FizzBuzz Big O Problem 5

September 22, 2017

Question

Given the following code, explain the run-time and Big O. This code computes the integer square root of a number. If the integer square root does not exist, our code returns -1. The code works by guessing the min or max value until we get a match.

Explanation and Algorithm

An explanation of Big-O:

Big O is how we describe the efficiency of algorithms. This is an important concept, because a program that is time and space efficient saves a company time and money. For time complexity, think of Big O in terms of best case (fastest), worst case (slowest), and average/expected case. For many coding algorithms, the worst case and average case will actually be the same.

In addition to time complexity, you will often be asked in an interview about space complexity. The Big O of time complexity and the Big O of space complexity can often be different.

When thinking about the Big O of your algorithm, write a formula for your run time and then drop the constants. For example, if you have an algorithm with two for loops, the run time is $0(n)$, not $O(2n)$. In addition to this, it is also important to remember to drop the non dominant terms. For example, $O(N^2 + N)$ should simplify to $O(N^2)$.

The Big O times in order from most efficient to least efficient are the following: $O(1)$, $O(logn)$, $O(nlogn)$, $O(n^2)$, $O(2^n)$, $O(n!)$.

Hints

1. Think about how many times an operation is conducted in this problem. What are the main operations?

2. An explanation of $O(n)$: This means an algorithm is being run in constant time.

3. An explanation of $O(logn)$: If n number of operations get divided at each run time.

4. An explanation of $O(n^2)$: If n times n operations are performed.

5. An explanation of $O(2^n)$: If the number of operations increases exponentially.

Code

```
/* Code */

int sqrt(int n){
   return sqrt_helper(n,1,n);
}

int sqrt_helper(int n, int min, int max){
   if (max<min) return -1; //no square root

    int guess = (min+max)/2;
    if(guess*guess==n){ //found it!
      return guess;
    } else if(guess*guess<n){
        return sqrt_helper(n, guess+1, max); //try higher
    } else { //too high
      return sqrt_helper(n, min, guess-1); //try lower
    }

}
```

Run time analysis

The best case run time of this algorithm is O(1). If max¡min, we simply return -1, and if guess*guess==n, we return guess.

Our average case and our worst case is O(log n). In each iteration, we keep returning min+max/2 until we get guess*guess/2. At each iteration, our range of numbers we are guessing from to obtain the square root gets halved. To put it simply, we perform a binary search to find the square root. Thus, our runtime is O(log n).