Research Paper on Visualization of Sorting Algorithm

Rahul Srivastava[*1], **Sachin Kumar**[*2], **Dr. Sadhana Rana**[*3]
[*1,*2] Student, Computer Science And Engineering, SRMCEM, Lucknow, India.
[*3] Assistant Professor, Computer Science And Engineering, SRMCEM, Lucknow, India.
rahulsrivast356@gmail.com , sachingpt771@gmail.com

Abstract:
In the realm of sorting algorithms, visualization projects serve as educational tools to comprehend and demonstrate various sorting techniques. This paper presents an analysis and review of sorting visualization projects that do not utilize parallelism. By exploring sorting algorithms without parallel processing, this review aims to provide insights into the efficiency, functionality, and visual representation of these algorithms. Understanding sorting algorithms without parallelism contributes to a foundational understanding of their sequential execution and computational complexities.

Keywords: Sorting Algorithms, React Visualizer, Selection Sort, Merge Sort, Bubble Sort, Insertion Sort, Heap Sort.

1. Introduction

In the ever-expanding landscape of computer science, sorting algorithms represent foundational pillars, enabling efficient data organization and retrieval. Visualizing these algorithms offers a compelling gateway into understanding their inner workings and complexities. Throughout the annals of human ingenuity, the evolution of tools has continually transformed our capabilities. Digital computers, especially, stand as beacons of innovation, performing tasks at speeds that surpass human capacity, including executing intricate sorting algorithms. This sorting visualization project delves into the realm of sorting algorithms, sans parallelism, seeking to unravel their sequential execution and computational intricacies through visual representation. By immersing ourselves in this exploration, we endeavor to decode the mechanisms that govern sorting algorithms, providing an educational resource that illuminates their functionality and nuances. Beyond mere computational exercises, the visualization of sorting algorithms encapsulates a profound narrative—a testament to human endeavor, mathematical elegance, and the quest for optimized data manipulation. Through this project, we aim to unlock the visual symphony underlying sorting methodologies, fostering a deeper comprehension accessible to both seasoned enthusiasts and budding computer science learners.
The sorting visualization project focuses on creating a user-friendly online tool aimed at explaining and visually demonstrating how sorting algorithms function in organizing and transforming sets of data. It's designed to cater to various learning styles, especially emphasizing visual learning preferences.

The project aims to serve as an educational resource for students, enthusiasts, or anyone interested in understanding sorting algorithms. By offering a visual representation, it facilitates a deeper understanding of these fundamental concepts in computer science.
Navigating through challenges often involves finding the most effective approach rather than delving into complexities. Consider the scenario of a broken car headlight—an everyday inconvenience made increasingly confounding by modern, abstract designs.
Resolving such issues typically involves consulting a car handbook, conducting research, or seeking guidance from someone experienced. In a similar vein, learning styles significantly influence how we absorb information. As a visual learner, I discovered my affinity for comprehending complex concepts through visual demonstrations rather than textual explanations. This realization sparked my curiosity in understanding sorting algorithms and prompted the creation of an online tool detailed in this paper—a tool designed to elucidate the transformative nature of sorting algorithms in organizing data sets. Imagine organizing a list of individuals by their ages in ascending order—this simple task mirrors the essence of sorting algorithms.

To aid visualization, I crafted a histogram representing numerical data, where each number translates into a bar's height, symbolizing its value. The journey of these data points, undergoing transformation from disarray to ordered sequences, mirrors the essence of Selection Sort, Bubble Sort, Insertion Sort, and Merge Sort—four well-known sorting algorithms. To simplify the visualization, imagine age labels on index cards. Selecting the youngest card and sequentially arranging them is akin to Selection Sort's process—an intuitive analogy that simplifies understanding. However, grasping more intricate algorithms, such as Quick Sort, which pivot around data reorganization, proves challenging through text alone. In a bid to cater to diverse learning styles, I developed an animation-based visualization tool, accessible via the web. Utilizing HTML5, JavaScript, and CSS, this web-based platform eliminates the need for additional software installations or complex setups. The decision to leverage web-based technologies aims to mitigate user anxiety, ensuring a seamless and user-friendly experience for individuals across various technological proficiencies.
Understanding the intricacies of sorting algorithms often proves to be a challenge, especially when attempting to visualize the complex maneuvering of data in text form alone. Hence, inspired by my innate affinity for visual learning and realizing the inherent complexities in comprehending these algorithms through conventional textual descriptions, I embarked on a mission to create an interactive, visually immersive online platform.

The image is a flowchart for a sorting website. It outlines the steps and options available on the website for sorting an array. Users can generate a new array to sort through direct input of array values, random generation of array values, or changing the array size. They can set the speed of sorting and specify the type of sort they want to use: selection sort, bubble sort, insertion sort, or merge sort. After these parameters are set, users can start sorting and the website will show comparison/exchange/merge count.

Below that there's a box labeled "Generate a new array to sort" with three options: - Direct input of array values - Random generation of array values - Change array size - Next step in flow is "Set the speed of sorting". - Followed by another box labeled "Specify sort type" with four options: - Selection Sort - Bubble Sort - Insertion Sort - Merge Sort - The final step in flow is an oval labeled "Start sorting", followed by text “show comparison/exchange/merge count”.

The figure (figure 1) below illustrates the Working and Visualization of Sorting Algorithm.

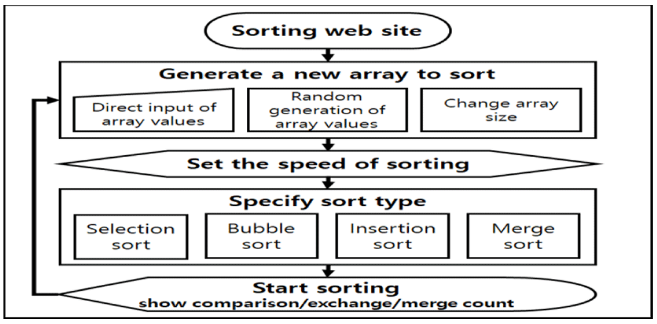


Figure1: Sorting Algorithm Working.

2. Module Description

A module diagram is a graphical representation of the components and dependencies of a software system. It shows how the modules interact with each other and what functions they provide. A module diagram for a visualization of sorting algorithm might look something like this (Figure 2).

Figure2: User Interface

The User Interface Module plays a pivotal role in facilitating user interaction and input. In a parallel context, this module could offer enhanced functionalities, enabling users to select between running sorting algorithms either sequentially or in parallel.

It might present options for users to specify the number of threads or processing units to engage in parallel sorting. Additionally, the interface could provide controls allowing users to adjust the level of parallelism, impacting sorting speed and efficiency dynamically.

The Sorting Module, responsible for generating and executing sorting algorithms, undergoes a substantial transformation with parallelism. It divides the array data into segments allocated to different threads or processors for concurrent processing. In a parallel setting, this module orchestrates the simultaneous execution of sorting algorithms across multiple segments, managing synchronization points where sorted segments are merged or combined to form the final ordered array.

Meanwhile, the Visualization Module brings the sorting process to life through graphics. In a parallel context, it visualizes the concurrent execution of sorting algorithms by various threads or processors. The module's parallel representation could include graphical depictions of multiple threads or processors engaging in simultaneous sorting actions. Visual animations might illustrate how these segments interact and eventually merge to produce the final sorted sequence.

The Input / Output Module manages user interactions and data operations. In parallelism, it could provide real-time statistics on parallel execution times or thread utilization. Users might be able to dynamically modify parallel settings during sorting, such as adjusting the number of parallel threads or processors engaged in the sorting process. The Sorting Logic Module is responsible for implementing sorting algorithms, adapting them for parallel execution. In a parallel environment, it segments the data for simultaneous sorting across multiple threads or processors. This module orchestrates synchronization and communication between these segments, ensuring accurate merging or combination into the final sorted sequence.

Lastly, the Graphics Library Module provides low-level graphics functions. In a parallel context, it may require optimization to handle concurrent rendering from multiple threads or processors efficiently. Enhancements might involve managing concurrent graphics rendering to ensure a smooth visualization of the parallel sorting process, aligning with the sorting logic and visualization modules for coherent parallel execution.

Typically, Visualization Project stands as the visual storyteller, translating sorting algorithms into graphical representations and animations. Its primary goal is to provide users with a visual journey through the sorting process, elucidating complex algorithmic operations through intuitive graphics. This module plays a crucial role in enhancing comprehension by illustrating step-by-step sorting procedures, making abstract algorithms more tangible and comprehensible. Through visual aids and animations, it ensures users grasp the inner workings of sorting algorithms, fostering a deeper understanding of their functionalities and complexities within the sorting visualization tool. This is the purpose of the explanation module in Figure 2 . The Sorting Module operates as the backbone responsible for executing sorting algorithms based on user inputs. Its core function involves dividing the dataset into segments for potential parallel processing, execution of sorting algorithms.

3. Used Methodologies

Frames In a Sorting Visualization project, several methodologies can be employed to develop and implement the functionality of visualizing sorting algorithms. Here are some commonly used methodologies:

1.Object-Oriented Programming (OOP): -

Description: OOP involves organizing code into objects that have attributes and methods. Each sorting algorithm, user interface element, or visualization component can be represented as an object.

Implementation: Implement sorting algorithms, user interface elements, and visualization components as separate objects. Utilize inheritance, encapsulation, and polymorphism to create a modular and maintainable codebase.

2. Model-View-Controller (MVC): -

Description: MVC separates the application into three interconnected components: Model (data), View (user interface), and Controller (logic).

Implementation: Use this pattern to separate sorting algorithm logic (Model), user interface elements (View), and user interactions and algorithm execution control (Controller). It helps in better organization and maintenance of the codebase.

3. Asynchronous Programming: -

Description: Asynchronous programming enables concurrent execution of tasks, useful for implementing parallel sorting algorithms or updating the visualization while the sorting process is ongoing.

Implementation: Utilize features like multithreading or asynchronous JavaScript (such as Web Workers) to perform parallel sorting algorithms. Update the visualization in real-time to reflect changes as the sorting progresses.

4. Data Structures and Algorithms: -

Description: Employ various data structures (arrays, linked lists) and sorting algorithms (Bubble Sort, Merge Sort, Quick Sort) to showcase different sorting techniques. Implementation: Implement these algorithms considering their time complexity and efficiency. Visualize the step-by-step process of sorting, highlighting comparisons, swaps, or partitioning steps.

5. Event-Driven Programming: -

Description: Event-driven programming reacts to user actions or system events, triggering responses accordingly. In a sorting visualization tool, this methodology allows users to interact with the interface and initiate sorting. Implementation: Implement event handlers that respond to user inputs (e.g., selecting a sorting algorithm, adjusting speed) to trigger the execution of sorting algorithms or update the visualization.

6. Responsive Design and Animation: -

Description: Incorporate responsive design principles for the user interface elements and animations to ensure a seamless experience across various devices.

Implementation: Utilize CSS for responsive layouts and transitions, and JavaScript or libraries like D3.js for creating dynamic and engaging visualizations of sorting algorithms.

7. Continuous Integration and Deployment (CI/CD): - Description: CI/CD practices facilitate automated testing, integration, and deployment of code changes, ensuring a stable and updated application. - **Implementation**: Set up automated testing for sorting algorithms, user interface components, and visualizations. Implement continuous deployment pipelines for timely updates and improvements. Implementing these methodologies contributes to the robustness, maintainability, and effectiveness of the Sorting Visualization project, ensuring a user-friendly, informative, and engaging experience for users exploring sorting algorithms.

ALGO-SORT
Visualization of Sorting Algorithm
SORTING
SEARCHING
HOME
ALGO-SORT
Visualization of Sorting Algorithm

Figure3: ALGO-SORT

Certainly! These methodologies contribute significantly to the development of a Sorting Visualization project. Object-Oriented Programming (OOP) structures code into reusable objects, enhancing modularity and maintainability. Model-View-Controller (MVC) divides the application into interconnected components, separating data, user interface, and logic, facilitating better code organization. Asynchronous Programming enables concurrent task execution, vital for implementing parallel sorting algorithms and updating real-time visualizations. Data Structures and Algorithms bring diversity to sorting techniques, showcasing different sorting methods and visualizing step-by-step processes. Event-Driven Programming reacts to user actions, initiating sorting algorithms and enhancing user interaction. Responsive Design and Animation ensures a seamless user experience, adapting the interface to various devices and using animations for engaging visualizations. Continuous Integration and Deployment automates testing and deployment, ensuring a stable and updated application. Integrating these methodologies results in an efficient, user-friendly, and informative Sorting Visualization tool.

The Continuous Integration and Deployment automates testing, integration, and deployment processes, ensuring code stability and timely updates. This methodology helps maintain a robust and updated application by automatically validating changes and deploying new versions without disruptions, thereby improving the overall quality and reliability of the sorting visualization tool.

4. Unit Testing

Unit testing for visualizations of sorting algorithms involves verifying the correctness and functionality of individual components responsible for visual representation. While traditional unit testing focuses on verifying code logic and functions, testing visualizations requires assessing whether the visual representations accurately depict the sorting process. Here's how unit testing can be approached for visualizations of sorting algorithms:

1).Test Component Behavior:
Validation of Sorting Logic: Ensure that the visualization accurately reflects the steps of the sorting algorithm. Validate that elements are rearranged according to the algorithm's logic (e.g., comparisons, swaps) at each iteration.
Handling Edge Cases: Test the visualization for handling edge cases, such as sorting an already sorted array, an array in reverse order, or an array with duplicate elements, ensuring the visualization responds appropriately.

2).Comparison with Expected Output:
Expected Visualization Output: Establish a reference or expected output for each step of the sorting algorithm. Compare the actual visualization output with the expected output to verify correctness.
Data Consistency Check: Confirm that the data representation and positioning of elements in the visualization match the expected outcome at each sorting step.

3).Interactive Component Testing:
User Interaction: If the visualization tool involves user interactions (e.g., adjusting sorting speed, selecting different algorithms), ensure these interactions trigger the correct visual changes.
Real-Time Updates: Test the visualization for real-time updates, ensuring that modifications initiated by user interactions are accurately reflected without errors or delays.

4).Handling Visual Rendering:
Cross-Browser and Device Testing: Verify that the visualization renders consistently across various browsers and devices, ensuring a uniform experience for users. Responsiveness and Animation Verification: Assess responsiveness by checking how the visualization adapts to different screen sizes. Ensure animations and transitions are smooth and correctly depict the sorting process.

5).Error Handling and Edge Cases:
Handling Errors: Test how the visualization handles unexpected scenarios, such as erroneous inputs or interrupted processes, ensuring graceful handling of errors without crashing the application.
Performance Testing: Evaluate the visualization's performance when dealing with large datasets to ensure it remains responsive and functional without significant degradation.

Implementing unit tests for sorting algorithm visualizations involves evaluating the visual output against expected outcomes, validating user interactions, and ensuring error-free and accurate representations of sorting processes. Additionally, it focuses on responsiveness, cross-compatibility, and handling various scenarios to ensure a robust and reliable visualization tool.

Unit testing for visualizations of sorting algorithms is a crucial aspect of ensuring the accuracy, functionality, and reliability of the graphical representations depicting the sorting processes. Unlike traditional unit testing that primarily focuses on verifying code logic and functionalities, testing visualizations involves evaluating whether the visual elements effectively illustrate the sorting algorithm's execution. The validation process includes several key components. Firstly, it assesses the behavior of visualization components, ensuring they accurately mirror the steps of the sorting algorithm, including comparisons, swaps, or any relevant operations. This validation extends to handling edge cases, confirming that the visualization appropriately responds to scenarios such as sorting pre-sorted arrays or arrays with duplicate elements. Moreover, comparing the actual visualization output with an expected reference output becomes crucial; it verifies that the data representation and positioning of elements align with the anticipated outcome at each sorting step.
Another critical aspect involves testing interactive elements if present within the visualization tool. This includes validating user interactions—such as altering sorting speed or selecting different algorithms—to ensure they trigger correct and expected visual changes in real time. Additionally, comprehensive testing covers the rendering of visual elements across various browsers and devices to confirm consistent performance and presentation. It includes assessing responsiveness across different screen sizes and ensuring smooth animations that accurately portray the sorting process. Furthermore, robust unit testing addresses error handling, checking how the visualization manages unexpected scenarios or erroneous inputs, guaranteeing the tool's resilience and stability. Finally, performance testing evaluates how the visualization handles large datasets, ensuring that it remains responsive and functional without compromising the user experience. Through these comprehensive testing methodologies, unit testing for sorting algorithm visualizations aims to ensure accuracy, interactivity, responsiveness, and reliability, culminating in an effective and trustworthy visualization tool for exploring Sorting algorithms.

Unit testing for visualizations of sorting algorithms is pivotal in validating the accuracy, functionality, and robustness of the graphical representations depicting sorting processes. Unlike conventional testing that scrutinizes code logic, visualization testing ensures that the visual elements faithfully reflect the steps of sorting algorithms.
This testing approach involves multifaceted assessments to verify different aspects of the visualization tool. Firstly, it meticulously inspects the behavior of visualization components, validating their synchronization with sorting algorithms' steps, encompassing comparisons, swaps, and other operations. This scrutiny extends to handling diverse scenarios, ensuring the visualization adeptly responds to

varying inputs, including already sorted arrays or atypical datasets with duplicates or outliers. Furthermore, unit testing compares the actual visualization output against predetermined reference outputs, meticulously scrutinizing data positioning and rendering accuracy at each sorting step.

5. Implementation

View

The implementation of the View component in the Sorting Algorithm project, adhering to the Model-View-Controller (MVC) architecture, concentrates on crafting the user interface and visualization elements. Within this component, the primary goal revolves around creating an intuitive and interactive user interface that effectively presents the sorting algorithm visualizations. Designing the user interface involves the utilization of HTML, CSS, and JavaScript, or relevant front-end technologies, to architect layouts, interactive elements, and visual representations.
These visualizations, which could be in the form of animated charts, diagrams, or graphical depictions, are meticulously crafted to accurately illustrate the step-by-step sorting process. Additionally, the implementation encompasses the incorporation of interactive components, such as buttons, dropdown menus, and sliders, empowering users to input data, select sorting algorithms, adjust settings (such as speed), and initiate sorting executions. Ensuring the interface's responsiveness across diverse devices and browsers remains paramount, facilitating a seamless and consistent user experience. Ultimately, the View component serves as the conduit for users to engage with the sorting algorithms visually, fostering comprehension and engagement through an intelligible and interactive interface.

Model

The implementation of the Model component in the Sorting Algorithm project, following the Model-View-Controller (MVC) architecture, is centered around housing and executing the sorting algorithms. Within the Model section, the primary emphasis lies in encapsulating the logic and functionalities of different sorting algorithms, including but not limited to Bubble Sort, Merge Sort, and Quick Sort. The Model component houses the core codebase responsible for the accurate execution of these algorithms. It involves meticulously coding the logic for each sorting method, defining methods or functions representing their specific sorting techniques, and rigorously testing these implementations with diverse datasets. Testing encompasses various scenarios, ensuring the algorithms correctly handle common cases like already sorted arrays or those containing duplicate elements.

Moreover, optimization efforts are crucial, focusing on refining algorithms to enhance their efficiency by minimizing unnecessary computations and considering time complexity to facilitate faster sorting processes. This Model-centric approach ensures the reliability, accuracy, and performance of the sorting algorithms embedded within the application.

Controller

The implementation of the Controller component in the Sorting Algorithm project, adhering to the Model-View-Controller (MVC) architecture, focuses on managing and orchestrating user interactions while facilitating communication between the View and Model components. Serving as the central coordinator, the Controller interprets

and manages user inputs within the application. It encapsulates the logic responsible for handling user interactions initiated through the user interface elements embedded within the View. This includes interpreting user selections of sorting algorithms, adjustments in settings, or triggering sorting executions. The Controller component acts as the bridge, facilitating seamless communication between the View, where users interact with the interface, and the underlying Model, housing the sorting algorithms.
It ensures that user-triggered events are appropriately translated into actions, invoking the relevant sorting algorithms from the Model. By coordinating these interactions, the Controller governs the flow of the application, directing the execution of sorting algorithms based on user inputs and updating the View with real-time sorting visualizations or relevant information derived from the Model's operations. This pivotal component ensures a cohesive and interactive user experience while efficiently orchestrating the sorting algorithm executions in response to user interactions.

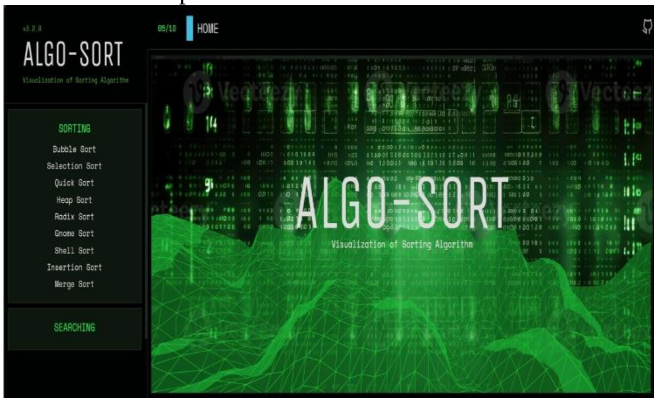


Figure4: Model Interface

The Model component in the Sorting Algorithm project plays a fundamental role in encapsulating the logic and implementation of various sorting algorithms. At its core, the Model section hosts a collection of meticulously crafted sorting algorithms such as Bubble Sort, Merge Sort, Quick Sort, and other variations. Each algorithm is meticulously translated into executable code segments within the Model, adhering to the programming language's syntax and design principles. Methods or functions are structured and organized to represent individual sorting techniques, ensuring clear and distinct encapsulation of each algorithm's logic. Rigorous testing and validation procedures are conducted within this Model segment to assess the algorithms' performance across different scenarios, including varying dataset sizes, pre-sorted arrays, or arrays containing duplicate elements.
Additionally, optimization strategies are employed within the Model to fine-tune these algorithms, focusing on enhancing their efficiency and minimizing redundant computations to elevate sorting speed and overall performance. Through this

meticulous implementation process, the Model component forms a robust foundation, ensuring the reliability, accuracy, and efficiency of the sorting algorithms integrated into the Sorting Algorithm project.

6. Comparative Analysis of Sorting Algorithm

Insertion Sort :

In the Sorting Visualization project, the implementation of the Insertion Sort algorithm involves a visual representation that elucidates its step-by-step functioning. Initially, the algorithm considers the first element of the dataset as a sorted sequence, marking its assumed correct position. Subsequently, it iterates through the unsorted elements, sequentially comparing each element with those in the sorted sequence. As the algorithm progresses, it scrutinizes each unsorted element against the sorted sequence, shifting greater elements to the right to create space for insertion when a smaller value is encountered. This process visually demonstrates the comparison and insertion steps, showcasing the transition of elements into their rightful positions within the sorted sequence. Graphical representations, often depicted as bars or array elements with their respective values, dynamically illustrate the algorithm's execution through animations or real-time updates. Users can visually track how elements are checked, shifted, and finally placed, culminating in the complete sorting of the dataset in ascending or descending order based on the specified criteria. This visual depiction enhances user comprehension of Insertion Sort's mechanisms, fostering a clearer understanding of sorting algorithms through an interactive and illustrative approach.

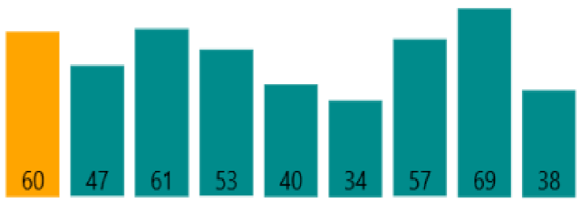


Figure5: Insertion Sort

Selection Sort:

The Selection Sort algorithm's functionality in a Sorting Visualization project involves a structured process aimed at sorting elements within a dataset in ascending or descending order, visualizing each step for user comprehension. Initially, the algorithm starts with an unsorted dataset, gradually forming a sorted sequence by repeatedly selecting the smallest (or largest) element from the unsorted section and placing it at the beginning (or end) of the sorted portion. As the algorithm progresses, it iterates through the dataset, meticulously identifying the smallest (or largest) element within the unsorted segment. Subsequently, this element is exchanged with the element at the current position in the sorted sequence. This step-by-step process continues until all elements are appropriately placed within the sorted sequence. Visually, the sorting visualization illustrates this iterative selection and swapping of elements, usually depicted graphically as bars or array elements transitioning into their correct sorted positions. By showcasing these transitions dynamically through animations or real-time updates, users can follow the algorithm's progression, comprehending how elements are selectively chosen and arranged, ultimately culminating in a fully sorted dataset.

This interactive visual representation facilitates a clearer understanding of the Selection Sort algorithm's mechanics and its iterative approach to sorting.

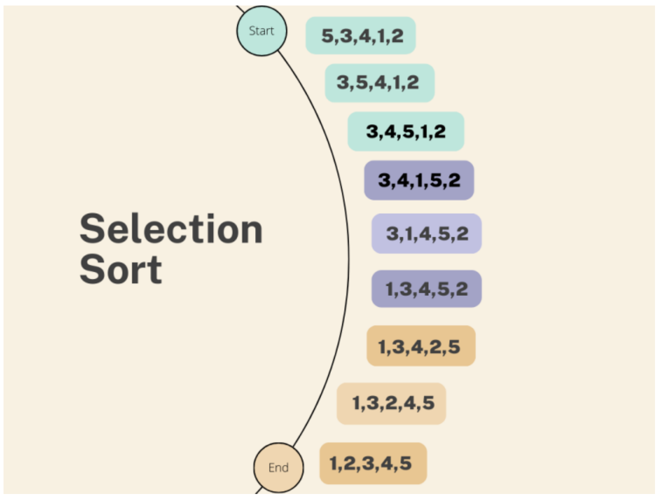


Figure6: Selection Sort

Quick Sort :

Quick Sort algorithm, when visualized within a Sorting Visualization project, employs a divide-and-conquer strategy to efficiently sort elements within a dataset. This method involves selecting a pivot element from the dataset and partitioning the elements into two groups based on their relation to the pivot. Subsequently, the algorithm recursively sorts these partitions until the entire dataset is sorted. Initially, the algorithm chooses a pivot element, often the last element in the dataset or selected randomly. Then, it rearranges the elements so that those smaller than the pivot are placed before it, while larger elements follow. This partitioning process divides the dataset into two segments. The Quick Sort algorithm then recursively applies this partitioning and sorting process to each segment, dividing them further until the entire dataset is sorted. Visually representing Quick Sort involves illustrating the partitioning of elements around the pivot, graphically showing the elements moving into their correct positions relative to the pivot. This visualization process, commonly displayed as bars or array elements transitioning through animations or real-time updates, aids users in comprehending how the algorithm efficiently divides and conquers the dataset to achieve the final sorted sequence. Through this visual depiction, users can observe the gradual sorting of elements, enhancing their understanding of Quick Sort's recursive nature and its proficiency in sorting large datasets of data pre-processing tools.

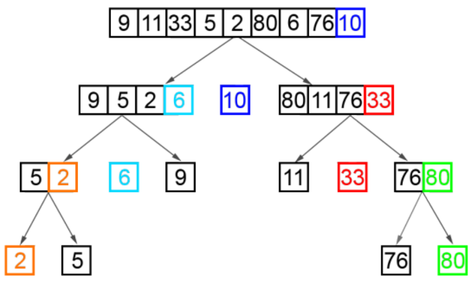


Figure7: Selection Sort

Merge Sort:

The Merge Sort algorithm, as visualized in a Sorting Visualization project, implements a divide-and-conquer strategy to sort elements within a dataset. It systematically divides the dataset into smaller segments, sorts them individually, and then merges them back together to achieve a fully sorted sequence. Initially, the algorithm divides the dataset recursively into halves until each segment contains only one element. Subsequently, it merges these smaller segments, sorting and combining them to form larger sorted segments. This merging process continues until the entire dataset is sorted. Visually, Merge Sort is represented by illustrating the recursive division of the dataset into smaller parts and the subsequent merging of these sorted segments. This visual representation, often depicted as bars or array elements transitioning through animations or real-time updates, demonstrates how smaller segments are sorted and then merged to form larger sorted sequences, ultimately resulting in a fully sorted dataset. This visual aid aids users in understanding the iterative nature of Merge Sort and its efficiency in sorting large datasets through division, sorting, and merging operations.

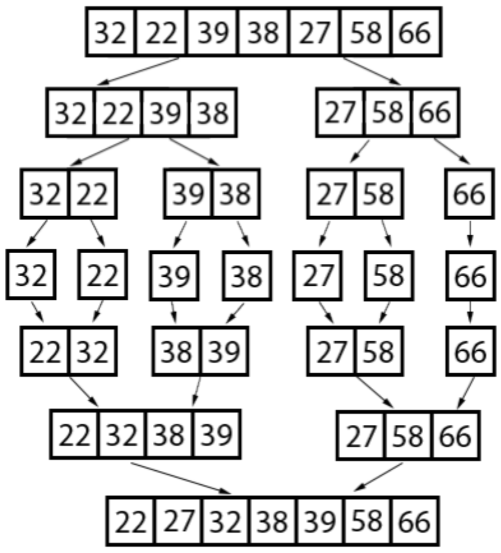


Figure8: Merge Sort

Heap Sort:

The Heap Sort algorithm, when visualized in a Sorting Visualization project, employs a binary heap data structure to sort elements within a dataset. It involves converting the dataset into a max-heap or min-heap structure, then repeatedly extracting the root element (the maximum or minimum element) and reconstructing the heap until all elements are sorted. Initially, the algorithm builds a heap by arranging the elements in a specific order—either as a max-heap, where the parent element is greater than its children, or a min-heap, where the parent element is smaller than its children. It then repeatedly extracts the root element and rearranges the remaining elements to maintain the heap property. Visually representing Heap Sort involves showcasing the construction and manipulation of the heap structure. This representation, often illustrated as bars or array elements transitioning through animations or real-time updates, demonstrates how elements are organized into a heap and then extracted and reordered to achieve a fully sorted dataset. Through this visualization, users can comprehend Heap Sort's systematic extraction of elements based on heap properties, leading to the eventual sorting of the entire dataset.

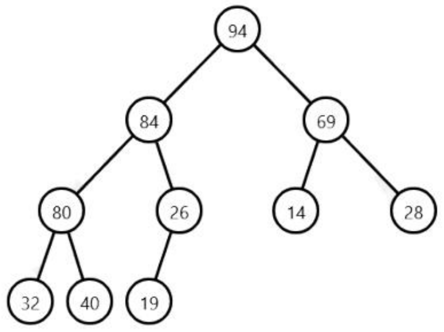


Figure9: Heap Sort

Implementing sorting algorithms in a Sorting Visualization project is a multifaceted process that intertwines algorithmic functionality with visual representation. The initial phase involves selecting and incorporating various sorting algorithms into the project scope, considering algorithms like Bubble Sort, Selection Sort, Merge Sort, Quick Sort, and others. Each algorithm demands meticulous implementation of its logical structure and functionality in the chosen programming language. This implementation guarantees the accurate execution of sorting methodologies, ensuring that the algorithms correctly sort the dataset. Concurrently, a significant aspect of this project lies in crafting a compelling visual representation of the dataset to be sorted. Typically, this involves graphical elements such as bars or array representations, visually reflecting the dataset's elements and their values. These elements act as dynamic components whose movements and positions change throughout the sorting process. Associating sorting actions, such as comparisons, swaps, or repositioning, with visual changes enhances user engagement and facilitates a clearer understanding of the algorithm's behavior. For instance, when executing Bubble Sort within the visual environment, the comparisons and exchanges between elements are visually translated, enabling users to observe how elements shift until the dataset is entirely sorted. This synchronization between algorithmic execution and the graphical representation of the dataset creates an interactive and educational experience. Users can witness the step-by-step progression of the sorting process, visually tracking elements as they transition into their correct positions within the sorted sequence. Through this amalgamation of algorithmic precision and visual feedback, users gain an intuitive understanding of the inner workings of sorting algorithms. This visual aid not only fosters a deeper comprehension of sorting methodologies but also promotes an engaging learning experience.

This fusion allows users to witness and comprehend the intricate steps involved in sorting datasets, providing a dynamic educational experience. Visualizing sorting algorithms through graphical elements or animations aids in demystifying complex concepts, making them accessible to users at various proficiency levels. The interactive nature of

this visual representation fosters a hands-on learning environment, empowering users to grasp sorting techniques by observing the algorithms' behavior in action. Ultimately, the combined approach of algorithmic implementation and visual depiction creates a compelling platform for comprehending and exploring the dynamics of sorting algorithms.

Limitations :

- Traditional algorithms need to have all data in main memory.
- Big datasets are an issue.

Solution

- Incremental schemes; having the datasets in several schemes or sizes.
- Stream algorithms; the use of MOA “Massive Online Analysis” (Coincidentally, Mao is not only a streaming algorithm but a flightless bird which also is extinct!)

7. Visualization Libraries

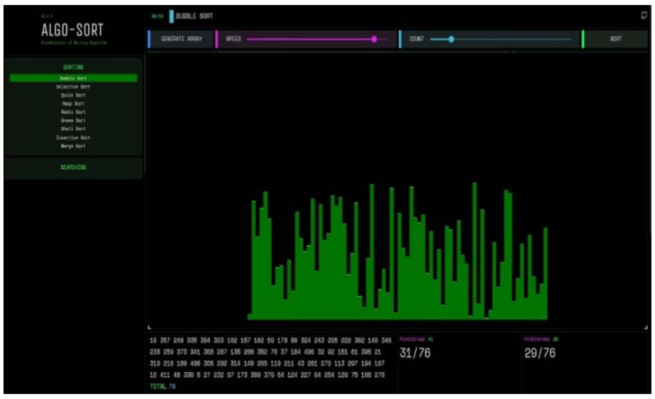
D3.js: D3.js is a robust JavaScript library specifically designed for data visualization on web browsers. It enables the creation of dynamic and interactive visualizations by binding data to HTML, SVG, and CSS elements. D3.js provides a wide range of tools for generating various charts, graphs, and animations, making it suitable for illustrating sorting algorithms dynamically

P5.js: p5.js is a JavaScript library designed for creative coding, making it an accessible and versatile tool for artists, designers, educators, and beginners interested in coding and visual arts. Developed with a focus on simplicity and flexibility, p5.js simplifies the process of creating visual elements and interactive experiences within web environments. It is based on the Processing programming language and inherits its core concepts, allowing users to engage in creative coding through visual expressions.

At its core, p5.js provides a rich set of functions and commands to create graphics, animations, and interactive content directly within web browsers. With its easy-to-understand syntax and immediate feedback, users can swiftly produce visual representations and interactive elements, making it an ideal platform for those new to programming or seeking a visual approach to learning coding concepts.The library offers various built-in functions for drawing shapes, creating animations, handling user interactions, and manipulating visual elements. Users can generate diverse graphical elements, including basic geometric shapes, images, text, and custom visualizations, leveraging p5.js's extensive capabilities. Moreover, p5.js facilitates interaction by responding to user inputs, such as mouse movements, clicks, and keyboard interactions, enabling the creation of interactive and responsive visuals.

Its versatility extends beyond graphics and animation; p5.js supports integration with HTML, CSS, and other web technologies, allowing seamless incorporation of visual creations into web pages and projects. Additionally, p5.js provides access to a supportive community, a wide range of documentation, tutorials, and examples, fostering a collaborative and educational environment for users to explore creative coding.

In essence, p5.js stands as a user-friendly and powerful JavaScript library, empowering individuals to express creativity through code-driven visual arts, animations, and interactive experiences on the web. Its simplicity, comprehensive functionality, and emphasis on creative expression make it a valuable tool for both learning programming concepts and producing engaging visual content within web-based projects.



```
16 357 249 335 384 303 102 157 102 59 178 86 324 243 205 222 362 146 348
238 259 373 341 368 287 138 200 382 70 37 184 406 32 92 181 81 396 21
319 216 189 408 308 292 314 140 265 119 211 43 281 270 113 297 194 167
10 411 48 330 5 27 232 97 173 389 379 64 124 227 84 254 120 75 108 278
TOTAL 76
```

31/76

29/76

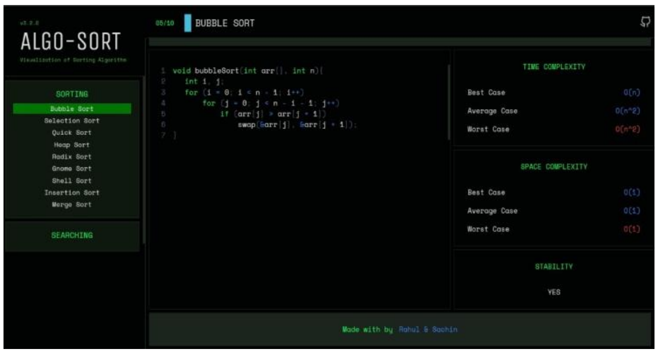
Figure10: Data Visualization

Next.Js: Next.js, a popular React framework, can be utilized effectively in implementing sorting algorithms. By leveraging Next.js's server-side rendering (SSR) capabilities, sorting algorithms can be seamlessly integrated into web applications, ensuring optimal performance and user experience.

Additionally, Next.js's dynamic routing feature enables efficient handling of sorting requests, allowing for real-time updates without the need for page reloads. Moreover, Next.js's built-in API routes facilitate communication between the frontend and backend, enabling the implementation of complex sorting logic while maintaining a clean and modular codebase. Overall, Next.js empowers developers to create robust and responsive sorting algorithms that enhance the functionality of web applications. Incorporating Next.js into sorting algorithm implementations offers a multifaceted approach to enhancing web applications.

Leveraging Next.js's efficient client-side navigation, sorting algorithms can seamlessly integrate into single-page applications (SPAs) Next.js's static site generation (SSG) capabilities enable pre-rendering of sorted data, ensuring swift load times and improved search engine optimization (SEO). Additionally, reducing the likelihood of runtime errors and enhancing code maintainability. By harnessing the power of Next.js, developers can create sophisticated sorting functionalities that elevate the overall performance and usability of web applications. Next.js serves as an ideal framework for implementing sorting algorithms due to its versatility and scalability. With Next.js's seamless integration of React components, developers can create dynamic sorting interfaces that respond instantly to user input, providing a smooth and intuitive sorting experience. Moreover, Next.js's support for server-side rendering (SSR) ensures that sorting algorithms perform efficiently even with large datasets, as sorting operations can be offloaded to cases.Result

Start by arranging the data, and then pick the visualization algorithm to use. Algorithm buttons provide sorting of data as it arrives on the interface. Asking to specify the ordering of elements takes precedence because when the algorithm has completed running the initialization process, the interface is now showing a new ordering, while the code has already completed running the initialization with the prior data set. There was considerable confusion caused by the way the ordering buttons and algorithm buttons were shown in the UI after the surveys were completed. When beginning the sorting process, the student noted that she was having problems starting because she believed that she was hitting the buttons in the wrong order. This then led to her failing to execute the animation. The answers can be found in Appendix D. Overall, my animation tool did not aid with the understanding of sorting algorithms. Among those who answered question 3, which questioned if their knowledge of a particular algorithm changed after using the tool, 5 of the 13 students (38%) stated that they had in some way altered their previous knowledge of the algorithm. Many thought the tool was a good concept, while the other 7 did not find it useful at all. It was said that one student stated a false positive about the instrument (whom I did not include in the 5 that said it was helpful).



```
void bubbleSort(int arr[], int n){
  int i, j;
  for (i = 0; i < n - 1; i++)
      for (j = 0; j < n - i - 1; j++)
          if (arr[j] > arr[j + 1])
              swap(&arr[j], &arr[j + 1]);
}
```

TIME COMPLEXITY

Best Case	O(n)
Average Case	O(n^2)
Worst Case	O(n^2)

SPACE COMPLEXITY

Best Case	O(1)
Average Case	O(1)
Worst Case	O(1)

STABILITY

YES

Figure11: Algorithm

8. Feedback

The assessment of the algorithm's memory constraints resulted in persistent impasses and system crashes. Ahead of the test, students were apprised of this issue, and three respondents shared their insights regarding the concern. Despite the 3.9 average rating received on question 4, evaluating the tool's usefulness left me pleasantly surprised. Moreover, features initially disregarded surfaced during my evaluation:

1. Introduction of adjustable animation speed for enhanced user control.
2. Incorporation of visual feedback when interacting with interface buttons to indicate selection status.
3. Implementation of a segmented visualization, akin to box-like splits, to amplify clarity during Merge Sort.

4. Utilization of color-coding to denote active comparisons within the visualization.

While most students expressed a desire for an earlier presentation of the animation tool, two students didn't voice such a request. They asserted that an earlier introduction could enhance topic accessibility. Notably, following my presentation, students encountered Merge Sort for the first time, which influenced 54% (7 out of 13) to incorporate it into their learning using the animation. Considering the author's suggestion, a phased approach to studying individual algorithms, rather than simultaneous mastery of all, might yield more effective learning outcomes. Interestingly, students were tasked with selecting the sorting algorithm for focused study—choosing between Selection Sort, Insertion Sort, or Merge/Insertion Sort, with Bubble Sort omitted as it's not part of the RIC curriculum. Despite its exclusion, presenting Bubble Sort alongside other algorithms evoked positive reactions from those encountering it for the first time. This observation underscores the viability of creating animations in elucidating concepts. Though memory limitations posed challenges, leveraging animation software enabled the introduction of innovative concepts.

Modern computational hardware is poorly suited for efficient implementation of sorting algorithms. For example, the CPUs and GPUs have dedicated instructions to add vectors of numbers, or compute maximum, but there is no instruction that sorts a vector of numbers, or finds permutation that would sort that vector.
If performance is critical, it is best to consider other algorithms

that perform better on modern hardware. For example, the universal statistics algorithm computes upper limits faster than an equivalent procedure implemented using quantiles. This is achieved by performing a few passes over the data using vector instructions.

Sorting algorithms are one of the fundamental topics in computer science, as they are used to organize data in a way that makes it easier to search, access, and manipulate. However, not all sorting algorithms are created equal, and choosing the right one can have a significant impact on your code performance. In this article, you will learn about some of the common sorting algorithms, their advantages and disadvantages, and how to measure their efficiency.

9. Limitations

Algorithm Coverage: Some sorting visualization projects might have limitations in terms of the number or variety of sorting algorithms depicted. Due to complexity or design constraints, only a subset of sorting algorithms might be showcased, potentially limiting the breadth of understanding for users.
Performance Issues: Visualization projects might face challenges in handling large datasets or executing complex animations efficiently. This can result in performance bottlenecks, sluggishness, or even system crashes, particularly when demonstrating algorithms with intensive computational requirements.
Visualization Clarity: The clarity of visual representations might be compromised, impacting user comprehension. Visual complexity, unclear animations, or insufficient labeling could hinder users' ability to grasp sorting concepts effectively.

Platform Dependency: Some sorting visualization projects might be designed for specific platforms or browsers, potentially excluding users who cannot access or interact with the visualization due to compatibility issues.

Addressing these limitations might involve improving algorithm coverage, optimizing performance, enhancing visualization clarity, increasing interactivity, ensuring cross-platform compatibility, enriching educational content, and prioritizing accessibility to cater to a broader user base effectively.

10. Benefits

Absolutely, here's a more detailed breakdown of the benefits of a Sorting Visualization project:

1. Enhanced Learning Experience: Visualizations offer a dynamic learning experience by presenting abstract concepts visually. Sorting Visualization projects provide a hands-on, visual representation of sorting algorithms, fostering a deeper understanding of how these algorithms function. By witnessing the step-by-step sorting process, learners gain a more comprehensive understanding compared to theoretical explanations alone.

2. Improved Comprehension: Visual representations aid in comprehension. Sorting Visualization projects enable users to observe the sorting process, allowing them to understand how data elements are manipulated during sorting. The visual cues make it easier to grasp algorithmic behaviors, such as comparisons, swaps, or shifting of elements, enhancing comprehension of algorithmic logic.

3. Engaging Educational Tool: The interactive and visually stimulating nature of sorting visualizations makes learning algorithms engaging. In academic settings or self-paced learning environments, these projects capture learners' attention, fostering active engagement and making the learning process enjoyable.

4. Interactive Exploration :Many Sorting Visualization projects allow users to interact with the animation. Features such as adjusting animation speed, pausing, or stepping through sorting steps empower users to control their learning pace. This interactive exploration encourages deeper engagement and a better understanding of sorting algorithms.

5. Demonstrating Algorithm Efficiency: Visualizations enable side-by-side comparisons of different sorting algorithms. Users can witness and compare the performance of algorithms, understanding their time complexity, efficiency, and suitability for different datasets. This comparative analysis provides insights into algorithmic strengths and weaknesses.

6. Facilitating Self-paced Learning: Visualizations accommodate diverse learning styles and speeds. Users can replay animations or steps as needed, facilitating self-directed learning. This flexibility enables learners to grasp concepts at their preferred pace, catering to individual learning preferences.

7. Real-world Application Understanding: By visualizing how sorting algorithms organize data, users gain practical insights into real-world applications. Understanding how these algorithms work is fundamental in computer science, data analysis, and software development. Sorting Visualization projects bridge the gap between theoretical understanding and practical application.

8. Error Identification and Debugging: Visualization tools offer a visual aid in identifying errors or inefficiencies within sorting algorithms. Users can observe and pinpoint potential flaws in algorithmic logic or implementation by visually tracking how data elements are sorted. This visual feedback helps in debugging and refining algorithms, enhancing the learning process by highlighting common pitfalls or misconceptions.

9. Cross-disciplinary Learning: Sorting Visualization projects transcend specific fields, allowing users from diverse backgrounds to comprehend and appreciate algorithms' significance. Students, professionals, or enthusiasts in various domains, including mathematics, engineering, or business analytics, can benefit from understanding sorting algorithms through visual representations. This cross-disciplinary

applicability fosters a deeper appreciation for algorithmic concepts and their wide-ranging relevance across disciplines.

These additional benefits underscore how Sorting Visualization projects facilitate not only a deeper understanding of sorting algorithms but also aid in error identification and have broad applicability across different disciplines.

In essence, Sorting Visualization projects leverage visual representations to make learning algorithms more engaging, accessible, and insightful, catering to a broad spectrum of learners. Certainly,

11. Applications

1. Algorithmic Complexity Demonstration: Sorting Visualization projects offer a practical means to demonstrate the complexities of various sorting algorithms. Users can visually witness how different algorithms perform under varying data sizes, showcasing their time complexities (like O(n^2), O(n log n), etc.) and helping learners understand their computational efficiency.

2. Algorithmic Evolution and Historical Context: These projects provide insights into the evolution and historical context of sorting algorithms. Users can observe the progression from basic algorithms like Bubble Sort to more sophisticated ones like Quick Sort or Merge Sort, fostering an appreciation for the iterative development of algorithms over time.

3. User Behavior Studies and Human-Computer Interaction (HCI): Sorting Visualization tools can be used for HCI studies to analyze how users interact with visual interfaces. Researchers observe user behaviors, preferences, and comprehension levels while navigating the visualization, aiding in understanding user interaction patterns and optimizing user interfaces for educational tools.

4. Visual Analytics and Pattern Recognition: Sorting Visualizations facilitate visual analytics, allowing users to recognize patterns within sorting processes. This capability extends to identifying common algorithmic patterns and strategies, fostering pattern recognition skills beneficial in algorithm design and analysis.

5. Accessibility and Inclusivity Considerations: Apart from functionality, these projects address accessibility standards and inclusivity by incorporating features catering to diverse user needs. This includes considerations for colorblindness, screen readers, keyboard navigation, and other accessibility features, ensuring usability for all users.

6. Ethical Considerations in Educational Technologies: The development of Sorting Visualization projects involves ethical considerations surrounding educational technologies. This includes considerations for data privacy, ensuring age-appropriate content, and adhering to ethical guidelines in educational software development.

7. Global Educational Impact: These projects have the potential for global educational impact by providing access to quality educational resources worldwide. They transcend geographical barriers, offering a shared learning experience to students and enthusiasts across the globe.

12. Conclusion

These aspects showcase the diverse dimensions beyond the direct benefits of Sorting Visualization projects, encompassing fields like algorithmic complexity, historical context, HCI, accessibility, cognitive science, ethics, and global educational outreach.

The web-based animation tool showcasing various sorting algorithms has been a culmination of substantial time and effort invested. Despite its memory-intensive nature, the feedback from students who engaged with the tool has predominantly been positive.

This echoes findings from previous research, suggesting no substantial discrepancy in learning outcomes through conventional methods versus animated presentations. Undoubtedly, there's a consensus that investigating and implementing animated tools holds significant promise for enhancing classroom education.remains addressing these memory difficulties, seen as the imminent challenge on the project's horizon.

Plans include implementing Merge/Insertion Sort, an algorithm integrating aspects of both Merge Sort and Insertion Sort. Additionally, the goal is to finalize the integration of Quick Sort, where the code readiness is established. The ultimate aspiration is to release the online tool to the public, accentuated by the desire to introduce a crucial feature—public accessibility.

However, this step might pose challenges due to the current availability of the tool strictly in a local environment. The hurdle lies in enabling the application to serve multiple concurrent requests from various users accessing the tool online. To achieve this, optimizing the code for efficiency becomes paramount while contemplating strategies to handle simultaneous usage by numerous individuals. Such a development would pave the way for a potential comparison study, allowing insights into the tool's effectiveness and user experiences across diverse user interactions. Regarding the Sorting Visualization project, a parallel narrative emerges. Similar to the sorting algorithm animation tool, considerable dedication and effort have been devoted to this project. Despite memory concerns, positive feedback has been received from engaged users, aligning with prior research suggesting the efficacy of visualization in educational settings. There's a shared consensus on the potential of animated presentations to augment classroom learning.

The project roadmap includes enhancing the visualization tool by integrating Merge/Insertion Sort and finalizing the integration of Quick Sort due to existing code readiness. Both projects aim to transition to public availability, yet challenges persist, particularly in enabling concurrent access by multiple users. Resolving memory limitations is a priority, with a focus on optimizing code efficiency to accommodate simultaneous usage. The successful achievement of these objectives would potentially facilitate comparative studies, providing valuable insights into the tool's effectiveness across varying user scenarios.

The ultimate aspiration remains the release of the Sorting Visualization tool to the public domain, accentuated by the critical feature of accessibility to a wider audience.

However, transitioning from a locally available tool to a publicly accessible online resource presents significant challenges. The primary hurdle is optimizing the tool's functionality to cater to concurrent requests from multiple users. Striving for code efficiency and contemplating strategies to manage simultaneous user interactions becomes paramount in overcoming these challenges. In the pursuit of making these projects accessible to a broader audience, the overarching goal is not solely the public release but also to conduct comprehensive studies. These studies could potentially analyze user experiences, effectiveness, and educational impact, thus contributing valuable insights into the efficacy of such educational tools.

The journey ahead involves meticulous optimization, strategic planning, and innovative problem-solving, all in service of creating impactful and accessible educational resources.

References

[1] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, D. L.; STEIN, C. Introduction to algorithms. Second Edition. 2001. ISBN 0-262-03293-7.

[2] Java documentation. Available from: hhttps://docs.oracle.com/javase/8/i.

[3] KNUTH, D. The Art of Computer Programming: Fundamental Algorithms. Third Edition. 2004. ISBN 0-201-89683-4

[4] SIPSER, M. Introduction to the Theory of Computation. Boston, MA: PWS Publishing Company, 1997. ISBN 0-534-94728-X.

[5] GeeksforGeeks. Available from: hhttps://www.geeksforgeeks.org/i.

[6] BĚLOHLÁVEK, R. Algoritmická matematika 1 : část 1. Available also from: hhttp://belohlavek.inf.upol.cz/vyuka/algoritmicka-matematika-1-1.pdfi.

[7] Stackoverflow. Available from: hhttps://stackoverflow.com/i.

[8] T. Bingmann. “The Sound of Sorting - ‘Audibilization’ and Visualization of Sorting Algorithms.” Panthemanet Weblog. Impressum, 22 May 2013. Web. 29 Mar. 2017.

[9] Bubble-sort with Hungarian (“Cs´ang´o”) Folk Dance. Dir. K´atai Zolt´an and T´oth L´aszl´o. YouTube. Sapientia University, 29 Mar. 2011. Web. 29 Mar.2017.

[10] A. Kerren and J. T. Stasko. (2002) Chapter 1 Algorithm Animation. In: Diehl S.(eds) Software Visualization. Lecture Notes in Computer Science, vol 2269. Springer, Berlin, Heidelberg.

[11] A. Moreno, E. Sutinen, R. Bednarik, and N. Myller. Conflictive animations as engaging learning tools. Proceedings of the Koli Calling ’07 Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88, Koli ‘07 (Koli National Park, Finland), pages 203-206.

[12] J. Stasko. Using Student-built Algorithm Animations As Learning Aids. Proceedings of the Twentyeighth SIGCSE Technical Symposium on Computer Science Education. SIGCSE ‘97 (San Jose, California), pages 25-29. http://doi.acm.org/10.1145/268084.268091

[13] J. Stasko, A. Badre, and C. Lewis. Do Algorithm Animations Assist Learning?: An Empirical Study and Analysis. Proceedings of the INTERACT ‘93 and CHI ‘93 Conference on Human Factors in Computing Systems, CHI-93 (Amsterdam, the Netherlands), pages 61- 6.Jeff Schneider, Cross Validation, Feb 7, 1997,

H. Danielsiek, W. Paul and J. Vahrenhold, Detecting and understanding students’ misconceptions related to algorithms and data structures, 43rd ACM technical symposium on Computer Science Education (2012), 21-26

[14] K. Booten, Harvesting ReRites, Johnston, DJ ReRites: Human+ AI Poetry+ Raw Output+ Responses. Montréal, QC: Anteism., (2019)

[15] G. Rößling, M. Schüler and B. Freisleben, The ANIMAL algorithm animation tool, 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education (2000), 37-40.

[16] W. C. Pierson and S. H. Rodger, Web-based animation of data structures using JAWAA, ACM SIGCSE Bulletin 30(1) (1998), 267-271.

[17] A. Moreno and M. S. Joy, Jeliot in a demanding educational setting, Electronic Notes in Theoretical Computer Science 178 (2007), 51-59.

[18] T. L. Naps, Jhavé: Supporting algorithm visualization, IEEE Computer Graphics and Applications 25(5) (2005), 49-55.