Problem 69: Sqrt(x)

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given a non-negative integer

x

, return

the square root of

x

rounded down to the nearest integer

. The returned integer should be

non-negative

as well.

You

must not use

any built-in exponent function or operator.

For example, do not use

pow(x, 0.5)

in c++ or

x ** 0.5

in python.

Example 1:

Input:

x = 4

Output:

2

Explanation:

The square root of 4 is 2, so we return 2.

Example 2:

Input:

x = 8

Output:

2

Explanation:

The square root of 8 is 2.82842..., and since we round it down to the nearest integer, 2 is returned.

Constraints:

0 <= x <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
int mySqrt(int x) {

}
};
```

Java:

```
class Solution {
public int mySqrt(int x) {

}
}
```

Python3:

```
class Solution:
def mySqrt(self, x: int) -> int:
```

Python:

```
class Solution(object):
def mySqrt(self, x):
"""
:type x: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} x
* @return {number}
*/
var mySqrt = function(x) {

};
```

TypeScript:

```
function mySqrt(x: number): number {

};
```

C#:

```
public class Solution {
public int MySqrt(int x) {

}
}
```

C:

```
int mySqrt(int x) {

}
```

Go:

```
func mySqrt(x int) int {

}
```

Kotlin:

```
class Solution {
fun mySqrt(x: Int): Int {

}
}
```

Swift:

```
class Solution {
func mySqrt(_ x: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn my_sqrt(x: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} x
# @return {Integer}
def my_sqrt(x)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @return Integer
*/
function mySqrt($x) {

}
}
```

Dart:

```
class Solution {
int mySqrt(int x) {

}
}
```

Scala:

```
object Solution {
def mySqrt(x: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec my_sqrt(x :: integer) :: integer
def my_sqrt(x) do

end
end
```

Erlang:

```
-spec my_sqrt(X :: integer()) -> integer().
my_sqrt(X) ->
.
```

Racket:

```
(define/contract (my-sqrt x)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int mySqrt(int x) {

}
};
```

Java Solution:

```
/**
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int mySqrt(int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Sqrt(x)
Difficulty: Easy
Tags: math, search

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def mySqrt(self, x: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def mySqrt(self, x):
"""
:type x: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} x
* @return {number}
*/
var mySqrt = function(x) {

};
```

TypeScript Solution:

```
/**
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

function mySqrt(x: number): number {

};
```

C# Solution:

```
/*
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MySqrt(int x) {

}
}
```

C Solution:

```
/*
* Problem: Sqrt(x)
* Difficulty: Easy
* Tags: math, search
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int mySqrt(int x) {

}
```

Go Solution:

```
// Problem: Sqrt(x)
// Difficulty: Easy
// Tags: math, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func mySqrt(x int) int {

}
```

Kotlin Solution:

```
class Solution {
fun mySqrt(x: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func mySqrt(_ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sqrt(x)
// Difficulty: Easy
// Tags: math, search
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn my_sqrt(x: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer} x
# @return {Integer}
def my_sqrt(x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @return Integer
*/
function mySqrt($x) {

}
}
```

Dart Solution:

```
class Solution {
int mySqrt(int x) {

}
}
```

Scala Solution:

```
object Solution {
def mySqrt(x: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec my_sqrt(x :: integer) :: integer
def my_sqrt(x) do

end
end
```

Erlang Solution:

```
-spec my_sqrt(X :: integer()) -> integer().
my_sqrt(X) ->
.
```

Racket Solution:

```
(define/contract (my-sqrt x)
(-> exact-integer? exact-integer?)
)
```