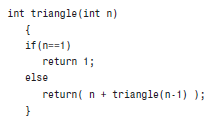
Lab 4

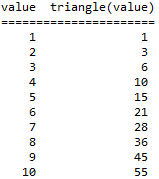
Recursion, Big O, Search

1. Recursion. (10 points) Thanks to Data Structures and Algorithms in Java by LaFore, 2nd edition.

Given the following recursive method:



Create a Java program that calls this method for the numbers 1-10 and displays the output. I am not picky about output formatting, but output could look like:



Your name and today's date must be a comment on the first line of the program.

Then, please answer the following questions:

1.1. What is the base case?  
  
 **- When n=1;**

1.2. Is n > 1 the base case or the recursive case?

**- The recursive case.**  
1.3. Calculate the value of triangle(3) by hand, and then run the program in Java to check your result. Did your answer match?

**- 6, and yes.**

1.4. What value is returned if n=0?

**-Java throws and exception.**

1.5. Based on your answer to 1.4: what is missing from the function?

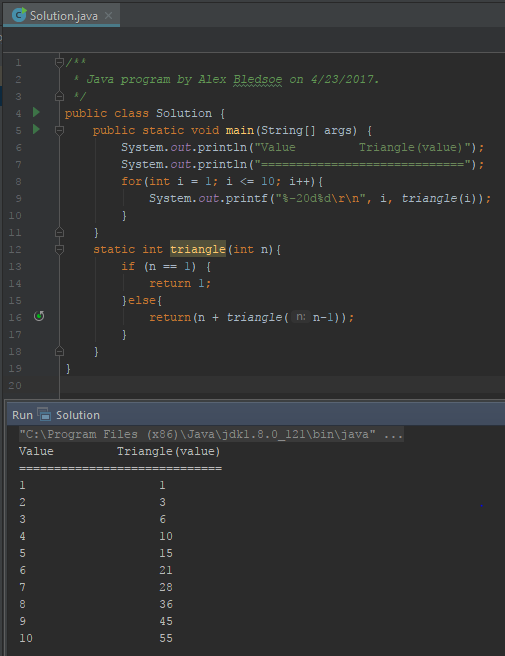
**- Either nest the contents of the triangle method in another if to check for negative numbers,**

**or -> ‘if(n <= 1)’ instead of ‘if(n == 1)’.**

Rubric:

Student name and today’s date is a comment in the first line of the programs: -5 points if fails  
Screenshot and program code: -5 points if fails  
Questions: 5 points, 1 point per question  
triangle defined per problem statement: 1 point  
Loop to test triangle(n) 1-10: 2 points  
Program output: 2 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your .java file, here.



**Solution.java:**

/\*\*

\* Java program by Alex Bledsoe on 4/23/2017.

\*/

public class Solution {

public static void main(String[] args) {

System.out.println("Value Triangle(value)");

System.out.println("=============================");

for(int i = 1; i <= 10; i++){

System.out.printf("%-20d%d\r\n", i, triangle(i));

}

}

static int triangle(int n){

if (n == 1) {

return 1;

}else{

return(n + triangle(n-1));

}

}

}

2. Recursive problem breakdown. (10 points)

Create a recursive method to implement binary search. Call the method from your main program with the following test cases:

|  |  |  |
| --- | --- | --- |
| Array | Search for value | Index returned |
| (0,1,2,3,4,5,6,7,8,9) | 5 | 5 |
| (0,1,2,3,4,5,6,7,8,9) | 9 | 9 |
| (0,1,2,3,4,5,6,7,8,9) | 0 | 0 |
| (0,1,2,3,4,5,6,7,8,9) | 10 | -1 |
| (3,5,7,9) | 2 | -1 |
| (3,5,7,9) | 3 | 0 |
| (3,5,7,9) | 9 | 3 |
| (2,4,6,8,12) | 2 | ? |
| (2,4,6,8,12) | 12 | ? |
| (2,4,6,8,12) | 10 | ? |
| (2,4,6,8,12) | -5 | ? |

After each call to BinarySearch, have your program output:

1. The array searched

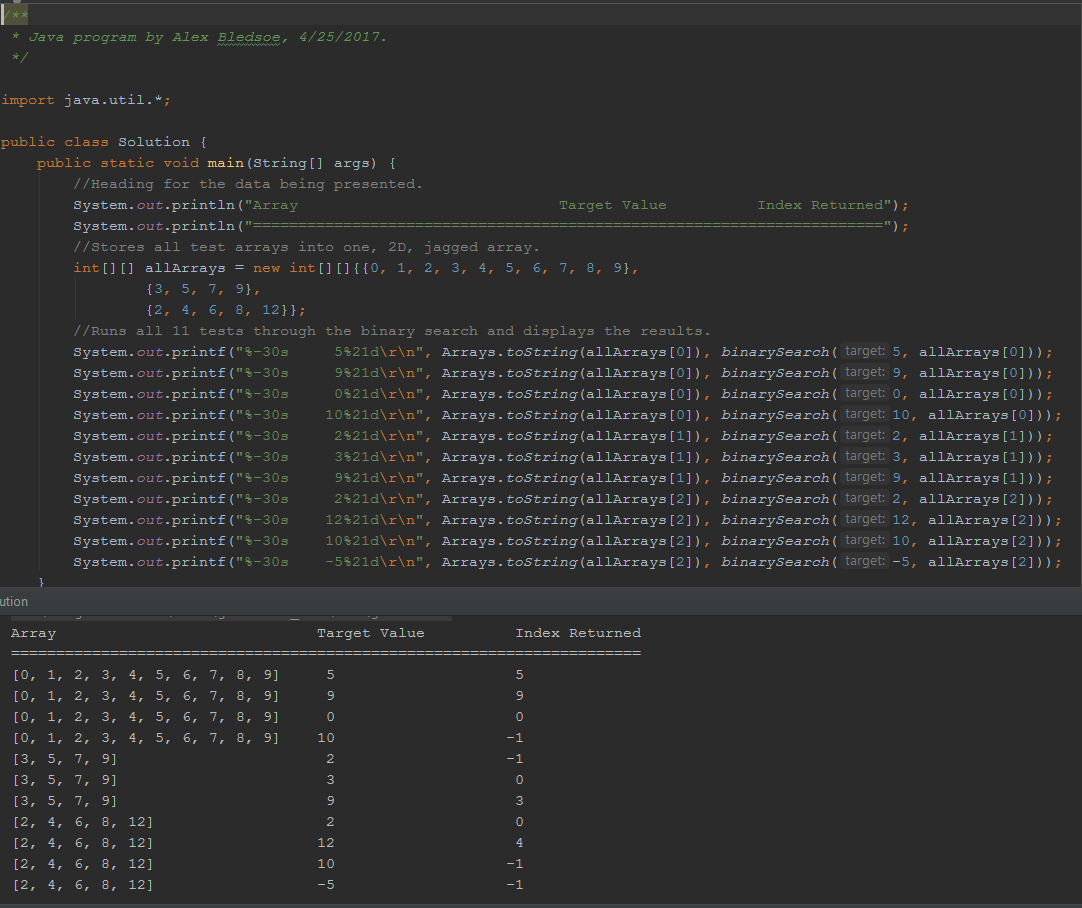
2. The value searched for

3. The index returned from BinarySearch

Create a table similar to the sample output from question 1.

Rubric:  
Student name and today’s date is a comment in the first line of the programs: -5 points if fails  
Screenshot and program code: -5 points if fails   
BinarySearch algorithm implemented correctly: 3 points  
Three arrays and 11 test cases: 3 points  
Output formatted in a nice table: 4 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here

****

**Solution.java:**

/\*\*

\* Java program by Alex Bledsoe, 4/25/2017.

\*/

import java.util.\*;

public class Solution {

public static void main(String[] args) {

//Heading for the data being presented.

System.out.println("Array Target Value Index Returned");

System.out.println("======================================================================");

//Stores all test arrays into one, 2D, jagged array.

int[][] allArrays = new int[][]{{0, 1, 2, 3, 4, 5, 6, 7, 8, 9},

{3, 5, 7, 9},

{2, 4, 6, 8, 12}};

//Runs all 11 tests through the binary search and displays the results.

System.out.printf("%-30s 5%21d\r\n", Arrays.toString(allArrays[0]), binarySearch(5, allArrays[0]));

System.out.printf("%-30s 9%21d\r\n", Arrays.toString(allArrays[0]), binarySearch(9, allArrays[0]));

System.out.printf("%-30s 0%21d\r\n", Arrays.toString(allArrays[0]), binarySearch(0, allArrays[0]));

System.out.printf("%-30s 10%21d\r\n", Arrays.toString(allArrays[0]), binarySearch(10, allArrays[0]));

System.out.printf("%-30s 2%21d\r\n", Arrays.toString(allArrays[1]), binarySearch(2, allArrays[1]));

System.out.printf("%-30s 3%21d\r\n", Arrays.toString(allArrays[1]), binarySearch(3, allArrays[1]));

System.out.printf("%-30s 9%21d\r\n", Arrays.toString(allArrays[1]), binarySearch(9, allArrays[1]));

System.out.printf("%-30s 2%21d\r\n", Arrays.toString(allArrays[2]), binarySearch(2, allArrays[2]));

System.out.printf("%-30s 12%21d\r\n", Arrays.toString(allArrays[2]), binarySearch(12, allArrays[2]));

System.out.printf("%-30s 10%21d\r\n", Arrays.toString(allArrays[2]), binarySearch(10, allArrays[2]));

System.out.printf("%-30s -5%21d\r\n", Arrays.toString(allArrays[2]), binarySearch(-5, allArrays[2]));

}

/\*

\* Takes target and array, then passes those plus min and max onto other binary search method.

\* (so that min and max don't have to be manually specified every time you want to do a search).

\*/

static int binarySearch(int target, int[] array) {

return binarySearch(target, array, 0, array.length - 1);

}

// Method that actually performs the binary search.

static int binarySearch(int target, int[] array, int min, int max) {

//Base case.

if (min > max) {

return -1;

}

int mid = (min + max) / 2;

//Cut the left half off and try again.

if (array[mid] > target) {

max = mid - 1;

return binarySearch(target, array, min, max);

//Cut the left half off and try again.

} else if (array[mid] < target) {

min = mid + 1;

return binarySearch(target, array, min, max);

}

//Target found.

return mid;

}

}

3. Big O Notation (10 points) Thanks to Reges, Building Java Programs, 2nd edition.

Estimate the big-O complexity for each of these algorithms, and **justify your answer**.

To confirm your calculations, answers are provided at the end of the rubric. Your justification can be mathematical or written, formal or informal.

|  |  |
| --- | --- |
| Problem | Code fragment |
| 3.1 | int sum = 0;  int j = 1;  while (j <= n) {  sum++;  j \*= 2;  } |
| Big-O Category:   * **O(log N)** |
| Justification (why did you pick the way you did?):   * **Because ‘j’ doubles with each iteration of the loop, it rapidly increases so the time for j to reach n increases slower and slower as n increases.** * **When n is 100, the loop iterates 7 times, but when n is 10,000, it only must iterate another 7 times, and only another 6 times to reach 1,000,000.** |
| 3.2 | int sum = 0;  for (int j = 1; j < n; j++) {  sum++;  if (j % 2 == 0) {  sum++;  }  } |
| Big-O Category:   * **O(N)** |
| Justification (why did you pick the way you did?):   * **The amount of time that the program will run is directly proportional to how large n is. When n is 10, the loop runs 9 times, but when n is 100, the loop must run 99 times.** |
| 3.3 | int sum = 0;  for (int i = 1; i <= n \* 2; i++) {  for (int j = 1; j <= n; j++) {  sum++;  }  } |
| Big-O Category:   * O(2N^2), but since constants are dropped, **O(N^2).** |
| Justification (why did you pick the way you did?):   * **The amount of times that the loop runs is proportional to 2 times the square of n. n^2 has a far greater impact on the runtime, which is why the 2 is dropped in the notation, but when n is 10, the loop runs 200 times (or 2\*10^2), and when n is 10,000 the loop has to run 200,000,000 times(or 2\*10,000^2).** |
| 3.4 | for (int j = 1; j < 100; j++) {  sum++;  sum++;  } |
| Big-O Category:   * **O(1)** |
| Justification (why did you pick the way you did?):   * **The loop will always iterate 99 times, because both the starting point and endpoint for j are defined. Regardless of how large some variable (n) is elsewhere in the program, the given loop will always have the same runtime**. |

Rubric:  
Correct Big-O classification of four problems: 2 points  
Justification of four problems: 4 points  
Big-O categories: 3.1. O(log n). 3.2. O(n). 3.3. O(n2). 3.4. O(1)

4. Linear and Binary search. (20 points)

Write methods that implement linear and binary search. Whether you use an iterative or recursive solution is up to you, but an iterative solution may be easier for this problem.

Have both methods display the value searched for, and the number of iterations needed to find the value. You will need to add System.out.println() statements just before both return statements for each algorithm. You will also need to add some kind of counter to track iterations.

To test this, create an array: (0, 1, 2, 3, 4, 5, 6, 7, 8 . . . ) of *at least 100 values*.

Search for the following values in the array, first using linear search, then using binary search.

Values:

5

10

17

20

25

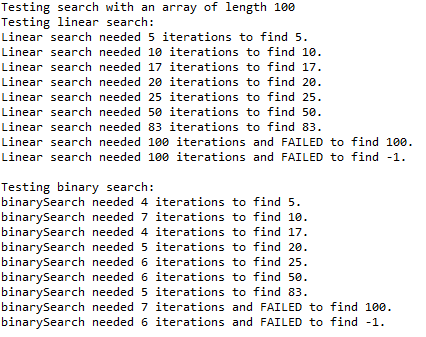
50

83

n, where n is the length of the array. (If the array has 100 values, search for the number 100.)

-1

Your program output could look like:



**You will need to resize the console window in Eclipse to view all the program output.**

Rubric:

Student name and today’s date are a comment on the first line of the program (-5 if fails)

linearSearch and binarySearch implemented correctly: (-10 if fails)

Screenshot shows all program output: (-8 if fails)

Recursive binary search with iteration tracker: +5 bonus

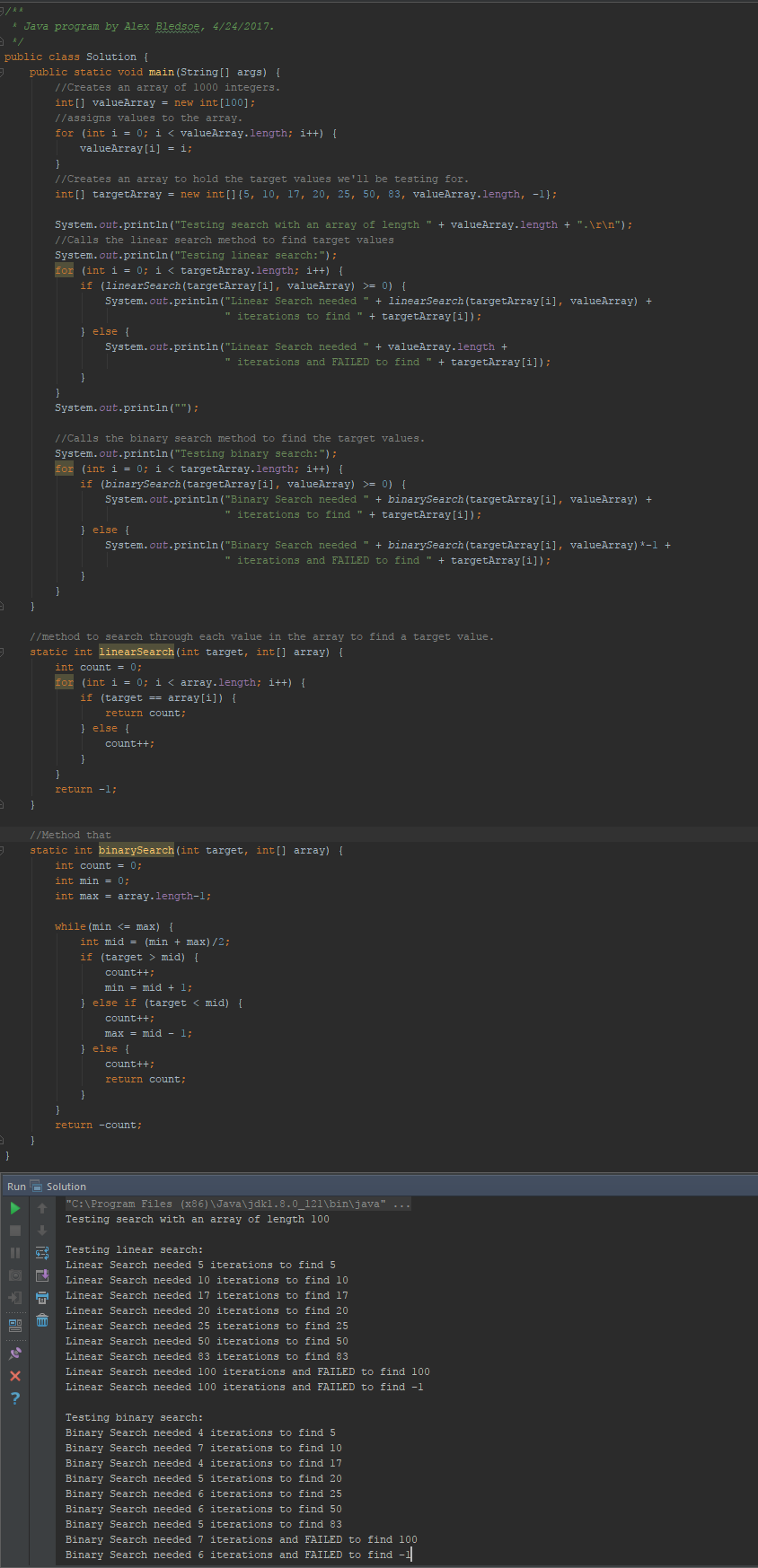
Iteration counter, linear search: 5 points

Iteration counter, binary search: 5 points  
Array of at least 100 values: 3 points

Test cases: 5 points  
Program output: 2 points

Please paste a screenshot of a successful program run, and copy-and-paste the source code from your main program's .java file, here.

**ITERATIVE SOLUTION:**



**Solution.java (for iterative solution):**

/\*\*

\* Java program by Alex Bledsoe, 4/24/2017.

\*/

public class Solution {

public static void main(String[] args) {

//Creates an array of 1000 integers.

int[] valueArray = new int[44650850]; //As large as IntelliJ will consistently let me make the array.

//assigns values to the array.

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

valueArray[i] = i;

}

//Creates an array to hold the target values we'll be testing for.

int[] targetArray = new int[]{5, 10, 17, 20, 25, 50, 83, valueArray.length, -1};

System.out.println("Testing search with an array of length " + valueArray.length + ".\r\n");

//Calls the linear search method to find target values

System.out.println("Testing linear search:");

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

if (linearSearch(targetArray[i], valueArray) >= 0) {

System.out.println("Linear Search needed " + linearSearch(targetArray[i], valueArray) +

" iterations to find " + targetArray[i]);

} else {

System.out.println("Linear Search needed " + valueArray.length +

" iterations and FAILED to find " + targetArray[i]);

}

}

System.out.println("");

//Calls the binary search method to find the target values.

System.out.println("Testing binary search:");

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

if (binarySearch(targetArray[i], valueArray) >= 0) {

System.out.println("Binary Search needed " + binarySearch(targetArray[i], valueArray) +

" iterations to find " + targetArray[i]);

} else {

System.out.println("Binary Search needed " + binarySearch(targetArray[i], valueArray)\*-1 +

" iterations and FAILED to find " + targetArray[i]);

}

}

}

//method to search through each value in the array to find a target value.

static int linearSearch(int target, int[] array) {

int count = 0;

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

if (target == array[i]) {

return count;

} else {

count++;

}

}

return -1;

}

//Method that

static int binarySearch(int target, int[] array) {

int count = 0;

int min = 0;

int max = array.length-1;

while(min <= max) {

int mid = (min + max)/2;

if (target > mid) {

count++;

min = mid + 1;

} else if (target < mid) {

count++;

max = mid - 1;

} else {

count++;

return count;

}

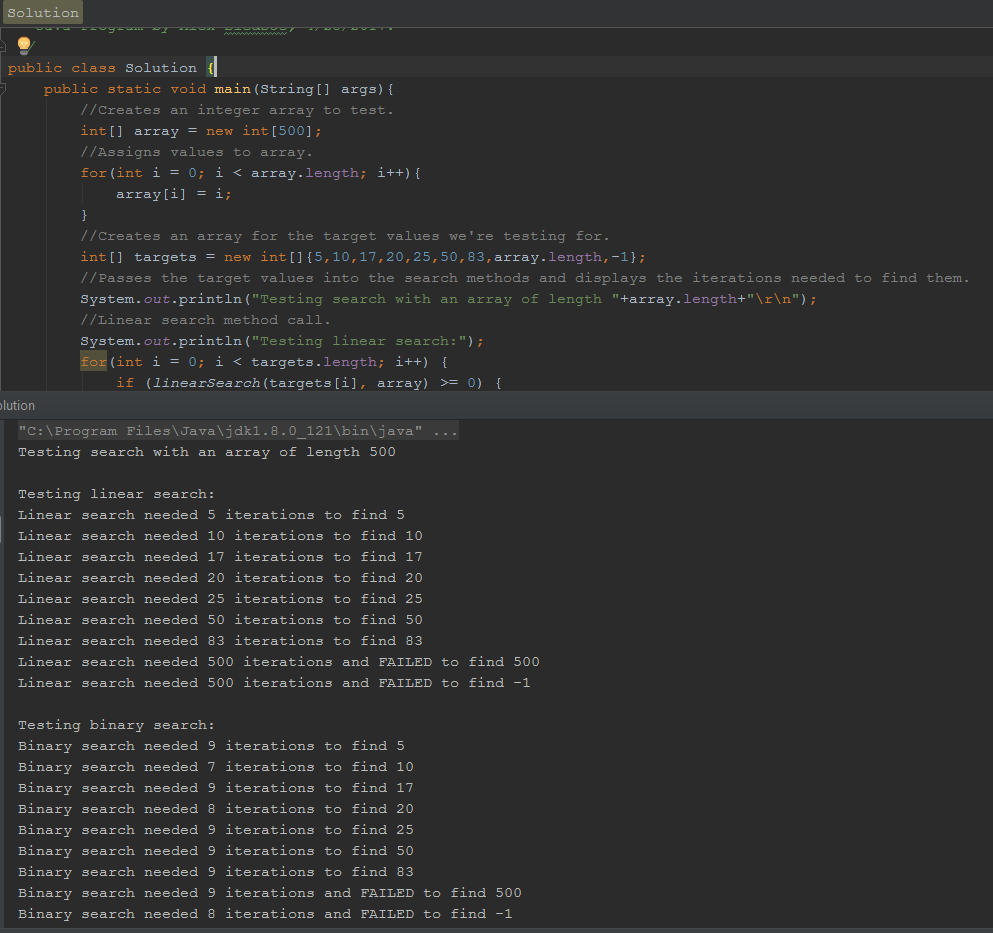
}

return -count;

}

}

**RECURSIVE SOLUTION:**

****

**Solution.java (for recursive solution):**

/\*\*

\* Java Program by Alex Bledsoe, 4/25/2017.

\*/

public class Solution {

public static void main(String[] args) {

//Creates an integer array to test.

int[] array = new int[500];

//Assigns values to array.

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

array[i] = i;

}

//Creates an array for the target values we're testing for.

int[] targets = new int[]{5, 10, 17, 20, 25, 50, 83, array.length, -1};

//Passes the target values into the search methods and displays the iterations needed to find them.

System.out.println("Testing search with an array of length " + array.length + "\r\n");

//Linear search method call.

System.out.println("Testing linear search:");

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

if (linearSearch(targets[i], array) >= 0) {

System.out.println("Linear search needed " + linearSearch(targets[i], array) +

" iterations to find " + targets[i]);

} else {

System.out.println("Linear search needed " + linearSearch(targets[i], array) \* -1 +

" iterations and FAILED to find " + targets[i]);

}

}

System.out.println();

//Binary search method call.

System.out.println("Testing binary search:");

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

if (binarySearch(targets[i], array) >= 0) {

System.out.println("Binary search needed " + binarySearch(targets[i], array) +

" iterations to find " + targets[i]);

} else {

System.out.println("Binary search needed " + binarySearch(targets[i], array) \* -1 +

" iterations and FAILED to find " + targets[i]);

}

}

}

/\*

\* Initial method that passes the target and array onto the method that actually performs the search,

\* but also passes an iterations counter as well so that only the target and array have

\* to be passed in the main method.

\*/

static int linearSearch(int target, int[] array) {

return linearSearch(target, array, 0);

}

/\*

\* RECURSIVE linear search method.

\*/

static int linearSearch(int target, int[] array, int iterations) {

if (array.length == iterations) {

return -iterations;

} else if (array[iterations] == target) {

return iterations;

} else {

return linearSearch(target, array, iterations + 1);

}

}

/\*

\* Initial method that passes the target and array onto the method that actually performs the search,

\* but also passes a min, max and iterations counter as well so that only the target and array have

\* to be passed in the main method.

\*/

static int binarySearch(int target, int[] array) {

return (binarySearch(target, array, 0, array.length - 1, 0));

}

//RECURSIVE method that actually performs the binary search.

static int binarySearch(int target, int[] array, int min, int max, int iterations) {

if (min > max) {

return -iterations;

}

int mid = (min + max) / 2;

if (array[mid] > target) {

iterations++;

max = mid - 1;

return binarySearch(target, array, min, max, iterations);

} else if (array[mid] < target) {

iterations++;

min = mid + 1;

return binarySearch(target, array, min, max, iterations);

}

iterations++;

return iterations;

}

}