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.

/\*

Instructions:

Your task is to complete the binary search implementation below within the searchForAlphabeticalIndex function. You should not need to edit code anywhere else than within

the recursiveBinarySearch code block. You will need to handle cases for:

-loop complete and letter not found - return false

-letter found - return true and set the binarySearchCharacterIndex to the index it was found on

-letter not found and loop not complete - call the function again on either the upper half or lower half of the previous iterations search and check same cases again

\*/

// do not edit this function

/\*\*

 \* Function that generates an array of letters ranging from charA to charZ. The passed in values don't necessarily need to be 'a' and 'z' but for this activity they should always be the default values.

 \* @param charA, beginning of the range to generate the alphabetical array

 \* @param charZ, end of the range for alphabetical array generated

 \*/

const generateAlphabetArray = (charA = 'a', charZ = 'z') => {

    let array = [], j = charZ.charCodeAt(0);

    for (let i = charA.charCodeAt(0); i <= j; ++i) {

        array.push(String.fromCharCode(i));

    }

    return array;

}

const searchForAlphabeticalIndex = ([stringToFindIndexForFirstLetter], alphabetArray = generateAlphabetArray()) => {

    // The binarySearchResults and linearSearchResults variables are incremented every time you iterate the binary and linear searches. As an iteration comes with the cost of computational time,

    // the one with lower iterations and therefore lower number has better performance.

    let binarySearchResults = 0;

    let binarySearchCharacterIndex = -1;

    let linearSearchResults = 0;

    let linearSearchCharacterIndex = -1;

    // first make sure stringToFindIndexForFirstLetter var is lowercase as your array generated above does not contain capital letters

    stringToFindIndexForFirstLetter = stringToFindIndexForFirstLetter.toLowerCase();

    // here is linear search implementation, you don't need to change this

    for (let i = 0; i < alphabetArray.length; i++) {

        // iterates the result to show how many iterations and therefore how efficient this search is

        linearSearchResults++;

        if (alphabetArray[i] === stringToFindIndexForFirstLetter) {

            linearSearchCharacterIndex = i;

            break; // keyword to jump out of a loop

        }

        // if you get through to the last iteration of the loop without breaking, set result to -1 for not found

        if (i === alphabetArray.length - 1) {

            linearSearchResults = -1;

        }

    }

    // \*\*\*\*\* your code goes inside the function below \*\*\*\*\*

    // here is the recursive binary search implementation

    let recursiveBinarySearch = (array, letter, startIndex, endIndex) => {

        binarySearchResults++; // this keeps track of how many iterations of Binary Search you have done, no need to change this code

        console.log(startIndex, endIndex);

        // establish base case for not finding the letter in the given array, if you are on an iteration where the startIndex is greater than the endIndex, first set the binarySearchResults variable to be -1 and then return false

        //find middle index between the given start and end, hint: use Math.floor(your calculation for the middle index between start and end) as that rounds down to the nearest integer, you are not searching decimals

        // for example for array from 0 to 49 (length is 50), you calculate midpoint by writing let midIndex = Math.floor((0+49) / 2)

        /\* CODE GOES HERE \*/

        //check if the value at midIndex is equal to the character being searched for, return true if so

        // before returning true inside the if statement, write the following line of code: binarySearchCharacterIndex = midIndex; // this sets the variable up top to the index the character was found at so you can see the result in browser

        // hint: make sure to use == or === to check if you found the number, single equals sign is only used for assigning values to variables, not checking equality

        /\* CODE GOES HERE \*/

        // determine the if/else condition that should house these two return statements that recursively call the same function after splitting the data in one direction or another

        // note that just like 2 < 4 will evaluate to true, 'a' < 'b' will also evaluate true, javascript can compare letters just like numbers by their position in the alphabet

        // you need to write the if statement that will determine what condition needs to happen for you to search the lower half (start to middle) of the array rather than the upper half (middle to end) next

        // if () {

        //     // this return statement will continue the recursion by narrowing our search to the lower (start to middle) portion of the dataset

        //     return recursiveBinarySearch(array, letter, startIndex, midIndex - 1);

        // } else {

        //     // this return statement will continue the recursion by narrowing our search to the upper (middle to end) portion of the dataset

        //     return recursiveBinarySearch(array, letter, midIndex + 1, endIndex);

        // }

        return false; // remove or comment out this line once you have un-commented the recursive calls above

    }

    // Invokes the recursiveBinarySearch you just defined, do not change the following

    if (recursiveBinarySearch(alphabetArray, stringToFindIndexForFirstLetter, 0, alphabetArray.length - 1)) {

        console.log('Letter: ', stringToFindIndexForFirstLetter, ' found after ', binarySearchResults, ' iterations.')

    } else {

        console.log('Letter: ', stringToFindIndexForFirstLetter, ' not found within array: ', alphabetArray);

    }

    // add the necessary inputs and and results to the DOM

    const numberToSearchForElm = document.getElementById('number-searching-for-id') || {};

    numberToSearchForElm.innerText += stringToFindIndexForFirstLetter;

    const arrayOfNumbersToSearchInsideElm = document.getElementById('searched-array-id') || {};

    arrayOfNumbersToSearchInsideElm.innerText += alphabetArray;

    const linearSearchResultsElm = document.getElementById('linear-search-results') || {};

    linearSearchResultsElm.innerText += linearSearchResults;

    const linearSearchPositionElm = document.getElementById('linear-search-position') || {};

    linearSearchPositionElm.innerText += linearSearchCharacterIndex;

    const binarySearchResultsElm = document.getElementById('binary-search-results') || {};

    binarySearchResultsElm.innerText += binarySearchResults;

    const binarySearchPositionElm = document.getElementById('binary-search-position') || {};

    binarySearchPositionElm.innerText += binarySearchCharacterIndex;

    return binarySearchResults;

}

// Don't change this code aside from the numeric parameter passed in which indicates the number to search for, may see how searching different numbers results in different iterations for linear and binary search once your implementation is complete

// note: searchForAlphabeticalIndex is defaulted to generate an array of length 100 which ranges from 0-99, inputting a number outside that range will not be found and should therefore show -1 in the results for both searches.

window.onload = () => {

    searchForAlphabeticalIndex("Way of Kings");

};

//don't change this line

if (typeof module !== 'undefined') {

    module.exports = { searchForAlphabeticalIndex };

}

<!DOCTYPE html>

<html>

  <head>

    <title>Compare Binary and Linear Search Performance</title>

  </head>

  <body>

    <p>Refresh this page to see how many iterations it takes for both linear and binary search to find the same number in the same set.</p>

    <div id='number-searching-for-id'>

      Letter to search for:

    </div>

    <br>

    <div id='searched-array-id'>

      Alphabet array to search inside:

    </div>

    <br>

    <div id='linear-search-results'>

      Linear Search Iterations:

    </div>

    <div id='linear-search-position'>

      Index of the searched Character found by Linear Search:

    </div>

    <br>

    <div id='binary-search-results'>

      Binary Search Iterations:

    </div>

    <div id='binary-search-position'>

      Index of the searched Character found by Binary Search:

    </div>

    <script src="./binarySearch.js"></script>

  </body>

</html>