Write an algorithm to round off a given decimal digit and plot the error/accuracy to number of decimal digit for the solution of a quadratic equation.

Parag Parihar
Roll No:- IIT2016095
iit2016095@iiita.ac.in

Rakshit Sai
Roll No:- IIT2016126
iit2016126@iiita.ac.in

Adarsh Agrawal
Roll No:- IIT2016516
iit2016516@iiita.ac.in

Nilotpal Pramanik
Roll No:- IRM2016501
irm2016501@iiita.ac.in

I. Introduction and Literature Survey

The given task for us to design and analyze a problem. The given problem was to write an algorithm to round of a given decimal digit and plot the error or accuracy to number of decimal digit for the solution of quadratic equation. The given problem actually consists of two sets one is for rounding of the decimal digit and the other one is to calculate the roots and plot the error or accuracy graph.

According to the first problem, we have to round of a given decimal digit up to a certain digit.

In the second one we've to find the solutions of the quadratic equation. A quadratic equation of the form $ax^2 + bx + c = 0$ might have 2 distinct real solutions or 2 equal real solutions or 2 complex solutions.

After all this we need to draw a graph between error or accuracy and the number of digits of decimal number and check how much accurate the solutions we obtained are by rounding of them to a certain digit at each run and check the closeness of the quadratic equation to 0 when the root is substituted into it.

II. Algorithm Design

An algorithm to round of a given decimal digit:-

Firstly, we are going to take the input decimal number which is going to round of as an array of characters named number.As well as we are scanning the number up to which we have to round of the decimal number as the long int for all type of test cases.In l we have defined the length of the number as the input string.Also we have assigned 0 in the first index of an array of characters named arr to declare an extra digit at beginning.The purpose of the arr to store the whole digits (without point) from the decimal input.

Now we are traversing the number till its length defined in l to check at which index we are getting the point(.) assigned in b.If the input contains the point then that index will be stored in point, an int data type which already has been defined as -1 by default.In other hand well store other digits as characters in the arr array from the 1st index(arr[1]).

If point has -1 value means the input is not containing any decimal point then we will check under this condition that if the length is equal to the digit then well print the number given to us as input directly.Otherwise if the digit is less than the number then simply well print INVALID,as it will be impossible.And where the digit is greater than the length we will print the number itself followed by the zeros of the difference between the digit and the length.

Where point is not -1 means the input number containing the decimal point,and it is containing its location,well check if the length of the numbers without the decimal point (through i-1) is equal to the digit or not.If true then well similarly print the number itself and where the digit is greater than the length we will print the number itself followed by the zeros of the difference between the digit and the length.Then for the last case i.e where the digit is less than the length of the digits only firstly we have to check if the index of the decimal point i.e point contains greater value than the digit then we have to print INVALID again.Otherwise means where the digit is greater than the value at point firstly we have to decrement the length and assign how many digits to be removed from the end of the number in x an int data type.

For while valid if the arr[i] contains value less than 5 well put 0 in that position and otherwise if arr[i] contains value greater than 5 we are calling update function taking attributes as length, the total length to be removed from the end of the number and the array of size 2, containing the values respectively in the 0th and 1st indexes.In the update function till the the previous index of the particular index contains 9 the particular index will be filled as 0 and well move to the beginning of the number by decrementing the length(i) and the value of x.Otherwise logically at that index 0 will be put and 1 will be added at the value of previous index.

Now if the value at the particular index is equal to 5 we will typecast it into int by subtracting 48 from its ASCII value and assign that at value and check weather it is even or not,If it is even to retain its value well simply put 0 otherwise again call the update function to execute accordingly.

To proceed further well decrement the value of length(i) and the value x accordingly and check weather the value of x is less than or equal to zero or not.If it is zero then well break the loop through break.

If the the extra digit at the beginning of the arr i.e arr[0] retains 0 suggest the first digit of the input is not 9 and then from the index 1 of arr upto digit well print the characters of the digits by checking the index of the point(.) to print it accordingly.Otherwise well print digits similarly as previous way expect starting from the 0th index as the starting digit of the given input is 9.As well as well check weather the index of the decimal point at point +1 ,i.e next index to the point is exceeding over the value of digit or not if true then it will print INVALID as through updating the decimal input finally the length of the number before the decimal point will be increased by 1 so logically having smaller digit to round of the input number will be impossible.

Algorithm 1 update function

```
INPUT:- i,x,a
while $arr[i] =' 9'$ do
    $arr[i] \leftarrow' 0'$
    $i \leftarrow i-1, x \leftarrow x-1$
end while
$arr[i] \leftarrow' 0'$
$arr[i-1] \leftarrow arr[i-1]+1$
$a[0] \leftarrow i, a[1] \leftarrow x$
```

Algorithm 2 Rounding off any given number up to some significant digit

```
INPUT: number, sigdigit
OUTPUT: Rounded off number
Initialization :
$j \leftarrow 1, l \leftarrow$ length of number, arr[0] $\leftarrow 0, i \leftarrow 0$
copy number into an character array arr without "." and store the index of "." in point
if $point = -1$ then
    if $lengthofnumber = sigdigit$ then return number
    else if $lengthofnumber > sigdigit$ then return invalid
    else
        print "number."
        add "rest zeroes"
    end if
else
    if $(lengthofnumber(excluding".") = sigdigit)$ then
        print number
    else if $(lengthofnumber(excluding".") < sigdigit)$ then
        print number
        add "rest zeroes"
    else
        if $(point > sigdigit)$ then return invalid
        else
            $i \leftarrow i-1, x \leftarrow (i-point)-(sigdigit-point)$
            while (1) do
                if $(arr[i] <' 5')$ then
                    $arr[i] \leftarrow' 0'$
                else if $(arr[i] >' 5')$ then
                    $update(i,x,a)$
                else
                    $value \leftarrow arr[i-1]-48$
                    if value is even update arr[i] to 0 otherwise call update(i,x,a)
                end if
                $i \leftarrow i-1, x \leftarrow x-1$
                if $x <= 0$ then
                    break
                end if
            end while
        end if
        if $(arr[0] = 0)$ then
            $i \leftarrow 1$
            while $(i <= sigdigit)$ do
                if $i = point$ then
                    print "."
                end if
                print $arr[i]$
            end while
        else
            if $point + 1 > sigdigit$ then
                print Invalid
            else
                print the arr upto sigdigit with "."
            end if
        end if
    end if
end if
```

The error/accuracy to number of decimal digit for the solution of a quadratic equation.:-
Here well take the coefficients of a quadratic equation as the inputs(a,b,c respectively) and assign the value of b^2-4ac in d.If the value of d is greater than or equal to zero then well assign the value of $\frac{-b+\sqrt{d}}{(2*a)}$ at root1 and $\frac{-b-\sqrt{d}}{(2*a)}$ at root2.The both of the roots will be real.In other hand where d is less than zero well print $\frac{-b}{2a}$ as the real part and $\frac{\sqrt{-d}}{2a}$ as the imaginary

part of the root1 similarly $\frac{-b}{2a}$ as the real part and $\frac{\sqrt{-d}}{2a}$ as the imaginary part of the root2.Here both the roots are imaginary.

Algorithm 3 Algorithm for finding root of given quardratic equation

```
**INPUT:-** a,b,c
**OUTPUT:-** root1,root2
$d \leftarrow b^2 - 4ac$
**if** $d >= 0$ **then**
    $root1 \leftarrow \frac{(-b+\sqrt{d})}{2*a}$
    $root2 \leftarrow \frac{(-b-\sqrt{d})}{2*a}$
**else**
    $root1 \leftarrow \frac{-b}{2*a} + \frac{\sqrt{d}}{2*a} j$
    $root2 \leftarrow \frac{-b}{2*a} - \frac{\sqrt{d}}{2*a} j$
**end if**
```

III. Analysis and Discussion

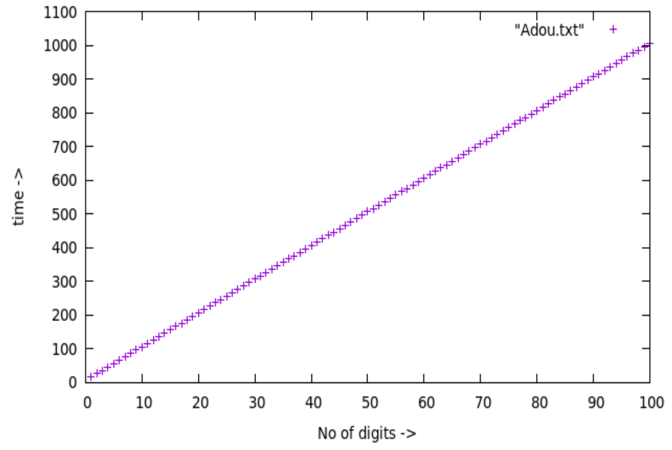
A. For Algorithm:- 2

*1) **BEST CASE-**: The number of units of time taken for the FOR LOOP is:-* $8n+2$

$$time_{for\_loop} \propto 8n+2$$

Total time is :-$8n+18$

$$time_{best\_case} \propto 8n+18$$



*2) **WORST CASE-**:* The number of units of time taken for the FOR LOOP is:- $8n+3$

$$time_{for\_loop} \propto 8n+3$$

The update function takes $9n+12$ units of time

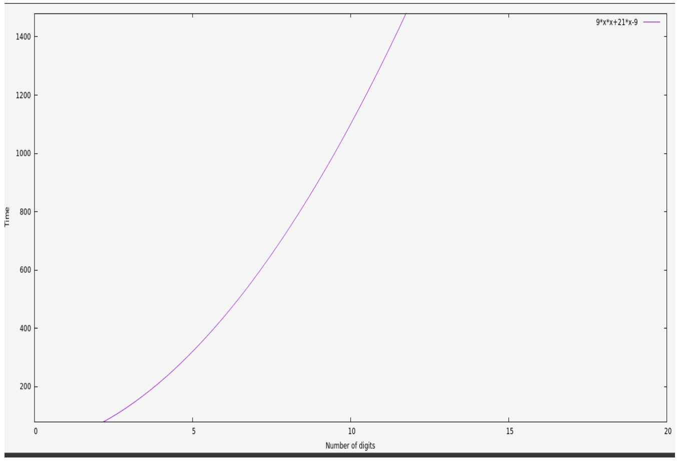
$$time_{update\_function} \propto 9n+12$$

The WHILE LOOP takes $(n-s)[9n+30]$ units of time

$$time_{while\_loop} \propto (n-s)[9n+30]$$

Total time is :-$9n^2+13n-9ns-16s+7$

$$time_{worst\_case} \propto 9n^2+13n-9ns-16s+7$$



*3) **AVERAGE CASE-**: The number of units of time taken for the FOR LOOP is:-* $8n+3$

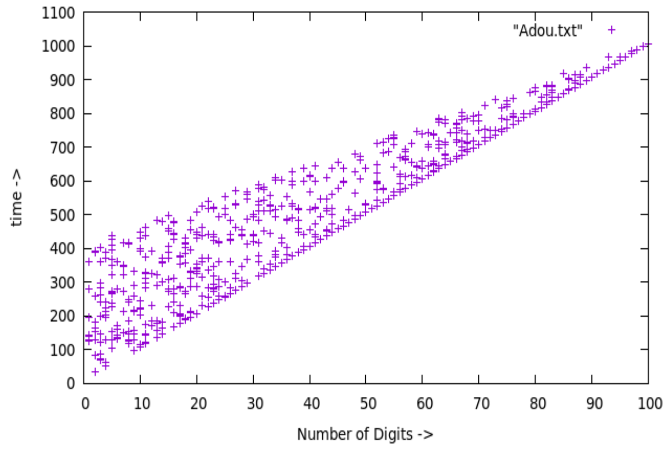
$$time_{for\_loop} \propto 8n+3$$

The number of units of time taken for the WHILE LOOP is:- $(n-s)(9)$

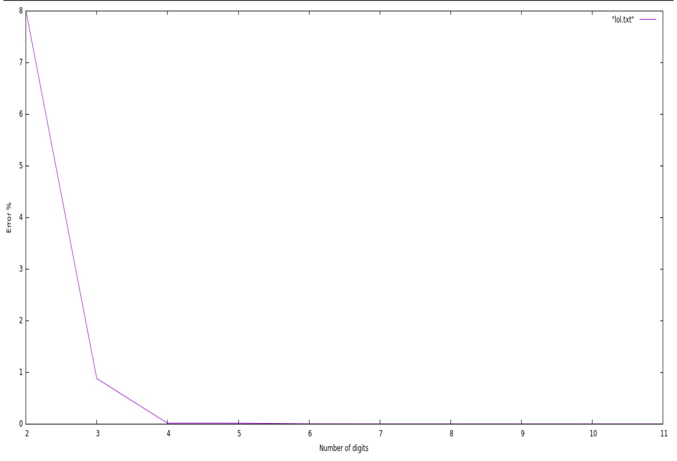
$$time_{while\_loop} \propto (n-s)(9)$$

Total time is :-$17n-9s+27$

$$time_{average\_case} \propto 17n-9s+27$$

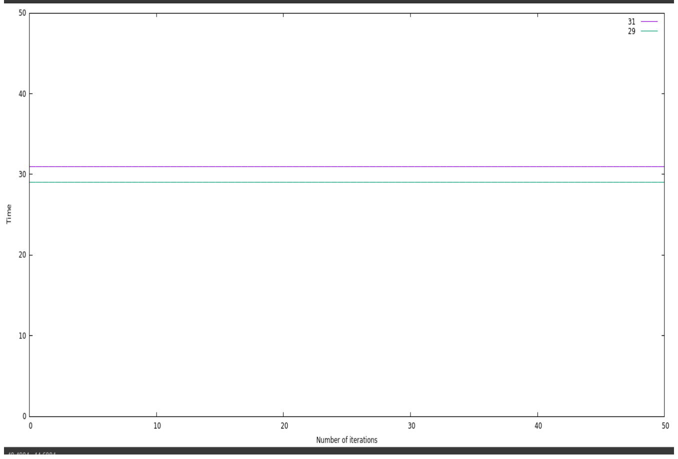


4) Error/Accuracy Graph:



B. For Algorithm:- 3

In the case of **REAL ROOTS** we need 31 units of time and in the case of **IMAGINARY ROOTS** we need 29 units of time.



IV. Experimental Study

TABLE I
PART A(For Algorithm 1)

Initial decimal number	Number of digits asked to round off	Total instructions
364	3	42
127345	6	66
143.465	9	81
135.1112	5	101
134.56	3	146
189.956	4	156
999.9956	4	223
	Average	116

In the **first part**, we are going to round off a given decimal number to a specific number of digits. Let us consider the example of 364 with rounding off to 3 decimal digits. Since the number has no decimal, so after copying through for loop, it will never take if condition and so the value of point will remain -1 and hence it will go to 1st if condition and check the length of number with digits required which is same and hence the output will be generated and hence it is best case with least instructions of 42, $\Omega(n)$. But, on considering example 999.9956 and to round off to 4 decimal digits. While copying the digits point will get a value of 3 as digits before point are 3, hence the first if condition will be false and in else part as the test case is not invalid it will go into the else part it will begin the rounding off of digits from the right end, but when it will come to 5(rounded off to 6), the 9 before it shall be incremented and made 0 and so all the followed 9s, but at the end the last 0(at arr[0]),it will become 1 and hence while printing also it will check the first condition and print 1000 as an output making it the worst case with 223 instruction $O(n^2)$. Considering any one average case like 135.1112 it has 101 instruction which is near to the average instructions we got of these 7 test cases.

Graph Description :

Best Case : Since there is only the first for loop running in that hence it is linear $O(n)$. **Worst Case :** The worst case will have the for loop, the while loop will also include the while loop of Update Function,which in this case case can run for n times and the above while will run for (n - required digits), hence it is of order $O(n^2) \rightarrow (n - s) * (n)$.

TABLE II
PART B(For Algorithm 2)

Quadratic Equations	Total instructions
a = 1, b = -7, c = 12	31
a = 4, b = 1, c = 1	29
a = 4, b = 4, c = 1	31
a = 7, b = 2, c = 2	29

In the **second part**,the time graph for this algorithm of order of $\theta(1)$.Consider any example, whether having complex or real roots, there will be some error in the roots of the quadratic equation, hence we are plotting error-accuracy against no of digits in the roots of the equation. There will be an increasing in accuracy but decreasing graph in error with increase in number of digits.

V. Conclusion

In the **first part** of the problem we have implemented it efficiently and try to analyze the algorithm correctly.By doing proper analysis of the code for rounding of the decimal number up to a given digit we found out that the time complexity for the worst case is proportional to $O(n^2)$ and the corresponding best case $\Omega(n)$.

In the **second part** of the problem throughout the analysis we can conclude that by increasing the digits the accuracy is increasing and error is decreasing because once we start rounding of the decimal roots of any quadratic equation the distance of the point from it's original value will be increased Therefore we get a exponentially decreasing graph and the accuracy graph is exponentially increasing.As well as we found out that the time complexity for the worst case is proportional to $O(1)$ and the best case is $\Omega(1)$ same as the worst case hence, the time complexity will be $\Theta(1)$.