335 — Algorithms — Sort Race

Project #3 – Sort Race

Introduction

This project is to write a JS+P5 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 the state of its current pass.

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

Algorithms

For this project, you will run three sorting algorithms: Gold's Poresort, Mergesort, and Quicksort. You should assume that the setup for Mergesort includes the "tree partitioning" into 1-element lists, as discussed in lecture.

Input

The input for a Sort Race will be one of several 12-character hexadecimal strings (e.g., "FD8A15934785"). 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 competition, you should display in an HTML page graphics canvas each of the three racing algorithms as a grid column 12-cells wide by 40+ rows high. All three grid columns can share the canvas if you put 1 or 2 cells between each column (e.g., a 40-cell wide canvas). Each grid cell should be at least 20x20 pixels wide, and as you will be putting a hex digit in a cell, you will have to ensure that the text digit character fits.

On the top row, the input hex string will appear in each sort algorithm's column. Thereafter, for each step of the algorithm, the next step's behavior should be indicated. For example, by highlighting the two cells that the algorithm compares, as well as the visible array text results the algorithm takes based on that comparison operation's outcome. This row-by-row, cell-by-cell display updating is intended to be 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/highlighted in that pass's row. The prior row for the prior "pass" result should be left as is, so that the "audience" can see the various "pass" differences. 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. Your program can include them as dedicated input data.

Race Manager

To make this single Stepping work (one Step per comparison operation), 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 1/5th of 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 need access to its current state, so that it can take a next Step. The state for each algorithm is specific to that particular algorithm. We recommend that a State object be created for each type of algorithm, and be initialized during setup. The Race Mgr will call the algorithm's Step function with the algorithm's State object as an argument. The Step function will perform its processing and return an updated State object to the Race Mgr. During Step processing, the Step function can call a GUI update function with the important changes needed to update that algorithm's column state. On the Race Mgr's next loop, it will pass that returned State object back to the algorithm for the next step's processing.

The Step will do one Big-O operation (one comparison) and related support processing.

When the Step function has completed the final sorting operation, the Step function will return a suitable null value to the Race Mgr.

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

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, not counting any GUI operations. 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 (above 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 converted to a Big-O running time.

Sample Inputs

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