Assignment 1

Name : Hrishikesh Rajan
Email : hrishikeshrajan3@gmail.com
LinkedIn : https://www.linkedin.com/in/hrishikesh-rajan-96aa70165

---

3) Given a positive integer num, write a program that returns True if num is a perfect square else return False. Do not use built-in functions like sqrt. Also, talk about the time complexity of your code.

Test Cases:

Input: 16
Output: True

Input: 14
Output: False

*A Javascript implementation is provided

Answer :

```
function calculate(num,low,high){
    if(low <= high){

        let mid = Math.floor(low + ((high -low)/2))

        if((mid * mid) === num){
            return true;
        }

        if((mid * mid) > num){
            return calculate(num,low,mid-1)
        }

        if( (mid * mid) < num){
            return calculate(num,mid+1,high)
        }
     }

    return false
}
```

```
function isPerfectSquare(num){
      return calculate(num,0,num);
}
const result = isPerfectSquare(16);
console.log(result)
```
Output : true

Time Complexity :
The recurrence relation is formed by

T(n) = T(n/2) + c , c = constant. (1)

The reason for T(n/2) is that at a time our search space for multiplication was limited to half of n, that is n/2, where n is the input number.

Applying Master's Theorem

T(n) = aT(n/b) + f(n) , where f(n) = Θ ($n^k log^p n$ **)** (2)

From the above recurrence relation (1)
a=1
b=2
k=0
p=0

Substituting values
$log\ {}^{b}_{a}$ **=** $log\ {}^{1}_{2}$ **=** 0
That means $log\ {}^{b}_{a}$ **= k** and **P > -1,** This relation comes under case 2.
Then the equation becomes **Θ (**$n^k log^{p+1} n$ **)**

= Θ ($n^0 log^{0+1} n$ **)**
= Θ ($1 * log^1 n$ **)**
= Θ ($log\ n$ **)**

Time Complexity = O(logn)
Space Complexity = O(1)