Bisection Method
Function: f(x) = $\mathrm{x}^{20} - x^{16} - x^8 - x$
Within a given tolerance and maximum number of iterations, this function employs the bisection method to locate the function's root. The root's location is in the range $[a, b]$. The tolerance and the maximum number of iterations are specified in the function.
Python code

```
 # Bisection method
import timeit
mycode='''def f(x):
    return x**20-x**16-x**8-x

a=1
b=2

tolerance = 1e-6
max_iterations = 100

for i in range(max_iterations):
    c = (a+b)/2

    if abs(f(c))<tolerance:
        print(f" Roof found at x={c: 6f}")
        break
    elif f(c)*f(a)<0:
        b=c
    else:
        a=c'''


print (timeit.timeit(stmt = mycode,number = 100))
```

Newton Raphson Method
Function: f(x) = $\mathrm{x}^{20} - x^{16} - x^8 - x$
Derivative: df(x) = $20\mathrm{x}^{19} - 16x^{15} - 8x^7 - 1$ In order to determine the function's root within a given tolerance and the maximum number of iterations, this function uses the Newton-Raphson approach. Initial guess x0, tolerance, and maximum iterations are provided as inputs. If the root cannot be located within the provided number of iterations, the function emits an error message instead of the root when it is discovered. Moreover, the function needs the derivative, which is represented by the symbol df (x).
Python code

```
import timeit
```

```
mycode = '''
def f(x):
    return x**20-x**16-x**8-x

def df(x):
    return 20*x**19-16*x**15-8*x**7-1

x0 = 2
tolerance = 1e-6
max_iterations = 100



for i in range (max_iterations):
    fx = f(x0)
    dfx=df(x0)
    x1=x0-fx/dfx
    if abs(f(x1))<tolerance:
        print(f"Root found at x={x1: 6f}")
        break
    else:
        x0=x1'''

print (timeit.timeit(stmt = mycode,number = 100))
```

Result

```
Roof found at x= 1.121195
4.2132000089623034e-05
Root found at x= 1.121195
1.9103999875369482e-05

Roof found at x= 1.121195
3.612000000430271e-05
Root found at x= 1.121195
1.942000017152168e-05

Roof found at x= 1.121195
4.206499988868018e-05
Root found at x= 1.121195
2.0526999833236914e-05

Roof found at x= 1.121195
5.9684999996534316e-05
Root found at x= 1.121195
3.610300018408452e-05
```

Conclusion

After running the function multiple times along I found that in each case the Bisection method ran slower than the Newton Raphson method. That is inspite both the equations arriving at the same root answer
"Root found at x= 1.121195"