Project 1 Explanation – The Binary Search

There won't be notes explaining every aspect of the project, as there are many ways to approach it. However, there is an example program, and these notes about the key concept: the binary search.

There are many types of searching and sorting things in programs, but divide-and-conquer algorithms tend to be some of the most efficient. On each repetition, these algorithms reduce the possible space they are looking in by a percentage, rather than a specific value. For example, a search algorithm that finds an integer between 0 and 100 would not divide and conquer if it simply looped from 0 to 100 and tested each number against what it is searching for. The most straightforward type of divide-and-conquer algorithm is the binary search, which simply cuts the search space in half each time it finds a value in the upper or lower half. You've probably encountered this algorithm before—for example, if you were guessing a number between 0 and 100, you can start with 50, and if the number is higher, you know it is between 50 and 100, so you can then guess 75, and so on. A binary search does just this. Of course, you need a way to test if your value is in the upper of lower half of the sample space, but if you have this information, you can easily find values—for example, you can find a number between 0 and 2048 in 11 or less repetitions of the search. Of course, there are many ways to implement this type of search in code.

How it is implemented in the example root program is pretty simple: the function first calculates the range of values the square root can be in—between 0 and the number (or 1 for values between 0 and 1)—and it tests the middle of the range against the correct value. To do this, it takes the middle (or test) value, and raises it to the power that would negate the root it is finding, and checks to see if it is closer to the perfect value than the maximum precision. Then, if the value is less than it should be, it makes the middle value the lower range, and restarts the process. Hence, it now will test the middle of the new range, and it will redefine the range to the top or bottom of that range, and so on and so on until it finally finds the root of the number.