Binary Search

Binary search is an efficient algorithm for finding an item from an ordered list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one. We used binary search in the guessing game in the introductory tutorial.

One of the most common ways to use binary search is to find an item in an array. For example, the Tycho-2 star catalog contains information about the brightest 2,539,913 stars in our galaxy. Suppose that you want to search the catalog for a particular star, based on the star's name. If the program examined every star in the star catalog in order starting with the first, an algorithm called linear search, the computer might have to examine all 2,539,913 stars to find the star you were looking for, in the worst case. If the catalog were sorted alphabetically by star names, binary search would not have to examine more than 22 stars, even in the worst case.

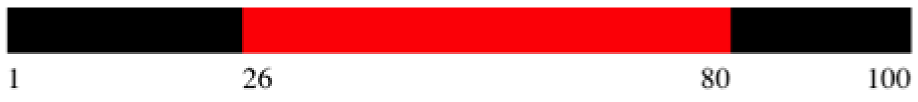
The next few articles discuss how to describe the algorithm carefully, how to implement the algorithm in JavaScript, and how to analyze efficiency.

Pseudocode for binary search

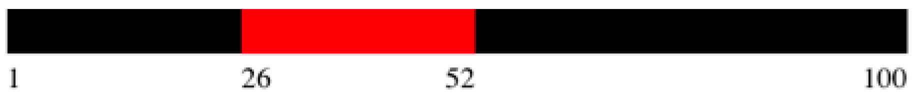
When describing an algorithm to a fellow human being, an incomplete description is often good enough. Some details may be left out of a recipe for a cake; the recipe assumes that you know how to open the refrigerator to get the eggs out and that you know how to crack the eggs. People might intuitively know how to fill in the missing details, but computer programs do not. That's why we need to describe computer algorithms completely.

In order to implement an algorithm in a programming language, you will need to understand an algorithm down to the details. What are the inputs to the problem? The outputs? What variables should be created, and what initial values should they have? What intermediate steps should be taken to compute other values and to ultimately compute the output? Do these steps repeat instructions that can be written in simplified form using a loop?

Let's look at how to describe binary search carefully. The main idea of binary search is to keep track of the current range of reasonable guesses. Let's say that I'm thinking of a number between one and 100, just like the **guessing game**. If you've already guessed 25 and I told you my number was higher, and you've already guessed 81 and I told you my number was lower, then the numbers in the range from 26 to 80 are the only reasonable guesses. Here, the red section of the number line contains the reasonable guesses, and the black section shows the guesses that we've ruled out.



In each turn, you choose a guess that divides the set of reasonable guesses into two ranges of roughly the same size. If your guess is not correct, then I tell you whether it's too high or too low, and you can eliminate about half of the reasonable guesses. For example, if the current range of reasonable guesses is 26 to 80, you would guess the halfway point, **(26+80)/2**, or **53**. If I then tell you that 53 is too high, you can eliminate all numbers from 53 to 80, leaving 26 to 52 as the new range of reasonable guesses, halving the size of the range.



For the guessing game, we can keep track of the set of reasonable guesses using a few variables. Let the variable min be the current minimum reasonable guess for this round, and let the variable max be the current maximum reasonable guess. The input to the problem is the number n, the highest possible number that your opponent is thinking of. We assume that the lowest possible number is one, but it would be easy to modify the algorithm to take the lowest possible number as a second input.

Here's a pseudocode description of binary search:

1. Let min = 1 and max = n.
2. Guess the average of max and min, rounded down so that it is an integer.
3. If you guessed the number, stop. You found it!
4. If the guess was too low, set min to be one larger than the guess.

5. If the guess was too high, set max to be one smaller than the guess.
6. Go back to step two.

We could make this pseudocode even more precise by clearly describing the inputs and the outputs for the algorithm and by clarifying what we mean by instructions like "guess a number" and "stop." But this will do for now.