Module 7 Project

Introduction

It’s time to practice and solidify the skills you learned in this module.

Warning: Independent work

This is an individual assignment. Please do not collaborate with your peers or share your work until the project is reviewed as a class.

Setup

1. Click here to download the code files you need to for this project.
2. Navigate to the files on the command line.
3. Setup the your project by running `npm install`.

Step 1: Runtime Analysis

In the materials you downloaded at the beginning, there is a file called ***runtime.js***. Read the code in this file carefully. After you have read over the file yourself and made some guesses on what's happening, hover below to read a summary.

Hint: *runtime.js* Summary

▼ *Click to hide*

At the top of the file, there are two functions, ***doublerAppend*** and ***doublerInsert*** that seem to do a similar thing– take in an array of numbers, multiple each number by two, and then add it to a new array.

However, the functions are in fact vastly different. One uses ***.push()*** to add the numbers to a new array, and the other uses ***.unshift()*** to add the numbers to a new array. If you haven't already, stop and do a Google search for ***.unshift()*** to learn about what it does in Javascript.

After the functions, you'll see several arrays being created using another function in the file, ***getSizedArray***. This is a "helper function" for this assignment.

Lastly, you'll see code that is starting and stopping a timer near the end of the file. This happens twice– once to call the first function, and another time to call the second function. The results of those timers are printed to the console.

1. It's time to run the file! Run the command `node runtime.js` in the same directory as where the file lives.
2. In your notes document, take note of the timing result for the ***extraLargeArray*** results– comparing when the ***extraLargeArray*** is passed to ***doublerAppend*** and ***doublerInsert***.
3. Next, edit the code in ***runtime.js*** to obtain timing results for calling the two functions with all of the differently sized arrays– ***tinyArray***, ***smallArray***, ***mediumArray***, ***largeArray***, and ***extraLargeArray***. Notate these in your document in some kind table table so that you can easily compare the different values for the timers in relation to the size of the array that was passed into each function.
4. Read over the results, and write a paragraph that explains the pattern you see. How does each function "scale"? Which of the two functions scales better? How can you tell?
5. For extra credit, do some review / research on why the slower function is so slow, and summarize the reasoning for this.

Be sure to include this report in the Github repo for this project.

Step 2: Write Code

Work through the following problems in Javascript (you have seen these problems before). Create a new file for your code. When you have finished with each function, leave a code comment with what you believe the runtime of your code to be.

1) Sum Zero

Write a function that takes in an array of numbers. The function should return True if any two numberss in list sum to 0, and false otherwise.

For example:

```
addToZero([]);
// -> False

addToZero([1]);
// -> False

addToZero([1, 2, 3]);
// -> False

addToZero([1, 2, 3, -2]);
// -> True
```

2) Unique Characters

Write a function that takes in a single word, as a string. It should return True if that word contains only unique characters. Return False otherwise.

For example:

```
hasUniqueChars("Monday");
// -> True

hasUniqueChars("Moonday");
// -> False
```

3) Pangram Sentence

A pangram is a sentence that contains all the letters of the English alphabet at least once, like “The quick brown fox jumps over the lazy dog.”

Write a function to check a sentence to see if it is a pangram or not.

For example:

```
isPangram("The quick brown fox jumps over the lazy dog!");
// -> True

isPangram("I like cats, but not mice");
// -> False
```

4) Longest Word

Write a function, find_longest_word, that takes a list of words and returns the length of the longest one.

For example:

```
findLongestWord(["hi", "hello"]);
// -> 5
```

Be sure to add this file to your Github repo for this project.

Extra Credit

List out the space complexity of each solution in Step 2.

Be sure to push your code to Github for this assignment!

To pass this assessment you must score at least 13/18.

OUTCOME	DETAIL	0 pts	1 pt	2 pts	
Runtime Analysis	Assessment includes timing results for calling the provided functions with the provided arrays (10 total, 2 functions, 5 arrays)	Did not attempt.	Assessment contains 5 or fewer timing results.	Assessment contains 6-9 timing results.	
	Assessment includes a paragraph explaining 1) how each function scales, 2) which function scaled better, and 3) reasoning	Did not attempt.	Attempted but only 1 part of the prompt is fully answered.	Attempted but only 2 parts of the prompt are fully answered.	
Write Code	Assessment includes 1) an addToZero function that 2) does what it is supposed to do and 3) has the correct runtime noted. (Function name does not have to match exactly.)	Did not attempt.	Attempted but only 1 of the requirements is met.	Attempted but only 2 of the requirements are met.	
	Assessment includes 1) a hasUniqueChars function that 2) does what it is supposed to do and 3) has the correct runtime noted. (Function name does not have to match exactly.)	Did not attempt.	Attempted but only 1 of the requirements is met.	Attempted but only 2 of the requirements are met.	
	Assessment includes 1) an isPangram function that 2) does what it is supposed to do and 3) has the correct runtime noted. (Function name does not have to match exactly.)	Did not attempt.	Attempted but only 1 of the requirements is met.	Attempted but only 2 of the requirements are met.	
	Assessment includes 1) a findLongestWord function that 2) does what it is supposed to do and 3) has the correct runtime noted. (Function name does not have to match exactly.)	Did not attempt.	Attempted but only 1 of the requirements is met.	Attempted but only 2 of the requirements are met.	
Extra Credit	Assessment includes an explanation of why the slower function from 'runtime.js' is slower.	Did not attempt.	Attempted but inaccurate **and** incomplete.	Attempted but inaccurate **or** incomplete.	
	Assessment includes the space complexity for all 4 functions from Step 2.	Did not attempt.	Attempted but only 2 or fewer correct answers given.	Attempted but only 3 correct answers given.	
Total		**x/18**			

© 2021 Devmountain