Average Rating: 4.67 (33 votes)

Implement int sqrt(int x). Compute and return the square root of x, where x is guaranteed to be a non-negative integer.

Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

Example 1:

Input: 4 Output: 2

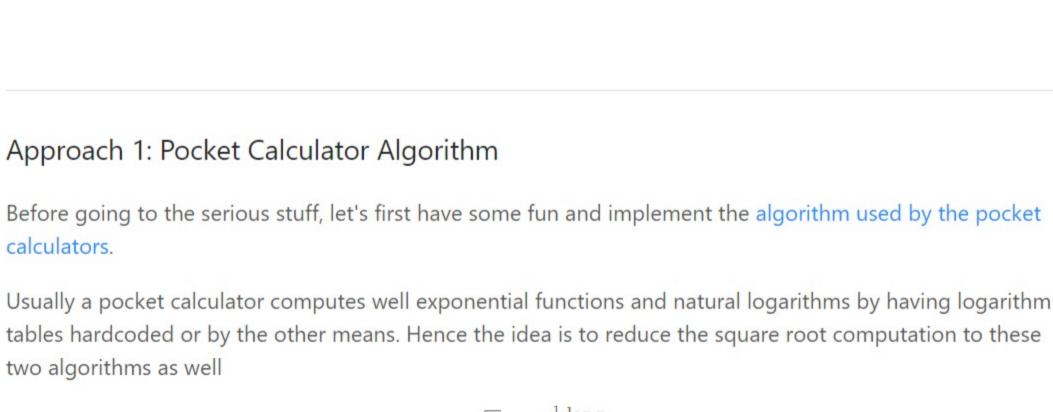
Example 2:

Input: 8 Output: 2 Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

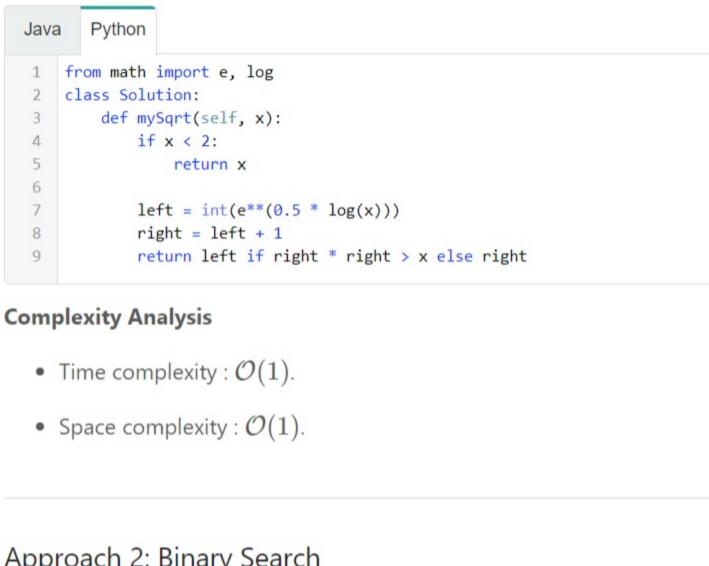
calculators.

9 x = 1016

4



|lee+code



x = 10

pivot

5 left, right = 2, $\times // 2$ 6 7 while left <= right: 8 9

if num > x:

elif num < x:

right = pivot -1

left = pivot + 1

return pivot

18 return right **Complexity Analysis**

that results in $\mathcal{O}(n^{\log_b a} \log^{d+1} N) = \mathcal{O}(\log N)$ time complexity.

Let's compute time complexity with the help of master theorem $T(N) = aT\left(rac{N}{b}
ight) + \Theta(N^d)$. The

equation represents dividing the problem up into a subproblems of size $\frac{N}{b}$ in $\Theta(N^d)$ time. Here at

step there is only one subproblem a = 1, its size is a half of the initial problem b = 2, and all this

happens in a constant time d=0 . That means that $\log_b a=d$ and hence we're dealing with case 2

How to go down? $mySqrt(x) = 2 \times mySqrt\left(\frac{x}{4}\right)$ x << y that means $x \times 2^y$ x >> y that means

Approach 4: Newton's Method Intuition

proceed iteratively.

Implementation

Java

1 2

3

4

5 6

8

9

10 11 12 Python

Complexity Analysis

class Solution:

def mySqrt(self, x):

return x

return int(x1)

x1 = (x0 + x / x0) / 2

while abs(x0 - x1) >= 1:

x1 = (x0 + x / x0) / 2

• Time complexity : $\mathcal{O}(\log N)$ since the set converges quadratically.

if x < 2:

x0 = x

Compare Approaches 2, 3 and 4

• Space complexity : $\mathcal{O}(1)$.

anyone can tell me why return right? **SHOW 7 REPLIES**

40

Type comment here... (Markdown is supported)

screem 🖈 12 🗿 August 10, 2019 6:52 PM

In Approach 2: can someone explain why

pivot = (left + right) / 2;

dilit * 37 • March 26, 2020 5:13 AM

why not simply

SHOW 1 REPLY

pivot = left + (right - left) / 2;

Read More 4 A V C Share Reply **SHOW 2 REPLIES** pianoguy ★ 7 ② January 15, 2020 2:12 PM There is a Chinese traditional method based on handy calculation: Yanghui Caosuan Solution%3A-Yanghui-Suancao-()

carlos_faria ★ 2 ② April 20, 2020 8:09 PM Why do we have those spikes in the Binary Search iterations? 1 A V C Share Share **SHOW 1 REPLY**

life.

Implementation

Approach 2: Binary Search Intuition Let's go back to the interview context. For $x \geq 2$ the square root is always smaller than x/2 and larger than 0: 0 < a < x/2.Since a is an integer, the problem goes down to the iteration over the sorted set of integer numbers. Here the binary search enters the scene. right left

left

5

Сору

Сору

Сору

120

140

Next **1**

Sort By ▼

Post

100

Algorithm • If x < 2, return x. • Set the left boundary to 2, and the right boundary to x / 2. While left <= right: • Take num = (left + right) / 2 as a guess. Compute num * num and compare it with x: ■ If num * num > x, move the right boundary right = pivot -1 ■ Else, if num * num < x, move the left boundary left = pivot + 1 ■ Otherwise num * num == x, the integer square root is here, let's return it Return right Implementation Python class Solution: def mySqrt(self, x): if x < 2: return x

• Time complexity : $\mathcal{O}(\log N)$.

Intuition

Java

1 2

3

4

10

11 12

13

14 15 16

17

Python

class Solution:

def mySqrt(self, x):

• Time complexity : $\mathcal{O}(\log N)$.

Java

2

• Space complexity : $\mathcal{O}(1)$.

Approach 3: Recursion + Bit Shifts

if x < 2: 3 4 return x 5 6 left = self.mySqrt(x >> 2) << 1right = left + 1 7 return left if right * right > x else right 8 **Complexity Analysis**

Let's compute time complexity with the help of master theorem $T(N)=aT\left(rac{N}{b}
ight)+\Theta(N^d)$. The

equation represents dividing the problem up into a subproblems of size $\frac{N}{b}$ in $\Theta(N^d)$ time. Here at

step there is only one subproblem a = 1, its size is a half of the initial problem b = 2, and all this

happens in a constant time d=0 . That means that $\log_b a=d$ and hence we're dealing with case 2

fun, so here is a link if you'd like to dive in.

that results in $\mathcal{O}(n^{\log_b a} \log^{d+1} N) = \mathcal{O}(\log N)$ time complexity.

• Space complexity : $\mathcal{O}(\log N)$ to keep the recursion stack.

Here we have three algorithms with a time performance $\mathcal{O}(\log N)$, and it's a bit confusing. Which one is performing less iterations? Let's run tests for the range of x in order to check that. Here are the results. The best one is Newton's method.

fmxy 🛊 52 ② May 12, 2020 7:13 PM I would be triggered if this problem came up in an interview 7 A V Share Share Reply

Here comes the codes: https://leetcode.com/problems/sqrtx/discuss/480993/A-Chinese-Traditional-

b23our58ne ★21 ② November 6, 2019 6:56 AM Another great improvement can be made to the Newton's method implementation here: since we know that for all integers x > 2, sqrt(x) is always < x / 2.

> Did a summary of Newton method and important steps of the derivation

Solution Integer Square Root The value a we're supposed to compute could be defined as $a^2 \leq x < (a+1)^2$. It is called *integer square* root. From geometrical points of view, it's the side of the largest integer-side square with a surface less than X.

Approach 1: Pocket Calculator Algorithm

Before going to the serious stuff, let's first have some fun and implement the algorithm used by the pocket calculators.

Usually a pocket calculator computes well exponential functions and natural logarithms by having logarithm tables hardcoded or by the other means. Hence the idea is to reduce the square root computation to these two algorithms as well
$$\sqrt{x} = e^{\frac{1}{2}\log x}$$

That's some sort of cheat because of non-elementary function usage but it's how that actually works in a real life.

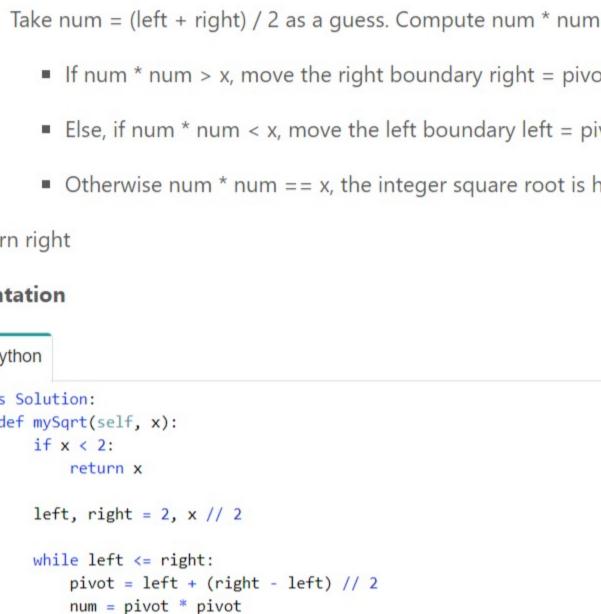
Сору Java

pivot

right

3

2



Let's use recursion. Bases cases are $\sqrt{x}=x$ for x<2. Now the idea is to decrease x recursively at each step to go down to the base cases. For example, let's notice that $\sqrt{x}=2 imes\sqrt{\frac{x}{4}}$, and hence square root could be computed recursively as One could already stop here, but let's use left and right shifts, which are quite fast manipulations with bits That means one could rewrite the recursion above as mySqrt(x) = mySqrt(x >> 2) << 1in order to fasten up the computations. Implementation

One of the best and widely used methods to compute sqrt is Newton's Method. Here we'll implement the version without the seed trimming to keep things simple. However, seed trimming is a bit of math and lot of Let's keep the mathematical proofs outside of the article and just use the textbook fact that the set $x_{k+1} = rac{1}{2} \left[x_k + rac{x}{x_k}
ight]$

converges to \sqrt{x} if $x_0=x$. Then the things are straightforward: define that error should be less than 1 and

number of iterations

Rate this article: * * * *

Preview

Binary Search **Newtons Method**

Recursion + Bit Shifts

log(x)

10

O Previous

Comments: 17

return Math.floor(Math.sqrt(x)) 12 A V C Share Reply **SHOW 7 REPLIES** motodev ★ 5 ② February 1, 2020 12:26 AM

Solution 1 is plain silly! You might as well do x ** 0.5 to save a step :-) nashshm1 🛊 2 🗿 August 23, 2019 8:51 AM Can anyone please explain why in Binary Search approach 'pivot' is multiplied twice? 2 A V C Share Reply

instead of initializing x0 = x at the beginning, we should initialize x0 = x // 2Read More SHOW 1 REPLY

https://leetcode.com/problems/sqrtx/discuss/359172/Python-Newton-Solution

amchoukir * 183 • August 15, 2019 5:32 PM