Assignment 3 - CT2109 Object Oriented Programming: Data Structures and Algorithms

Daniel Hannon (19484286)

March 2021

1 Problem Analysis

1.1 Overview

For this assignment, you are going to write an application which tests if a sequence of numbers is a palindrome or not. Specifically, you are going to write fourdifferent methods(with meaningful names)which take a String as a parameter and returns a Boolean which represents whether or not the String is a pallindrome

1.2 The Four different methods

- **Method1:** Reverse all characters in a string and then compare the string to the original and determine if it's a pallindrome
- **Method2:** We compare every element on an element by element basis using a loop, first to the last, second to the second last and so on. If at any point two do not match it returns false immediately
- **Method3:** We are going to use ArrayStack and ArrayQueue to compare strings to see if they are valid or not
- **Method4:** We recursively reverse the string and then compare it

1.3 Algorithm analysis

For method1 I figured availing of the String.substring() method would work and encapsulating it in a for loop
Method 2 required the use of two indexes and would run so long as the rightmost index was larger the leftmost index, as any checks of the other way would be redundant.
Method 3: required me to splice the string and store the values in a stack and a queue then pulling them, this required use of the String.charAt() method.
Method 4: required the use of recursive string splicing, I did this by taking the first character of the string out and adding it to the end of the result of the method called with that character spliced out.

1.4 Pre Calculation analysis

I reckon that Method 2 would be fastest as its worst case is if the value given is a pallindrome so it would often not even complete two cycles in the while loop.

1.5 Graphing

In order to get an accurate graph(As shown below), I generated random binary strings of n size in increments of five and then plotted them in matlab the worst amount of operations was seemingly 4n and the best was very small.

2 Code

```java
import java.lang.Math;

public class Main {
  //private static long Method1Steps,Method2Steps,Method3Steps,Method4Steps = 0;
  private static long Method1Time ,Method2Time, Method3Time, Method4Time = 0;

  public static boolean checkPallindromeLinearStringReverse(String val) {
    String valReversed = ""; //Assignment takes 1 Step
    //Method1Steps++;
    //Initialization, method call, subtraction, comparison
    //Method1Steps+=4;
    for(int i = val.length() - 1; i > -1; i--) {
      //Comparison, subtraction
      //Method1Steps+=2;
      valReversed += val.substring(i,i+1);
      //Addition Call of Method
      //Method1Steps+=2;
    }
    //Check equals return true/false
    //Method1Steps +=2;
    return val.equals(valReversed);
  }

  public static boolean checkPallindromeCompareFirstLast(String val) {
    int i = 0;
    int j = val.length() - 1;
    if (j == 0) {
      return true;
    }
    //Two Assignments a Method Call and a subtraction
    //Method2Steps+=3;
    while(i < j) {
      //Comparison
      //Method2Steps++;
      if(val.charAt(i) != val.charAt(j)) {
        //Valid If/else and return
        //Method2Steps+=4;
        return false;
      }
      //Two Method calls and a comparison
      //Method2Steps+=2;
      i++;
      j--;
      //Two Mathematical operations
      //Method2Steps+=2;
    }
    //return statement
    //Method2Steps++;
    return true;
  }

  public static boolean checkPallindromeStackAndQueue(String val) {
    ArrayStack stack = new ArrayStack(1000000);
    ArrayQueue queue = new ArrayQueue(1000000);
    //Two Initialization calls and two constructor calls
    //Method3Steps+=3;
    //Initialization, comparison, method call
    //Method3Steps+=3;
    for(int i = 0; i < val.length(); i++) {
      //Comparison method call, subtraction
      //Method3Steps+=3;
      stack.push(val.charAt(i));
      queue.enqueue(val.charAt(i));
      // Four method invocations
      //Method3Steps+=4;
    }
    while(!stack.isEmpty()) {
      //Method call and invert
      //Method3Steps+=2;
      if ((char)queue.dequeue() != (char)stack.pop()) {
        //Two Method calls, two typecasts, comparison, return value
        //Method3Steps+=6;
        return false;
      }
      //Two Method Calls, Two Typecasts and a comparison
      //Method3Steps+=5;
    }
    //Comparison while loop
    //Method3Steps+=2;
    //Return
    //Method3Steps+=1;
    return true;
  }

  public static boolean checkPallindromeRecursiveStringReverse(String val) {
    String valReversed = recursiveStringReverse(val);
    //Method4Steps+=2; //Initialization + method call
    //Method Call + return value
    //Method4Steps+=2;
    return val.equals(valReversed);
  }

  public static String recursiveStringReverse(String val) {
    if(val.length() == 1) {
      //If/else and Return
      //Method4Steps+=2;
      return val;
    }
    //If/else 3 method calls addition and Return
    //Method4Steps+=5;
    return recursiveStringReverse(val.substring(1)) + val.substring(0,1);
  }

  public static String intToBinaryString(int val) {
    /*
      Basically this gets the index of the highest power of two
      then the next, and so on and pads zeroes in between to make
      it a binary representation of a number
    */
    if(val == 0) {
      return "0";
    }
    String output = "";
    int curr = (int)(Math.log(val)/Math.log(2));
    output += "1";
    val -= Math.pow(2,curr);
    int prev = 0;
    while(val != 0) {
      prev = curr;
      curr = (int)(Math.log(val)/Math.log(2));
      while(prev > curr+1) {
        output+="0";
        prev--;
      }
      output+="1";
      val -= Math.pow(2,curr);
    }
    /*Adds trailing zeroes*/
    if(curr > 0) {
      while(curr > 0) {
        output+="0";
        curr--;
      }
    }
    return output;
  }

  /*This was used to generate the data for the graph values*/
  public static String generateBinaryStringXLength(int val) {
    String inpt = "";
    for(; val > 0; val--) {
      if(Math.random()>Math.random()) {
        inpt+="1";
      } else {
        inpt+="0";
      }
    }
    return inpt;
  }
  public static void main(String[] args) {
    System.out.println("length\tmethod1\tmethod2\tmethod3\tmethod4");
    for(int i = 1; i < 1000000; i++) {
      String binaryString = intToBinaryString(i);
      String numstring = String.valueOf(i);
      //String binaryString = intToBinaryString(i);
      //String numstring = Integer.toString(num);
      //Method1Time = System.nanoTime();
      //boolean valid1 = checkPallindromeStackAndQueue(binaryString);
      //Method1Time = System.nanoTime() - Method1Time;
      //Method2Time = System.nanoTime();
      boolean valid2 = checkPallindromeCompareFirstLast(binaryString);
      //Method2Time = System.nanoTime() - Method2Time;
      //Method3Time = System.nanoTime();
      //boolean valid3 = checkPallindromeLinearStringReverse(binaryString);
      //Method3Time = System.nanoTime() - Method3Time;
      //Method4Time = System.nanoTime();
      //boolean valid4 = checkPallindromeRecursiveStringReverse(binaryString);
      //Method4Time = System.nanoTime() - Method4Time;
      /*if(valid1&&valid2&&valid3&&valid4) {
        System.out.println(binaryString + "Is a pallindrome");
      }
      System.out.println("Times: "+Method1Time+"\t"+Method2Time+"\t"+Method3Time+"\t"+Method4Time);*/
      boolean valid_num_string = checkPallindromeCompareFirstLast(numstring);
      if(valid_num_string && valid2) {
        System.out.println(numstring + " & " + binaryString + " Are Both Pallindromes");
      }
      /*if(valid1) {
        System.out.println(binaryString + " is a pallindrome");
      }
      if(valid_num_string) {
        System.out.println(numstring + " is a pallindrome");
      }
      System.out.println(binaryString.length() + "\t"+Method1Steps+"\t"+Method2Steps+"\t"+Method3Steps+
      Method1Steps = 0;
      Method2Steps = 0;
      Method3Steps = 0;
      Method4Steps = 0;*/
    }
  }
```

```
}
```

3 Outputs

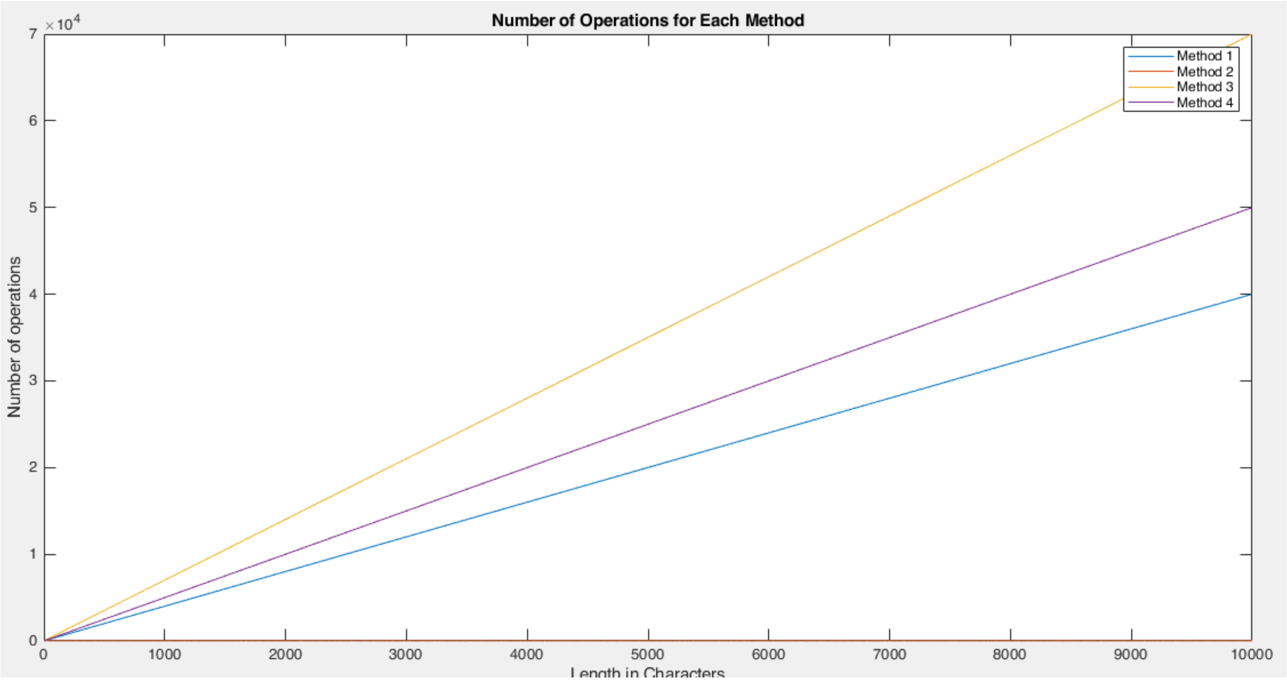
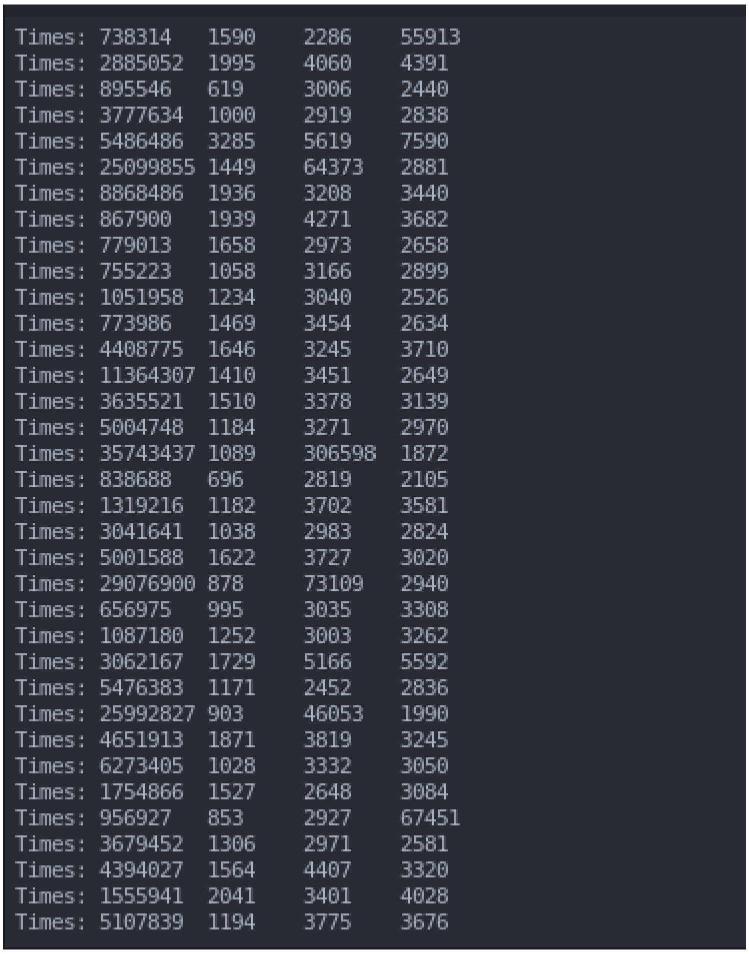
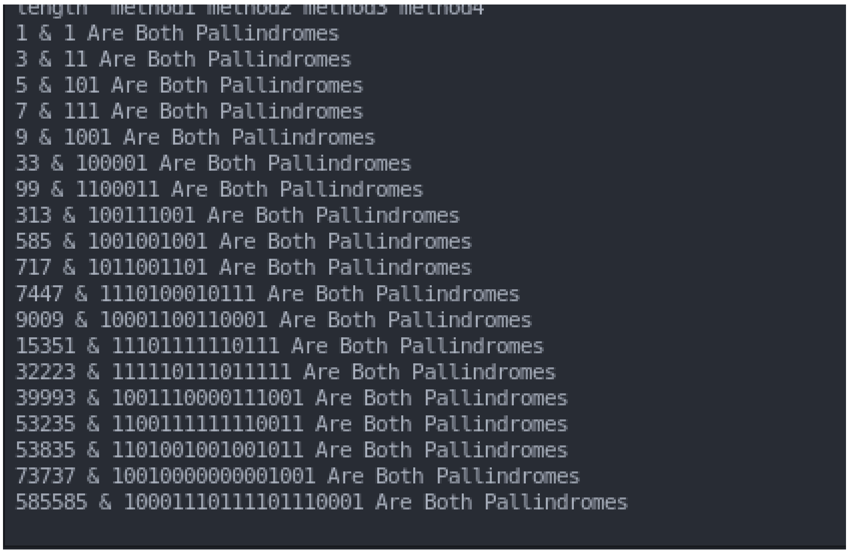


Figure 1: The four methods graphed by number of operations



```
Times: 738314   1590    2286    55913
Times: 2885052  1995    4060    4391
Times: 895546   619     3006    2440
Times: 3777634  1000    2919    2838
Times: 5486486  3285    5619    7590
Times: 25099855 1449    64373   2881
Times: 8868486  1936    3208    3440
Times: 867900   1939    4271    3682
Times: 779013   1658    2973    2658
Times: 755223   1058    3166    2899
Times: 1051958  1234    3040    2526
Times: 773986   1469    3454    2634
Times: 4408775  1646    3245    3710
Times: 11364307 1410    3451    2649
Times: 3635521  1510    3378    3139
Times: 5004748  1184    3271    2970
Times: 35743437 1089    306598  1872
Times: 838688   696     2819    2105
Times: 1319216  1182    3702    3581
Times: 3041641  1038    2983    2824
Times: 5001588  1622    3727    3020
Times: 29076900 878     73109   2940
Times: 656975   995     3035    3308
Times: 1087180  1252    3003    3262
Times: 3062167  1729    5166    5592
Times: 5476383  1171    2452    2836
Times: 25992827 903     46053   1990
Times: 4651913  1871    3819    3245
Times: 6273405  1028    3332    3050
Times: 1754866  1527    2648    3084
Times: 956927   853     2927    67451
Times: 3679452  1306    2971    2581
Times: 4394027  1564    4407    3320
Times: 1555941  2041    3401    4028
Times: 5107839  1194    3775    3676
```

Figure 2: A sample of the execution times for each method



```
length  method1 method2 method3 method4
1 & 1 Are Both Pallindromes
3 & 11 Are Both Pallindromes
5 & 101 Are Both Pallindromes
7 & 111 Are Both Pallindromes
9 & 1001 Are Both Pallindromes
33 & 100001 Are Both Pallindromes
99 & 1100011 Are Both Pallindromes
313 & 100111001 Are Both Pallindromes
585 & 1001001001 Are Both Pallindromes
717 & 1011001101 Are Both Pallindromes
7447 & 1110100010111 Are Both Pallindromes
9009 & 10001100110001 Are Both Pallindromes
15351 & 11101111110111 Are Both Pallindromes
32223 & 111110111011111 Are Both Pallindromes
39993 & 1001110000111001 Are Both Pallindromes
53235 & 1100111111110011 Are Both Pallindromes
53835 & 1101001001001011 Are Both Pallindromes
73737 & 10010000000001001 Are Both Pallindromes
585585 & 10001110111101110001 Are Both Pallindromes
```

Figure 3: List of dual pallindromes