will be kicked out of the while //loop

A.set(j+1, A.get(j));

j--;

}

//so since we changed, the key now moves to the next //string

A.set(j+1, key);

}

//System.out.println(numSwap);

**return** A;

}

// Function that return average of an array.

**public** **static** **double** average(ArrayList<Integer> averageArray, **int** size)

{

// Find sum of array element

**int** sum = 0;

//for the the size of

**for** (**int** i = 0; i < size; i++)

{

//we will be adding our newest found item

//to our total sum at the time

sum += averageArray.get(i);

}

//returns our average aka our sum of ints / our arraySize

**return** sum / size;

}

}

//Assignment02

//Rocco Piccirillo

//BinarySearch

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**import** java.util.function.UnaryOperator;

**public** **class** BinarySearch

{

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting //added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//sets all strings in wordList to upperCase

UnaryOperator<String> upper = (x) -> x.toUpperCase();

wordList.replaceAll(upper);

//creates a new arrayList to store our random items

ArrayList<String> randList = **new** ArrayList<String>();

//while the size of the randList is less than 42

**while** (randList.size() < 42)

{

//a random Index of wordList will be selected

**int** randIndex = (**int**) (Math.*random*()\*(wordList.size()-1));

**if** (randList.indexOf(randIndex) == -1)

{

//the randList adds the random indexed number to the List

randList.add(wordList.get(randIndex));

}

}

//first we will be sorting the arrayList

//this is being done so that finding everything in the list won't be //incredibly skewed

*sort*(wordList);

//this list is being done so that we can sort the

//amount of comparisons being performed

ArrayList<Integer> comparisons = **new** ArrayList<Integer>();

//a basic counter being made for the while loop

**int** counter = 0;

//made so we can store the index of randList and

//end up incrementing it as we continue

**int** index = 0;

//while the counter has not yet reached 42 comparisons yet

**while**(counter < 42)

{

System.***out***.print(randList.get(index) + " was found after ");

//this lets us call our binarySearch

//using the wordList to search and picking

//a string from randList to be found

*binSearch*(wordList, randList.get(index), comparisons);

//after being found we increment randList's index

index++;

//once our comparisons

**if**(comparisons.size() == 42)

{

System.***out***.print("The average of comparisons:" + *average*(comparisons, comparisons.size()));

}

//the counter also gets increased so eventually we exit the loop

counter++;

}

}

//our binarySearch method

**public** **static** **boolean** binSearch(ArrayList<String> A, String target, ArrayList<Integer> compare)

{

**int** numSwap = 0;

//start is at the beginning of the array being called

**int** start = 0;

//stop is at the end of the arrayList

**int** stop = A.size() -1;

//midpoint point is the exact middle of our list

**int** midpoint = (start + stop)/2;

//while the start is less than or equal we stay in our loop

**while**(start <= stop)

{

//numSwap++;

//if A is before the target alphabetically

//we move the start up 1

**if**(A.get(midpoint).compareTo(target) < 0)

{

numSwap++;

start = midpoint + 1;

}

//if the arrays string equals our target string

**else** **if**(A.get(midpoint).equals(target))

{

numSwap++;

midpoint++;

//we print out the number of comparisons it took to //be found

System.***out***.println((numSwap) + " comparisons");

compare.add(numSwap);

//this lets us exit the method since our target was //found

**return** **true**;

}**else**

{ //if midpoint is greater than the target point

numSwap++;

stop = midpoint -1;

}

//midpoint gets reset back

midpoint = (start + stop) / 2;

}

//if not found, we return as false

**return** **false**;

}

**public** **static** ArrayList<String> sort(ArrayList<String> A)

{

//loops over the arrayList

**for**(**int** i = 1; i < A.size(); i++)

{

//key is set to the current position of i

String key = A.get(i);

//j is the point before key

**int** j = i-1;

//makes sure that j is in index 0 or greater

//and that j is > key before initiating this loop

**while**(j >= 0 && A.get(j).compareTo(key) > 0 )

{

//this then sets the index value of the greater word to +1

//so that we will be kicked out of the while loop

A.set(j+1, A.get(j));

j--;

}

//so since we changed, the key now moves to the next string

A.set(j+1, key);

}

**return** A;

}

// Function that return average of an array.

**public** **static** **double** average(ArrayList<Integer> averageArray, **int** size)

{

// Find sum of array element

**int** sum = 0;

//for the the size of

**for** (**int** i = 0; i < size; i++)

{

//we will be adding our newest found item

//to our total sum at the time

sum += averageArray.get(i);

}

//returns our average aka our sum of ints / our arraySize

**return** sum / size;

}

}

//Assignment02

//Rocco Piccirillo

//LinkedHash

//creates my key, value, and next

**public** **class** LinkedHash

{

String key;

**int** value;

LinkedHash next;

/\* Constructor \*/

LinkedHash(String key, **int** value)

{

**this**.key = key;

**this**.value = value;

**this**.next = **null**;

}

}

//created a class for my HashTable

**class** HashTable

{

//creates an int for size of the my hash

//and creates my hash table

**int** hashSize;

LinkedHash[] table;

//set a constructor for my hashTable

**public** HashTable(**int** ts)

{

//sets my hashTable size to limit it

hashSize = 250;

//calls my hashTable and gives it a size of 250

table = **new** LinkedHash[hashSize];

}

//this function was made to find the int value

//of whatever specified string

**public** **int** get(String key)

{

//creates an int to store the index

//of the selected key

**int** hash = (myhash( key ) % hashSize);

//if the the key doesn't exist, exit

**if** (table[hash] == **null**)

**return** -1;

**else**

{

//checks the linkedList at hands value

//if that is not it, we move to the next

//once found, the value of the entry gets returned

LinkedHash entry = table[hash];

**while** (entry != **null** && !entry.key.equals(key))

{

entry = entry.next;

}

**if** (entry == **null**)

{

**return** -1;

}

**else**

{

**return** entry.value;

}

}

}

//this is the same as getting the string but instead

//returning the string back we return the number

//of comparisons it takes to find our string

**public** **int** getComparisons(String key)

{

//counter made to count the compares and get

**int** counter = 0;

**int** hash = (myhash( key ) % hashSize);

**if** (table[hash] == **null**)

**return** -1;

**else**

{

//adds a compare bc we are getting the LinkedList it is in

counter++;

LinkedHash entry = table[hash];

**while** (entry != **null** && !entry.key.equals(key))

{

//adds another compare bc we have not yet found the string

counter++;

entry = entry.next;

}

**if** (entry == **null**)

{

**return** -1;

}

**else**

{

//once found, that means we compared one last time

counter++;

**return** counter;

}

}

}

//this is my method for inserting a string and its own value

//into the index

**public** **void** insert(String key, **int** value)

{

**int** hash = (myhash( key ) % hashSize);

**if** (table[hash] == **null**)

table[hash] = **new** LinkedHash(key, value);

**else**

{

LinkedHash entry = table[hash];

**while** (entry.next != **null** && !entry.key.equals(key))

entry = entry.next;

**if** (entry.key.equals(key))

entry.value = value;

**else**

entry.next = **new** LinkedHash(key, value);

}

}

//through using the ASCII value of each string,

//we assign to a specific value of the hash

**private** **int** myhash(String str )

{

//sets string to upper case

str = str.toUpperCase();

//gets the length of the string

**int** length = str.length();

//sets a letter total int

**int** letterTotal = 0;

//Iterate over all letters in the string, totaling their ASCII values.

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

{

**char** thisLetter = str.charAt(i);

**int** thisValue = (**int**)thisLetter;

letterTotal = letterTotal + thisValue;

}

// Scale letterTotal to fit in hashTableSize.

**int** hashCode = (letterTotal \* 1) % hashSize;

**return** hashCode;

}

//prints all of my hash table out for the user to view

**public** **void** printHashTable()

{

//traverses the whole hash

**for** (**int** i = 0; i < hashSize; i++)

{

//then prints out bucket # has X entries with their indexes

System.***out***.print("\nBucket "+ (i + 1) +": ");

LinkedHash entry = table[i];

//when there are already entries, we chain

**while** (entry != **null**)

{

System.***out***.print(entry.value +" ");

entry = entry.next;

}

}

}

}

//Assignment02

//Rocco Piccirillo

//LinkedHashTable

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** LinkedHashTable

{

**public** **static** **void** main(String[] args) **throws** FileNotFoundException

{

//the scanner is storing the magicitems.txt file temporarily

Scanner scanner = **new** Scanner(**new** File("magicitems"));

//made to actually store the magic items

ArrayList<String> wordList = **new** ArrayList<String>();

//while there is still another line of text more keeps getting added

**while**(scanner.hasNextLine())

{

wordList.add(scanner.nextLine());

}

//creates an instance of my hashTable

HashTable hasher = **new** HashTable(0);

//created a for loop to insert all

//string of my arrayList into the hash

**for**(**int** i = 0; i < wordList.size(); i++)

{

//inserts my string, and a value for it

//I chose the value to be the index of the string

//thought it would be easier to follow

hasher.insert(wordList.get(i), i);

}

//prints out my hash for users to view

hasher.printHashTable();

//creates a new arrayList to store our random items

ArrayList<String> randList = **new** ArrayList<String>();

//while the size of the randList is less than 42

**while** (randList.size() < 42)

{

//a random Index of wordList will be selected

**int** randIndex = (**int**) (Math.*random*()\*(wordList.size()-1));

**if** (randList.indexOf(randIndex) == -1)

{

//the randList adds the random indexed number to the List

randList.add(wordList.get(randIndex));

}

}

//stores the amount of comparisons it takes for the randList string

//to be found in the hash's linkedList

ArrayList<Integer> compareCount = **new** ArrayList<Integer>();

//sets a counter for my while to run 42 times

**int** counter = 0;

//creates an index for my randList to begin checking at

**int** index = 0;

//tests for my word at hand being in the hash

**while**(counter < 42)

{

//if the hasher has an index for the randList string

//we enter the if statement

**if**(hasher.get(randList.get(index)) != -1 )

{

//adds the amount of comparisons it took to find the string

//inside of this arrayList

compareCount.add(hasher.getComparisons(randList.get(index)));

//prints out the string being found and the amount of comparisons/get it took

System.***out***.print("\n" + randList.get(index) + " found after " + hasher.getComparisons(randList.get(index)) + " comparisons.");

//increments index so we can move onto the next string

index++;

//increments counter so we can eventually leave the while loop

counter++;

}

}

//prints out the average amount of comparisons

System.***out***.print("\nThe average amount of comparisons is " + *average*(compareCount, compareCount.size()));

}

//Function that return average of an array.

**public** **static** **double** average(ArrayList<Integer> averageArray, **int** size)

{

// Find sum of array element

**int** sum = 0;

//for the the size of

**for** (**int** i = 0; i < size; i++)

{

//we will be adding our newest found item

//to our total sum at the time

sum += averageArray.get(i);

}

//returns our average aka our sum of ints / our arraySize

**return** sum / size;

}

}