Palindrome Complexity Analysis

Author - Abishek Bupathi

I. Problem Statement

The function of the program is to check whether the number in decimal representation is palindrome and check if it's binary representation also is palindrome.

Before checking for palindrome in binary representation a function needs to be created to convert decimal numbers to binary.

The 4 methods used to check for palindrome:

1. Reverse the string using a loop and comparing the reversed string with the original string
2. Compare the first character with the last character, second with the second last character and so on.
3. Using stack and queue
4. Using recursion

After implementing, each method needs to be timed and number of operations needs to be counted for checking the number of palindromes between 0 and 1000000 in order to find the efficiency of each method.

II. Analysis and Design notes

4 function were created which implemented the 4 methods mentioned in the problem:

1. reverse_string() Function:

A loop is run from length-1 to 0 i.e. from last element to first element and each element was added to another string called reverse_input. Then the reverse input is compared with the original input. If it's the same, then the function returns true else false.

2. ***element_comparison() Function:***

   A variable j is initialized with the value of input.length(). Then a loop is run from 0 to length/2, If the character at pos i and j are equal till the last iteration of the loop then the function returns true else the loop stops and returns false. j variable is decremented by 1 for each iteration of the loop.

3. ***stack_queue() Function:***

   The characters of the string/ digits of the number are simultaneously pushed into a stack and enqueued into a queue using loop. Then Using another loop that runs till both queue and stack are empty is used to pop and dequeue elements. While doing these actions popped and dequeued elements are compared. If there are same till the last iteration of the loop then the function returns true else the loop stops and returns false

4. ***using_recurssion() Function:***

   This function send the input value to a method called reverse() which returns the reversed string, Then compare the returned value with the original string and return true if both are same else false.

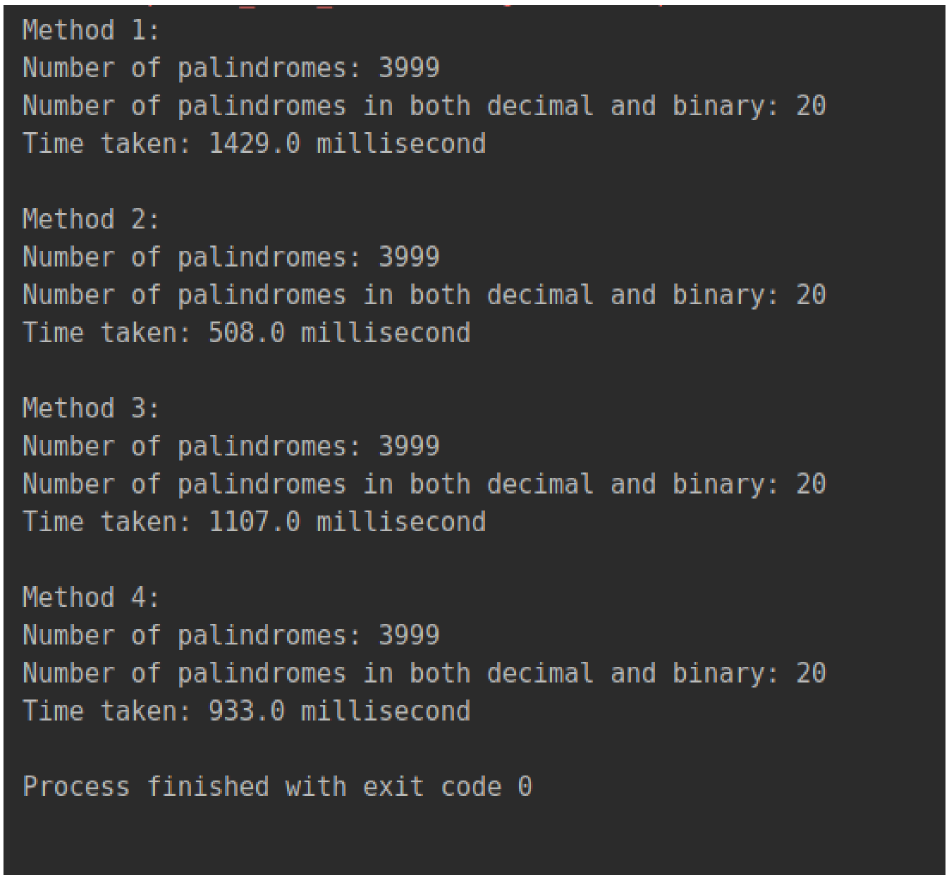
   reverse() method has base case that returns back to the function that initially called this method (i.e. using_recursion() function), it calls itself till the string become empty while changing its argument value to input_string.substring(1) and returning that value added with character at 0 position of the input_string.

Another function called ***convertBinary()*** takes in decimal representation of a number as a string and coverts into integer, then runs a while loop until the integer becomes 0. Inside the loop the value of the integer%2 is added to a string, then the integer is divided by 2. After the loop finishes the string is then reversed then returned, this value is the binary representation of the integer value taken in.

The number of operations are counter using separate variables for each method as counter which is incremented by 1 for each operation taken place then those values are stored in separate arrays for different methods and sent to *createExcel()* function which puts data from arrays into excel sheet using jxl libraries.

III. Testing

Output of the program:

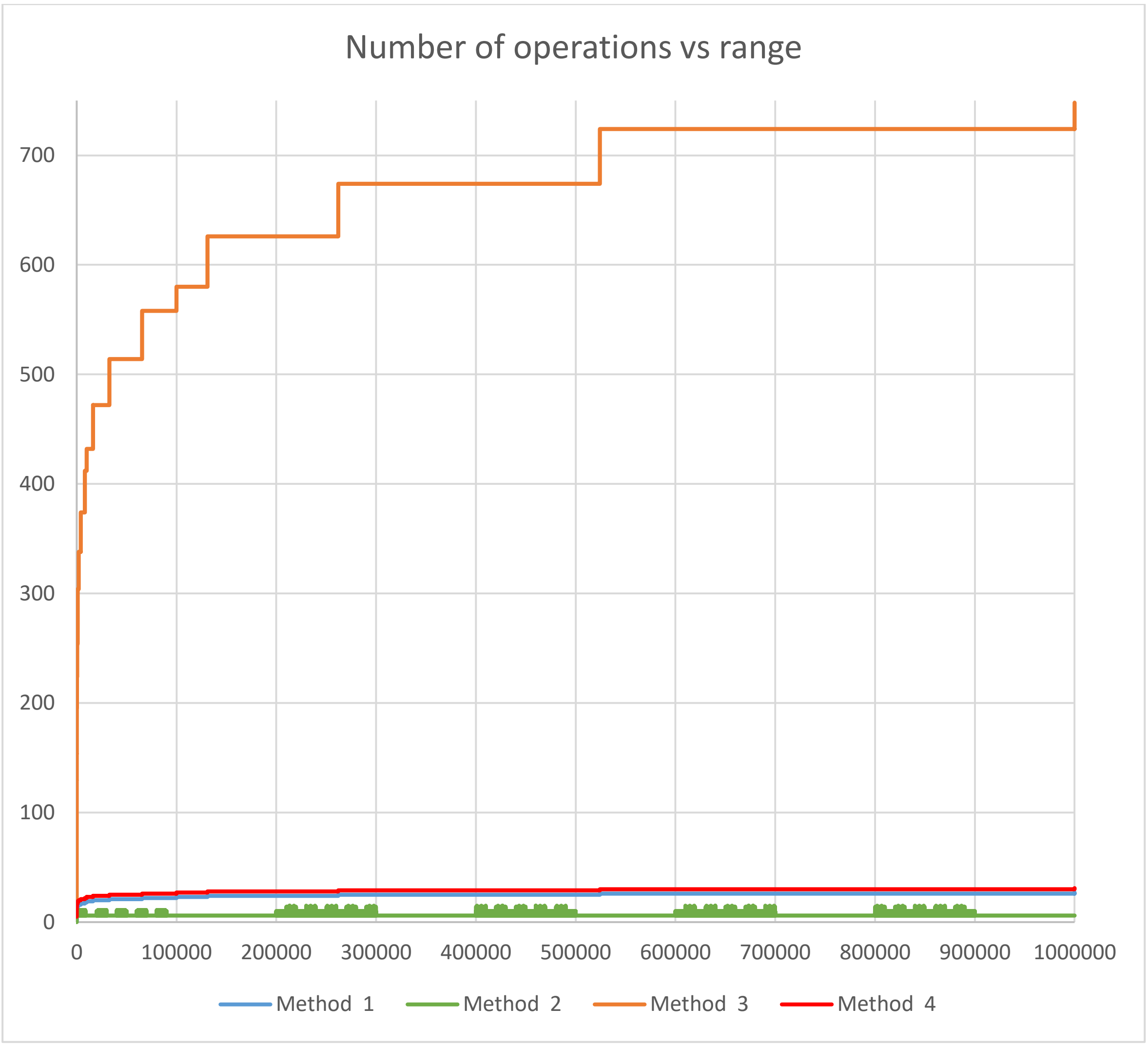


```
Method 1:
Number of palindromes: 3999
Number of palindromes in both decimal and binary: 20
Time taken: 1429.0 millisecond

Method 2:
Number of palindromes: 3999
Number of palindromes in both decimal and binary: 20
Time taken: 508.0 millisecond

Method 3:
Number of palindromes: 3999
Number of palindromes in both decimal and binary: 20
Time taken: 1107.0 millisecond

Method 4:
Number of palindromes: 3999
Number of palindromes in both decimal and binary: 20
Time taken: 933.0 millisecond

Process finished with exit code 0
```

The 1st method took the maximum time to find the number of palindrome because the loop used must go through each digit and add it to another string.

The 2nd method took the minimum time and proves to the perfect method if time was a concern

Graph:



Method 3 which implemented stack and queue has the maximum number of operation and increases significantly with increase in range of numbers.

Method 4 which implemented recursion has second largest number of operations and it remains almost constant after 500000

Method 2 which implemented comparing first digit to last digit, second to second last digit has the lowest number of operations

Method I which implemented reversing string and comparing has the second lowest number of operations.

From the graph we come to know that method 3 consumes the most memory and method 2 consumes the least memory.