Answer to Problem 3

Arnab Roy, 40184043

1 Algorithms

1.1 Pseudocode

1.1.1 The algorithm for sin(x)/cos(x)

Algorithm 1 tangent(angle) using sin(angle)/cos(angle)

```
result = calculateSin(angle)/calculateCos(angle)
if result is equal to ∞ then
    throw error
end if
Round result to 7 digits
return result
```

Algorithm 2 calculateSin(angle)

```
convert angle to 360 degrees
if angle is less than 0 then
    store the sign of angle
    angle = absolute value of angle
end if
determine the quadrant of the angle
signInQuadrant = calculate the sign of the angle in determined quadrant
shouldSubtract = true
angleInRadian = angle * 3.141592654 / 180
squareOfRadianAngle = angleInRadian * angleInRadian
numerator = angleInRadian * angleInRadian * angleInRadian
digitToFactorial = 3
result = angleInRadian
for i = 1...9 do
    nTerm = numerator / factorials[digitToFactorial]
    if shouldSubtract then
        nTerm *= -1
        shouldSubtract = false
        else
        shouldSubtract = true
    end if
    result += nTerm
    numerator = numerator * squareOfRadianAngle digitToFactorial += 2
end for
result = roundTo7Digits(result)
return result == 0 ? 0 : result * sign * signInQuadrant
```

Algorithm 3 calculateCos(angle)

```
convert angle to 360 degrees
if angle is less than 0 then
    angle = absolute value of angle
end if
determine the quadrant of the angle
signInQuadrant = calculate the sign of the angle in determined quadrant
shouldSubtract = true
angleInRadian = angle * 3.141592654 / 180
squareOfRadianAngle = angleInRadian * angleInRadian
numerator = squareOfRadianAngle
digitToFactorial = 2
result = 1
for i = 1...9 do
    nTerm = numerator / factorials[digitToFactorial]
    if shouldSubtract then
        nTerm *= -1
        shouldSubtract = false

    else
        shouldSubtract = true
    end if
    result += nTerm
    numerator = numerator * squareOfRadianAngle
    digitToFactorial += 2
end for
result = roundTo7Digits(result)
return result == 0 ? 0 : result * signInQuadrant
```

1.1.2 The algorithm for the Taylor series of tangent function

Algorithm 4 tangent(angle) using Taylor series

```
nominator[13] = [1, 1, 2, 17, 62, 1382, 21844, 929569, 6404582, 443861162, 18888466084,
    113927491862, 58870668456604]
denominator[13] = [1, 3, 15, 315, 2835, 155925, 6081075, 638512875, 10854718875,
    1856156927625, 194896477400625, 49308808782358125, 3698160658676859375]
result = 0
squareOfAngle = angle * angle
for test = 0...12 do
    result += angle * nominator[test] / denominator[test]
    angle *= squareOfAngle
end for
return result
```

1.2 Description of the implemented algorithm

We have implemented the tangent function using the sin/cos formula. For this, we need the Taylor series for sine and cosine. At first, the angle in degrees provided by the user is converted within 90 degrees and the quadrant is determined. The it is converted to radians. The algorithm considers upto 9 terms and achieves an accuracy upto 6 decimal digits.

1.2.1 Time complexity

O(N)

1.2.2 Space complexity

O(T), where T is the number of factorials generated.

1.3 The advantages and disadvantages

We have implemented the tangent(angle) using the sin(angle)/cos(angle) algorithm. It has several advantages of the Taylor series of the tangent function. To understand this, we will first discuss about the disadvantage of the Taylor series of tangent.

1.3.1 Disadvantage of Taylor series of tangent

- The algorithm is not accurate for smaller terms. A large number of terms (approximately 30) needs to be taken to get an accurate result.
- The denominator of the 13th term of the series is 3698160658676859375. It is evident that the denominators of the larger terms will exceed the capacity of the primitive data types in Java.
- Keeping the Taylor series small results in output whose fractional parts are largely deviated from the accurate result.

1.3.2 Advantage of sin(x)/cos(x)

- Needs only 9 terms for the Taylor series of sine and cosine to make the output accurate upto 6 fractional parts
- Smaller number of terms mean faster execution