## A Guessing Game

Let's play a little game to give you an idea of how different algorithms for the same problem can have wildly different efficiencies. The computer is going to randomly select an integer from 1 to 16. You have to guess the number by making guesses until you find the number that the computer chose. Let's begin.



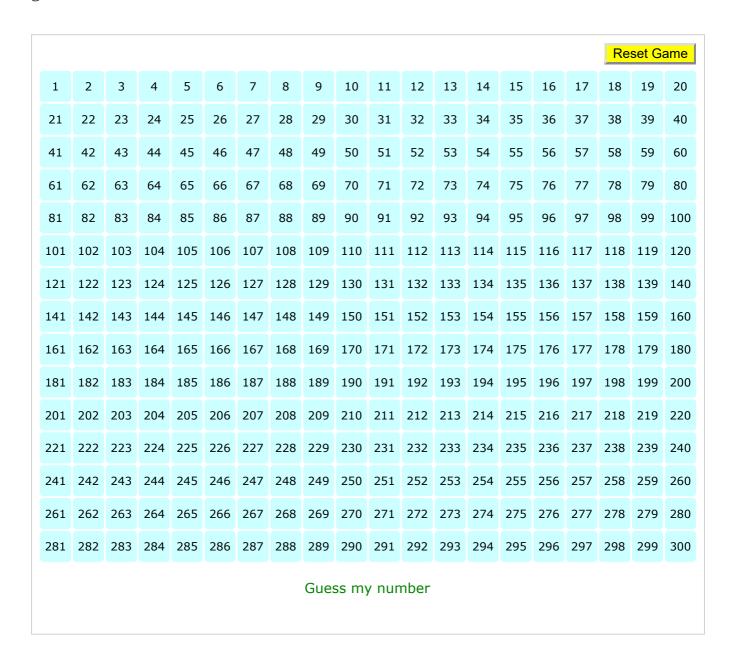
Maybe you guessed 1, then 2, then 3, then 4, and so on, until you guessed the right number. We call this approach **linear search**, because you guess all the numbers as if they were lined up in a row. It would work. But what is the highest number of guesses you could need? If the computer selects 16, you would need 16 guesses. Then again, you could be really lucky, which would be when the computer selects 1 and you get the number on your first guess. How about on average? If the computer is equally likely to select any number from 1 to 16, then on average you'll need 8 guesses.

But you could do something more efficient than just guessing 1, 2, 3, 4, ..., right? Since the computer tells you whether a guess is too low, too high, or correct, you can start off by guessing 15. If the number that the computer selected is less than 15, then because you know that 15 is too high, you can eliminate all the numbers from 15 to 30 from further consideration. If the number selected by the computer is greater than 15, then you can eliminate 1 through 15. Either way, you can eliminate about half the numbers. On your next guess, eliminate half of the remaining numbers. Keep going, always eliminating half of the remaining numbers. We call this halving approach

**binary search**, and no matter which number from 1 to 30 the computer has selected, you should be able to find the number in at most 5 guesses with this technique.

eminiating half of the remaining numbers, we can this harving approach

Here, try it for a number from 1 to 300. You should need no more than 9 guesses.



How many guesses did it take you to find the number this time? Why should you never need more than 9 guesses? (Can you think of a mathematical explanation)?

We'll return to binary search, and we'll see how you can use it to efficiently search for an item in an array. But first, let's look at an algorithm for a trickier problem.