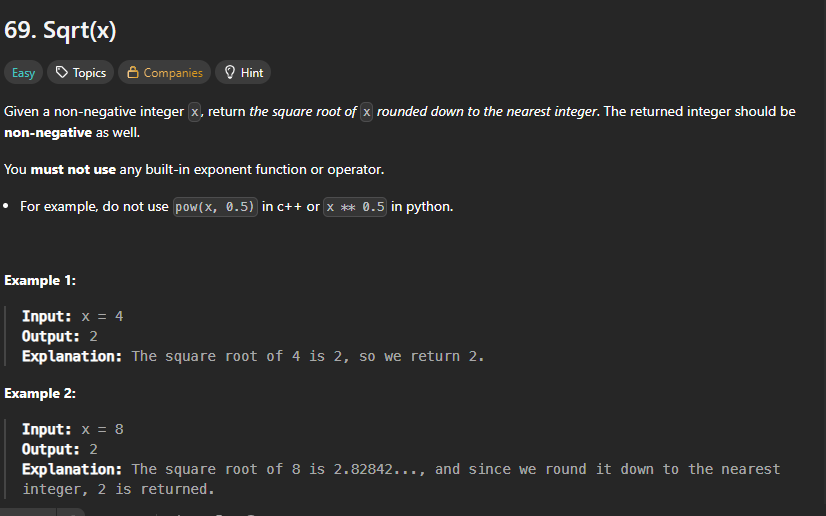
**LeetCode Problem 69 : Sqrt(x)** **[ Time taken: 3 hrs 3 m 58 s ]**

**Brute Force Approach O(n)**

class Solution {

public:

    int mySqrt(int x) {

        // brute force approack

        // my intiution is let's square sorted natural numbers one by one and

        // check if it reaches rounded off result

        for (int i = 1; i <= x; i++) {

            if (i \* i > x) {

                return --i;

            }

            if (i \* i == x) {

                return i;

            }

        }

        return 0;

    }

};

**Runtime Errors Encountered:**

Line 9: Char 19: runtime error: signed integer overflow: 46341 \* 46341 cannot be represented in type 'int' (solution.cpp)

My Learnings :

**❌ Problem: Integer Overflow in i \* i**

When i = 46341, then:

cpp

i \* i = 46341 \* 46341 = 2,147,488,081

This value **exceeds** INT\_MAX, which is 2,147,483,647. Since i is an int, i \* i causes **signed integer overflow**, which is undefined behavior in C++ — and that’s exactly what LeetCode’s runtime error is reporting:

runtime error: signed integer overflow: 46341 \* 46341 cannot be represented in type 'int'

**✅ How to Fix It (without changing your brute-force logic)**

You can safely cast i to long long before squaring:

cpp

if ((long long)i \* i > x) {

return --i;

}

if ((long long)i \* i == x) {

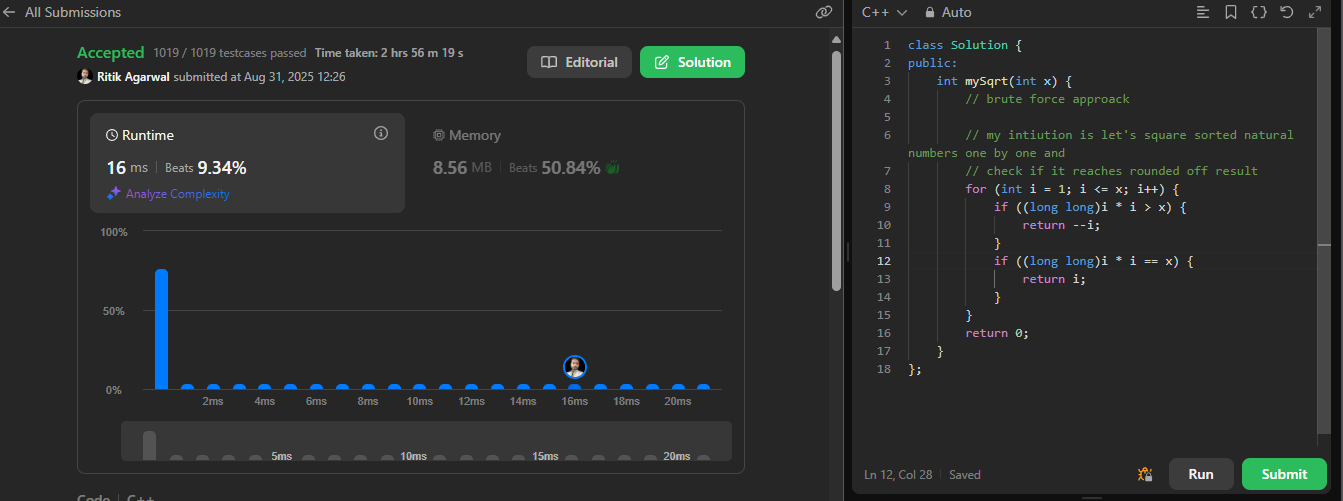
return i;

}

This ensures that the multiplication happens in 64-bit space, avoiding overflow.

**⚡ Bonus Tip: Binary Search Optimization**

If you're ready to move beyond brute force, a binary search approach reduces time complexity from **O(√x)** to **O(log x)**. But your current logic is great for understanding the fundamentals.



Optimised Solution O(log n)

Binary Search algorithm worked well here.

Code snippet

class Solution {

public:

    int mySqrt(int x) {

        int low = 0, high = x;

        int val;

        while (low <= high) {

            val = low + (high - low) / 2; // safer midpoint2;

            if ((long long)val \* val == x) {

                return val;

            }

            if ((long long)val \* val < x) {

                low = val+1;

            }

            if ((long long)val \* val > x) {

                high = val-1;

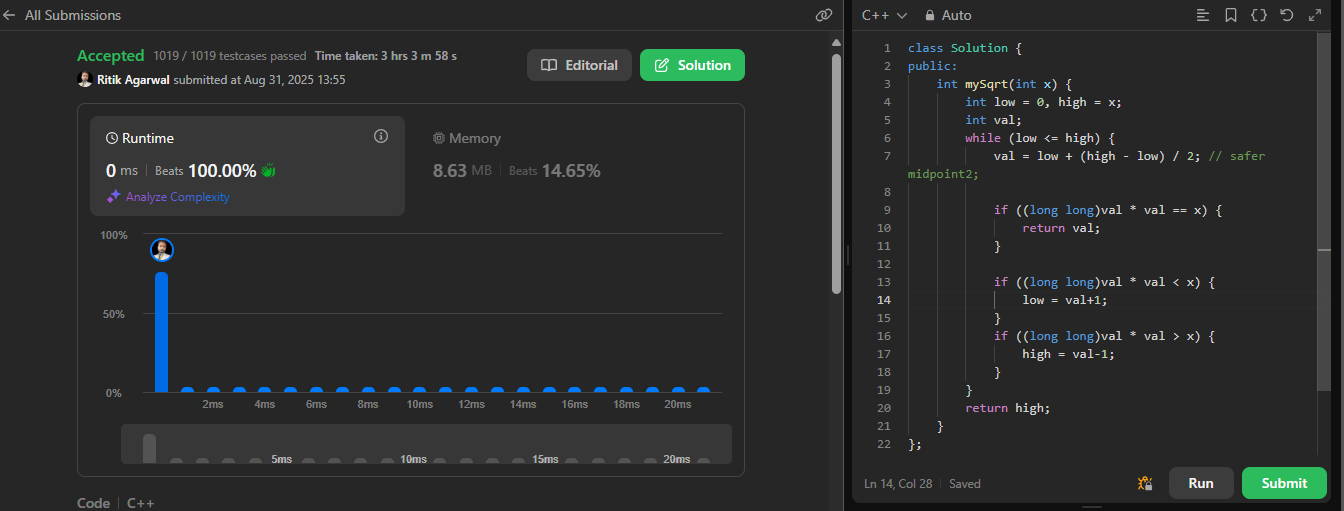
            }

        }

        return high;

    }

};



My learnings:

1. Faced too much trouble in lower & upper bounds.