Programming Exercise 08
Bisection, Guess My Number Game

ISTA-220, C# Step by Step

Bisection Algorithm This exercise implements three programs implementing the bisection algorithm. The bisection method is an efficient way of finding a particular value in a sorted list. It takes a sorted list and a value, and finds the value in the list. First, it checks the "middle" element in the list. There are three possibilities: the value could match the "middle" element, the value could be higher than the 'middle" element of the list, or the value could be lower than the "middle" element. If the value matches, the function returns. If it's higher, the algorithm (recursively) calls itself with the top half of the list. If it's lower, the algorithm calls itself with the bottom half of the list.

For example, here is the output of a function call, the searched value being `7` and the sorted list being `1 2 3 4 5 6 7 8 9 10`.

```
>> int value = 7;
>> int[] list = {1,2,3,4,5,6,7,8,9,10};
>> bisection_search(value, list);
<< value is higher than 5
        //the ‘‘middle’’ value is 5.5, but we are doing integer division
        //the list is now set to {6,7,8,9,10}
>> bisection_search(value, list);
<< value is lower than 8
        //the middle value is 8
        //the list is now set to {6,7}
>> bisection_search(value, list);
<< value is higher than 6
        //the ‘‘middle’’ value is 6.5, but we are doing integer division
        //the list is now set to {7}
>> bisection_search(value, list);
<< value is equal to 7
        //the middle value is 7
        //the list is now set to {7}
<< the value searched for, 7, has been found
```

The bisection algorithm is nice because it is guaranteed to find an answer (or return if there is no answer) in log_2 of the size of the list. Only 10 repetitions of the function are necessary to find a result in a list of 1024 items, and only 20 repetitions to find a result in a list of 1,000,000 items.

Implement bisection algorithm: 70 points Write a console application implementing the bisection algorithm. As the initial list, use an integer array from 1 to 10, like this: `int[] list = 1,2,3,4,5,6,7,8,9,10;`. As input, have the user select a number from 1 to 10. Have the application print each step. Use appropriate exception handling to guard against invalid input from the user.

Guess my number, human plays: 80 points Implement a version of Guess My Number, where the computer randomly choses a number between 1 and 1000, and the human guesses the number. In this case, the program should print a hint with each repetition, either `<Your guess was too high>`, `<Your guess`

`was too low>`, or `<You guessed the number>`. The human should then input the next guess. Run this multiple times and compute the average number of repetitions necessary for you to guess the number.

What is the maximum number of guesses you need to guess a number between 1 and 1000? Recall that $\log_2 1000 = 9.966$ and that $2^{10} = 1024$.

Guess my number, computer plays: 100 points Implement a version of Guess My Number, where the human choses a number between 1 and 100, and the computer guesses the number. The human should be able to tell the computer whether the computer's guess was too high, too low, or was the correct answer. Run this multiple times and compute the average number of repetitions necessary for the computer to guess the number. Have the program print the value, the guess, and the list on each repetition.

Is the human as good as the computer in finding the number?