Square root

Given an integer **X**, find its square root. If **X** is not a perfect square, then return **floor(√x)**.

Examples :

> ***Input:*** *x = 4*
> ***Output:*** *2*
> ***Explanation:*** *The square root of 4 is 2.*
>
> ***Input:*** *x = 11*
> ***Output:*** *3*
> ***Explanation:*** *The square root of 11 lies in between 3 and 4 so floor of the square root is 3.*

Naive Approach: To find the floor of the square root, try with all-natural numbers starting from 1. Continue incrementing the number until the square of that number is greater than the given number.

Follow the steps below to implement the above idea

1. Create a variable (counter) ***i*** and take care of some base cases, (i.e when the given number is 0 or 1).
2. Run a loop until ***i*i <= n***, where n is the given number. Increment i by 1.
3. The floor of the square root of the number is *i – 1*

Below is the implementation of the above approach:

C++ | Java

```
// A C++ program to find floor(sqrt(x)
#include <bits/stdc++.h>
using namespace std;

// Returns floor of square root of x
int floorSqrt(int x)
```

```
    if (x == 0 || x == 1)
        return x;

    // i*i is greater than or equal to x.
    int i = 1, result = 1;
    while (result <= x) {
        i++;
        result = i * i;
    }
    return i - 1;
}

// Driver program
int main()
{
    int x = 11;
    cout << floorSqrt(x) << endl;
    return 0;
}
```

Output

```
3
```

Complexity Analysis:

- **Time Complexity:** O(√X). Only one traversal of the solution is needed, so the time complexity is O(√X).
- **Auxiliary Space:** O(1).

Square root an integer using Binary search:

> *The idea is to find the largest integer **i** whose square is less than or equal to the given number. The values of **i * i** is monotonically increasing, so the problem can be solved using binary search.*

Below is the implementation of the above idea:

1. Base cases for the given problem are when the given number is **0** or **1**, then

2. Create some variables, for storing the lower bound say ***l = 0,*** *and for upper*

4. Check if the square of mid (***mid = (l + r)/2***) is less than or equal to **X**, If yes then search for a larger value in the second half of the search space, i.e **l = mid + 1**, update **ans = mid**
5. Else if the square of mid is more than **X** then search for a smaller value in the first half of the search space, i.e **r = mid – 1**
6. Finally, Return the **ans**

Below is the implementation of the above approach:

C++ | Java


```
#include <iostream>

using namespace std;
int floorSqrt(int x)
{
    // Base cases
    if (x == 0 || x == 1)
        return x;

    // Do Binary Search for floor(sqrt(x))
    int start = 1, end = x / 2, ans;
    while (start <= end) {
        int mid = (start + end) / 2;

        // If x is a perfect square
        int sqr = mid * mid;
        if (sqr == x)
            return mid;

        // Since we need floor, we update answer when
        // mid*mid is smaller than x, and move closer to
        // sqrt(x)

        /*
        if(mid*mid<=x)
                {
```

```
        }
        situation we can use long or we can just divide
            the number by mid which is same as checking
        mid*mid < x

        */
        if (sqr <= x) {
            start = mid + 1;
            ans = mid;
        }
        else // If mid*mid is greater than x
            end = mid - 1;
    }
    return ans;
}

// Driver program
int main()
{
    int x = 11;
    cout << floorSqrt(x) << endl;
    return 0;
}
```

Output

```
3
```

Complexity Analysis:

- **Time Complexity:** O(log(X)).
- **Auxiliary Space:** O(1).

Dash

If you are facing any issue on this page. Please let us know.

All

Articles

Videos

Problems

Quiz

Contest

Menu

Track Progress

23 of **60** Complete. (39%)