ECE326 Lab 2 Report

Define a python function newton_raphson() to compute the square root of 12345. You may use 100 (or another number of your choice for convergence) as initial guess x_0.
newton_raphson() function is defined in the Python file.

Now define a randomized version of the newton_raphson_randomized() which will make a random guess of the initial approximation. Determine the probability (Hint: number of times converged/total number of runs) of convergence of newton_raphson_randomized().
The probability depends on the range of numbers used for selecting a random number from. More specifically, it depends on (1) if 0 is included in the range and (2) how big the range is. If 0 is included in the range, then there is a case where the function will not converge (because you are dividing by 0). Otherwise, the function will always converge with a probability of 100%. If 0 is included in the range, the probability of convergence increases as the range size increases. If the range is an infinite size, then the probability of convergence goes to 100%.

Analyze and compare accuracy and runtime performance of newton_raphson() and newton_raphson_randomized() methods.
newton_raphson() always produces the same result and newton_raphson_randomized() produces a different result each time. Sometimes newton_raphson_randomized() produces a result that is more accurate than that of newton_raphson(). The runtime for newton_raphson_randomized() varies depending on the randomly chosen initial root. When the randomly chosen root is closer to the real root value than the initial choice used in newton_raphson(), the runtime of newton_raphson_randomized() is faster than newton_raphson(). This is more rare. The usual case is that newton_raphson_randomized() is slower than newton_raphson().

The Quercus file type for submission is PDF, attached below is a screenshot of the code in my Python file.

```python
import random
import time

# Define a python function newton_raphson() to compute square root of 12345.
# You may use 100 (or another number of your choice for convergence) as initial guess x_0.
# x_(n+1) = x_n - (f(x_n)/f'(x_n))
# f(x) = x^2 ; f'(x) = 2x
def newton_raphson(n):
    x_0 = 100
    root = x_0 - (((x_0*x_0)-n)/(2*x_0))

    while (abs(root-x_0) >= 0.5):
        x_0 = root
        root = x_0 - (((x_0*x_0)-n)/(2*x_0))

    return root

# Now define a randomized version which will make a random guess of the initial approximation.
def newton_raphson_randomized(n):
    x_0 = random.uniform(1, n)
    root = x_0 - (((x_0*x_0)-n)/(2*x_0))

    while (abs(root-x_0) >= 0.5):
        x_0 = root
        root = x_0 - (((x_0*x_0)-n)/(2*x_0))

    return root
```