Assignment 3 - 18343763

Problem Statement

The problem involves finding all palindromes between 0 - NUM (1000000) that are palindromes in decimal and binary. I will have to make methods that find palindromes using different ways and then compare after based on the number of operations taken to complete & time taken to complete.

Analysis & Design Notes

I will have to make a method that is able to test that all methods are working. This method would check that every method is returning the correct palindromes as the point of this assignment is to show the amount of operations needed to find palindromes, not actually find the palindromes. I would compare the list of palindromes to a list of valid palindromes where the numbers are the same in binary and decimal.
I will need to make the methods (method 1,2,3,4) and name them according to what they do.

Method 1: LoopString

Method 1 loops through the string given, flips it and returns true or false based on if the input string is equal to the output string (flipped input string).

Method 2: checkIndividually

Method 2 loops over every character in the input and compares it to that position flipped in the string, for example: it would compare position 3 to the (input length - 3)th position. If at any point the characters are not the same, it exits, as it is pointless to continue, it is not a palindrome.

Method 3: stackCheck

Method 3 loops over every character, adds them to a stack. Once done, it loops over every character again, pops from the stack and compares it to the character in the string, if at any point the characters are not the same, it exits. After is has looped, it double checks the stack is empty and returns true if it is.

Method 4: recursiveStringCheck

Method 4 returns true if reverse returns true when the input is the same.

The decimal to binary method pushes a 0 or 1 to a stack based on if the input is divisible by 2. After it has generated the stack will then be empty and appended to a string and then returned.

PseudoCode

```
Main {
	For (int i = 0; i < 1000000; i++) {
		if (loopString(String.valueOf(i)) && loopString(decToBin(String.valueOf(i)))) {
			myListM1.add(i); // add to array
		}
		if (checkIndividually(String.valueOf(i)) &&
	checkIndividually(decToBin(String.valueOf(i)))) {
			myListM1.add(i); // add to array
		}
		if (stackCheck(String.valueOf(i)) && stackCheck(decToBin(String.valueOf(i)))) {
			myListM1.add(i); // add to array
		}
		if (recursiveStringCheck(String.valueOf(i)) &&
	recursiveStringCheck(decToBin(String.valueOf(i)))) {
			myListM1.add(i); // add to array
		}
	}

	// Do for every list
	for (int i : myListM1) {
		if (existsInValidPalindromes(i)) System.out.printf("i: %d\n", i);
		else System.out.printf("error i: %d", i);
	}
}

 static boolean existsInValidPalindromes(int i) {
	for (int y : validPalindromes) {
		if (i == y) return true;
	}
	return false;
}
```

Code

```
import java.util.*;

public class App {
```

```
	//Declare any global variables required (e.g. operation counts for each method)
	static long startTimeM1, endTimeM1, startTimeM2, endTimeM2, startTimeM3,
endTimeM3, startTimeM4, endTimeM4;
	static int countM1 = 0, countM2 = 0, countM3 = 0, countM4 = 0;
	static List<Integer> myListM1 = new ArrayList<>(), myListM2 = new ArrayList<>(),
myListM3 = new ArrayList<>(), myListM4 = new ArrayList<>();
	static int[] validPalindromes = {0, 1, 3, 5,7, 9, 33, 99, 313, 585, 717, 7447, 9009, 15351,
32223, 39993, 53235, 53835, 73737, 585585};

	// Main Method
	public static void main (String[] args) {
	//Declare any variables used (e.g. for timing etc.)

	//Test each method (looping over the binary/decimal numbers for each)
	//by calling your defined methods below
	testMethods();

	// Display results for each method
	System.out.print("\nMethod 1 errors (if any): \n");
	for (int i : myListM1) {
	if (existsInValidPalindromes(i)) System.out.printf("error i: %d", i);
	}

	System.out.print("\nMethod 2 errors (if any): \n");
	for (int i : myListM2) {
	if (existsInValidPalindromes(i)) System.out.printf("error i: %d", i);
	}

	System.out.print("\nMethod 3 errors (if any): \n");
	for (int i : myListM3) {
	if (existsInValidPalindromes(i)) System.out.printf("error i: %d", i);
	}

	System.out.print("\nMethod 4 errors (if any): \n");
	for (int i : myListM4) {
	if (existsInValidPalindromes(i)) System.out.printf("error i: %d", i);
	}

	// Note: Think carefully about the design of your main method
	//If designed correctly, you will be able to automate the running of
	//experiments over many number ranges instead of having to manually
	//change the values for each run. The data produced can then be used for
	// graphing in Excel.
```

```
        }

        static boolean existsInValidPalindromes(int i) {
        for (int y : validPalindromes) {
        if (i == y) return false;
        }
        return true;
        }

        static void testMethods() {
        startTimeM1 = System.currentTimeMillis(); // Init count
        countM1++;
        countM1++;
        System.out.print("Method1\nInterval\tNum Operations\n");
        for (int i = 0; i < 1000000; i++) {
        countM1++; // init i && i < num check
        countM1 += 5;
        if (loopString(String.valueOf(i)) && loopString(decToBin(String.valueOf(i)))) { // call 2 x
loopString, 2 x convert to string, convert to binary
                myListM1.add(i); // add to array
                countM1++;
        }
        if (i % 50000 == 0) {
                System.out.printf("%d\t%d\n", i, countM1);
        }
        countM1++; // increment i
        }
        endTimeM1 = System.currentTimeMillis() - startTimeM1;
        System.out.printf("Time Taken: %d ms\n\n", endTimeM1);

        startTimeM2 = System.currentTimeMillis();
        countM2++;
        System.out.print("Method 2\nInterval\tNum Operations\n");
        for (int i = 0; i < 1000000; i++) {
        countM2 += 2; // init i && i < num check
        countM2 += 5;
        if (checkIndividually(String.valueOf(i)) && checkIndividually(decToBin(String.valueOf(i))))
{
                myListM2.add(i); // add to array
                countM2++; // add to array
        }
        if (i % 50000 == 0) {
                System.out.printf("%d\t%d\n", i, countM2);
```

```
        }
        countM2++; //increment i
        }
        endTimeM2 = System.currentTimeMillis() - startTimeM2;
        System.out.printf("Time Taken: %d ms\n\n", endTimeM2);

        startTimeM3 = System.currentTimeMillis();
        countM3++;
        System.out.print("Method 3\nInterval\tNum Operations\n");
        for (int i = 0; i < 1000000; i++) {
        countM3 += 2; // init i && i < num check
        countM3 += 5;
        if (stackCheck(String.valueOf(i)) && stackCheck(decToBin(String.valueOf(i)))) {
                myListM3.add(i);
                countM3++;
        }
        if (i % 50000 == 0) {
                System.out.printf("%d\t%d\n", i, countM3);
        }
        countM3++; // increment i
        }
        endTimeM3 = System.currentTimeMillis() - startTimeM3;
        System.out.printf("Time Taken: %d ms\n\n", endTimeM3);

        startTimeM4 = System.currentTimeMillis();
        countM4++;
        System.out.print("Method 4\nInterval\tNum Operations\n");
        for (int i = 0; i < 1000000; i++) {
        countM4 += 2; // init i && i < num check
        countM4 += 5;
        if (recursiveStringCheck(String.valueOf(i)) &&
recursiveStringCheck(decToBin(String.valueOf(i)))) {
                myListM4.add(i);
                countM4++;
        }
        if (i % 50000 == 0) {
                System.out.printf("%d\t%d\n", i, countM4);
        }
        countM4++; // increment i
        }
        endTimeM4 = System.currentTimeMillis() - startTimeM4;
        System.out.printf("Time Taken: %d ms\n\n", endTimeM4);
        }
```

```
	//Static method for: Palindrome Method 1 (give it a name based on how it works)
	//Takes a String as a parameter and return a Boolean value
	static Boolean loopString(String in) {
	StringBuilder out = new StringBuilder(); // init string builder
	countM1++;
	countM1++;
	for (int i = in.length() - 1; i >= 0; i--) { // init i, call in.length, check i,
	countM1 += 2;
	out.append(in.charAt(i)); // get character and append
	countM1 += 2;
	countM1++; // increment i
	}
	countM1 += 3; // check if equals, call toString, return true or false
	return in.equals(out.toString());
	}

	//Static method for: Palindrome Method 2 (give it a name based on how it works)
	//Takes a String as a parameter and return a Boolean value
	static Boolean checkIndividually(String in) {
	countM2++;
	for (int i = 0; i < in.length(); i++) { // init i, call in.length, check i,
	countM2 += 2;
	countM2 += 2;
	if (in.length() == 1) { // call in.length, check
		countM2++; // return true
		return true;
	}
	countM2 += 4;
	if (in.charAt(i) != in.charAt(in.length() - i - 1)) { // 2 x get chatAt i, get in.length, check if the
same
		return false;
	}
	}
	countM2++;
	return true;
	}

	//Static method for: Palindrome Method 3 (give it a name based on how it works)
	//Takes a String as a parameter and return a Boolean value
	static Boolean stackCheck(String in) {
	Stack<Character> s = new Stack<>(); // init stack
	countM3++;
	countM3++;
```

```
        for (char x : in.toCharArray()) { // call toCharArray, get x
        countM3+=2;
        s.push(x);
        }
        for (char x : in.toCharArray()) {
        countM3 += 2;
        countM3+=4;
        if (!s.isEmpty() && x == s.peek()) { // check is not empty, call s.peek, call isempty, check
is not equal to x
                countM3++;
                s.pop(); // check is not empty, check top of stack
        }
        else {
                countM3++;
                return false;
        }
        }
        countM3++;
        return s.isEmpty();
        }

        //Static method for: Palindrome Method 4 (give it a name based on how it works)
        //Takes a String as a parameter and return a Boolean value
        static Boolean recursiveStringCheck(String in) {
        countM4 += 3;
        return reverse(in).equals(in); // return, call reverse, check if equal
        }

        //Static method for: Recursively reversing a String (to be used by Method 4)
        //Takes a String and returns a String value of it reversed (must use recursion)
        static String reverse(String in) {
        countM4++; // if check
        if (in.isEmpty()) {
        countM4++;
        return in;
        }
        countM4 += 5;
        return reverse(in.substring(1)) + in.charAt(0);
        }

        //Static method for: Converting a decimal number into its equivalent binary
representation
        //Takes a String representation of a number as a parameter and return a String value
```

```
        static String decToBin(String input) {
        Stack<Integer> s = new Stack<>();
        int in = Integer.parseInt(input);
        StringBuilder out = new StringBuilder();

        if (!input.equals("0")) {
        while (in != 0) {
                s.push(in % 2);
                in /= 2;
        }
        while (!s.isEmpty()) {
                out.append(s.peek());
                s.pop();
        }
        } else {
        out.append(input);
        }
        return out.toString();
        }
}
```

Testing

Below I am demonstrating that if there is a number in the array that is not a valid palindrome, it returns true and will output an error to the console, else it outputs no errors to the console.

```
for (int i : myListM1) {
    if (existsInValidPalindromes(i)) System.out.printf( s: "error i: %d", i);
}
System.out.print("\nMethod 2 errors (if any): \n");
for (int i : myListM2) {
    if (existsInValidPalindromes(i)) System.out.printf( s: "error i: %d", i);
}
System.out.print("\nMethod 3 errors (if any): \n");
for (int i : myListM3) {
    if (existsInValidPalindromes(i)) System.out.printf( s: "error i: %d", i);
}
System.out.print("\nMethod 4 errors (if any): \n");
for (int i : myListM4) {
    if (existsInValidPalindromes(i)) System.out.printf( s: "error i: %d", i);
}
```

Example of an error occuring and the error being output to console:

```
myListM1.add(20);
for (int i : myListM1) {
    if (existsInValidPalindromes(i)) System.out.printf( s: "error i: %d", i);
}
```

```
Method 1 errors (if any):
error i: 20
Method 2 errors (if any):

Method 3 errors (if any):

Method 4 errors (if any):
```

Output from program:
Outputs the interval, time taken and number of operations at that interval. This is then used to generate the graph.

```
Method1
Interval    Num Operations
0   29
50000   1838369
100000  3732602
150000  5837197
200000  7941947
250000  10046697
300000  12151637
350000  14256637
400000  16361637
450000  18466637
500000  20571637
550000  22676767
600000  24782018
650000  26887268
700000  28992518
750000  31097768
800000  33203018
850000  35308268
900000  37413518
950000  39518768
Time Taken: 417 ms

Method 2
Interval    Num Operations
0   21
50000   913143
100000  1825173
150000  2721617
200000  3618109
250000  4513809
300000  5409509
350000  6305913
400000  7202349
450000  8098049
500000  8993749
550000  9890273
600000  10786783
650000  11682483
700000  12578183
750000  13474595
800000  14371103
850000  15266803
900000  16162503
950000  17058947
Time Taken: 236 ms
```

```
Method 3
Interval    Num Operations
0   33
50000   1400185
100000  2821779
150000  4314218
200000  5806761
250000  7298611
300000  8790537
350000  10283103
400000  11775697
450000  13267647
500000  14759597
550000  16252320
600000  17745079
650000  19237129
700000  20729179
750000  22221852
800000  23714609
850000  25206659
900000  26698709
950000  28191410
Time Taken: 248 ms

Method 4
Interval    Num Operations
0   31
50000   2135438
100000  4338017
150000  6793481
200000  9249131
250000  11704781
300000  14160659
350000  16616609
400000  19072559
450000  21528509
500000  23984459
550000  26440565
600000  28896816
650000  31353066
700000  33809316
750000  36265566
800000  38721816
850000  41178066
900000  43634316
950000  46090566
Time Taken: 794 ms
```

The graph shows that the number of operations are exponential.

Method 4 takes the most amount of operations as for every letter in the string, there are at least 5 operations (call reverse, call substring, call charAt, add operation in.substring(1)) + in.charAt(0) and return the answer).
Method 2 takes the least amount of time as all it does is flip the string in one for loop and return true if the flipped string is equal to the input.