**Algorithm Simulation**

1. **Abstract**

Algorithm is the core part of every software development process. An algorithm is a structured way of solving a problem in a step by step manner. Having a standard algorithm for a problem helps programmers in many ways. It provides the core logic to solve all related problems so that we can use it across different projects/features/problems, it provides a cleaner solution, it gives a measure to compare performances of different approaches, it can be used as a baseline for the future work etc.

Although having an algorithm makes the problem-solving process easier, the algorithm itself can become complex and difficult to understand based on the difficulty of the problem statement or the approach we are taking to solve it. For example, a simple sorting problem, when solving using bubble sort algorithm is easy but when solving using merge sort algorithm can be difficult to understand as it involves recursion. One of the reasons why we think some algorithms are difficult is because it’s hard to visualize how different parts of those algorithms actually works. It will be helpful if we get some visual representation of the working the algorithm, and that’s the main focus of this project. To provide a platform where a user can see how different algorithms works, to play around with the algorithm with different inputs, to keep track of how the input changes at each step.

1. **Introduction**

The solution provided is a web app built using react.js library. The user interface is basically divided into three parts – menu bar, input/output area and the notifications area. A user can select any algorithm from the menu bar which has a general category of different problems as main menu and different algorithms to solve that problem as the sub-menus. The input/output area is where user provides input data and see the animation happening. Appearance of both input and output section changes depends on the category of the algorithm. The output section has controls to go to next step, previous step, pause animation etc. The notifications’ bar is to provide all the necessary information specific to algorithm. There are three types of notifications – NOTIFY, DESCRIBE and ERROR. The first type is a general notification which disappears after certain timeout which is customizable. The second type is the most important which displays snapshots of input data and main variables/objects used at each step. These notifications will be cleared when different algorithm is selected. The third type is to display error messages which will appear in red and will disappear after some timeout which is again customizable.

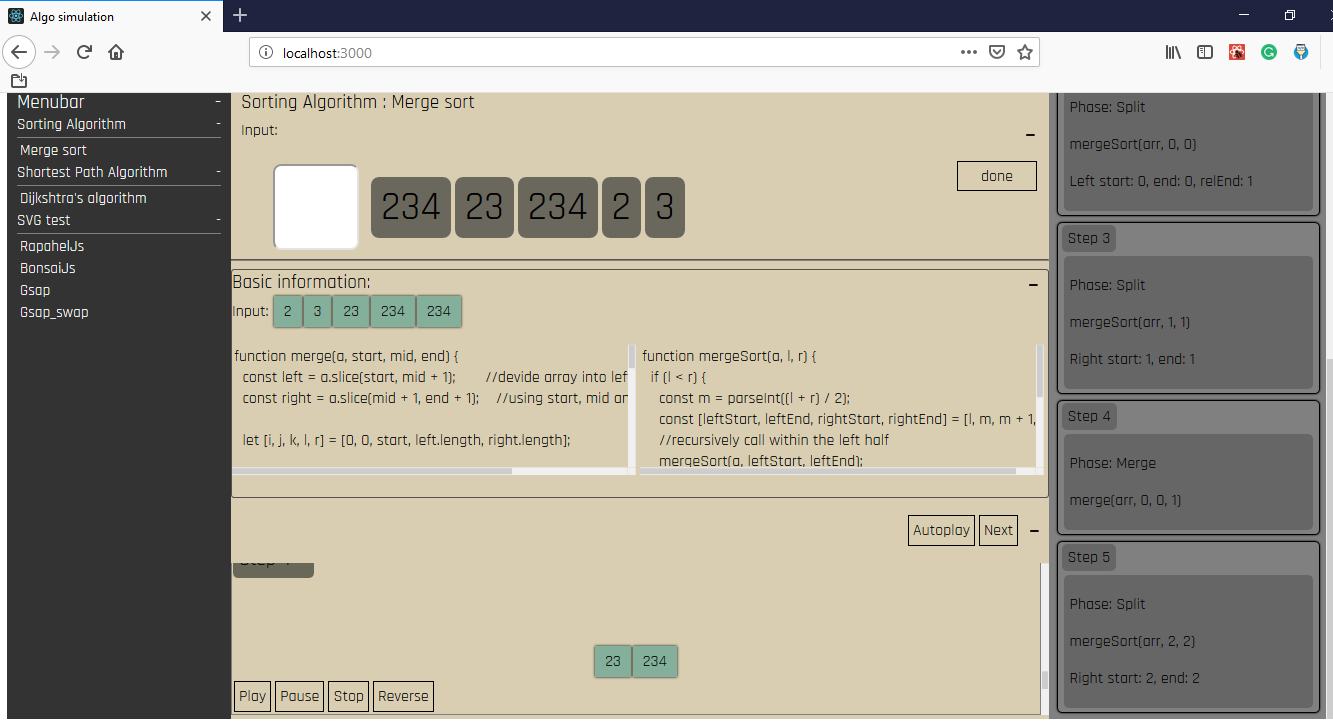


Figure a: Merge sort

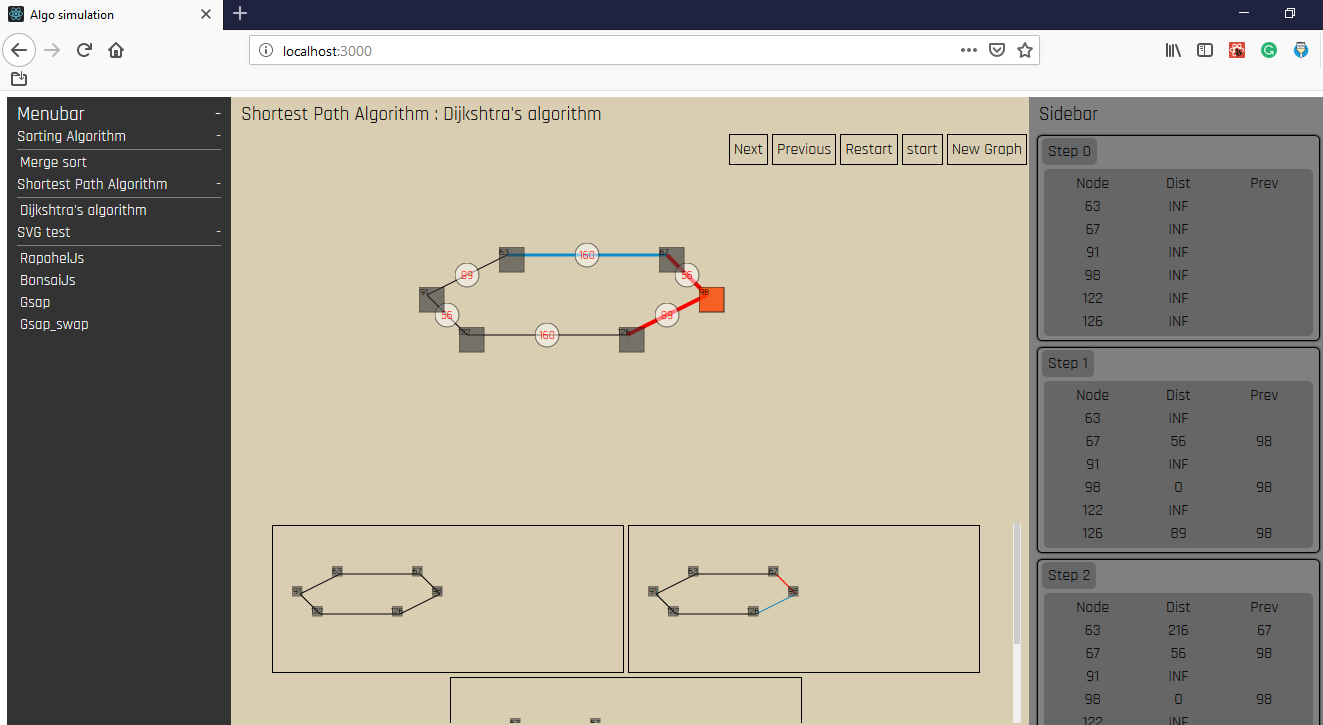


Figure b: Shortest path

**Why react.js?**

Speed

The entire project involves lot of changes to the DOM. React intelligently handles changes to the DOM with the help state and minimizes updates. React’s virtual DOM helps improve performance and make the animation smoother. For a small project with very limited scope a plain vanilla javascript based solution is sufficient but if we are building a complete having some front-end development framework, helps us focus on the core problem while the framework takes care of the rest of things.

Reusability

Even though there are many moving parts in the project, there are few parts which are reused extensively across different algorithms. With react.js or similar libraries/frameworks (like vue or angular), we can convert the reusable functionality into a component and make the code DRY.

Learning curve:

React is not a complete framework and hence does not have so steep learning curve as compared to Angular. Basic understanding of react’s core features is sufficient for implementing this project.

1. **Requirement gathering**

Following are the major areas that are needed to be addressed in order to build the tool.

**3.1 Core features**

* **Animation controls like play-pause-reverse options**

The whole purpose of this project is to help students or people who are learning the algorithm for the first time or the people who are having trouble understanding the working of the algorithm. Hence giving full control of the flow with the help of play, pause and reverse buttons is the core feature. With this, the user can spend as much time as he/she wants on each step and if needed go back to previous step as well.

* **Logging information of every step**

Logging information along with animation has following advantages

It’s difficult to animate every change happened during the execution. So, showing the main flow of the algorithm in the animation and logging the rest of the information is a good solution.

It helps quickly checking what part of input changed after last step.

This feature can be used as a debugging tool. If the user wants to implement his on his own, he has the entire log ready him. He can just cross check the output produced by his algorithm to check if his algorithm is working properly or not.

* **Ease of giving input data**

Generating input for the algorithm can be tedious for some algorithms. This issue is handled with the help of following steps

* A ready-made sample input:

With each problem, a sample input is provided by default which covers important cases and corner cases. This saves some of user’s time.

* The entire process of generating input is tied to simple button press and mouse click events. For example, in merge sort, when the user is on its homepage, the cursor is already set to input box where user can input number and press enter to add it to the list.

Similarly, for shortest path algorithm, user just need to click on the node to select it and an edge will be drawn automatically in between the last two selected nodes. The edge weight is again provided automatically based on their absolute distance. The edge can be removed by double clicking on the weight of edge. At any point, user can clear the entire graph and start over. Once the graph is ready, it can be locked with the help of “lock” button to avoid further modification in between the animation. User can again start redrawing at that point as well by clicking on “New graph” (This will remove the lock). A graph in a locked state, can be tested against any node as a starting node just by clicking on it. This helps in understanding how shortest path from one node differs from other node as a starting point for the same graph. These features are added considering the common issues we face while learning any graph related problems.

Some of features related to this point are added as part of “future work” in the latter section of the report.

**3.2 Approach for animation**

* **Execute first animate later**

In this approach entire algorithm is executed first and a log is generated which has all the information need to perform the animation.

Advantages:

* Decoupling - The code for animation and the actual algorithm is totally separated.
* Reusability - With this approach we just need to focus on building a single platform for all algorithms that falls in same category and then generating the log of all those algorithms in that platform. For example, once we decide how we want the data for graph related problems, we can reuse the same code and components for animating Prim’s/Kruskal’s/Dijkshatra’s algorithm.
* Using this approach, we don’t need to actually run the algorithm every time, now we can save the log and directly use it as an input to the animation code.
* We don’t have to worry about syncing the both parts with each other as they are happening totally separately.

Disadvantages:

* It requires additional overhead to run the algorithm and generate log.
* Requires additional storage.
* Relatively slow.
* **Animate along with the execution of the algorithm**

In this approach, the execution of algorithm and the animation happen at the same time. After each logical step, a call is made to the animation function to perform necessary actions.

Advantages:

* Does not require any additional storage.
* relatively faster as the animation happens almost parallelly.
* It is a good approach for algorithms which depend on user input at each step. With prior approach we might need to run the entire algorithm at every step

Disadvantages:

* Animation and algorithm’s code are tightly coupled.
* The speed of execution of algorithm needs to be adjusted according to the speed of animation. In case of an “AutoPlay” mode, the execution of the algorithm and its animation should happen in sync.
* With this approach we have to spend time on every algorithm (even those which fall in the same category) for adding the calls to animation related functions

**3.3 Animation library**

Following animation libraries were studied as part of requirement gathering process. Each of the libraries was tested for support for above mentioned features.

* 1. **RaphaelJs**

Advantages:

* Open source
* Better solution for algorithms which require drawing elements along with the animation for example graph/tree related problems.
* Supports basic animation.

Disadvantages:

* Silent failures - The react-raphael library fails silently without throwing any error on the console. For example, if the react components are not properly nested, the entire DOM (until the root i.e. <Paper> element) painting fails without showing any error.
* Performance reduces linearly with respect to number of components on the DOM. More details on this in the “Performance” section below.
  1. **BonsaiJs**

This library is similar to raphael.js

Advantages:

* Open source
* Bonsai-react library is built on newer version of react.js
* Supports code splitting with the help of dynamic import hence performance improvement

Disadvantages:

* Not good support with react.js
* Not widely used and less documentation on the internet.
* Bonsai-react library is currently managed by a single contributor.
  1. **GSAP (GreenSock Animation Platform)**

Advantages:

* Inbuild support for pause and reverse animation
* Additional animation effects part from basic ease-in, ease-out etc.
* Lightweight
* Has good support with react, It has even provided lower level hooks to perform custom implementation
* Good documentation

Disadvantages:

* It’s not totally open source library
* Open source tools do not include vector graphics, hence not suitable solution for graph/tree related algorithms

1. **Performance**

This section only focuses on react-raphael library’s performance evaluation. The performance of the library when animating DOM elements is good, but when it comes to adding large number of react-raphael components, to the DOM, its performance reduces linearly. For example, in shortest path algorithm, after clicking on “New” button, 250 react-raphel components of type Rect, Text and Set are added to the DOM. Every component on an average takes 1ms to load which adds up to around 1 sec. As a result of that, we can see in the below screenshot that during first load, the frame rate drops from average of 50fps to 2fps.

Due to short span of the project, the performance was not tested against other libraries. But for a long-term project, this can be an area of improvement. Having said that, this has not hampered the overall performance at all as it happens during loading the entire grid only.

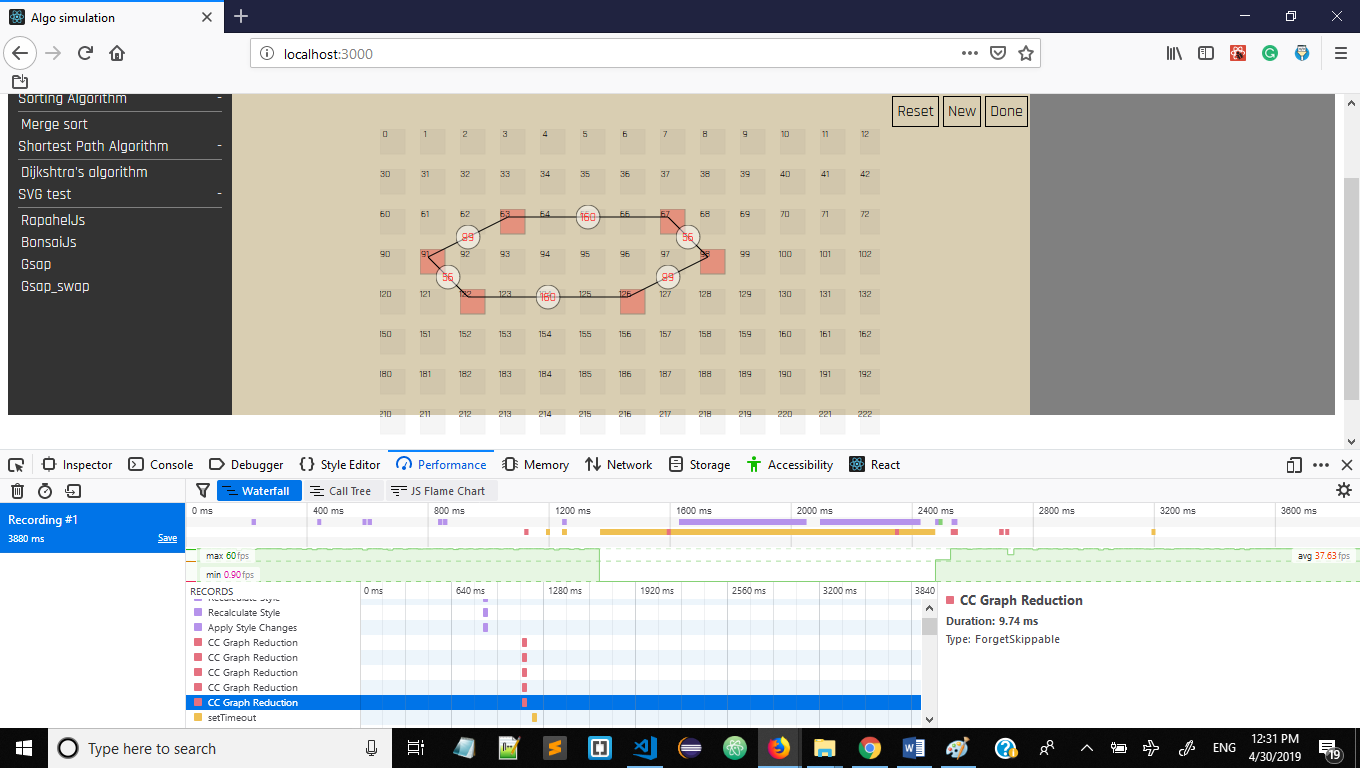


Figure c: grid-load-time - firefox devtools

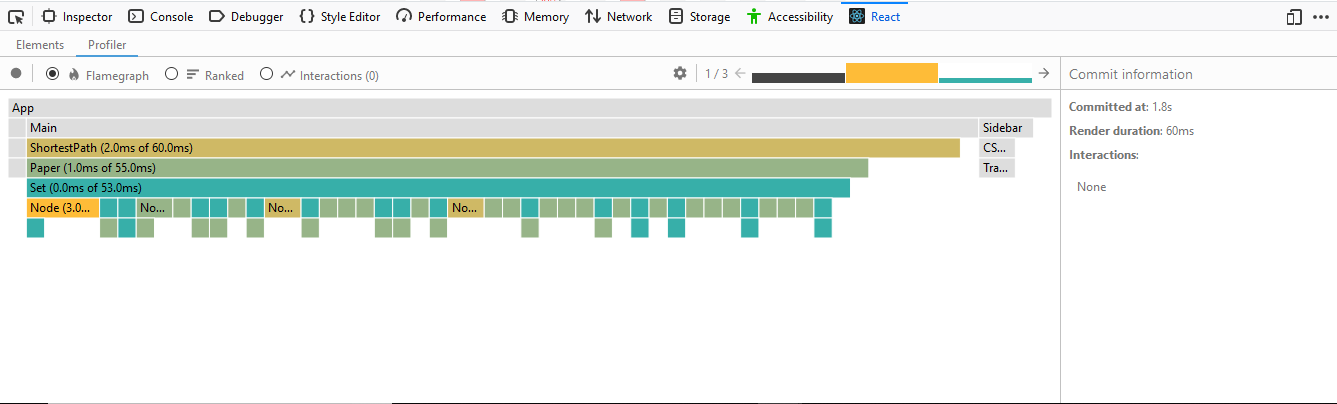


Figure d: grid-load-time – react devtools profiler output

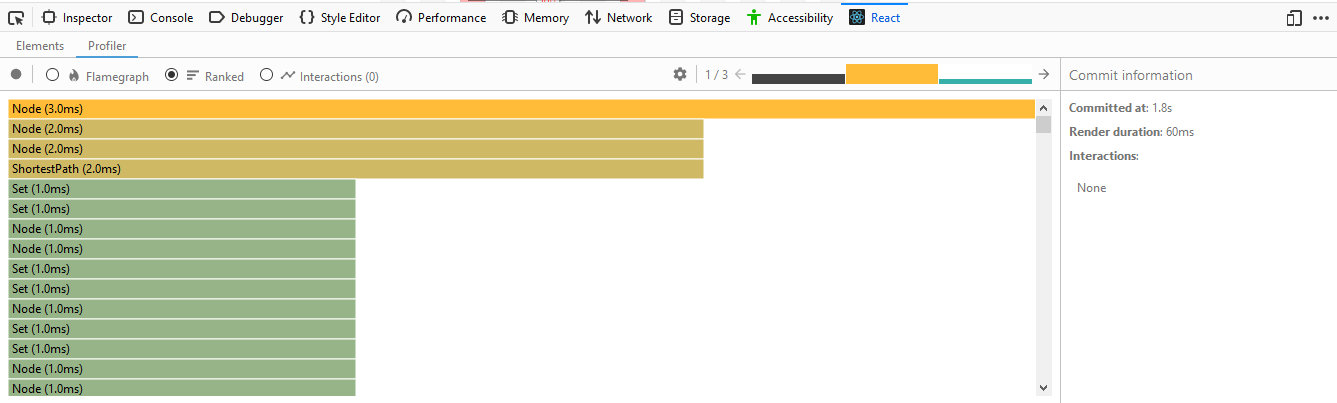


Figure e: grid-load-time sorted in descending order at component level

1. **Future work**

* Customizable logger

Instead of showing the entire information in the notification area, provide an option to the user to select the information he/she wants to see. This will remove unnecessary information, make UI cleaner and provide more free space for the animation purpose.

* Auto play feature

If the user already knows the working of algorithm and just interested in the final state, or want to use it for demo purpose, this feature would save some time reduce the interactions with the user interface.

* Persistent storage of input/output

Saving input and output as a file or database have following advantages

Avoid generating new input every time

Save standard example for each case. This will be helpful to teachers as well. We can save each corner case as an input file and at the time of explaining, directly load it and start the animation.

1. **Project links:**

The code for the project can be found at : <https://github.com/a-ad-adi/algo-simulation>

Current working solution can be found at : <http://algo-simulation-reactapp.s3-website.us-east-2.amazonaws.com/>

**7. References**

[1] <https://reactjs.org/docs/getting-started.html>

[2] <https://www.npmjs.com/package/react-raphael>

[3] <https://greensock.com/react>

[4] <https://www.npmjs.com/package/react-gsap>

[5] <http://raphaeljs.com/>

[6] <https://bonsaijs.org/>