CPSC-335 — Algorithms — Sort Race

Project #2 – Sort Race
Introduction

This project is to write (in HTML+JS+P5) program display the 'generational' (i.e., **one row per 'pass'**) progress of several **different sorting algorithms (each in their own vertical column) side-by-side**. Your program will run several algorithms in an interleaved fashion, where **each algorithm will get a turn** to perform a single Pass (approximately **'touching' all** the elements of its array to be sorted) in its Pass, and then show the newly updated array (the **Pass results**) in the next row of that algorithm's column.

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

Algorithms

For this project, you will **run five** sorting algorithms: Insertion Sort, Selection Sort, Gold's Poresort, Mergesort, and Quicksort.

Input

The input for a Sort Race will be a 16-character hexadecimal string (for example "3**B**5F5947**B**807D8A1"). Note, **duplicate** characters are allowed in the input (eg, “B” and “8” in the example). To get the input, you can follow the (new) example provided which shows how to easily create an input text-box and a “submit” button on the webpage, and how to retrieve the input context (as a string) for use in starting the Sort Race. If the input is less than or greather than 16 chars, you may truncate the end, or pad with '0' characters.

Output

To show the race competition, you should display in an HTML page graphics canvas each of the racing algorithms, side-by-side, with each as a grid column with one cell wide by **50 or more rows high**. All four grid columns (one for each algo) can share the canvas if you put 1 or 2 cells between each algorithm column. Each grid cell should be **at least 10x10 pixels** wide, and as you will be putting **one hex digit in a cell**, you will have to ensure that the text digit character fits within the cell.

On the top (row 0, or “row -1”) row, the **algorithm's name** will appear (or enough of it to fit within your 16-cell width); and please do not overwrite this row when you wrap around from the bottom back to the top.

On the next (row 1, or row 0) the **input hex string will appear** in each sort algorithm's column. Thereafter, **for each pass of all** the algorithms, the updated order of the hexadecimal string should **be shown** in each column on the next row.

Note that you run all the algorithms, each for **one pass, and then display** all these single pass results in the next row (for each of the columns) **BEFORE** you have the algorithm continue to run for the next pass. (That is, you do not run and save all the individual pass strings and then display these strings after the algorithms have fully run.)

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.

At the end are sample outputs in various styles.

Output Delay

After each pass is displayed (for all the algorithms) you should **delay** the start of the **next pass** computations for **between one half and one second**, in order for the audience to see and understand the results.

Setup

Your program should select one of the sample inputs provided below at random (or you can ask the user for a hex “index” digit to pick one of them). Each is a list of 16 hex digits. Your program can include them as dedicated input data (ie, hardcoded).

Architecture

To simplify your code, and because **you cannot run** an algorithm **through the entire sorting** process **at one time/one go**, we recommend that you create a **Pass/Algo function** for each algorithm, which keeps track of any between-Pass details (e.g., the array being sorted, or Mergesort might track a sublist size value) and is capable of **running one pass** of its algorithm if given the string, probably as an array, to work on.

(EG, you could create a small family of global vars for each sort routine, and use-modify these during the sort pass.)

It then becomes a bit simpler to run a single pass for each algorithm. You can do this for each algorithm by creating **Row Mgr** (manager) with a row global, which keep track of what row needs to be updated, and can call each of the algorithms to get access to the pass-updated string for that algorithm. The Row Mgr could then get the results from a Pass object and display the updates to the algorithm's row.

You might also want a **Race Mgr** to setup the initial strings, and display, for each algorithm, and then call the Row Mgr enough times to get all the algorithms to finish sorting the string.

You **do not have to use** this architecture, but note that you cannot run an algorithm to completion (without displaying its progress) and then and display each of its passes, before starting the next algorithm. The algorithms must each be "paused" in some fashion after each pass so that those pass results can be displayed. This will be discussed in Lab.

Running Time

You should prepare a 1-page (at most) paper describing your analysis of the **rough running time** (not Big-O) of each algorithm as you have implemented it, not counting any GUI operations. Your basic operation for a Sort algorithm is the 2-item comparison operation, as usual. 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 to Try

`"A62A7BC2B6F0305C"`	`"E1AD0342D7859286"`
`"DE66B71F040BA065B"`	`"530BC4786AF2130E"`
`"4FB89C3D5754E0"`	`"DE47C65C10BA9328"`
`"9A2D18D3E4B6507C"`	`"2756FD18E90BA34F"`
`"48E7862ED261609F"`	`"A34F07E56F18090B"`
`"B3D47905FC2861FA"`	`"59286E2FD0342D78"`

Team

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

Academic Rules

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

Project Reporting Data, Readme File, Submission & Readme File, Grading

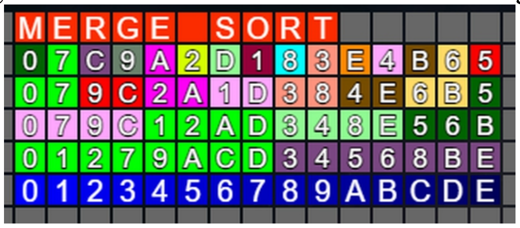
Same as for Project #1.

CPSC-335 — Algorithms — Sort Race

Here are some examples of output

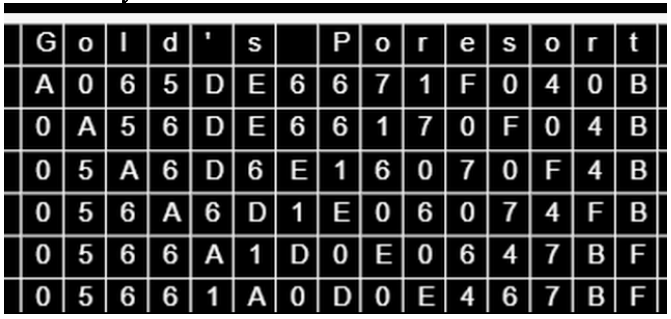
One letter in a grid cell, sort algo title on top, blank line after fully sorted, first line is the input string, and a 1-cell blank margin/gutter on either side.

Here, grid cell color is used to indicate sub-list grouping by size for Mergesort.



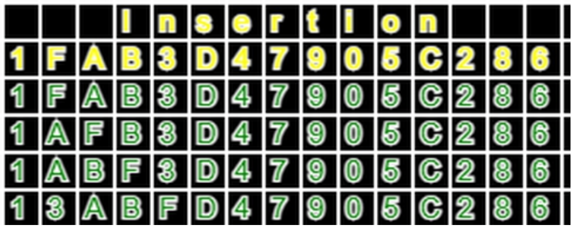
```
 M E R G E   S O R T
 0 7 C 9 A 2 D 1 8 3 E 4 B 6 5
 0 7 9 C 2 A 1 D 3 8 4 E 6 B 5
 0 7 9 C 1 2 A D 3 4 8 E 5 6 B
 0 1 2 7 9 A C D 3 4 5 6 8 B E
 0 1 2 3 4 5 6 7 8 9 A B C D E
```

Here, each half-pass of Gold's Poresort is shown (first the odd+next, then the even+next).
No color but easy to read.



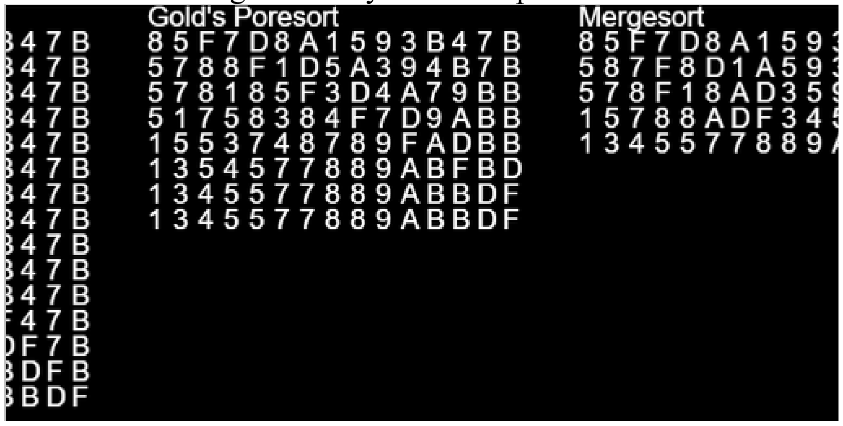
```
 G o l d ' s   P o r e s o r t
 A 0 6 5 D E 6 6 7 1 F 0 4 0 B
 0 A 5 6 D E 6 6 1 7 0 F 0 4 B
 0 5 A 6 D 6 E 1 6 0 7 0 F 4 B
 0 5 6 A 6 D 1 E 0 6 0 7 4 F B
 0 5 6 6 A 1 D 0 E 0 6 4 7 B F
 0 5 6 6 1 A 0 D 0 E 4 6 7 B F
```

Here, the characters are a bit harder to read, but still “fun”. For Insertion Sort, it would be nice to highlight the Tail[0] cell so as to easily see what the next pass is: 1 then F then A then B then 3, etc.



```
       I n s e r t i o n
 1 F A B 3 D 4 7 9 0 5 C 2 8 6
 1 F A B 3 D 4 7 9 0 5 C 2 8 6
 1 A F B 3 D 4 7 9 0 5 C 2 8 6
 1 A B F 3 D 4 7 9 0 5 C 2 8 6
 1 3 A B F D 4 7 9 0 5 C 2 8 6
```

Here is an example of the side-by-side nature of the algorithm race, with a separate column for each algorithm, where the already finished sort algos just stopped producing any more output. Missing grid lines, though. And here, Poresort doesn't show the half-passes (instead it shows the results of both half-passes – which is fair but it is also harder to see what is happening) – that is, notice that in the first and second rows the LHS '8' moved right two positions due to being moved by each half-pass.



```
             Gold's Poresort                     Mergesort
3 4 7 B      8 5 F 7 D 8 A 1 5 9 3 B 4 7 B       8 5 F 7 D 8 A 1 5 9
3 4 7 B      5 7 8 8 F 1 D 5 A 3 9 4 B 7 B       5 8 7 F 8 D 1 A 5 9
3 4 7 B      5 7 8 1 8 5 F 3 D 4 A 7 9 B B       5 7 8 F 1 8 A D 3 5
3 4 7 B      5 1 7 5 8 3 8 4 F 7 D 9 A B B       1 5 7 8 8 A D F 3 4
3 4 7 B      1 5 5 3 7 4 8 7 8 9 F A D B B       1 3 4 5 5 7 7 8 8 9
3 4 7 B      1 3 5 4 5 7 7 8 8 9 A B F B D
3 4 7 B      1 3 4 5 5 7 7 8 8 9 A B B D F
3 4 7 B      1 3 4 5 5 7 7 8 8 9 A B B D F
3 4 7 B
3 4 7 B
3 4 7 B
F 4 7 B
D F 7 B
3 D F B
3 B D F
```