//testcases:

***(Float 64)***

***Input for 5:***

**Stirling's Number:** 118.0191679575901

**N!:** 120

**Absolute Error:** 1.9808320424099009

**Relative Error:** 0.01678398582780814

***Input for 100:***

**Stirling’s Number:** 9.32484762526942e+157

**N!:**93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

**Absolute Error:**  7.773919124995513e+154

**Relative Error:** 0.0008336778720039303

As the numbers increased, there was an inverse relationship between the absolute error and the relative error, because logically, with bigger numbers, there would obviously be a bigger absolute error, while the relative error is smaller due to the absolute error / the input being a tremendously large number. If we worked with a float 32 system, the numbers would also be more off, because every iteration would be losing precision in the decimals, due to the lack of bytes for the remaining integers.

Bisection Method).

a). Approximate root is: 1.0541271240799688

b). Approximate root is: 3.057103549974272

Newtons Method).

a). Approximate root is: 1.054127124091213

b). Approximate root is: 3.057103549994738

Secant Method).

a). Approximate root is: 1.054127124091213

b). Approximate root is: 3.057103549994738