Assignment 3 – Oisín Mc Laughlin – 22441106

Problem Statement

A palindrome is a sequence that will read the same forwards as it does backwards. An example of one is 86668 and it's binary equivalent 100111001.

For this assignment we will be writing a program that will check to see if a number is a palindrome using values 1 to 1000000 and converting them ot binary strings and these will be passed into four methods to check if they are a palindrome. Each method will be different in design and test the efficiency of each method. The four methods t will all take a string input and return a boolean output depending on if it is a palindrome (true) or not (false).

The idea of this assignment is to look at

Analysis and Design Notes

Method 1:
Using a loop, reverse all characters in the string using a loop that will loop from input.length and decrement until 0. It then will compare the reversed input to the original input using .equals and return true if they match otherwise returning false if they don't.

Method 2:
Again using a loop, compare each character this time comparing first element to last, second to second last and so on. This will be done using a while loop where i will be incrementing from the start and j decrementing from the end returning true if they match and false if they don't.

Method 3:
This will use ArrayStack and ArrayQueue which are provided on canvas. This method pushes / queues each element to the stack and also the queue, after fully added, each element is popped / dequeued and compared to each other. If a mismatch is found, return false otherwise return true.

Method 4:
A separate method is first made called reverse and this recursive method will peel off first char each call and append first char to result of the call and build reversed string from end to beginning.

The actual method 4 will then compare the reversed string to the original input.

Utility Method:
This converts a decimal to a binary using .toBinaryString and returns it.

Code

```
import java.lang.reflect.Array;

public class Main extends ArrayStack implements Stack {
    static long oppC1 = 0;
    static long oppC2 = 0;
    static long oppC3 = 0;
    static long oppC4 = 0;

    public static boolean method1(String input) {
        String reversed = "";
        oppC1++;

        //Looping from end and adding to reversed
        for (int i = input.length() - 1; i >= 0; i--) {
            reversed += input.charAt(i);
            oppC1 += 2;
        }

        //If input is the same as reversed input return true, otherwise
return false
        oppC1++;
        return (input.equals(reversed));
    }

    public static boolean method2(String input) {
        int i = 0;
        int j = input.length() - 1;
        oppC2 += 2;

        //End char compared to first char, return false if mismatch
        while (i < j) {
            if (input.charAt(i) != input.charAt(j)) {
                oppC2 += 3;
                return false;
            }
            i++;
            j--;
            oppC2 += 3;
        }
        //Otherwise return true
        oppC2++;
        return true;
    }

    public static boolean method3(String input) {
        ArrayStack stack = new ArrayStack();
        ArrayQueue queue = new ArrayQueue();
        oppC3 += 3;

        for (int i = 0; i < input.length(); i++) {
            //Add each character to both array stack and array queue
            char c = input.charAt(i);

            stack.push(c);
            queue.enqueue(c);
            oppC3 += 4;
        }

        //Pop and deque values and compare using .equals, return false if
mismatch
```

```
        while (!stack.isEmpty() && !queue.isEmpty()) {
            if (!stack.pop().equals(queue.dequeue())) {
                oppC3 += 3;
                return false;
            }
        }
        //Otherwise return true
        oppC3++;
        return true;
    }

    public static boolean method4(String input) {
        //If input is the same as reversed input return true, otherwise
return false
        oppC4++;
        return input.equals(reverse(input));
    }

    public static String reverse(String input) {
        //If empty (base case) return the input
        if (input.isEmpty()) {
            oppC4++;
            return input;
        }
        oppC4 += 3;
        //Recursion case will peel off first char each call and append
first char to result of the call and build reversed string from end to
begin
        return reverse(input.substring(1)) + input.charAt(0);
    }

    public static String utility(String input) {
        //Takes decimal string and changes to binary string
        return Integer.toBinaryString(Integer.parseInt(input));
    }

    public static void resetOps() {
        //Reset operations count for accuracy
        oppC1 = 0;
        oppC2 = 0;
        oppC3 = 0;
        oppC4 = 0;
    }

    public static boolean checkPal(int i, String input) {
        //Chooses one of the methods based on the index of the for loop
        if (i == 1) {
            return method1(input);
        }
        else if (i == 2) {
            return method2(input);
        }
        else if (i == 3) {
            return method3(input);
        }
        else if (i == 4) {
            return method4(input);
        }
        else {
            return false;
        }
```

```
    }

    private static long getOppC(int i) {
        //Gets the operations count based on the index of the for loop
        if (i == 1) {
            return oppC1;
        }
        else if (i == 2) {
            return oppC2;
        }
        else if (i == 3) {
            return oppC3;
        }
        else if (i == 4) {
            return oppC4;
        }
        else {
            return 0;
        }
    }

    public static void main(String[] args) {
        int increment = 50000; // Define the increment step
        int maxRange = 1000000; // Define the maximum range

        // Print the CSV header
        System.out.println("Problem Size,Method 1 Operations,Method 2
Operations,Method 3 Operations,Method 4 Operations");

        // Loop through the problem sizes incrementally
        for (int upperB = 0; upperB <= maxRange; upperB += increment) {
            resetOps(); // Reset the operation counts for all methods

            // Test numbers up to the current problem size
            for (int j = 0; j <= upperB; j++) {
                String dec = Integer.toString(j); // Convert the number to
a decimal string
                // Check palindromes for each method
                checkPal(1, dec);
                checkPal(2, dec);
                checkPal(3, dec);
                checkPal(4, dec);
            }

            // Print the operation counts for each method after each
increment
            System.out.printf("%d,%d,%d,%d,%d%n", upperB, oppC1, oppC2,
oppC3, oppC4);
        }

        //ORIGINAL CODE, had to change because graphs wouldn't turn out
right
        /*
        long start;
        long end;

        int decimalC;
        int binaryC;
        int both;

        long totalT;
```

```
        System.out.println("Problem Size,Method 1 Operations,Method 2
Operations,Method 3 Operations,Method 4 Operations");
        for (int i = 1; i <= 4; i++) {
            //Reset string and counters, the operator counts for accuracy
and start timer
            String str = "";
            decimalC = 0;
            binaryC = 0;
            both = 0;

            resetOps();
            start = System.currentTimeMillis();

            //Test each num 1000000 times as decimal and binary using
utility method
            for (int j = 0; j < 1000000; j++) {
                String dec = Integer.toString(j);
                String bin = utility(dec);

                //Check if palidrome for both
                boolean decPal = checkPal(i, dec);
                boolean binPal = checkPal(i, bin);

                //Increment if palidrome
                if (decPal) {
                    decimalC++;
                }
                if (binPal) {
                    binaryC++;
                }
                if (decPal && binPal) {
                    both++;
                }
            }
            //Work out how much time it took
            end = System.currentTimeMillis();
            totalT = end - start;

            //Print result
                str += "-= Method " + i + " =-\n";
                str += "Decimal Palindromes: " + decimalC + "\n";
                str += "Binary Palindromes: " + binaryC + "\n";
                str += "Both: " + both + "\n";
                str += "Time taken: " + totalT + "ms\n";
                str += "Operations: " + getOppC(i) + "\n";
                str += "-= * =-\n\n";


            System.out.println(str);
            }
        }
        */
    }
}
```

Testing

Here is my outputs and graphs:

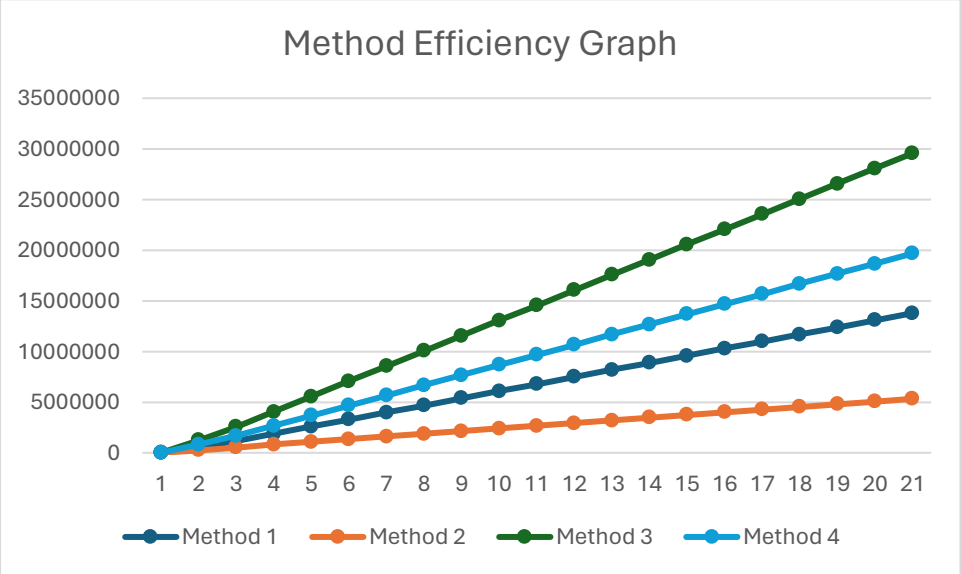
```
/Users/oisinmcl/Library/Java/Java
-= Method 1 =-
Decimal Palindromes: 1999
Binary Palindromes: 2000
Both: 20
Time taken: 1202ms
Operations: 53680632
-= * =-


-= Method 2 =-
Decimal Palindromes: 1999
Binary Palindromes: 2000
Both: 20
Time taken: 325ms
Operations: 13318678
-= * =-


-= Method 3 =-
Decimal Palindromes: 1999
Binary Palindromes: 2000
Both: 20
Time taken: 2114ms
Operations: 111353266
-= * =-
```

```
-= Method 4 =-
Decimal Palindromes: 1999
Binary Palindromes: 2000
Both: 20
Time taken: 670ms
Operations: 78520948
-= * =-
```

As you can see from the outputs, each method works as each of the palindromes are the same number. Method 2 is the most efficient with the least amount of time took and method 3 is the least efficient with the amount of time took.



Each point represents number of operations at each 50’000 intervals from the range 0 – 1,000,000. From the graph you can see method 2 is the most efficient (orange) and method 3 is the least efficient (green).