Karthik, Charan - USDC 2023 Technical Assessment Documentation
This document outlines my process behind crafting a solution for the given project-based assessment.

Task: Write a JavaScript function that takes in two inputs - a search term and an array of JSON objects containing phrases from a book - and returns one output (a JSON object containing the lines that match the search word).

Notes:
- The search term is a string. *Assumption: The search term will be one word.*
- The input array will contain 0 or more objects. Each object may or may not have scanned text to search through.
- The output will always be an object, even if no matching results were found.
- Search term matches are case-sensitive.
- *Assumption: If there are duplicates, return all instances in the result array (i.e. if the search term appears more than once in a given line, the result array will contain each instance for which the word appears).*

Edge Cases:
- Words that have punctuation attached at the end should still appear in the result.
- Words that span across multiple lines should appear in the result.

Problem-Solving Thought Process:

Since one of the inputs is an array of objects, we must iterate through that array. The time complexity of this process is O(n) where n is the length of the input array.

Similarly, for each book object that we iterate through, we'll have to iterate through its 'Content' (which is also an array of objects). At this point, the time complexity is O(n*c) where c is the length of the 'Content' array.

Next, since each 'Content' object contains a line of text, we'll need to split that line of text by each word and store the words in an array. Now, the time complexity (on average) is O(n*c*t) where t is the length of the string of text. The space complexity has also now changed from O(1) to O(w) where w is the length of the newly created array (this represents the number of words in that line of text).

Afterward, we need to go through each word in the array and check if that matches the search term. If there's a match, an object containing the required information will be added to the results array. The time complexity is now O(n*c*t*w) and the space complexity remains at O(w) - note that the output object is not considered to be extra memory. (Note: there's also the potential to consider the time complexity of comparing strings as it can be O(s) [where s is the length of the string] in some cases, but for this situation, we'll assume it happens in O(1) time).

The above steps outline the general thought process for a rudimentary solution. To fine-tune the function such that it accounts for the edge cases mentioned above, a few modifications need to be made.

To tackle the edge cases, I created a helper function to check if a given character is a letter (‘a-z’ or ‘A-Z’) - using ES6’s ability to interpret Regular Expression - or a hyphen (‘-’) - using string comparison; this function returns a Boolean value (True or False). Then, in the main function, when we are going through each word in the array of words, I check if the last character is not a letter or hyphen. If the last character is not a letter or hyphen, I recreate the word to not include that last character (which is assumed to be punctuation) before I continue checking to see if it matches the search term.

The aforementioned helper function specifically checks to see if the last character is a hyphen as well because it’s assumed that if the last character of a word in the array is a hyphen, it’s a split word that spans across two lines. For this scenario, I keep track of what the previous word was so that we can handle reconstructing that word in the next iteration (as applicable).