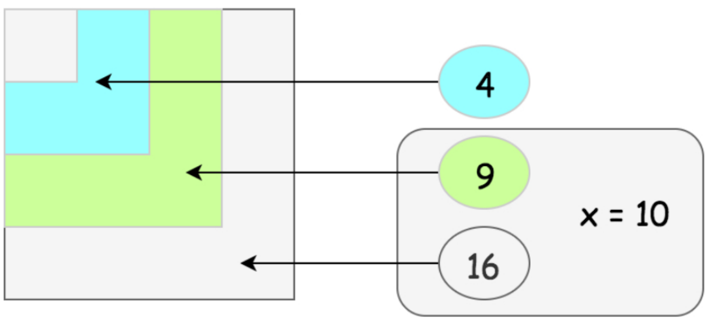
Average Rating: 4.39 (66 votes) Premium

Solution

Integer Square Root

The value a we're supposed to compute could be defined as $a^2 \le x < (a+1)^2$. 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.



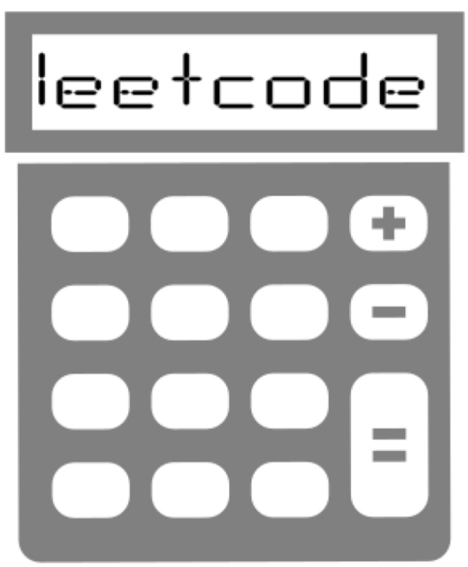
Approach 1: Pocket Calculator Algorithm

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.



Implementation

Java Python Copy

```java
class Solution {
  public int mySqrt(int x) {
    if (x < 2) return x;

    int left = (int)Math.pow(Math.E, 0.5 * Math.log(x));
    int right = left + 1;
    return (long)right * right > x ? left : right;
  }
}
```

Complexity Analysis

- Time complexity : $\mathcal{O}(1)$

C++

Autocomplete

```cpp
class Solution {
public:

    double mySqrtD(double value, double EPS = 1e-9) {
        double start = 0, end = value;

        while (end - start > EPS) {
            double mid = start + (end - start) / 2;
            if (mid * mid - value < 0.0)
                start = mid;
            else
                end = mid;
        }
        return start + 1e-9;
    }

    int mySqrt(int x) {
        return mySqrtD(x);
    }
};
```

Testcase Run Code Result Debugger

Accepted Runtime: 4 ms

Your input	4
Output	2
Expected	2

Diff

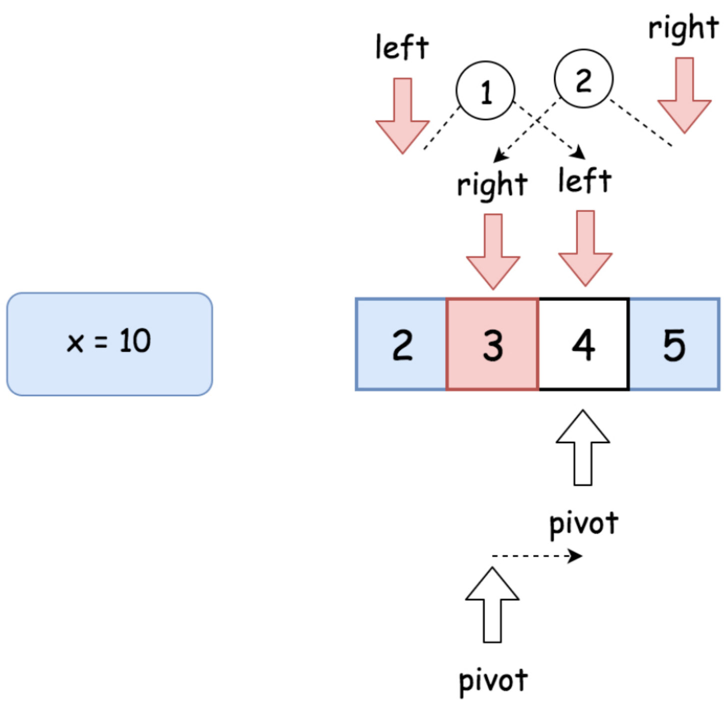
Console Use Example Testcases

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

Approach 2: Binary Search

Intuition

Let's go back to the interview context. For $x \ge 2$ 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.



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

Java | Python

Copy

```java
class Solution {
  public int mySqrt(int x) {
    if (x < 2) return x;

    long num;
    int pivot, left = 2, right = x / 2;
    while (left <= right) {
      pivot = left + (right - left) / 2;
      num = (long)pivot * pivot;
      if (num > x) right = pivot - 1;
      else if (num < x) left = pivot + 1;
      else return pivot;
    }

    return right;
  }
}
```

Complexity Analysis

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

  Let's compute time complexity with the help of master theorem $T(N) = aT\left(\frac{N}{b}\right) + \Theta(N^d)$. 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 that results in $\mathcal{O}(n^{\log_b a} \log^{d+1} N)$ = $\mathcal{O}(\log N)$ time complexity.

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

Approach 3: Recursion + Bit Shifts

Intuition

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.

> How to go down?

For example, let's notice that $\sqrt{x} = 2 \times \sqrt{\frac{x}{4}}$, and hence square root could be computed recursively as

$$\text{mySqrt}(x) = 2 \times \text{mySqrt}\left(\frac{x}{4}\right)$$

One could already stop here, but let's use left and right shifts, which are quite fast manipulations with bits

$$x << y \qquad \text{that means} \qquad x \times 2^y$$

$$x >> y \qquad \text{that means} \qquad \frac{x}{2^y}$$

That means one could rewrite the recursion above as

$$\text{mySqrt}(x) = \text{mySqrt}(x >> 2) << 1$$

in order to fasten up the computations.

Implementation

Java | Python

Copy

```java
class Solution {
  public int mySqrt(int x) {
    if (x < 2) return x;

    int left = mySqrt(x >> 2) << 1;
    int right = left + 1;
    return (long)right * right > x ? left : right;
  }
}
```

Complexity Analysis

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

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

Approach 4: Newton's Method

Intuition

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

Let's keep the mathematical proofs outside of the article and just use the textbook fact that the set

$$x_{k+1} = \frac{1}{2}\left[x_k + \frac{x}{x_k}\right]$$

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

Implementation

Java | Python

Copy

```java
class Solution {
  public int mySqrt(int x) {
    if (x < 2) return x;

    double x0 = x;
    double x1 = (x0 + x / x0) / 2.0;
    while (Math.abs(x0 - x1) >= 1) {
      x0 = x1;
      x1 = (x0 + x / x0) / 2.0;
    }

    return (int)x1;
  }
}
```

Complexity Analysis

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

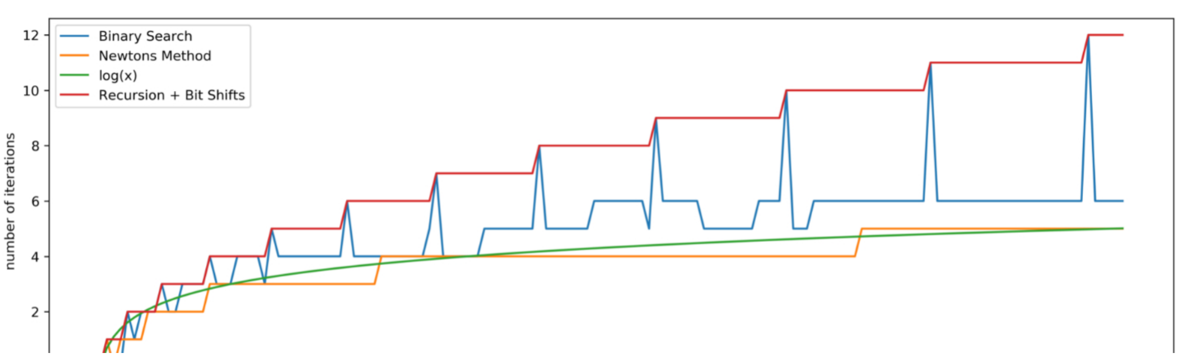
- Space complexity : $\mathcal{O}(1)$.

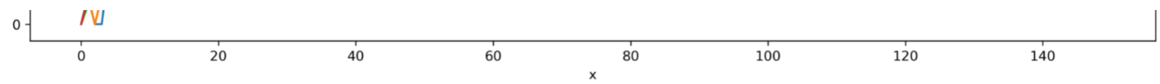
Compare Approaches 2, 3 and 4

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.





Report Article Issue

Comments: 53

Best | Most Votes | Newest to Oldest | Oldest to Newest

Type comment here... (Markdown is supported)

Post

frederikmuller ★ 486 May 12, 2020 6:43 AM

I would be triggered if this problem came up in an interview

205 Show 9 replies Reply

patrickzzz ★ 72 August 15, 2019 11:15 AM

anyone can tell me why return right ?

48 Show 14 replies Reply

motodev ★ 29 Last Edit: January 31, 2020 10:57 AM

In Approach 2: can someone explain why

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

why not simply

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

what's difference?

27 Show 7 replies Reply

pianoguy ★ 26 January 15, 2020 12:42 AM

There is a Chinese traditional method based on handy calculation: Yanghui Caosuan

Here comes the codes: https://leetcode.com/problems/sqrtx/discuss/480993/A-Chinese-Traditional-Solution%3A-Yanghui-Suancao-()

18 Reply

vik101 ★ 9 September 24, 2020 3:01 AM

For the binary search method, in order to avoid the overflowing of an int and casting to long, we can replace lines 10-12 with:

```
if(x / pivot == pivot) return pivot;
else if(x / pivot > pivot) left = pivot;
else right = pivot;
```

This way we use division to rather than multiplication. Is this solution sound?

9 Show 1 reply Reply

ztztzt8888 ★ 260 January 24, 2021 1:21 AM

I don't understand why the binary search has be so complicated. I understand the 2nd solution from math, the curve is under x/2 after 1. But it doesn't speed up much from the standard binary search, because it only takes one step to cut the range in half anyway. The time complicity doesn't change. Standard no frial, good old binary search works just fine.

```
    public int mySqrt(int x) {
        int left = 0, right = x;
        while (left <= right) {
            int pivot = left + (right - left) / 2;
            long product = (long) pivot * pivot;
            if (product > x) {
                right = pivot - 1;
            } else if (product < x) {
```

Read More

5 Show 2 replies Reply

dilit ★ 291 March 25, 2020 4:43 PM

Solution 1 is plain silly! You might as well do x ** 0.5 to save a step :-)

10 Show 1 reply Reply

screem ★ 29 August 10, 2019 6:22 AM

return Math.floor(Math.sqrt(x))

20 Show 10 replies Reply

nashshm1 ★ 3 August 22, 2019 8:21 PM

Can anyone please explain why in Binary Search approach 'pivot' is multiplied twice?

3 Show 1 reply Reply

10shibu ★ 3 June 17, 2020 4:57 PM

If I don't do the cast" (long)pivot * pivot;", I got wrong results. Could anyone explain why? thanks!

2 Show 2 replies Reply