Challenge’s Documentation

1- Reading file: “data/cities_canada-usa.tsv” all columns are split by ‘\t’
Used async function call to read the data and insert them into array. This async function is called the second we run the app.js, if a query was requested while data is being read (which would take almost a second to read), the request will be rejected and returned empty array, because there is Boolean Flag “isDataReady=false” which gets toggled to true if read file done successfully, which allows the app to continue functioning properly.

2- Duplicated “name” column’s data into lower cased data to improve search algorithm (Case insensitive search). Used the original normal cased data for the final results only.

3- Instead of using array.filter() to search and loop through all the data, I used and Modified Binary Search algorithm and used it.

   a. Array.filter() or any other function that loops a full loop in the whole array in order to search for an input value “instantly” and is exposed to outside world traffic is a big mistake, because since it loops through all data which is “n” the data size is almost 1125 KB it would cost a lot of RAM and CPU to search instantly. Especially if web app got high traffic.

   b. Binary Search algorithm needs a sorted data, which I did sort it according to “lowered case name column” (Only once for the whole data) and kept the sorted data in the memory (array). From each outside request, Binary Search algorithm would cost log(n) + c1 + c2.

   c. Explaining c1 and c2 by Explaining What Modifications were done to Binary Search algorithm:
      i. Giving example:

0	1	2	3	4	5	6	7
111	22222	AAA	AAAAA	AAAAAA	AAAAAAA	BBBB	BBBBB

As we all know Binary Search Algorithm looks for an Exact value to search, but since we need multiple results and the data is sorted, I made it search for one similar value of the input. Then it stops in some “middle” index which is similar to the input value.

I added two extra for loops: for the above example let’s say we were looking for “AAA” and the algorithm somehow stopped as the middle index number [4] and found similar value of “AAAAAA”

These added two for loops:
First for loop: loops backwards starting from middle index (middle value included) to the left towards 0 index to check if more similar values exists till if it finds one non similar value then it will break, first loop called c1.

Second for loop: loops forward to the right towards [array.length – 1] index to check if more similar values exists till it doesn’t find any, second loop called c2.

Added: special condition also: if exact same value was found? It would be pushed to the head of the result array even before sorting it or evaluating scores.

Notes: in the example above, the data we are looping through to look for similar values to “AAA”, c1 loop will never reach to 0 index, it will stop at index 1, because “2222” is not similar to “AAA” and since data is sorted, no need to proceed.

Tarik Seyceri – tarik@seyceri.info – Istanbul, Turkey.

Same thing goes to c2 loop; it will never reach index 7. It will stop at index 6, and same thing goes for very big array of sorted data. The search method is well utilized and fast.

4- Could have deployed DBMS and inserted all the data in the file to a database which SQL Queries would make the search and indexing much easier, I didn't do it, because maybe the purpose of this challenge is to see how we handle bulky data with no DBMS.

5- Could have used Objects in Array, instead of Arrays in array, but Arrays Are Faster than Objects because of:

   a. By default, data is indexed with numbers in arrays, using number as index/key is way faster than string index/key
   b. Searching in arrays in general is way faster than searching in objects.

6- To prevent high number of Search Results, I limited the 'q' query input length to MUST BE AT LEAST 3 Chars in order to proceed with the search, otherwise it will return empty. Because dumping high number of search results which is not needed for 2 or 1 character(s), and to prevent overuse of resources while high traffic happens.

7- For the search result array, I used array in array in array logic. Because when we search for a data and it got found, we don't need to copy the data from the main array, we just copy the reference, but we need to add additional data to the found data, so we used array in array in array. The following will explain what's meant By the previous statement.

```
var allDataArrays = []; // Main Array of arrays of all data

// for the statements sake let's assume the array is full with data like this
let recordArray = [];

// Frequently used values are in the beginning of the array
recordArray[0] = name;
recordArray[1] = lat;
recordArray[2] = lon;
recordArray[3] = state;
recordArray[4] = country;

allDataArrays.push(recordArray);

///////////////////

var searchResults = [];

//When we search for a value and let's assume we found it, we do this:

searchResults.push([allDataArrays[i], 0, 0]);

    // Where
    //                                    0          1        2

    // Note: Search Results' Structure is: [[RecordArray], Score, Distance]

    // (pass by reference + new data (by value))

    // searchResults[i][0][0] name (tolowered) // by reference from allDataArrays[]
    // searchResults[i][0][1] lat // by reference from allDataArrays[]

    // searchResults[i][0][2] long // by reference from allDataArrays[]

    // searchResults[i][0][3] country // by reference from allDataArrays[]

    // searchResults[i][0][4] state // by reference from allDataArrays[]

    // searchResults[i][1] score // by value // new data with init value 0

    // searchResults[i][2] dist // by value // new data with init value 0
```

So in order to prevent data redundancy in RAM, we used pass by reference method for the search results.

it would help a lot to prevent data swelling while high traffic.

8- A) Scoring: if there were no latitude and longitude given in the query, I gave scores according to number of search results and char length difference.

   a. If only 1 search result has been found and exact same input value, score = 1
   b. If only 1 search result has been found and not exact value, score = 0.9
   c. If only 2 search results have been found?
      i. If first result is the exact same of input value, score = 1, else score = 0.9
      ii. If second result is the exact same of input value, score = 1, else score = 0.7
   d. Else results more than 2
      i. Used for loop to loop all searchResults and evaluating them:
         1. If first value is exact same, score = 1
         2. Else, used char length difference, between input value and each found result's name

```
let lenDif = searchResults[i][0][0].length - searchFor.length;

if(lenDif == 1){

        searchResults[i][1] = 0.9;

}

else if(lenDif == 2){

        searchResults[i][1] = 0.7;

}

else if(lenDif == 3){

        searchResults[i][1] = 0.5;

}

else if(lenDif == 4){

        searchResults[i][1] = 0.4;

}

else if(lenDif > 4 && lenDif <= 7){

        searchResults[i][1] = 0.3;

}

else if(lenDif > 7){
```

```
                        searchResults[i][1] = 0.1;
                }
```

B) if latitude and longitude provided:

e. First we find the distance between the given coordinates and each searchResults' coordinates.
f. Applied similar rules of(a,b,c) when only 1 or 2 searchResults found
g. If more than 2 results then for loop is used, similar to previous for loop but added distance difference between input coordinates and searchResults' coordinates.

```
// Giving score according to char length difference
let lenDif = searchResults[i][0][0].length - searchFor.length;

if(lenDif == 0 || lenDif == 1){
        searchResults[i][1] = 0.3;
}
else if(lenDif == 2){
        searchResults[i][1] = 0.2;

}

else if(lenDif > 2 && lenDif <= 7){

        searchResults[i][1] = 0.1;

}

// Adding score according to distance difference
if(searchResults[i][2] <= 200){
        searchResults[i][1] = searchResults[i][1] + 0.5;
}
else if(searchResults[i][2] <= 350 && searchResults[i][2] > 200){
        searchResults[i][1] = searchResults[i][1] + 0.4;
}
else if(searchResults[i][2] <= 1000 && searchResults[i][2] > 350){
        searchResults[i][1] = searchResults[i][1] + 0.2;
}
else {
        searchResults[i][1] = searchResults[i][1] + 0.1;
}
```

9- Sorting data, used sorting function only on searchResults once if latitude and longitude not provided (sorting according to DESC Score), and two times if coordinates was given one for (sorting according to distance) and one for (sorting according to score).

10- Pushed the final searchResults' data output one by one using for loop into Array of objects in order to output the way it's required.