Solution Documentation for Square Root Calculation

Problem Description

Given a non-negative integer x, the task is to return the square root of x rounded down to the nearest integer. The returned integer should also be non-negative. The solution must not use any built-in exponent function or operator (e.g., pow(x, 0.5) in C++ or x 0.5 in Python).

Examples

Example 1:

- *Input:* x = 4
- *Output:* 2
- *Explanation:* The square root of 4 is 2, so we return 2.

Example 2:

- *Input:* x = 8
- *Output:* 2
- *Explanation:* The square root of 8 is approximately 2.82842. Rounded down to the nearest integer, we return 2.

Constraints

- (0 leq x leq 2^{31} - 1)

Solution

The solution employs a binary search algorithm to find the square root of x. Here's a step-by-step explanation of the approach:

1. **Edge Case Handling:**
   - If x is less than 2, return x directly since the square root of 0 is 0 and the square root of 1 is 1.

2. **Binary Search Initialization:**
   - Initialize two pointers, left and right, to 0 and x respectively.

3. **Binary Search Process:**
   - While left is less than or equal to right:
   - Calculate the midpoint mid as the integer division of (left + right) // 2.
   - Compute mid_squared as mid * mid.
   - Compare mid_squared with x:
   - If mid_squared is equal to x, return mid as the square root.
   - If mid_squared is less than x, move the left pointer to mid + 1.
   - If mid_squared is greater than x, move the right pointer to mid - 1.

4. **Return Result:**
   - After exiting the loop, return right as it will be the largest integer for which right * right is less than or equal to x.

Explanation of the Code

1. **Edge Case Handling:**

- The function first checks if x is less than 2. If true, it directly returns x because the square root of x is x itself.

2. **Binary Search:**

- The left pointer starts at 0 and the right pointer starts at x.
- The loop runs as long as left is less than or equal to right.
- Inside the loop, mid is calculated as the integer division of (left + right) // 2.
- mid_squared is the square of mid.

3. **Conditions Inside the Loop:**

- If mid_squared is equal to x, the exact square root is found, and mid is returned.
- If mid_squared is less than x, it means the true square root is larger, so left is moved to mid + 1.
- If mid_squared is greater than x, it means the true square root is smaller, so right is moved to mid - 1.

4. **Returning the Result:**

- When the loop exits, right is the largest integer such that right * right is less than or equal to x, making it the integer square root of x.