Problem 2 (stack): Analysis and compare to output of sinh(x) and cosh(x)

Recursive Definition:

→ h(x) (sinh):

return 2*h(angR/2)* g(angR/2);

This definition calls both h(), g() recursively. call to h() generate additional call to g(), g() call h() again. this interaction increase number of calls for sinh(x)

→ g(x) (cosh):

float b = h(angR/2);

return 1+2*b*b;

this definition make a single call to h(), it does not call g() itself recursively. So, g() produce fewer recursive call compared to h()

→ Both function have a base case,

if(angR>-tol && angR<tol)

But recursive compounding effect is much stronger in h() as reliance on both h() and g().

→ call counts analysis example:

When angR = -1.0,

For sinh(x) (h(-1.0)):

At depth n, h(), calls both h() and g() recursively

number of calls grows approximately as 2^n or $2^n - 1$ where n is the depth of recursion.

The call count for h() and g() increase rapidly.

For cosh(x) (g(-1.0)):

At depthe n, g() calls h() only once

the number of calls grows as approximately as n, where n is the depth of recursion

h(x) make 2 recursive calls which form binary tree structure

g(x) make only 1 call to h(x/2), form linear growth

For example:

For sinh(-1.0) (h(-1.0)):

⇒ h(-1.0) calls both h(-0.5) and g(-0.5)

⇒ h(-0.5) calls h(-0.25) and g(-0.25)....

⇒ This process double with each recursive step

For cosh(-1.0) (g(-1.0)):

⇒ g(-1.0) calls only h(-0.5)

⇒ Since only one recursive branch occurs, do not double result in fewer calls.

Theoritical insight:

⇒ Recursive function like h(x) create a binary recursion tree

⇒ g(x) create a single recursion path.

⇒ total number of call follow the pattern:

$$\text{Total calls} \approx 2^n - 1 \text{ for } h(x)$$

⇒ the Recursion stop when the input angle is smaller than the tolerance 1e-6

Here is the below table for sinh(x) and cosh(x)

Table for sinh(x):

Angle	sinh(x)	h(x) Calls	g(x) Calls
-1.0	-1.2	28656	17710
-0.9	-1.0	28656	17710
-0.8	-0.9	28656	17710
-0.7	-0.8	28656	17710
-0.6	-0.6	28656	17710
-0.5	-0.5	17710	10945
-0.4	-0.4	17710	10945
-0.3	-0.3	17710	10945
-0.2	-0.2	10945	6764
-0.1	-0.1	6764	4180
0.0	0.0	1	0
0.1	0.1	6764	4180
0.2	0.2	10945	6764
0.3	0.3	17710	10945
0.4	0.4	17710	10945
0.5	0.5	17710	10945
0.6	0.6	28656	17710
0.7	0.8	28656	17710
0.8	0.9	28656	17710
0.9	1.0	28656	17710
1.0	1.2	28656	17710

Table for cosh(x):

Angle	cosh(x)	h(x) Calls	g(x) Calls
-1.0	1.5	17710	10946
-0.9	1.4	17710	10946
-0.8	1.3	17710	10946
-0.7	1.3	17710	10946
-0.6	1.2	17710	10946
-0.5	1.1	10945	6765
-0.4	1.1	10945	6765
-0.3	1.0	10945	6765
-0.2	1.0	6764	4181
-0.1	1.0	4180	2584
0.0	1.0	0	1
0.1	1.0	4180	2584
0.2	1.0	6764	4181
0.3	1.0	10945	6765
0.4	1.1	10945	6765
0.5	1.1	10945	6765
0.6	1.2	17710	10946
0.7	1.3	17710	10946
0.8	1.3	17710	10946
0.9	1.4	17710	10946
1.0	1.5	17710	10946

Fig.1 Output of Stack Program

☐ comparing Table Output to Inputs:

→ As seen from the table output, functions calls increase as the input angle |x| increases.

→ Near $x \approx 0$, function calls are few — exactly 1 for h(x) and 0 or 1 for g(x)

→ At $|x| = 1.0$, the function calls reach 28656 for h(x) and 17710 for g(x)

☐ What do I see and why:

→ At $x = 0.0$, calls are minimal as base case check: if(angle < tol), stop the recursion.

→ 1 call for h(x) and 0 calls for g(x) when calculating sinh(0.0)

→ As x increase from 0: The function calls grow rapidly as recursive depth doubling. The number of function calls double every time the input angle halves during recursion

☐ Does the number of function calls agree with prediction?

Yes, observed output aligns perfectly with theoretical expectation, as exponential growth for sinh(x) and linear growth for cosh(x)