Name: Paulius Zabinskas Student ID: 20120267

Assignment 3

- **Problem Statement:**
  1. Must design 4 methods/classes that would return TRUE if a given string, is palindrome. The design of all methods definition is given.
  2. Design a UTILITY method taking argument string, converting to binary string.
  3. Develop a TEST method to test all methods with scale.
  4. Determine Big O, graph it.

- **Analysis and Design Notes:**
  1. From top-down design create interfaces for all methods.
  2. Implement structural similarities to Method classes.
  3. Design method classes to given description.
  4. Create working utility method for conversion from decimal to binary.
  5. Create test method.

- **Code:**
  Method 3 is using Stack Array, Stack Queue Classes which will not be mentioned but will be added to code submission.

```
import java.util.Scanner;

public class Main {



    public static void main(String[] args) {
        MethodsClass methodOne = new MethodOne();
        MethodsClass methodTwo = new MethodTwo();
        MethodsClass methodThree = new MethodThree();
        MethodsClass methodFour = new MethodFour();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter number of iterations For test");

        while (!sc.hasNextInt()){
            System.out.println("is not int");
            sc = new Scanner(System.in);
        }
        int testTimes = sc.nextInt();


        System.out.println("\nMethod One: ");test(methodOne, testTimes);
        System.out.println("\nMethod Two: ");test(methodTwo, testTimes);
        System.out.println("\nMethod Three: ");test(methodThree, testTimes);
        System.out.println("\nMethod Four: ");test(methodFour, testTimes);

    }

    public static String utility(String input){
        int decimal = Integer.parseInt(input);
        StringBuilder binary = new StringBuilder();

        // Handle the case of 0 separately
        if (decimal == 0) {
            return "0";
        }
```

```
        // Convert the decimal number to binary using repeated division by 2
        while (decimal > 0) {
            int remainder = decimal % 2;
            binary.insert(0, remainder); // insert the remainder at the
beginning of the StringBuilder
            decimal /= 2;
        }

        return binary.toString();
    }

    public static void test(MethodsClass method, int testTimes){
        int count = 0;
        long start = System.currentTimeMillis();
        for(int i = 0; i< testTimes; i++){

            String j = Integer.toString(i);

            if (method.Method(j)){
                count++;
            }
        }
        long end = System.currentTimeMillis();
        System.out.println("Decimal Numbers:");
        System.out.println("\tPalindrome Count: "+ count);
        System.out.println("\tTime in milliseconds Count: "+ (end-start)+"
ms");
        System.out.println("\tOperation Count: "+ method.get_oCount() );
        System.out.println("\tO(n)): "+ method.getNCount() );
        count = 0;

        start = System.currentTimeMillis();
        for(int i = 0; i<= 1000000; i++ ){

            String j = Integer.toString(i);

            if (method.Method(utility(j))){
                count++;
            }
        }
        end = System.currentTimeMillis();
        System.out.println("Binary Numbers:");
        System.out.println("\tPalindrome Count: "+ count);
        System.out.println("\tTime in milliseconds Count: "+ (end-start)+"
ms");
        System.out.println("\tOperation Count: "+ method.get_oCount());
        System.out.println("\tO(n) Count: "+ method.getNCount() );
    }
}
```

```
public class MethodOne implements MethodsClass {

    protected int oCount = 0;
    protected int nCount = 0;
    public MethodOne() {

    }
    @Override
    public boolean Method(String input){
        String reverseString = "";
        oCount += 1;
        for(int i = input.length()-1; i > -1; i--){
            reverseString += input.charAt(i);
            oCount+=4;
            nCount ++;
```

```
        }
        oCount++;
        return reverseString.equals(input);
    }
    public int get_oCount(){
        return this.oCount;
    }
    public int getNCount(){
        return nCount;
    }
}
public class MethodTwo implements MethodsClass {
    protected int oCount = 0;
    protected int nCount = 0;
    public MethodTwo(){

    }

    @Override
    public boolean Method(String input) {
        for (int i = 0; i < input.length() / 2; i++) { // 3 operations
            if (input.charAt(i) != input.charAt(input.length() - 1 - i)) { // 2
operations
                oCount ++;
                return false;

            }
            oCount += 5;
            nCount ++;
        }
        return true;
    }

    @Override
    public int get_oCount() {
        return this.oCount;
    }

    @Override
    public int getNCount() {
        return nCount;
    }
}
```

```
public class MethodThree implements MethodsClass {
    protected int oCount = 0;
    protected int nCount = 0;
    public MethodThree() {
    }

    @Override
    public boolean Method(String input) {
        ArrayStack arrayStack = new ArrayStack();
        ArrayQueue arrayQueue = new ArrayQueue();
        oCount +=2;

        for(int i = 0; i < input.length(); i++){
            arrayStack.push(input.charAt(i));
            arrayQueue.enqueue(input.charAt(i));
            oCount += 4;
            nCount ++;
        }

        while(!arrayStack.isEmpty() && !arrayQueue.isEmpty()){
            if(arrayStack.pop() != arrayQueue.dequeue()){
                oCount++;
```

```
                return false;
            }
            nCount ++;
            oCount += 3;
        }

        return true;
    }

    @Override
    public int get_oCount() {
        return this.oCount;
    }

    @Override
    public int getNCount() {
        return nCount;
    }
}
```

```
public class MethodFour implements MethodsClass{
    protected static int oCount = 0;

    protected int nCount = 0;
    public MethodFour(){}

    @Override
    public boolean Method(String input) {

        String reversedStr = reverse(input);
        return input.equals(reversedStr);
    }

    public static String reverse(String input){
        if (input.isEmpty() || input.length() == 1) {
            return input;
        }

        StringBuilder reversedSubstring = new
StringBuilder(reverse(input.substring(1)));
        reversedSubstring.append(input.charAt(0));
        return reversedSubstring.toString();
        // O(n)
    }

    @Override
    public int get_oCount() {
        return this.oCount;
    }

    @Override
    public int getNCount() {
        return nCount;
    }
}
```

- **Testing:**

Name: Paulius Zabinskas Student ID: 20120267

Test_1: Only integers are accepted	Enter number of iterations For test *asd* is not int *asd8* is not int *84asd* is not int *48*
Test_2: Programme runs with shown outputs.	Method One: Decimal Numbers: Palindrome Count: 24 Time in milliseconds Count: 11 ms Operation Count: 1590 O(n)): 325 Binary Numbers: Palindrome Count: 24 Time in milliseconds Count: 2 ms Operation Count: 5538 O(n) Count: 1239 Method Two: Decimal Numbers: Palindrome Count: 24 Time in milliseconds Count: 0 ms Operation Count: 191 O(n)): 14 Binary Numbers: Palindrome Count: 24 Time in milliseconds Count: 0 ms Operation Count: 928 O(n) Count: 137 Method Three: Decimal Numbers: Palindrome Count: 24 Time in milliseconds Count: 3 ms Operation Count: 1840 O(n)): 368 Binary Numbers: Palindrome Count: 24 Time in milliseconds Count: 1 ms Operation Count: 6492 O(n) Count: 1476

```
Method Four:
Decimal Numbers:
    Palindrome Count: 24
    Time in milliseconds Count: 0 ms
    Operation Count: 0
    O(n)): 0
Binary Numbers:
    Palindrome Count: 24
    Time in milliseconds Count: 1 ms
    Operation Count: 0
    O(n) Count: 0

Process finished with exit code 0
```

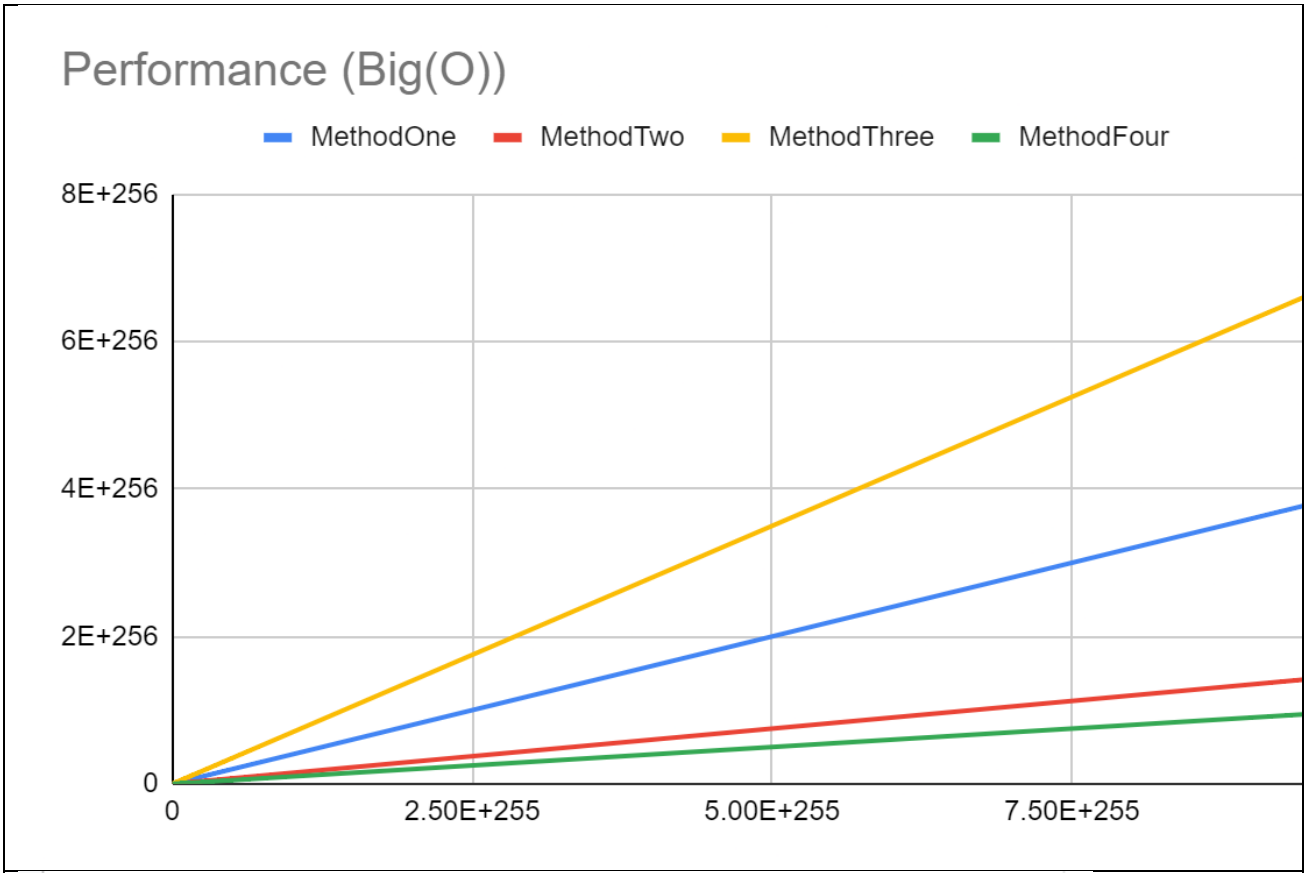
Name: Paulius Zabinskas Student ID: 20120267

Testing Default Option of input (0).

```
Enter Number Of Iterations For Test Or Enter 0 For Default Setting
0

Method One:
Decimal Numbers:
    Palindrome Count: 1999
    Time in milliseconds Count: 173 ms
    Operation Count: 25555560
    O(n)): 5888890
Binary Numbers:
    Palindrome Count: 2000
    Time in milliseconds Count: 694 ms
    Operation Count: 103361346
    O(n) Count: 24840336

Method Two:
Decimal Numbers:
    Palindrome Count: 1999
    Time in milliseconds Count: 27 ms
    Operation Count: 1552446
    O(n)): 110889
Binary Numbers:
    Palindrome Count: 2000
    Time in milliseconds Count: 450 ms
    Operation Count: 7540462
    O(n) Count: 1108892

Method Three:
Decimal Numbers:
    Palindrome Count: 1999
    Time in milliseconds Count: 1379 ms
    Operation Count: 32792455
    O(n)): 6005558
Binary Numbers:
    Palindrome Count: 2000
    Time in milliseconds Count: 1790 ms
    Operation Count: 133595592
    O(n) Count: 25972972
```

Note: I could not find a way to add recursive function calls.

Name: Paulius Zabinskas Student ID: 20120267



Operation Count	4n	3n/2	7n/+3	(n+2)
BIG O	O(n)	O(n)	O(n)	O(n)
n:	MethodOne	MethodTwo	MethodThree	MethodFour
0	0	0	0	2
10	40	15	70	12
100	400	150	700	102
10000	40000	15000	70000	10002
100000000	400000000	150000000	700000000	100000002
1E+16	4E+16	1.5E+16	7E+16	1E+16
1E+32	4E+32	1.5E+32	7E+32	1E+32
1E+64	4E+64	1.5E+64	7E+64	1E+64
1E+128	4E+128	1.5E+128	7E+128	1E+128
1E+256	4E+256	1.5E+256	7E+256	1E+256

- **Graph interpretation:**
  By calculations carried out, Method Two-time complexity is (4n), Method Two-time complexity is (3n/2), Method Three-time complexity is (7n+3), Method Four time complexity is (n+2).

  BIG O: Method One Big O (n), Method Two Big O (n), Method Three Big O (n), Method Four Big O (n).

  According to the analysis Method Four has the smallest Big O method applied testing if a string is a palindrome. Following by Method Two, Method One, Method Three in descending performance order.