PROGRAMMING ASSIGNMENT - 1

In this programming assignment, you will implement a square root function to calculate the square root of any integer input using two different numerical methods and compare the performance.

The square root problem can be converted to a root-finding problem as follows.

If the square root of an integer number *a* is *x*, then the following equation holds:

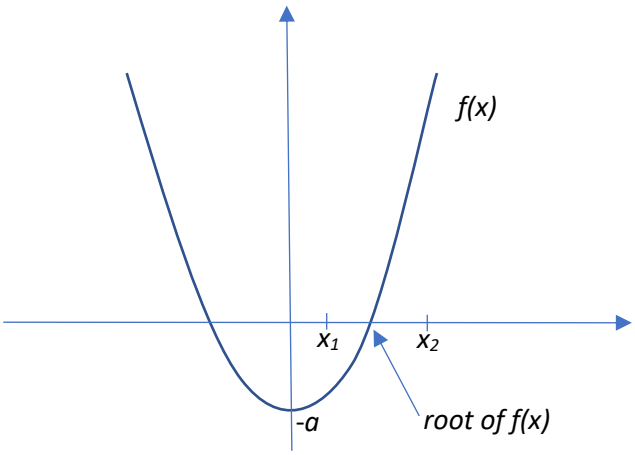
$$x^2 = a$$

By moving the constant *a* to the left side of the equality, we get:

$$f(x) = x^2 - a = 0$$

Therefore, the roots of the above quadratic function will be the square roots of *a.*

This function's graph is a simple parabola as show below.



If $f(x_1)$ and $f(x_2)$ have opposite signs, then there is at least one root in the interval (x_1, x_2) as shown above. If the interval is small enough, it is likely to contain a single root. Thus, the roots of $f(x)$ can be detected by repetitively evaluating the function at shrinking intervals (x_1, x_2) and looking for a change in sign between $f(x_1)$ and $f(x_2)$.

A better approach for finding the roots of $f(x)$ is the Newton-Raphson method. The first order Taylor series approximation of $f(x)$ about x_i is:

$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i)$

If x_{i+1} is a root of $f(x)$, the equation becomes $0 = f(x_i) + f'(x_i)(x_{i+1} - x_i)$

Leaving x_{i+1} alone, the result is the Newton-Raphson formula:

$x_{i+1} = x_i - f(x_i) / f'(x_i)$ (1)

By starting with an initial estimate of x_i and applying the above formula repetitively, one can approach the root of $f(x)$.

Part1) Implement the interval search method in C. Read a positive integer *a* from the user of which square root we would like to find. Take the initial interval as (0, *a*) and divide this interval into 10 equal size sections at each iteration. Repeat this search until the root is found or the difference between x_1 and x_2 becomes less than 0.0001. Print the resulting value of the root (if one is found) or the value of x_1 if the root cannot be found exactly.

Hint: You need to check each section to see if the root resides within that section by evaluating *f(x)* at the section boundaries and looking at the signs of the evaluations. If the section contains the root, then divide the section into 10 new sections and repeat the process.

Part2) Implement the Newton-Raphson formula in C. Read a positive integer *a* from the user of which square root we would like to find. Start with the initial estimate of root as *a* (i.e., $x_i = a$) and apply the formula in (1) to find the next estimate of the root at each iteration. Repeat the iterations until the difference between the current estimate and the previous estimate of the root becomes less than 0.0001. Print the resulting value of the root (if one is found) or the value of x_1 if the root cannot be found exactly.

Part3) Run both implementations in Part1 and Part2 repetitively for finding the square roots of all integers between 1 and 100000, but do not print the square roots to the screen. Only measure the total execution time for each method and print it to the screen. See which method runs faster.

Use the **clock()** function to profile the execution time. The clock() function does not take any arguments and returns the current value of the CPU ticks as a custom type **clock_t** (we will see custom types when we look at structs and typedefs. For now, you may assume this type exists and works just like any other primitive type such as int, float, etc, i.e you can subtract two clock_t values from each other to find the difference). You can read the CPU ticks at the beginning of your implementation and at the end and take the difference to calculate the execution time in terms of CPU ticks as below.

```
clock_t t1, t2;

int executionTime;

t1 = clock();

…. //Profiled code here

t2 = clock();

executionTime = (int)(t2 – t1); //Since executionTime is an int, cast the result to an int
```

Important: In order to use the clock() function, you need to include **time.h** header file in your source code.

DELIVERABLES

Implement all of the above parts as a single C source code file and submit your C source file to ninova. Do not include binaries or any other file. You may use multiple C functions in your source code if you want. When your code is compiled and run, the code should execute all parts and print the results to the screen.

Important: Your code should be properly commented. Uncommented code will get partial credit. You need to do your assignment alone. Code sharing among students or using code from any other source is not allowed. You may NOT use sqrt() or any other math function.