

## CT2109 – Assignment 2 – Luke Gibbons - 15553883:

### Double-base Palindromes and Complexity Analysis

#### Problem Statement:

The problem requires me to code three methods to detect whether or not a number (both decimal and binary) is a palindrome or not. The methods are a reversed string comparison, a char comparison, and a stack and queue comparison. After I will need to extensively test each method and determine which is the fastest and most efficient.

#### Analysis and Design:

To begin I will need to look at writing up some basic pseudo code for the 3 methods.

##### 1. String reverse method

I believe the most efficient way of reversing a string would be to create a new empty string and, using a loop, go backwards through the original inputted string. For each iteration the char in the string is added to the reversed string.

---

```
String reverse = ""
```

```
J = length of input string - 1
```

```
While(J >= 0)
```

```
    Reverse += char at J in input string.
```

```
    J--
```

```
Return (input string is equal to reverse)
```

---

## 2. Char comparison method

For this method I will use 2 increments, one to be positioned at the strings first char and the other at its last. I will then loop until the first increment is equal to the second increment, comparing them each time. If at any instance they are not equal false will be returned.

---

I = 0

J = length of string – 1

While(I <= J)

    If(char at I != char at J)

        Return false

    I++

    J--

Return true

---

## 3. Stack and queue comparison

This method will require 2 loops. One is for adding the elements to the stack and queue, and the other is for taking them out and comparing them.

---

ArrayStack stack = new ArrayStack

ArrayQueue queue = new ArrayQueue

I = 0

While(I < string length)

    Push char at I into stack

Enqueue char at I into queue

While(the stack is not empty) //or queue

Char c1 = stack.pop

Char c2 = queue.dequeue

If(c1 != c2)

Return false

Return true

---

## Counting Primitive Operations

Now I will need to look at the primitive operations in each method. Allow n to be the length of the number (string) entered.

### 1. String method

String reverse = "" +1

J = length of input string – 1

+1 for =, +1 for method and +1 for subtraction... +3

While(J >= 0) Will run n+1 times (+1 for last loop to establish false)

Reverse += char at J in input string. 3n (1 +, 1 =, 1 method)

J - - 2n

Return (input string is equal to reverse)

+1 for return, +1 for compare... +2

---

Therefore the final equation will be:

1+3+n+1+3n+2n+2

= 6n + 7

## 2. Char comparison

I = 0 +1

J = length of string – 1 +3 (1 =, 1 -, 1 method)

While(I <= J) If n is even  $n = n/2$ , if it is odd  $n = n/2 + \frac{1}{2}$  ... also +1  
here for final check

If(char at I != char at J) +3n (2 methods and 1 compare)

Return false +1(not needed for worst case)

I++ +2n

J- - +2n

Return true +1

---

So for worst case with n being an even number:

$$1+3+(n/2 + 1)+3(n/2)+2(n/2)+2(n/2)+1$$

$$= \underline{4n+6}$$

And for an odd number:

$$1+3+(n/2+1/2)+3(n/2+1/2)+2(n/2+1/2)+2(n/2 + 1/2)+1$$

$$= \underline{4n+9}$$

## 3. Stack and queue comparison

ArrayStack stack = new ArrayStack() +2... assuming +1 for object creation

ArrayQueue queue = new ArrayQueue() +2

I = 0 +1

While( $l < \text{string length}$ )  $2n+2$  ( $2n$  as 1 compare and 1 method)

Push char at  $l$  into stack  $+2n$  methods (push and get char method)

Enqueue char at  $l$  into queue  $+2n$  methods

While(the stack is not empty) //or queue  $2n+2$

Char  $c1 = \text{stack.pop}$   $+2n(1 = \text{and 1 method})$

Char  $c2 = \text{queue.dequeue}$   $+2n$

If( $c1 \neq c2$ )  $+1n$

Return false  $+1(\text{won't use})$

Return true  $+1$

---

So worst case scenario:

$2+2+1+2n+2+2+2n+2n+2n+2+2n+2n+n+1$

$= 13n+10$

## Converting decimal to binary

This is an extremely easy piece of code to write as it will only require a couple of lines. I will firstly need to convert the string into an integer using `parseInt (string)`. Lastly, I will convert it into a binary string using `toBinaryString (int)` and return it.

## Main method implementation

To make testing easier I am considering creating a GUI using `JOptionPane`. I could ask the user what method they would like to use and the number of numbers to calculate from 0 before displaying the results like time taken, number of palindromes etc.

Another thing I will need to work out is how I will count the number of palindromes. Something like the below pseudo code could be a good guideline for my main method.

---

```
Int palindromes, binary_palindromes, dbl_palindromes
Long timeBefore, timeAfter
Int loops, i=0

String gui = JOptionPaneInput("Choose a method", display 3
methods);
Loops = JOptionPaneInput("Choose amount of numbers")

If(input equals StringMethod)
    timeBefore = milliseconds since start
    while(i <= loops)
        if(StringMethod(i) is true)
            palindromes++
        if(StringMethod(convertToBinary(i)) is true)
            binary_palindromes++
        if(both above are true)
            dbl_palindromes++
        i++
    timeAfter = milliseconds since start

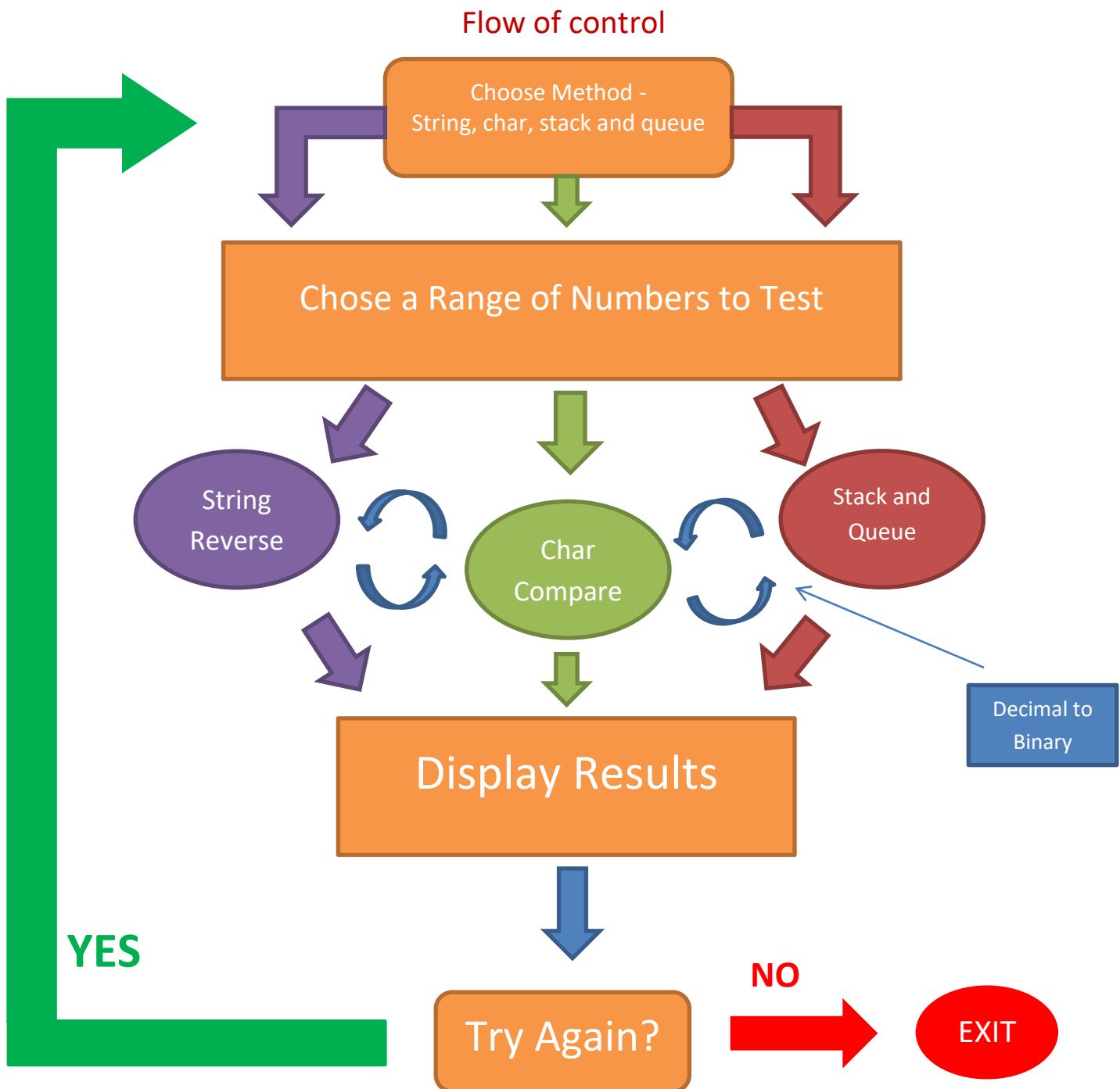
else if(input equals CharMethod)
    ..... Same as Above .....

Else //run stack_queue method
    ..... Same as Above .....

//Finally print out results
```

```
JOptionPane.showMessageDialog("Time taken = " + (timeAfter -  
timeBefore) + "\n" + "Number of palindromes " + palindromes +  
"Number of binary..." + binary_palindromes..... )
```

---







```

        palindromes++;
        palindrome = true;
    }
    if (charCheck(decimal2Binary(Integer.toString(i)))) {
        binaryPalindromes++;
        binPalindrome = true;
    }
    if (palindrome && binPalindrome) {
        doublePalindrome++;
    }
    i++;
    palindrome = false;
    binPalindrome = false;
}
timeAfter = System.currentTimeMillis(); //get time it finished
}
//if stack and queue comparison is chosen
else {
    timeBefore = System.currentTimeMillis(); //get time it starts
    while (i <= loops) {
        if (stack_queueCheck(Integer.toString(i))) {
            palindromes++;
            palindrome = true;
        }
        if (stack_queueCheck(decimal2Binary(Integer.toString(i)))) {
            binaryPalindromes++;
            binPalindrome = true;
        }
        if (palindrome && binPalindrome) {
            doublePalindrome++;
        }
        i++;
        palindrome = false;
        binPalindrome = false;
    }
    timeAfter = System.currentTimeMillis(); //get time it finished
}

//Output the results
JOptionPane.showMessageDialog(null, "Using: " + input + "\n" +
    "Time Taken: " + (timeAfter - timeBefore) + " milliseconds.\n"
+
    "From 0 to " + loops + " there are" + "\n" +
    palindromes + " palindromes." + "\n" +
    binaryPalindromes + " binary palindromes." + "\n" +
    doublePalindrome + " double palindromes.");

//Try again prompt
int reply = JOptionPane.showConfirmDialog(null, "Would you like to try
again?", "Try Again?", JOptionPane.YES_NO_OPTION);
if (reply == JOptionPane.NO_OPTION) {
    again = false;
}

}

}

//Static method for: Palindrome Method 1 - Reverses input string and compares
to see if equal
//Takes a String as a parameter and return a Boolean value
public static boolean String_stringCheck(String str){
    String reversed = "";
    int i = str.length() - 1; //set i to last string char position

```

```

        while(i>=0){
            reversed += str.charAt(i); //add char to reversed string starting from
end of str
            i--;
        }

        //String reversed = new StringBuilder(str).reverse().toString();

        return str.equals(reversed); //return true if it they are equal
    }

    //Static method for: Palindrome Method 2 - Compare each character in string to
its opposite to test for equality
    //Takes a String as a parameter and return a Boolean value
    public static boolean charCheck(String str){

        int i = 0; //i will be at start of string
        int j = str.length() - 1; //j will be at end
        while(i<=j){
            if(str.charAt(i) != str.charAt(j)){ //if at any stage !=, it is not a
palindrome
                return false;
            }
            i++;
            j--;
        }

        return true;
    }

    //Static method for: Palindrome Method 3
    //Takes a String as a parameter and return a Boolean value
    public static boolean stack_queueCheck(String str){
        ArrayStack stack = new ArrayStack(); //Initialise a stack
        ArrayQueue queue = new ArrayQueue(); //initialise a queue
        int i = 0;

        //push each element into stack and enqueue it into the queue
        while(i<str.length()){
            stack.push(str.charAt(i));
            queue.enqueue(str.charAt(i));
            i++;
        }

        //for every element in the stack..
        while(!stack.isEmpty()){
            char c1 = (char)stack.pop(); //pop the element from the stack
            char c2 = (char)queue.dequeue(); //dequeue the element from queue
            if(c1 != c2){ //Compare the two. If they are not equal - not a
palindrome
                return false;
            }
        }

        return true;
    }

    //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 public String decimal2Binary(String str){
        int num = Integer.parseInt(str); //convert string to int

        return Integer.toBinaryString(num); //return the binary string
    }

```

```
}  
}
```

## Testing

### 1. Does each method correctly detect palindromes?

Using 2 strings (1 palindrome, 1 not a palindrome). The results were as follows:

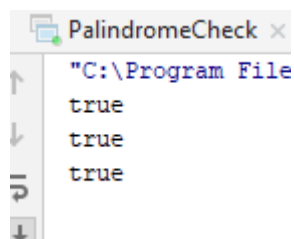
#### Inputted Code:

```
System.out.println(String_stringCheck("133494331"));  
System.out.println(charCheck("133494331"));  
System.out.println(stack_queueCheck("133494331"));
```

#### Expected result:

3 trues

#### Actual Result:



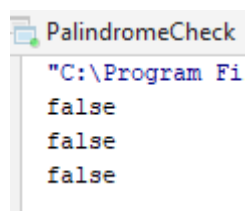
#### Inputted code:

```
System.out.println(String_stringCheck("59732"));  
System.out.println(charCheck("59732"));  
System.out.println(stack_queueCheck("59732"));
```

#### Expected Result:

3 false statements

#### Actual result:



## 2. Does the decimal2Binary method work correctly?

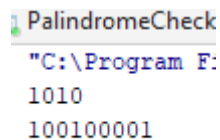
### Inputted Code:

```
String s = decimal2Binary("10");  
String s1 = decimal2Binary("289");  
System.out.println(s + "\n" + s1);
```

### Expected result:

1010 and 100100001

### Actual result:



```
PalindromeCheck  
"C:\Program F:  
1010  
100100001
```

## 3. Does the palindrome count work?

To see if the counts for palindromes, binary palindromes and double palindromes work I will run the program using a small range of value (0-10) so that I can easily calculate if it is correct.



### Inputted code:

None needed

### Expected result:

There should be 10 palindromes (0-9), 6 binary palindromes (0,1,3,5,7,9) and 6 double palindromes.

### Actual result:

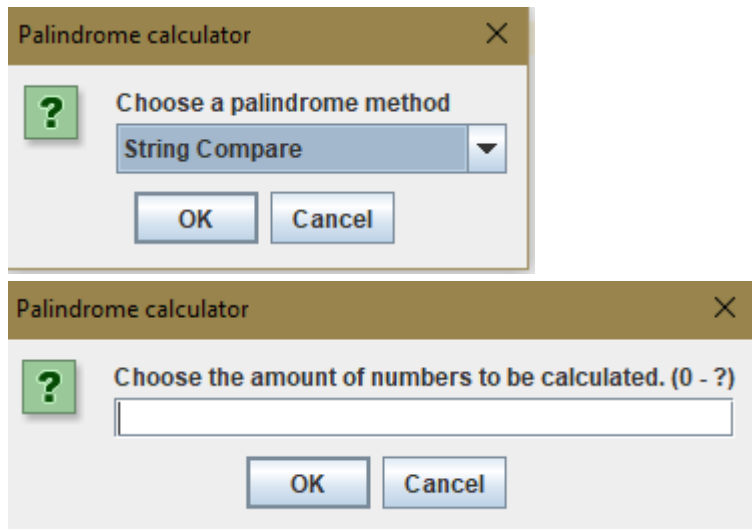
 Using: String Compare Time Taken: 5 milliseconds. From 0 to 10 there are 10 palindromes. 6 binary palindromes. 6 double palindromes.	 Using: Character Compare Time Taken: 6 milliseconds. From 0 to 10 there are 10 palindromes. 6 binary palindromes. 6 double palindromes.	Using: Stack & Queue Compare Time Taken: 642 milliseconds. From 0 to 10 there are 10 palindromes. 6 binary palindromes. 6 double palindromes.
---	--	--

#### 4. Do all methods have the same result?

Verified by above test

#### 5. Do all the JOptionPane GUI parts work as they should?

This test is already verified by test number 3. The GUI allows user to choose method and number of loops to be executed.



(I was unable to capture an image of the drop down menu as it kept closing)

#### 6. Does the try again loop work?

**Inputted code:**

None

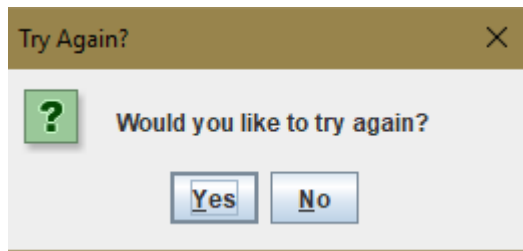
**Expected result:**

When yes is clicked, the user is brought back to the 'choose a palindrome' screen as shown above in test 5.

If no is clicked then the program should exit.

**Actual result:**

'Yes' brings user to the start again ('choose a palindrome') and 'no' exits the program.



## Speed analysis

Below are the speeds in milliseconds that each method took to test numbers 0 to 1 million for palindromes (decimal, binary and double).

	Test 1	Test 2	Test 3	Average
String	1174	1056	930	1053.4
Char	376	275	363	338
Stack and Queue	3707	3274	3435	3472

## Palindrome count from 0 – 1 million

- 1999 palindromes
- 2000 binary palindromes
- 20 double palindromes

## Primitive Operations

To count the primitive operations I had to add additional code. I didn't add them to begin with as I didn't want it to alter the execution speeds of the methods.

### Changes in code (full code at very end)

I started by adding global variable 'ops':

```
//operation counts for each method  
static int ops;
```

I pondered on the idea of using 3 separate variables, 1 for each method, but I couldn't find an easy way to incorporate it into my code so I stuck with using the one.

Inside the while loop I set ops = 0; so that for every rerun it starts from 0. At the bottom of the loop I added a line to the results GUI to display the operations.

```
JOptionPane.showMessageDialog(null, "Using: " + input + "\n" +
    "Time Taken: " + (timeAfter - timeBefore) + " milliseconds.\n" +
    "From 0 to " + loops + " there are" + "\n" +
    palindromes + " palindromes." + "\n" +
    binaryPalindromes + " binary palindromes." + "\n" +
    doublePalindrome + " double palindromes." + "\n" +
    ops + " operations completed.");
```

Finally, I added the increments of ops to each method using my calculations in the analysis stage to guide me.

### String method:

```
public static boolean String_stringCheck(String str){
    String reversed = "";
    ops++;

    int i = str.length() - 1; //set i to last string char position
    ops+=3;

    while(i>=0){
        ops++; //for loop
        reversed += str.charAt(i); //add char to reversed string starting from end
of str
        ops+=3;
        i--;
        ops+=2;
    }
    ops++; //end of loop checks and sees false

    ops+=2;
    return str.equals(reversed); //return true if it they are equal
}
```

### Char method:

```
public static boolean charCheck(String str){
    int i = 0; //i will be at start of string
    ops++;

    int j = str.length() - 1; //j will be at end
    ops+=3;
```

```

    while(i<=j){
        ops++; //for loop
        ops+=3;
        if(str.charAt(i) != str.charAt(j)){//if at any stage !=, it is not a
palindrome
            ops++;
            return false;
        }
        i++;
        ops+=2;
        j--;
        ops+=2;
    }
    ops++; //for last operation in loop- false
    ops++; //for return statement
    return true;
}

```

## Stack and Queue method:

```

public static boolean stack_queueCheck(String str){
    ArrayStack stack = new ArrayStack(); //Initialise a stack
    ops+=2;
    ArrayQueue queue = new ArrayQueue();//initialise a queue
    ops+=2;

    int i = 0;
    ops++;

    //push each element into stack and enqueue it into the queue
    while(i<str.length()){
        ops+=2; //for loop statement = true
        stack.push(str.charAt(i));
        ops+=2;
        queue.enqueue(str.charAt(i));
        ops+=2;
        i++;
        ops+=2;
    }
    ops+=2; //for loop statement = false

    //for every element in the stack..
    while(!stack.isEmpty()){
        ops+=2; //for loop = true
        char c1 = (char)stack.pop(); //pop the element from the stack
        ops+=2; //could be +=3 (char) cast?
        char c2 = (char)queue.dequeue(); //dequeue the element from queue
        ops+=2; //could be +=3 with (char) cast?
        ops++; //for if statement
        if(c1 != c2){//Compare the two. If they are not equal - not a palindrome
            ops++; //for return statement
            return false;
        }
    }
    ops+=2; //for loop = false;
    ops++; //for return statement
    return true;
}

```



## Does it work?

Using the equations that I created earlier for the worst case, I can test the numbers 0-10 (only decimals) to get a rough estimate of the number of primitive operation each method should use.

### Expected output:

#### String method

$6n+7$ :

$N = 1$  for 10 numbers and  $N = 2$  for 1 number.

$10[6(1) + 7] + 1[6(2) + 7] = 149$  operations (roughly)

#### Char method:

Even –  $4n+6$ , odd –  $4n+9$ :

$N$  is odd (1) for 10 numbers and even (2) for 1.

$10[4(1) + 9] + 1[4(2) + 6] = 144$  operations (roughly)

#### Stack and Queue method:

$13n + 10$ :

$10[13(1) + 10] + 1[13(2) + 10] = 266$  operations (roughly)

### Actual results:

#### String method

149 operations completed.

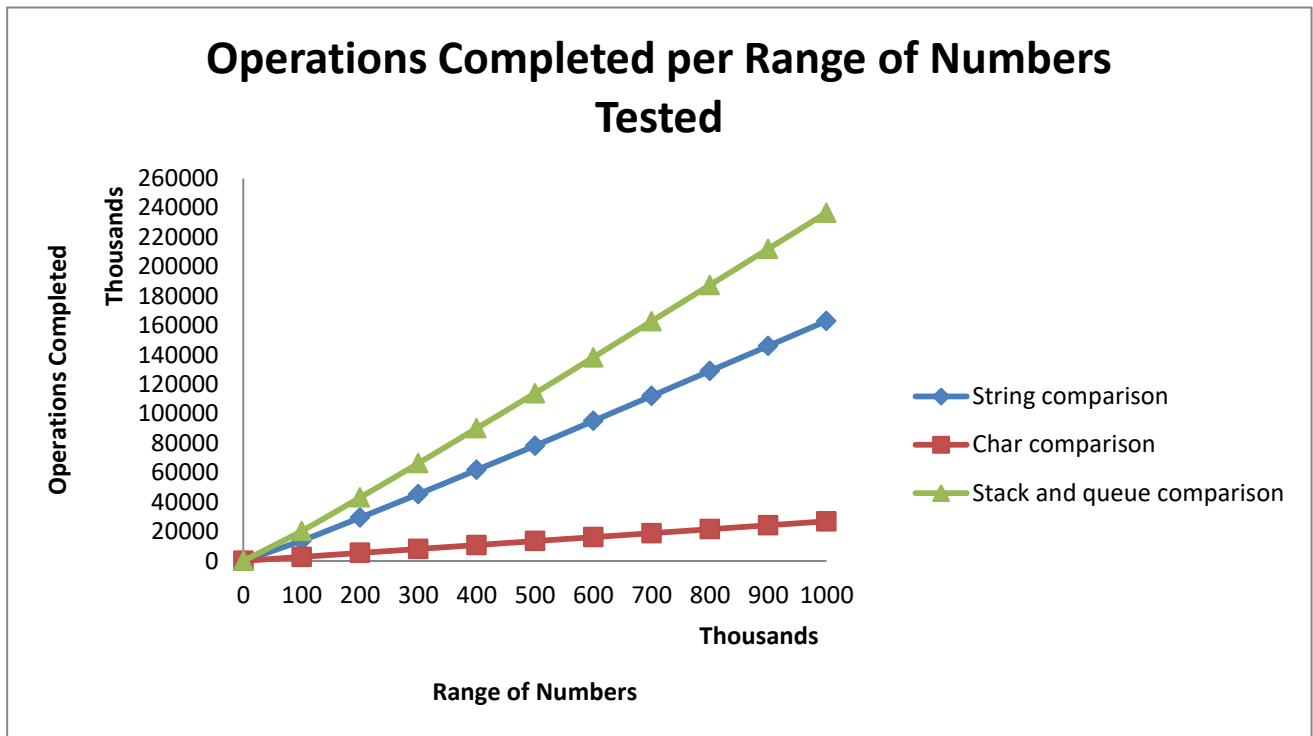
#### Char method:

149 operations completed.

#### Stack and Queue method:

281 operations completed.

## Graphing the results



## Conclusions

From both the graph and the speed test we can conclude that the char comparison method was by far the most efficient and quickest of the three. The char method has an incredibly low slope in the graph compared to the string reverse and the stack and queue method which have much sharper slopes. This shows that the char method requires much less operations as the range of numbers increases in contrast to the other two methods. The next most efficient method is the string reversal method. This has a slope not too far off of the stack and queue method but its speed is much faster with an average of 1053.5 milliseconds compared to the stack and queue's 3472 millisecond average execution speed. The stack and queue method was far off the other two methods and the graph shows this with its slope being the greatest. However, this comes and no great surprise as it requires two loops, one to enter the numbers into the stack and queue and the other to pull them out and compare them. This ramped up its number of operations and caused it to be far inferior to the char method and the string reverse method.

## Full Code changed for testing:

```
import javax.swing.*;

public class PalindromeCheck {

    //operation counts for each method... not in use until testing
    static int ops;

    //Main Method
    public static void main (String[] args)
    {

        int loops; //number of numbers to test, user inputted
        long timeBefore=0, timeAfter=0;
        boolean palindrome = false, binPalindrome = false; //to monitor any double
palindromes
        boolean again = true; //For try again feature
        String[] choices = {"String Compare", "Character Compare", "Stack & Queue
Compare"}; //Choices of algorithm

        while(again) {
            int i = 0;
            //Counts for palindrome types
            int doublePalindrome = 0;
            int palindromes = 0;
            int binaryPalindromes = 0;
            ops = 0;

            //GUI to choose algorithm and amount of numbers to test
            String input = (String) JOptionPane.showInputDialog(null, "Choose a
palindrome method",
                "Palindrome calculator", JOptionPane.QUESTION_MESSAGE, null,
choices, choices[0]);
            loops = Integer.parseInt(JOptionPane.showInputDialog(null, "Choose the
amount of numbers " +
                "to be calculated. (0 - ?)", "Palindrome calculator",
JOptionPane.QUESTION_MESSAGE));

            //if string comparison is chosen
            if (input.equals(choices[0])) {
                timeBefore = System.currentTimeMillis(); //Get time it starts
                while (i <= loops) {
                    //If it is a palindrome
                    if (String_stringCheck(Integer.toString(i))) {
                        palindromes++;
                        palindrome = true;
                    }
                    //if its converted binary number is a palindrome
                    if (String_stringCheck(decimal2Binary(Integer.toString(i)))) {
                        binaryPalindromes++;
                        binPalindrome = true;
                    }
                    //if both the decimal and binary are palindromes
                    if (palindrome && binPalindrome) {
                        doublePalindrome++;
                    }
                    i++;
                    palindrome = false;
                    binPalindrome = false;
                }
                timeAfter = System.currentTimeMillis(); //get time it finishes
            }
            //If char comparison is chosen
            else if (input.equals(choices[1])) {
                timeBefore = System.currentTimeMillis(); //get time it starts
                while (i <= loops) {
```

```

        if (charCheck(Integer.toString(i))) {
            palindromes++;
            palindrome = true;
        }
        if (charCheck(decimal2Binary(Integer.toString(i)))) {
            binaryPalindromes++;
            binPalindrome = true;
        }
        if (palindrome && binPalindrome) {
            doublePalindrome++;
        }
        i++;
        palindrome = false;
        binPalindrome = false;
    }
    timeAfter = System.currentTimeMillis(); //get time it finished
}
//if stack and queue comparison is chosen
else{
    timeBefore = System.currentTimeMillis(); //get time it starts
    while (i <= loops) {
        if (stack_queueCheck(Integer.toString(i))) {
            palindromes++;
            palindrome = true;
        }
        if (stack_queueCheck(decimal2Binary(Integer.toString(i)))) {
            binaryPalindromes++;
            binPalindrome = true;
        }
        if (palindrome && binPalindrome) {
            doublePalindrome++;
        }
        i++;
        palindrome = false;
        binPalindrome = false;
    }
    timeAfter = System.currentTimeMillis(); //get time it finished
}

//Output the results
JOptionPane.showMessageDialog(null, "Using: " + input + "\n" +
    "Time Taken: " + (timeAfter - timeBefore) + " milliseconds.\n"

+
    "From 0 to " + loops + " there are" + "\n" +
    palindromes + " palindromes." + "\n" +
    binaryPalindromes + " binary palindromes." + "\n" +
    doublePalindrome + " double palindromes." + "\n" +
    ops + " operations completed.");

//Try again prompt
int reply = JOptionPane.showConfirmDialog(null, "Would you like to try
again?", "Try Again?", JOptionPane.YES_NO_OPTION);
if (reply == JOptionPane.NO_OPTION) {
    again = false;
}

}

}

//Static method for: Palindrome Method 1 - Reverses input string and compares
to see if equal
//Takes a String as a parameter and return a Boolean value
public static boolean String_stringCheck(String str){
    String reversed = "";
    ops++;

```

```

        int i = str.length() - 1; //set i to last string char position
        ops+=3;

        while(i>=0){
            ops++; //for loop
            reversed += str.charAt(i); //add char to reversed string starting from
end of str
            ops+=3;
            i--;
            ops+=2;
        }
        ops++; //end of loop checks and sees false

        ops+=2;
        return str.equals(reversed); //return true if it they are equal
    }

    //Static method for: Palindrome Method 2 - Compare each character in string to
    its opposite to test for equality
    //Takes a String as a parameter and return a Boolean value
    public static boolean charCheck(String str){
        int i = 0; //i will be at start of string
        ops++;

        int j = str.length() - 1; //j will be at end
        ops+=3;

        while(i<=j){
            ops++; //for loop
            ops+=3;
            if(str.charAt(i) != str.charAt(j)){//if at any stage !=, it is not a
palindrome
                ops++;
                return false;
            }
            i++;
            ops+=2;
            j--;
            ops+=2;
        }
        ops++; //for last operation in loop- false
        ops++; //for return statement
        return true;
    }

    //Static method for: Palindrome Method 3
    //Takes a String as a parameter and return a Boolean value
    public static boolean stack_queueCheck(String str){
        ArrayStack stack = new ArrayStack(); //Initialise a stack
        ops+=2;
        ArrayQueue queue = new ArrayQueue(); //initialise a queue
        ops+=2;

        int i = 0;
        ops++;

        //push each element into stack and enqueue it into the queue
        while(i<str.length()){
            ops+=2; //for loop statement = true
            stack.push(str.charAt(i));
            ops+=2;
            queue.enqueue(str.charAt(i));
            ops+=2;
            i++;
            ops+=2;
        }
        ops+=2; //for loop statement = false

```

```

        //for every element in the stack..
        while(!stack.isEmpty()){
            ops+=2; //for loop = true
            char c1 = (char)stack.pop(); //pop the element from the stack
            ops+=2; //could be +=3 (char) cast?
            char c2 = (char)queue.dequeue(); //dequeue the element from queue
            ops+=2; //could be +=3 with (char) cast?
            ops++; //for if statement
            if(c1 != c2){//Compare the two. If they are not equal - not a
palindrome
                ops++; //for return statement
                return false;
            }
        }
        ops+=2; //for loop = false;
        ops++; //for return statement
        return true;
    }

    //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 public String decimal2Binary(String str){
        int num = Integer.parseInt(str); //convert string to int

        return Integer.toString(num); //return the binary string
    }
}

```