Previous Next

367. Valid Perfect Square

Oct. 18, 2019 | 21.7K views

Average Rating: 4.93 (14 votes)

Given a **positive** integer *num*, write a function which returns True if *num* is a perfect square else False.

Follow up: Do not use any built-in library function such as `sqrt`.

Example 1:

```
Input: num = 16
Output: true
```

Example 2:

```
Input: num = 14
Output: false
```

Constraints:

- `1 <= num <= 2^31 - 1`

Solution

Overview

Square root related problems usually could be solved in logarithmic time. There are three standard logarithmic time approaches, listed here from the worst to the best:

- Recursion. The slowest one.
- Binary Search. The simplest one.
- Newton's Method. The fastest one, and therefore widely used in dynamical simulations.

The last two algorithms are interesting ones, let's discuss them in details.

These solutions have the same starting point. If one knows an integer square x of num, the answer is straightforward: num is a perfect square if $x * x == \mathrm{num}$. Hence the problem is to compute this integer square.

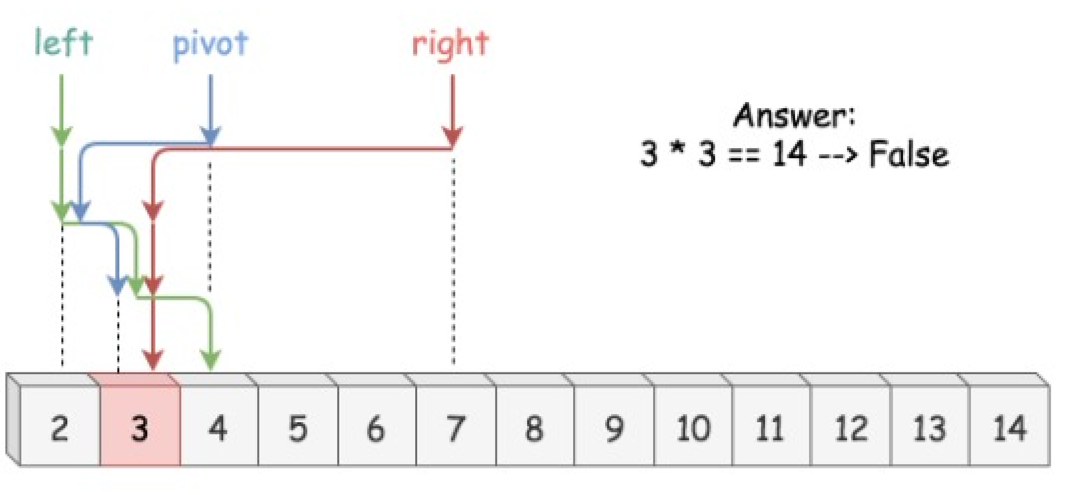
Approach 1: Binary Search

For $\mathrm{num} > 2$ the square root a is always less than $\mathrm{num}/2$ and greater than 1: $1 < x < \mathrm{num}/2$. Since x is an integer, the problem goes down to the search in the sorted set of integer numbers. Binary search is a standard way to proceed in such a situation.

Algorithm

- If num < 2, return True.
- Set the left boundary to 2, and the right boundary to num / 2.
- While left <= right:
  - Take x = (left + right) / 2 as a guess. Compute guess_squared = x * x and compare it with num:
    - If guess_squared == num, then the perfect square is right here, return True.
    - If guess_squared > num, move the right boundary right = x - 1.
    - Otherwise, move the left boundary left = x + 1.
- If we're here, the number is not a prefect square. Return False.

Implementation



Java | Python | Copy

```python
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        if num < 2:
            return True

        left, right = 2, num // 2

        while left <= right:
            x = left + (right - left) // 2
            guess_squared = x * x
            if guess_squared == num:
                return True
            if guess_squared > num:
                right = x - 1
            else:
                left = x + 1

        return False
```

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 2: Newton's Method

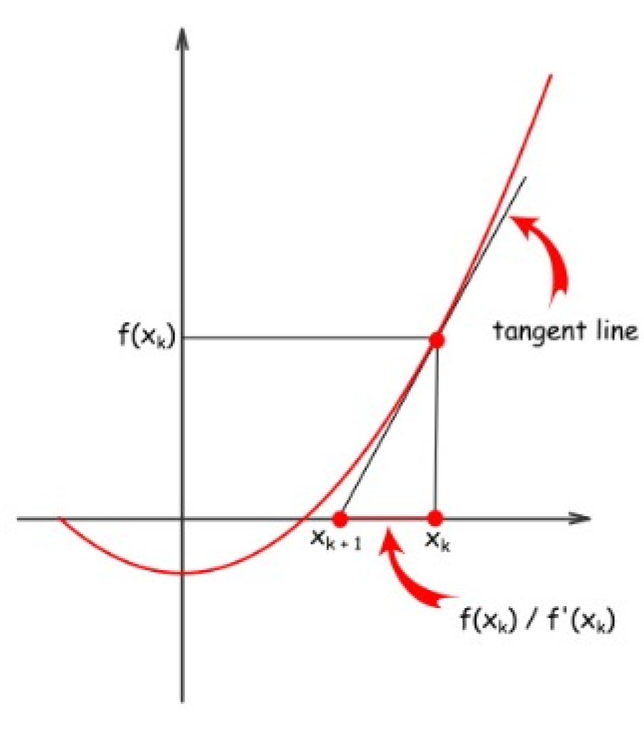
Newton's Algorithm: How to Figure out the Formula

Let's do a very rough derivation of Newton's sequence which could be done in two minutes during the interview. Please note that it's more a way to memorize than a strict mathematical proof.

The problem is to find a root of

$$f(x) = x^2 - \mathrm{num} = 0$$

The idea of Newton's algorithm is to start from a seed (= initial guess) and then to compute a root as a sequence of improved guesses.



For example, there is a guess x_k. To compute next guess x_{k+1}, let's approximate $f(x_k)$ by its tangent line, that would result in

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Now use $f(x_k) = x_k^2 - \mathrm{num}$ and $f'(x_k) = 2x_k$, and voila the result

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{\mathrm{num}}{x_k}\right)$$

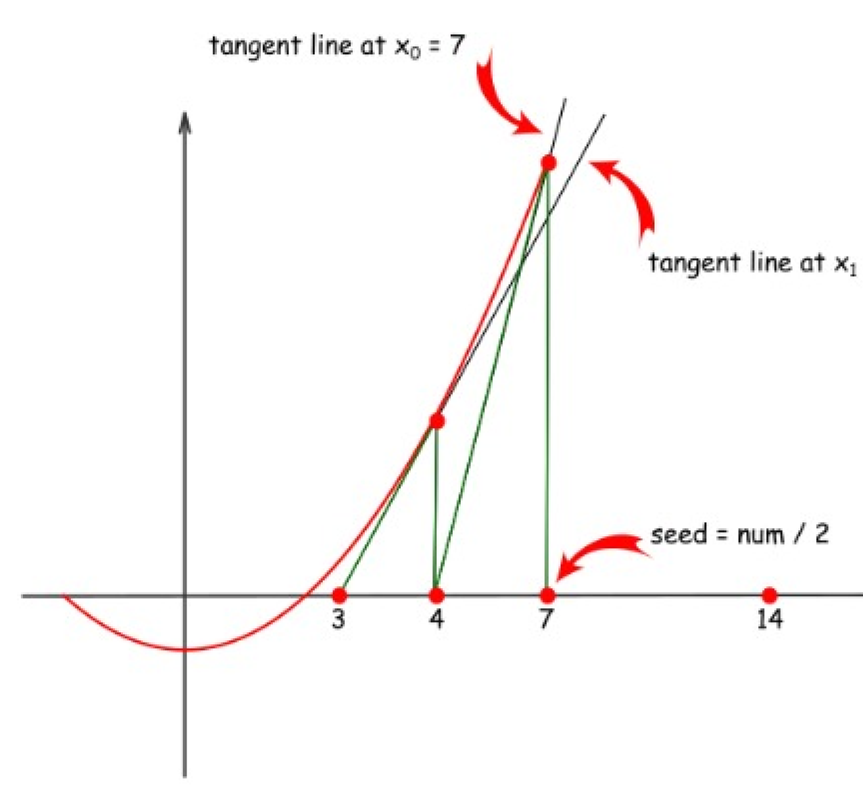
Choose a seed

How to choose a seed? Since the function $f(x) = x^2 - \mathrm{num}$ is monotonous, the smaller seed the better, so let's take $\mathrm{num}/2$.

Algorithm

- Take num / 2 as a seed.
- While x * x > num, compute the next guess using Newton's method: $x = \frac{1}{2}\left(x + \frac{\mathrm{num}}{x}\right)$.
- Return x * x == num

Implementation



Java | Python | Copy

```python
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        if num < 2:
            return True

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

Complexity Analysis

- Time complexity : $\mathcal{O}(\log N)$ because guess sequence converges quadratically.
- Space complexity : $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 9

Sort By

Type comment here... (Markdown is supported)

Preview Post

yu_chien ★ 18 October 20, 2019 6:50 PM

you god dam genius

5 Share Reply

mustafafirtina ★ 3 October 19, 2019 8:47 PM

Hi excellent solution and explanation but I think we dont need to check "return (right * right == num)" after binary search. Because all cases should be done at the binary search. Just return false.

2 Share Reply

SHOW 1 REPLY

nazariyv ★ 12 May 9, 2020 11:29 PM

If you need to jog your memory about the Newton's method derivation:
https://www.youtube.com/watch?v=cOmAk82cr9M

Read More

3 Share Reply

SHOW 1 REPLY

ady_sn ★ 30 April 30, 2020 1:53 AM

How is Zero not a perfect Square. The Test Cases Expects False for 0

1 Share Reply

SHOW 4 REPLIES

hungry_coder_n ★ 11 June 30, 2020 11:54 AM

When should we use long over int? If I use `int` in the first approach, It failed for the test case `808201`. I couldn't understand why this test case failed though there's no overflow. I was using `start + (end - start) / 2`

0 Share Reply

SHOW 1 REPLY

133c7 ★ 17 May 26, 2020 11:34 AM

Something that got me: The reason we are using `long` data type in the second approach to hold the current approximated value is to prevent integer overflow. The question states that `num` fits in a 32-bit signed integer (max value `2^31-1`). Therefore `x` could be as large as `((2^31-1)/2)^2 = 1152921503533105200` which will cause wrap around if we use `int` which is 32-bit.

0 Share Reply

nazariyv ★ 12 May 10, 2020 1:17 AM

I think this reads a bit easier

```
    def isPerfectSquare(self, num: int) -> bool:
        if num < 2: return True
```

Read More

0 Share Reply

c0re5 ★ 8 May 9, 2020 6:44 PM

Are you expected to remember Master's theorem in an FANG interview setting?

0 Share Reply

SHOW 6 REPLIES

nazariyv ★ 12 May 9, 2020 5:04 PM

Approach 1 correction: the inequality sign has to be less or equal, not just less. Take `num = 4`, then `num / 2 = 2`, according to you `1 < x < num / 2`, but `num / 2 = 2`, so the inequality must be `1 < x <= num / 2`

0 Share Reply