335-04 — Algorithm Engineering — 3-Sort Race

Project #3 – 3-Sort Race
Introduction

This project is to write a Javascript program for a racing competition between three different sorting algorithms. You program will run several algorithms in an interleaved fashion, where each algorithm will get a turn to perform a single Step (one of its complexity operations) in its run, and thereby it may rearrange its current pass Sort Array.

Any Sort Array changes for each algorithm's Step will be displayed on the program's HTML web page in a browser.

Algorithms

For this project, you will run three O(N*logN) sorting algorithms: Mergesort, Quicksort, and Bubblesort.

Input

The input for a Sort Race will be a 12 character hexadecimal string (e.g., "8A1593479085"). You will be provided with a set of them for testing. Note, there may be duplicate characters in the input.

Output

To show the race, you should display in an HTML page each of the three racing algorithms as a 12-item wide by 20+ rows high array in its own grid starting at the top row with the raw input string. The row-by-row display is similar to the cellular automata display of Project #1, but in 3 columns, one for each algorithm.

Below are sample inputs that your program should be able to race. For a race, each sorting algorithm will start with the same input. You should be able to run any of the sample inputs.

A new row for an algorithm should be provided as each “pass” started. While the steps/comparisons of a pass of an algorithm are being done, changes in item positions (e.g., 2-item swaps) should be displayed. The prior row for the prior “pass” result should be left as is, so that the user can see the various “pass” differences. The 12-item "string" should be large enough for it to be easily seen. As items are repositioned due to a Step, they should be highlighted in some way. Also include a column header string indicating the algorithm.

Setup

Your program should select one of the sample inputs provided below at random. Each is a list of 12 hex digits. You will create an array of these integers.

Race Manager

To make this single Stepping work, this project will require a Race Manager (Mgr) (function or object method) that will loop. At each iteration, the Race Mgr will call each algorithm's Step function. The Race Mgr will notice that an algorithm finishes when its Step function returns a zero value. The Race Mgr will continue calling each algorithm's Step function while it returns a non-zero value.

The Race Mgr will pause for at least 1/2 a second after each Step call, so that the audience can easily see what changes are being made.

Step Functions

Each algorithm Step function will have access to its current state, so that it can take a next Step. The state for each algorithm is specific to that particular algorithm. It comes in two parts, (semi-)public and private. Changes to the algorithm's "public" state will be transferred to the GUI Mgr as described below. The algorithm's private state will hold any other state that is useful to the algorithm's Step.

The Step will do one Big-O operation and related support processing. For example, if the operation is comparing a pair of elements, then the Step will do the next compare and update its algorithm's state.

When the Step function has completed the final sorting operation, the Step function will return a zero value.

Until then, it returns a non-zero value after completing its one Step operation.

Roughly speaking, the Step function would be the body of a loop inside a sort algorithm.

MVC & GUI Manager

The Step will also notify the Race Mgr (for example, by calling a function) which will in turn notify the GUI Mgr of what changes were made to the Step Algorithm's public state, passing the data for those changes. We communicate this way to the GUI because we use an MVC architecture pattern. A Step function will not communicate with the GUI directly.

Managers

Each manager can be either a Javascript class object or merely a set of related functions and data variables with a common 3- or 4-letter prefix (e.g., "rac" for Race Mgr, or "mrg" for Mergesort algorithm). Note that Javascript classes and methods are only "mostly" like C++.

Running Time

You should prepare a 1-page (at most) paper describing your analysis of the running time (not Big-O) of each algorithm as you have implemented it. Your basic operation for a Sort is the 2-item comparison operation. If you feel that other operations should also be included -- perhaps because they end up taking a significant (abouf 5% of the total) time -- then they should be included as well. Once you have an expression for running time in terms of the number of operations used (including the algorithm's setup), then show (briefly) how this running time is convered to a Big-O running time.

Sample Inputs

```
   (0, B, A, 3, 2, 8, 4, 7, 6, 5, 1, 9)      (6, 9, 8, 7, 2, B, 3, A, 5, 4, 1, 0)
   (0, A, 9, 8, 1, A, 3, 9, 2, 0, 1, 1)      (6, A, 2, 3, 0, 5, 3, 0, 4, 7, 8, 1)
   (1, 6, 3, 8, 9, 4, 0, A, 5, 2, B, 7)      (7, 0, 1, A, 6, 9, 3, 5, 4, 2, B, 8)
   (1, 9, 8, 4, 1, B, 3, 8, 2, 6, 2, 5)      (7, 8, 5, 2, 8, 6, 1, 0, 3, 4, 2, 9)
   (2, 9, 7, B, 4, 0, 1, 6, 3, 8, A, 5)      (8, 7, 3, A, 9, 4, 2, 5, B, 1, 6, 0)
   (2, 6, 1, 0, 9, 4, 8, 7, 8, 6, 2, 6)      (8, A, 1, 5, 9, 3, 4, 7, 9, 0, 8, 5)
   (3, 4, 5, 7, 1, 9, 2, 0, 6, 8, B, A)      (9, 0, B, 3, 4, 2, 7, 5, 6, 1, 8, A)
   (3, 5, 6, A, A, 0, 2, 3, B, 7, 2, 4)      (9, 9, B, 5, 3, 5, 1, A, 3, 3, A, B)
   (4, B, 6, 0, 7, 9, A, 2, 1, 8, 3, 5)      (A, 4, 0, B, 5, 8, 6, 1, 7, 9, 2, 3)
   (4, 0, B, 0, 6, 5, 6, 6, 7, 1, 0, A)      (A, 3, 9, 5, 9, A, 2, 2, A, 4, 4, 4)
   (5, 2, 8, 1, A, B, 3, 4, 7, 9, 0, 6)      (B, 8, A, 4, 6, 3, 7, 9, 0, 1, 5, 2)
   (5, 7, 5, 0, 6, 8, 4, B, 8, 9, 3, 4)      (B, 6, 0, 0, 5, A, 6, 2, 7, B, 2, 3)
```

Team

The team size is the same as before, but you can change team members from the previous project if you wish.

Project Reporting Data

Same as for Project #1.

Readme File

As before.

Academic Rules

Correctly and properly attribute all third party material and references, if any, lest points be taken off.

Submission & Readme File

As before.

Grading

As before.