Sqrt(x)

Implement `int sqrt(int x)`.

Compute and return the square root of *x*.

Solution 1

Instead of using fancy Newton's method, this plain binary search approach also works.

```
public int sqrt(int x) {
    if (x == 0)
        return 0;
    int left = 1, right = Integer.MAX_VALUE;
    while (true) {
        int mid = left + (right - left)/2;
        if (mid > x/mid) {
            right = mid - 1;
        } else {
            if (mid + 1 > x/(mid + 1))
                return mid;
            left = mid + 1;
        }
    }
}
```

written by AlexTheGreat original link here

Solution 2

Basic Idea:

Since sqrt(x) is composed of binary bits, I calculate sqrt(x) by deciding every bit from the most significant to least significant. **Since an integer n can have O(log n) bits with each bit decided within constant time, this algorithm has time limit O(log n), actually, because an Integer can have at most 32 bits, I can also say this algorithm takes O(32)=O(1) time.**

```
public int sqrt(int x) {
    if(x==0)
        return 0;
    int h=0;
    while((long)(1<<h)*(long)(1<<h)<=x) // firstly, find the most significant bit
        h++;
    h--;
    int b=h-1;
    int res=(1<<h);
    while(b>=0){  // find the remaining bits
        if((long)(res | (1<<b))*(long)(res |(1<<b))<=x)
            res|=(1<<b);
        b--;
    }
    return res;
}
```

written by yuyibestman original link here

Solution 3

Quite a few people used Newton already, but I didn't see someone make it this short. Same solution in every language. Explanation under the solutions.

C++ and C

```
long r = x;
while (r*r > x)
    r = (r + x/r) / 2;
return r;
```

Python

```
r = x
while r*r > x:
    r = (r + x/r) / 2
return r
```

Ruby

```
r = x
r = (r + x/r) / 2 while r*r > x
r
```

Java and C#

```
long r = x;
while (r*r > x)
    r = (r + x/r) / 2;
return (int) r;
```

JavaScript

```
r = x;
while (r*r > x)
    r = ((r + x/r) / 2) | 0;
return r;
```

Explanation

Apparently, using only integer division for the Newton method works. And I guessed that if I start at x, the root candidate will decrease monotonically and never get too small.

The above solutions all got accepted, and in C++ I also verified it locally on my PC for all possible inputs (0 to 2147483647):

```cpp
#include <iostream>
#include <climits>
using namespace std;

int mySqrt(int x) {
    long long r = x;
    while (r*r > x)
        r = (r + x/r) / 2;
    return r;
}

int main() {
    for (long long x=0; x<=INT_MAX; ++x) {
        long long r = mySqrt(x);
        if (r<0 || r*r > x || (r+1)*(r+1) <= x)
            cout << "false: " << x << " " << r << endl;
        if (x % 10000000 == 0)
            cout << x << endl;
    }
    cout << "all checked" << endl;
}
```

written by StefanPochmann original link here

From Leetcoder.