



Courses

Practice

Roadmap

Pro



Algorithms and Data Structures for Beginners

15 / 35

15 - Search Range

07:33

☐

Mark Lesson Complete





View Code

Prev

Next

16 Binary Tree

Suggested Problems

Status	Star	Problem ↕	Difficulty ↕	Video Solution	Code
<input type="checkbox"/>	☆	Guess Number Higher Or Lower	Easy		
<input type="checkbox"/>	☆	First Bad Version	Easy		
<input type="checkbox"/>	☆	Koko Eating Bananas	Medium		

Binary Search (Search Range)

Concept

Imagine your friend gave you a range from 1 - 100 and asked you to guess the number they were thinking of. There are three outcomes. Either your guess is correct, too small or too big.

If your guess is too small, you will have to guess higher and if your guess it too big, you will have to guess lower. The difference here is that you do not know what

number your friend is thinking of (aka the `target` value). All you have is feedback from them each time you guess, and you adjust your next guess accordingly.

The question here is, how would we go about setting our boundaries? Well, we can apply the standard binary search template that we learned in the previous chapter, except instead of checking if your `mid` equals `target`, you would ask your friend if your guess is correct.

Of course, your friend has a procedure for determining if your guess is too small or too big, which is simple. All they do is compare your guess to the number they are thinking of and let you know if you should guess higher or lower. The pseudocode for that method would look like the following:

```
fn isCorrect(n):  
    // If your guess is too big  
    if n > 10:  
        return 1  
    // If your guess is too small  
    else if n < 10:  
        return -1  
    // Finally, if your guess is correct  
    else:  
        return 0
```

Now, all we need to do is call this function in our binary search function.

```
fn binarySearch(low = 1, high = 100):  
    while low <= high:  
        mid = (low + high) / 2  
        if isCorrect(mid) > 0:  
            high = mid - 1  
        else if isCorrect(mid) < 0:  
            low = mid + 1  
        else:  
            return mid  
    return -1
```

As observed, based on the return value we get from the `isCorrect` function, we can choose to adjust our `high` and `low` accordingly, which, as stated in the previous chapter is just another way of representing the `L` and `R` pointers.

This technique can be confusing and can come across as very subtle because you have to figure out how to modify the typical implementation of binary search. For questions like these, a predefined method API is given, in this case, `isCorrect` and you are required to treat the function as a black-box and use it within your own binary search method.

Time Space Complexity

The work being done is the same as the previous section, which is standard binary search procedure. Therefore, this procedure is also in $O(\log n)$.

Closing Notes

This chapter is mostly review because the algorithm behind this pattern is binary search, which we already covered in detail. This is just a variation that you may see in coding interviews. For now, to solidify your understanding, tackle the questions linked below the video.

Copyright © 2023 NeetCode.io All rights reserved.

Contact: neetcodebusiness@gmail.com

[Github](#) [Privacy](#) [Terms](#)