Square Root

Implement squareroot(x).

Compute and return the square root of x.

If x is not a perfect square, return floor(sqrt(x)).<br>

<b>Note:</b>

DO NOT USE SQRT FUNCTION FROM STANDARD LIBRARY

An integer A .

Squareroot of A.

0 <= A <10000000000

#include<bits/stdc++.h>

#define ll long long int

using namespace std;

ll findSquareRoot(ll A) {

long long int low = 1;

long long int high = A;

long long int ans;

while(low <= high) {

long long int mid = low + (high - low) / 2;

if(mid\*mid == A){

return mid;

}

else if(mid\*mid < A){

low = mid + 1;

ans = mid;

}

else {

high = mid - 1;

}

}

if(A==0) return 0; else return ans;

}

int main() {

ll s;

cin >> s;

ll ans = findSquareRoot(s);

cout << ans;

}

#include<bits/stdc++.h>

using namespace std;

int squareRoot(int A) {

}

int main () {

int A;

cin >> A;

cout << squareRoot(A);

return 0;

}

Here we can't use inbuilt sqrt function .

- Main idea is to use Binary Search

- We can easily discard the whole part if square of certain number is greater than the given number.

- Using this trick we can find the square root of a given number in log(n) time complexity.

<br><br>

Code:

<br>

<pre><code class="lang-cpp">

#include&lt;bits/stdc++.h&gt;

#define ll long long int

using namespace std;

ll findSquareRoot(ll A) {

long long int low = 1;

long long int high = A;

long long int ans;

while(low <= high) {

long long int mid = low + (high - low) / 2;

if(mid \* mid == A){

return mid;

}

else if(mid \* mid < A){

low = mid + 1;

ans = mid;

}

else {

high = mid - 1;

}

}

if(A==0) return 0; else return ans;

}

int main() {

ll s;

cin >> s;

ll ans = findSquareRoot(s);

cout << ans;

}