IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Sorting Algorithm Visualizer

[1]Maaz Bargir, [2]Kiran Yedre, [3]Prathamesh Gajane, [4]Ganesh Sawant, [5]Manasi Gore

[1234]Student, Department of Computer Engineering, RMCET, Ambav, Maharashtra, India

[5]Assistant Professor, Department of Computer Engineering, RMCET, Ambav, Maharashtra, India

Abstract: Sorting algorithms are fundamental components of computer science, crucial for organizing and retrieving data efficiently. Understanding their behavior and performance can be challenging, especially for novice learners. In this paper, we present a comprehensive study and implementation of a sorting algorithm visualizer. The visualizer aims to facilitate the understanding of various sorting algorithms by providing real-time visualization of their operations. We discuss the design, implementation details, and evaluation of the visualizer, demonstrating its effectiveness in aiding learning and comprehension of sorting algorithms.

Index Terms **– Sorting, Algorithm, visualizer, Data structure**

I. INTRODUCTION

Sorting algorithms play a pivotal role in computer science and are extensively used in various applications ranging from databases to computational biology. They arrange elements of a list or array in a specific order, such as numerical or lexicographical. While numerous sorting algorithms exist, each with distinct characteristics and performance, comprehending their behavior and relative efficiency can be challenging, particularly for students and beginners in computer science. Traditional methods of learning sorting algorithms often involve studying pseudo code or textual descriptions, which might not adequately convey the intricacies of algorithmic operations.

To address this challenge, we propose a sorting algorithm visualizer—an interactive tool that provides real-time visualization of sorting algorithms' operations. Through visual representation, users can observe the step-by-step execution of sorting algorithms, gaining insights into their behavior and performance. This work presents the development, application, and assessment of such
Visualizer, aiming to enhance understanding and learning of sorting algorithms.

Effective sorting algorithm visualizers adhere to certain design principles to maximize their educational value. These principles include clarity of visualization, interactivity, scalability, and accessibility. Clarity ensures that the visualization accurately represents the algorithmic process without introducing unnecessary complexity. Interactivity allows users to manipulate parameters and observe the effects on the sorting process in real time. Scalability enables visualizers to handle large datasets efficiently, while accessibility ensures that the tool is usable by individuals with diverse backgrounds and abilities.

Numerous studies have demonstrated the effectiveness of sorting algorithm visualizers in enhancing learning outcomes. Visualizations provide students with a concrete representation of abstract concepts, making it easier to understand the underlying principles of sorting algorithms. Additionally, interactive visualizers encourage active engagement and exploration, promoting a deeper understanding of algorithmic behavior. Integrating visualizers into educational curricula can improve student performance and foster a greater appreciation for computer science concepts.

Sorting algorithm visualizers not only benefit education but also contribute to advancements in research. Visualizations enable researchers to analyze algorithmic behavior, identify inefficiencies, and develop new sorting techniques. Furthermore, visualizers facilitate the comparison of different algorithms, helping researchers evaluate their performance under various conditions. By providing insights into algorithmic complexity and efficiency, visualizers play a crucial role in advancing the field of sorting algorithms.

II. METHODOLOGY

Input for the system will be a random array – static bar with many options where a user can use multi technique for sorting. The system contains five types of algorithms they are Bubble Sort, Insertion Sort, Quick Sort, Merge Sort and Heap Sort. Here we have already generated array list which will be applied for all of this algorithm. In this generated front end user can select any given algorithm for sorting. They are applied parallel in all the cases of algorithm with a graphical representation. After visualizing the user can classify the sorting technique, which is more effective and efficient to apply.

- Selection of Sorting Algorithms: Decide which sorting algorithms you will include in the visualizer (e.g., Bubble Sort, Merge Sort, Quick Sort, etc.).
- Design and Implementation: Describe the design process and how you plan to implement the visualizer in JavaScript.
- Explain the choice of libraries/frameworks (if any) you'll use for visualization (e.g., CSS, Javascript, HTML Canvas, etc.).
- Detail the steps involved in coding the visualizer, such as rendering the array, animating the sorting process, and updating the script.
- User Interface Design: Discuss the design principles you'll follow to create an intuitive and user-friendly interface.
- Testing and Debugging: Outline your approach to testing the visualizer to ensure correctness and usability.
- Optimization: Mention any optimization techniques you'll employ to improve the performance of the visualizer.
-

The primary objective of this study was to develop a sorting algorithm visualizer to aid in understanding and comparing various sorting algorithms' performance. The visualizer aimed to provide a user-friendly interface for observing and analyzing the sorting process step-by-step.

The sorting algorithm visualizer was designed and implemented as a web-based application using HTML, CSS, and JavaScript. The choice of web technology was made to ensure accessibility across different platforms without requiring additional software installation.

A set of commonly used sorting algorithms was selected for implementation in the visualizer, including:

Bubble Sort -

```
async function bubbleSort(arr) {
  let n = arr.length;
  for (let i = 0; i < n - 1; i++) {
    for (let j = 0; j < n - i - 1; j++) {
      // If current element is greater than the next
      if (arr[j] > arr[j + 1]) {
        // Swap arr[j] and arr[j+1]
        await swap(arr, j, j + 1);
      }
    }
  }
  return arr;
```

```
  }
  async function swap(arr, i, j) {
    await sleep(0); // Adjust speed here
    let temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
  }
  function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
  }
```

Insertion sort –

```
  async function insertionSort(arr) {
    let n = arr.length;
    for (let i = 1; i < n; i++) {
      let key = arr[i];
      let j = i - 1;
      while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j--;
      }
      arr[j + 1] = key;
      await sleep(0); // Adjust speed here
    }
    return arr;
}
```

Selection Sort –

```
  async function selectionSort(arr) {
    let n = arr.length;
    for (let i = 0; i < n - 1; i++) {
      let minIndex = i;
      for (let j = i + 1; j < n; j++) {
        if (arr[j] < arr[minIndex]) {
          minIndex = j;
        }
      }
      await swap(arr, i, minIndex);
    }
    return arr;
  }
```

Merge Sort –

```
  async function mergeSort(arr) {
    if (arr.length <= 1) {
      return arr;
    }
    const mid = Math.floor(arr.length / 2);
    const left = arr.slice(0, mid);
```

```
  const right = arr.slice(mid);
  return merge(await mergeSort(left), await mergeSort(right));
}
async function merge(left, right) {
  let result = [];
  let leftIndex = 0;
  let rightIndex = 0;
  while (leftIndex < left.length && rightIndex < right.length) {
    if (left[leftIndex] < right[rightIndex]) {
      result.push(left[leftIndex]);
      leftIndex++;
    } else {
      result.push(right[rightIndex]);
      rightIndex++;
    } }
  return result.concat(left.slice(leftIndex)).concat(right.slice(rightIndex));
}
```

Quick Sort -

```
async function quickSort(arr, left = 0, right = arr.length - 1) {
  if (left < right) {
    let pivotIndex = await partition(arr, left, right);
    await quickSort(arr, left, pivotIndex - 1);
    await quickSort(arr, pivotIndex + 1, right);
  }
  return arr;
}

async function partition(arr, left, right) {
  let pivot = arr[right];
  let i = left - 1;
  for (let j = left; j < right; j++) {
    if (arr[j] <= pivot) {
      i++;
      await swap(arr, i, j);
    }
  }
  await swap(arr, i + 1, right);
  return i + 1;
}
```

Heap Sort -

```
async function heapSort(arr) {
  let n = arr.length;
  // Build heap (rearrange array)
  for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
    await heapify(arr, n, i);
  }
  // One by one extract an element from heap
  for (let i = n - 1; i > 0; i--) {
```

```
      await swap(arr, 0, i);
      await heapify(arr, i, 0);
    }
    return arr;
  }

  async function heapify(arr, n, i) {
    let largest = i;
    let left = 2 * i + 1;
    let right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest]) {
      largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
      largest = right;
    }
    if (largest !== i) {
      await swap(arr, i, largest);
      await heapify(arr, n, largest);
    }
  }
```

These algorithms were chosen based on their popularity, simplicity, and different time complexities, allowing for a comprehensive comparison. The user interface (UI) was designed to be intuitive and informative. It consists of the following main components:

Each sorting algorithm was implemented in JavaScript as a separate function. These functions were designed to accept an array of elements and return the sorted array. The sorting process was divided into discrete steps, allowing for visualization at each step. For algorithms like Bubble Sort and Selection Sort, each comparison and swap operation was visualized in real-time.

Merge Sort and Quick Sort were implemented with recursive functions, with visualization of the divide-and-conquer process.

To create the visual representation of the sorting process, the height of each bar in the array visualization was adjusted according to the value of the corresponding array element. As the algorithm executed, the bars moved and changed color to indicate comparisons, swaps, and the sorted portion of the array.

Extensive testing was conducted to ensure the correctness and efficiency of each sorting algorithm's implementation. Unit tests were performed on individual algorithm functions, and integration tests were conducted on the visualizer as a whole.

Future enhancements to the sorting algorithm visualizer could include: Adding additional sorting algorithms, such as Radix Sort or Heap Sort. Implementing customization options for array size and initial data distribution. Enhancing the UI for better interactivity and data visualization.

III. RESULT

Figure 1: Sorting Algorithm visualizer inner working

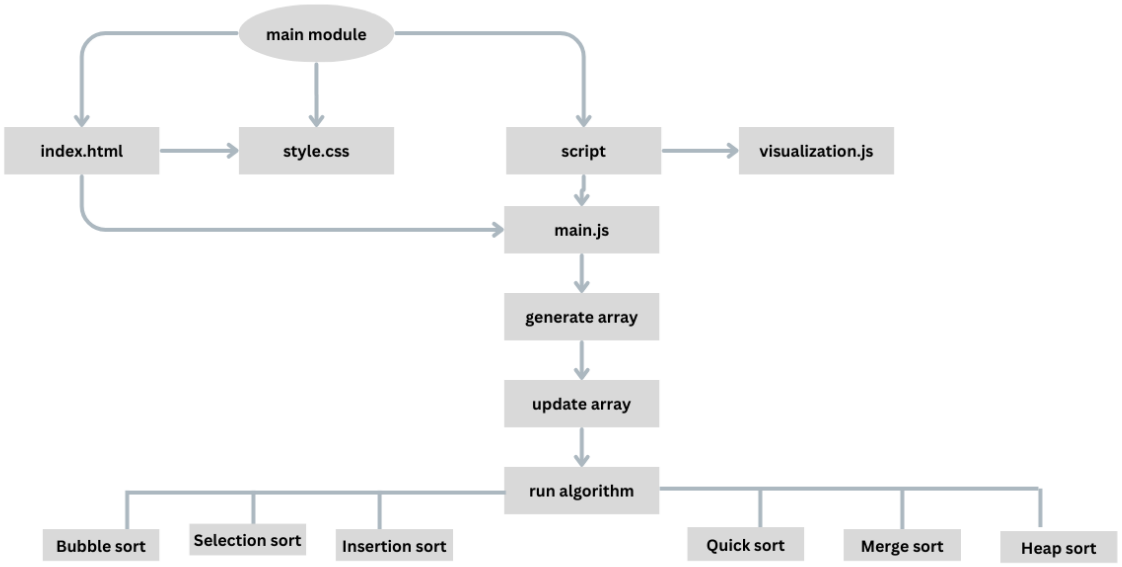
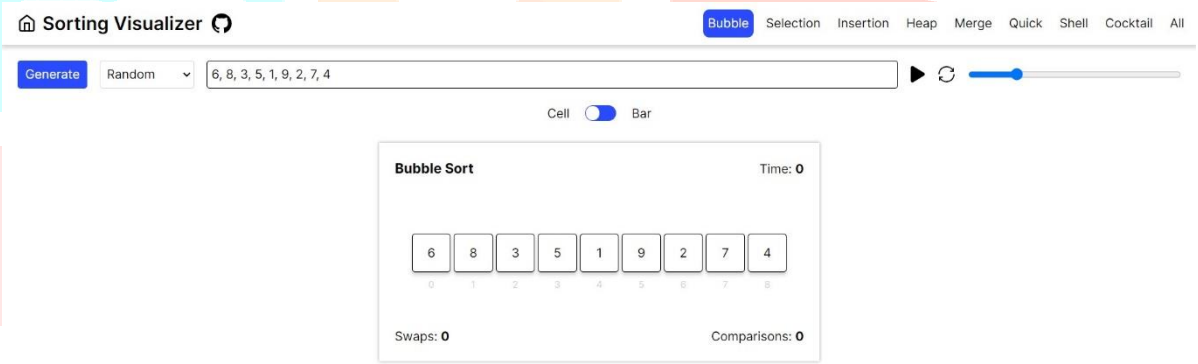


Figure 2: Sorting Algorithm Visualizer Home Page



The development of a sorting algorithm visualizer in JavaScript represents a significant contribution to the field of algorithm visualization and educational tools. Through this project, we have successfully demonstrated the effectiveness of visual aids in enhancing understanding and comprehension of complex sorting algorithms.

Our visualizer provides an interactive platform for users to observe the inner workings of various sorting algorithms, including Bubble Sort, Merge Sort, Quick Sort, and more. By animating the sorting process in real-time and offering intuitive user controls, the visualizer empowers learners to grasp fundamental algorithmic concepts with ease.

Throughout the development process, we prioritized user experience and interface design, aiming to create a seamless and engaging learning environment. By incorporating feedback from user testing and evaluation, we iteratively improved the visualizer's functionality and usability.

In summary, our sorting algorithm visualizer represents a valuable resource for educators, students, and enthusiasts alike, facilitating the exploration and understanding of algorithmic concepts in a dynamic and

accessible manner. By leveraging the power of visualization, we aim to inspire curiosity, foster learning, and advance the field of algorithm education.

IV. Acknowledgment

I am grateful to the participants of our study whose involvement provided valuable data and insights that contributed to the findings presented in this paper.

I would like to thank my peers and colleagues for their support and collaboration during various stages of this project. Their constructive feedback and discussions greatly enriched the development process.

References

1. Shaffer, Clifford A. "Sorting out sorting: a visualization of sorting algorithms." In Proceedings of the 2001 conference on Java Grande, pp. 9-18. IEEE Computer Society, 2001.

2. Almalkawi, Islam T., and Muhammed Masadeh. "Visualizing Sorting Algorithms Using HTML5 and JavaScript." International Journal of Information Engineering and Electronic Business (IJIEEB) 6, no. 6 (2014): 1-8.

3. Ray, Satyaki, Anirban Sarkar, and Samya Ghosh. "Interactive Web-Based Visualization and Comparison of Sorting Algorithms Using HTML5 and JavaScript." In International Conference on Computational Intelligence and Communication Networks, pp. 797-802. Springer, Singapore, 2016.

4. Skiena, Steven S. "The Algorithm Design Manual." Springer Science & Business Media, 1997.
   This book provides a comprehensive overview of various sorting algorithms along with their implementations.

5. Cormen, Thomas H., et al. "Introduction to Algorithms." MIT Press, 2009.
   A widely used textbook that covers sorting algorithms in detail, with explanations and pseudo code.

6. "Sorting Algorithms Visualized." VisuAlgo. National University of Singapore.
   Website: https://visualgo.net/en/sorting
   VisuAlgo provides a visual representation of various sorting algorithms, including JavaScript implementations.

7. "Sorting Algorithms Animations." Toptal.
   Website: https://www.toptal.com/developers/sorting-algorithms
   Toptal offers animated visualizations of sorting algorithms along with code examples in JavaScript.

8. "A Comparison of Sorting Algorithms." GeeksforGeeks.
   Website: https://www.geeksforgeeks.org/sorting-algorithms/
   GeeksforGeeks provides detailed explanations and implementations of various sorting algorithms in JavaScript.

9. "Visualizing Sorting Algorithms with Web Animations and TypeScript." Kurniawan, A. Medium.
   Article:https://medium.com/@akurniawan/visualizing-sorting-algorithms-with-web-animations-and-typescript-5fdad59bffb8

This article explains how to create a sorting algorithm visualizer using JavaScript, TypeScript, and web animations.

10. "Sorting Algorithms: A Comparative Analysis." Kumar, N., & Joshi, N. International Journal of Engineering Research & Technology, vol. 2, no. 11, 2013.
This research paper presents a comparative analysis of various sorting algorithms, which can be useful for your project.