Algorithm 1 Square root algorithm

```
1: function CalculateStandardDeviation
2:     samplesNumber ← Scanner
3:     array ← 0                                                    ▷ Integer array
4:     while i < samplesNumber do
5:         array[i] ← nextInt
6:     end while
7:     average ← GetAverage(array)
8:     temp = 0
9:     for i < lengthofarray do
10:        temp ← (array[i] − average) ∗ (array[i] − average) + temp
11:    end for
12:    beforeRoot ← temp/lengthofArray
13:    result ← MathSqure(beforeRoot)
14:    return result
15: end function
16:
17: function GetAverage(arry)
18:    temp = 0
19:    for i < lengthofarray do
20:        temp ← array[i] + temp
21:    end for
22:    result ← temp/lenghtofArray
23:    return result
24: end function
25:
26: function MathSqure(a, b)      ▷ a: Number be square rooted      ▷ b: accuracy of decimal points
27:    array ← 0
28:    integer ← 0
29:    if b > 0 then
30:        array ← DecimalAccuracy(b)
31:    end if
32:    integer ← IntegerPart(a)
33:    return GetResult(a, integer, array)
34: end function
35:
36: function DecimalAccuracy(b)
37:    array ← 0
38:    integer ← 0
39:    while integer ≠ b do
40:        f ← 1
41:        for i <= integer do
42:            f ← f ∗ 10
43:        end for
44:        array[integer] ← 1/f
45:        integer ← integer + 1
46:    end while
47:    return array
48: end function
49:
50: function IntegerPart(a)
51:    if a = 1 then
52:        return 1
53:    end if
54:    temp ← 0
55:    for i <= a/2 + 1 do
56:        if i ∗ i = a then
```

```
            temp ← i
            break
        end if
        if i * i > a then
            temp ← i − 1
            break
        end if
    end for
    return temp
end function

function GetResult(a, integer, array)
    temp ← integer
    for p < lengthofarray do
        if p > 0 then
            integer ← temp
        end if
        for j <= 9 do
            temp ← j * array[i] + integer
            if temp * temp = a then
                return temp
            end if
            if temp * temp > a then
                d1 ← toString(temp)                    ▷ d1 is BigDecimal
                d2 ← toString(array[p])                ▷ d2 is BigDecimal
                temp ← d1 − d2
                break
            end if
        end for
    end for
    return temp
end function
```

Technical reason

The important thing is the approach of root, there are several approaches to calculate square root, such as Newton's method or binary method. But i choose a customized approach to get the square root. First of all, this method can get the integer part of the square root result, which is simple than others. Then, we can get a template of accuracy. For example, if the decimal digital accuracy is 3, the template is 0.001. Finally, we can pass integer part, decimal part and the beforeRoot number to the last function, it calculates the result by combination the integer part and decimal part, which the decimal part is dynamic in order to get the accurate decimal part.

Advantages and disadvantages

The BigDecimal is used to avoid accuracy lost. It also can get the integer part of the result in a efficient way. In the meanwhile, this approach can control the number of decimal by user selection, which is use-friendly. The disadvantage is: it is a little bit more time-comsuming than the binary approach since this method use a traditional way to calculate the decimal part.