Binary Search

#### **Learning Outcome Addressed**

 4. Write recursive functions

Now, it’s time to practice another challenge you might encounter in an interview: Binary search. In this activity, you can explore the differences between linear and binary search and then apply recursion to solve the problem.

Click the link below to launch the activity in Next Tech and read the instructions.

### Implementing Binary Search And Comparing Performances

Now that you've seen some examples of recursion, you'll now see how it is used in a binary search implementation.

Hint:

* To have the greatest chance of success you should read the entire explanation and instructions before beginning to code.

#### Linear Search

If you are searching for a book in the Library, you can (1) start at the front of the library and pick up the first book you see, (2) check if that book is the one you are looking for, (3) put that book down and pick up the next, and, (4) repeat this process for every book in the library until you've found your book or run out of books to check. This is a **linear search**.

**Binary search** is a better way to search for the book you want.

#### Binary Search

With binary search, you would be able to narrow down what section the book is in by checking the first letter of the book's name. If the library has a section for each letter of the alphabet, you can split the alphabet in half and end up at the 13th letter, the letter M. In this way, you first check if your book title starts with M. If it does, you have found the section for your book. Otherwise, you can determine whether your letter precedes M in the alphabet or follows it. Based on that answer, you can decide to focus your search on the sections before M or after M.

Binary search does the following:

1. Splits the dataset with each iteration by calculating the mid-point
2. Checks whether the mid-point is what you are searching for
3. Checks if the mid-point is not what you are searching for, binary search checks which half of the split data contains the target
4. Uses that portion of the data and begins again at Step 1

The recursive nature of this function is that after completing the steps above (one iteration), it calls itself (on smaller and smaller data-sets) until either the target is found, or the search is complete without the target being found.

#### Task Instructions

In this activity you'll complete a binary search partially defined in the recursiveBinarySearch function in order to compare the iterations it takes to run "Binary Search vs. Linear Search" on an array containing the letters of the alphabet. Provided in the template is the implementation for linear search as well as some of the framework for binary search. You need to finish the implementation of a binary search.

##### Base Case

The base case in a recursive function occurs when an iteration of the function returns a value rather than an additional call of itself. In this activity, your base cases will return true if the number is found and return false if you complete the search without finding the number.

You will need to add the following code within the recursiveBinarySearch function:

* Establish a base case for the letter not found. If the the startIndex is greater than the endIndex, set binarySearchResults to -1, then return false.
  + Hint: You will need to evaluation the condition, using an *if* statement, of the startIndex being greater than the endIndex.
  + Note: -1 is the standard for an index not found in an array search.
* Calculate the middle index stored in the midIndex variable. It can be calculated by adding the start and end indexes and then dividing them by 2. You may use the Math.floor function to ensure you only work with whole numbers.
* See if the letter is found by checking if the letter stored in your middle index is equal to the letter you are searching for. The function should return true if they are equal.
* Return the results. Provided in the recursiveBinarySearch function are two commented out return statements. Your task here is write an if/else code block that will call the correct return statement based on the contents of your if condition. This is the recursive portion.

You may change the letter being searched for by changing the book title passed into the searchForAlphabeticalIndex() function which is invoked within window.onload near the bottom of the binarySearch.js file.

If you are searching the book titled Reckless and need to find the first letter, R, in the alphabet array, then the linear search iterations in your browser should read: 18 and your binary search iterations should read 4.

Similarly, the following input and results are expected:

searchForAlphabeticalIndex('Way of Kings')

Linear Search Iterations: 23

Binary Search Iterations: 3

searchForAlphabeticalIndex('Lord of the Rings')

Linear Search Iterations: 12

Binary Search Iterations: 5

searchForAlphabeticalIndex('Harry Potter')

Linear Search Iterations: 8

Binary Search Iterations: 5

searchForAlphabeticalIndex('Differential Equations')

Linear Search Iterations: 4

Binary Search Iterations: 4

searchForAlphabeticalIndex('!')

Linear Search Iterations: -1

Binary Search Iterations: -1

Note: The iterations for both linear search and binary search are coded to show -1 iterations if the letter being searched does not exist in the alphabet array you are searching. You can test that by passing special characters into your searchForAlphabeticalIndex function.

Hints:

* Implementing the four bulleted points above should be all you need to make your recursive binary search work in this template.
* Be sure to test your own inputs against the provided inputs and results examples by editing the string passed into the *searchForAlphabeticalIndex()* function inside window.onload.

Task

Complete the recursiveBinarySearch implementation such that you see identical results as shown in the instructions.