**GENERAL INSTRUCTIONS**:

Please complete the attached coding assignment. This is a two-part assignment, which should take roughly one hour to complete. The goal of the first part is to assess your ability to implement a working solution for a typical computer science programming exercise. The goal of the second part is to assess your ability to test your solution.

**WHAT TO SUBMIT:**

You should submit one attachment for Part One and a second attachment for Part Two (if you had time to complete it).

* The attachment for Part One should include the source code of your solution. You may use any programming language that you choose. But the code must be syntactically valid, not pseudo-code.
* The attachment for Part Two can be a simple text document that describes your test cases.

Along with these attachments, please provide a note on how to run or load your code. Also include a description of any assumptions you have made.

We understand that an hour may not be enough time for all candidates to complete both parts of this assignment. But keep in mind that the first part is the most important.

**HOW YOU ARE TIMED**:

You have one hour to complete this assignment. If there are circumstances, which prevent you from completing within an hour, please include those thoughts in your submission.

H**OW YOU WILL BE MEASURED:**

Your solution will be assessed with following criteria:

* Does the solution solve the given problem?
* How efficient is the design?
* Are proper coding practices followed?
* How readable or maintainable is the code?
* The quality of the test cases, if any.

**ASSIGNMENT PART ONE**:

Submit a program that outputs 26 coordinates listed in order, closest-to-farthest (physically on the Cartesian Plane) from a single input origin.

Please use this list of labeled coordinates in a file called “coordinates.json”:

[

{"id":"a","value":"31,49"},

{"id":"b","value":"44,67"},

{"id":"c","value":"93,6"},

{"id":"d","value":"20,16"},

{"id":"e","value":"68,53"},

{"id":"f","value":"71,8"},

{"id":"g","value":"61,90"},

{"id":"h","value":"34,97"},

{"id":"i","value":"21,63"},

{"id":"j","value":"19,84"},

{"id":"k","value":"0,81"},

{"id":"l","value":"6,76"},

{"id":"m","value":"43,64"},

{"id":"n","value":"18,64"},

{"id":"o","value":"10,61"},

{"id":"p","value":"37,27"},

{"id":"q","value":"44,88"},

{"id":"r","value":"75,63"},

{"id":"s","value":"99,46"},

{"id":"t","value":"28,51"},

{"id":"u","value":"88,79"},

{"id":"v","value":"47,21"},

{"id":"w","value":"18,66"},

{"id":"x","value":"84,100"},

{"id":"y","value":"75,92"},

{"id":"z","value":"32,33"}

]

Create a program that will take in a single x and y coordinate as the origin. After the input is provided, the output should be all of the coordinates (all 26 coordinates read from the “coordinates.json” file), in order of closest-to-farthest from the origin.

So, if the origin is (6,33), then the first item in the output would be the coordinate closest to (6,33), then the second closest, then third closest, and so on for all 26 input coordinates.

The origin should vary; do not hardcode a static origin. Also, the coordinates must be sourced from “coordinates.json”.

Codes: Using Javascript

'use strict'

var fs = require('fs');

if (!isFinite(process.argv[2]) || !isFinite(process.argv[3] || process.argv.length != 4)) {

console.log("Invalid input!");

return;

}

var x=process.argv[2], y=process.argv[3];

var bufferString;

var myMap = {};

var outputResult = function(map) {

let result = [];

console.log(map);

let keys = Object.keys(map);

keys.sort(function(a, b) {

return a - b;

})

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

result.push(map[keys[i]]);

}

console.log(result);

}

var calculateDistance = function(item) {

if (item.value!=="undefined") {

let point = item.value.split(',');

let distance = Math.pow(point[0]-x,2) + Math.pow(point[1]-y,2);

typeof myMap[distance];

if (typeof myMap[distance] === "undefined") {

myMap[distance] = [];

}

myMap[distance].push(item.id);

}

}

var calculateEachPoint = function(x, y) {

var jsonObj = JSON.parse(bufferString);

jsonObj.forEach(calculateDistance);

outputResult(myMap);

}

function readFile (jsonfile, callback) {

fs.readFile(jsonfile, 'utf8', function (err, data) {

if (err) {

console.log(err);

return;

}

bufferString = data.toString();

callback.apply(this, [x,y]);

});

}

readFile("coordinates.json", calculateEachPoint);

///////////////////////////////////////////////////////\* usage and output:

$ node computeDistances.js 3.5 81.7

{ '1825.5400000000002': [ 'a' ],

'1856.3400000000001': [ 'b' ],

'13740.740000000002': [ 'c' ],

'4588.740000000001': [ 'd' ],

'4983.9400000000005': [ 'e' ],

'9987.94': [ 'f' ],

'3375.14': [ 'g' ],

'1164.34': [ 'h' ],

'655.94': [ 'i' ],

'245.54': [ 'j' ],

'12.740000000000004': [ 'k' ],

'38.74000000000003': [ 'l' ],

'1873.54': [ 'm' ],

'523.5400000000001': [ 'n' ],

'470.7400000000001': [ 'o' ],

'4114.34': [ 'p' ],

'1679.94': [ 'q' ],

'5461.9400000000005': [ 'r' ],

'10394.74': [ 's' ],

'1542.7400000000002': [ 't' ],

'7147.54': [ 'u' ],

'5576.74': [ 'v' ],

'456.7400000000001': [ 'w' ],

'6815.139999999999': [ 'x' ],

'5218.34': [ 'y' ],

'3183.94': [ 'z' ] }

[ [ 'k' ],

[ 'l' ],

[ 'j' ],

[ 'w' ],

[ 'o' ],

[ 'n' ],

[ 'i' ],

[ 'h' ],

[ 't' ],

[ 'q' ],

[ 'a' ],

[ 'b' ],

[ 'm' ],

[ 'z' ],

[ 'g' ],

[ 'p' ],

[ 'd' ],

[ 'e' ],

[ 'y' ],

[ 'r' ],

[ 'v' ],

[ 'x' ],

[ 'u' ],

[ 'f' ],

[ 's' ],

[ 'c' ] ]

////////////////////////////////////////////////////////////////////////\*/

**ASSIGNMENTPART TWO**:

If time permits, write a suite of tests for your code in Part One. The test cases can be written descriptions, and not necessarily in code.

1. I assume the json input file is always valid so I don’t validate the json again. Can be done if that could happen.
2. I will test the following scenarios:
   1. Invalid input for x and y (node computeDistances.js NaN 3)
   2. Float point for x or y or both
   3. Not enough input (node computeDistances.js 3)
   4. Correct input and output is as expected