Given:

$g(x) = \cos(x) - x^3$ {has root on (0, 1)}

Root $r \sim 0.865474$

Find answer within 4 decimal places; Satisfy $|x_c - r| < 0.5 \times 10^{-4}$

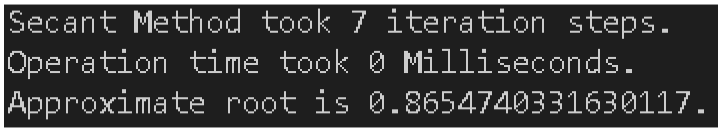
Problem Statement 3: Use the Secant method with two given initial roots $x_0 = 0$ & roots $x_1 = 1$.

Algorithm Description: Similar to Newton's Method, except it's trying to find without having the need to have a derivative with just 2 starting points. Newton's Method was to figure out x_i such that it was close enough to converging from x_{i-1} resulting in it being an approximation of root. In Newton's definition, a derivative was required. We simply replace the derivative with it's limit definition and that is then the secant method where we need 2 initial points now to get x_i to converge to root value, $x_i = x_{i-1} - (g(x_{i-1})(x_{i-1} - x_{i-2})) / (g(x_{i-1}) - g(x_{i-2}))$.

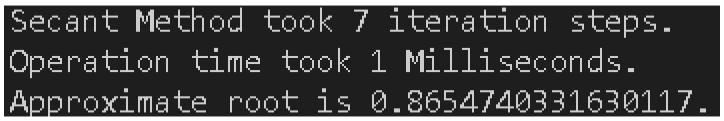
Pseudocode:

```
Function secantMethod(previous1, previous2, count):
    If abs(previous1 - previous2) < tolerance AND count > 1:
        Return previous1

    Return secantMethod(previous1 - ((g(previous1)(previous1 -
previous2)) / (g(previous1) - g(previous2))), previous1, count + 1);
End

Function g(x):
    Return cos(x) - $x^3$
End
```

Results: (p13.java)



```
Secant Method took 7 iteration steps.
Operation time took 0 Milliseconds.
Approximate root is 0.8654740331630117.
```



```
Secant Method took 7 iteration steps.
Operation time took 1 Milliseconds.
Approximate root is 0.8654740331630117.
```

Observations & Analysis

The root estimated by *p13.java* is approximately 0.86547403 which is well within 4 decimal points of the approximately correct root of 0.865474. As for the program efficiency, I would say it is sufficient as it only took no more than a millisecond to conduct 7 iterations of 17 floating point operations at most (last iteration could be fewer operations returning on first if statement) on each iteration which is about at least 119,000 operations per second. The algorithm is good because it only takes no more than 1 millisecond to figure out the root. I am guessing there are ways to write algorithms using bitwise operators for probably faster results and other more ways unknown to me that could result in faster runtimes. This program is scalable to any predefined internal points generally and runs with any java configured environment.