374. Guess Number Higher or Lower

July 21, 2016 | 74K views

Average Rating: 4.89 (45 votes)

We are playing the Guess Game. The game is as follows:

I pick a number from **1** to *n*. You have to guess which number I picked.

Every time you guess wrong, I'll tell you whether the number is higher or lower.

You call a pre-defined API `guess(int num)` which returns 3 possible results (`-1`, `1`, or `0`):

```
-1 : My number is lower
 1 : My number is higher
 0 : Congrats! You got it!
```

Example :

```
Input: n = 10, pick = 6
Output: 6
```

Solution

Approach 1: Brute Force

We check every number from 1 to n-1 and pass it to the *guess* function. The number for which a 0 is returned is the required answer.

Java

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        for (int i = 1; i < n; i++)
            if (guess(i) == 0)
                return i;
        return n;
    }
}
```

Complexity Analysis

- Time complexity : $O(n)$. We scan all the numbers from 1 to n.
- Space complexity : $O(1)$. No extra space is used.

Approach 2: Using Binary Search

Algorithm

We can apply Binary Search to find the given number. We start with the mid number. We pass that number to the *guess* function. If it returns a -1, it implies that the guessed number is larger than the required one. Thus, we use Binary Search for numbers lower than itself. Similarly, if it returns a 1, we use Binary Search for numbers higher than itself.

Java

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int low = 1;
        int high = n;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            int res = guess(mid);
            if (res == 0)
                return mid;
            else if (res < 0)
                high = mid - 1;
            else
                low = mid + 1;
        }
        return -1;
    }
}
```

Complexity Analysis

- Time complexity : $O(\log_2 n)$. Binary Search is used.
- Space complexity : $O(1)$. No extra space is used.

Approach 3: Ternary Search

Algorithm

In Binary Search, we choose the middle element as the pivot in splitting. In Ternary Search, we choose two pivots (say $m1$ and $m2$) such that the given range is divided into three equal parts. If the required number num is less than $m1$ then we apply ternary search on the left segment of $m1$. If num lies between $m1$ and $m2$, we apply ternary search between $m1$ and $m2$. Otherwise we will search in the segment right to $m2$.

Java

```
/* The guess API is defined in the parent class GuessGame.
   @param num, your guess
   @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
      int guess(int num); */

public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int low = 1;
        int high = n;
        while (low <= high) {
            int mid1 = low + (high - low) / 3;
            int mid2 = high - (high - low) / 3;
            int res1 = guess(mid1);
            int res2 = guess(mid2);
            if (res1 == 0)
                return mid1;
            if (res2 == 0)
                return mid2;
            else if (res1 < 0)
                high = mid1 - 1;
            else if (res2 > 0)
                low = mid2 + 1;
            else {
                low = mid1 + 1;
                high = mid2 - 1;
            }
        }
        return -1;
    }
}
```

Complexity Analysis

- Time complexity : $O(\log_3 n)$. Ternary Search is used.
- Space complexity : $O(1)$. No extra space is used.

Follow up

It seems that ternary search is able to terminate earlier compared to binary search. But why is binary search more widely used?

Comparisons between Binary Search and Ternary Search

Ternary Search is worse than Binary Search. The following outlines the recursive formula to count comparisons of Binary Search in the worst case.

$$T(n) = T\left(\frac{n}{2}\right) + 2, \quad T(1) = 1$$

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2^2}\right) + 2$$

$$\begin{aligned} \therefore \quad T(n) &= T\left(\frac{n}{2^2}\right) + 2 \times 2 \\ &= T\left(\frac{n}{2^3}\right) + 3 \times 2 \\ &= \dots \\ &= T\left(\frac{n}{2^{\log_2 n}}\right) + 2\log_2 n \\ &= T(1) + 2\log_2 n \\ &= 1 + 2\log_2 n \end{aligned}$$

The following outlines the recursive formula to count comparisons of Ternary Search in the worst case.

$$T(n) = T\left(\frac{n}{3}\right) + 4, \quad T(1) = 1$$

$$T\left(\frac{n}{3}\right) = T\left(\frac{n}{3^2}\right) + 4$$

$$\begin{aligned} \therefore \quad T(n) &= T\left(\frac{n}{3^2}\right) + 2 \times 4 \\ &= T\left(\frac{n}{3^3}\right) + 3 \times 4 \\ &= \dots \\ &= T\left(\frac{n}{3^{\log_3 n}}\right) + 4\log_3 n \\ &= T(1) + 4\log_3 n \\ &= 1 + 4\log_3 n \end{aligned}$$

As shown above, the total comparisons in the worst case for ternary and binary search are $1 + 4\log_3 n$ and $1 + 2\log_2 n$ comparisons respectively. To determine which is larger, we can just look at the expression $2\log_3 n$ and $\log_2 n$. The expression $2\log_3 n$ can be written as $\frac{2}{\log_2 3} \times \log_2 n$. Since the value of $\frac{2}{\log_2 3}$ is greater than one, Ternary Search does more comparisons than Binary Search in the worst case.

Analysis written by: @vinod23

Rate this article: