Sorting Visualizer

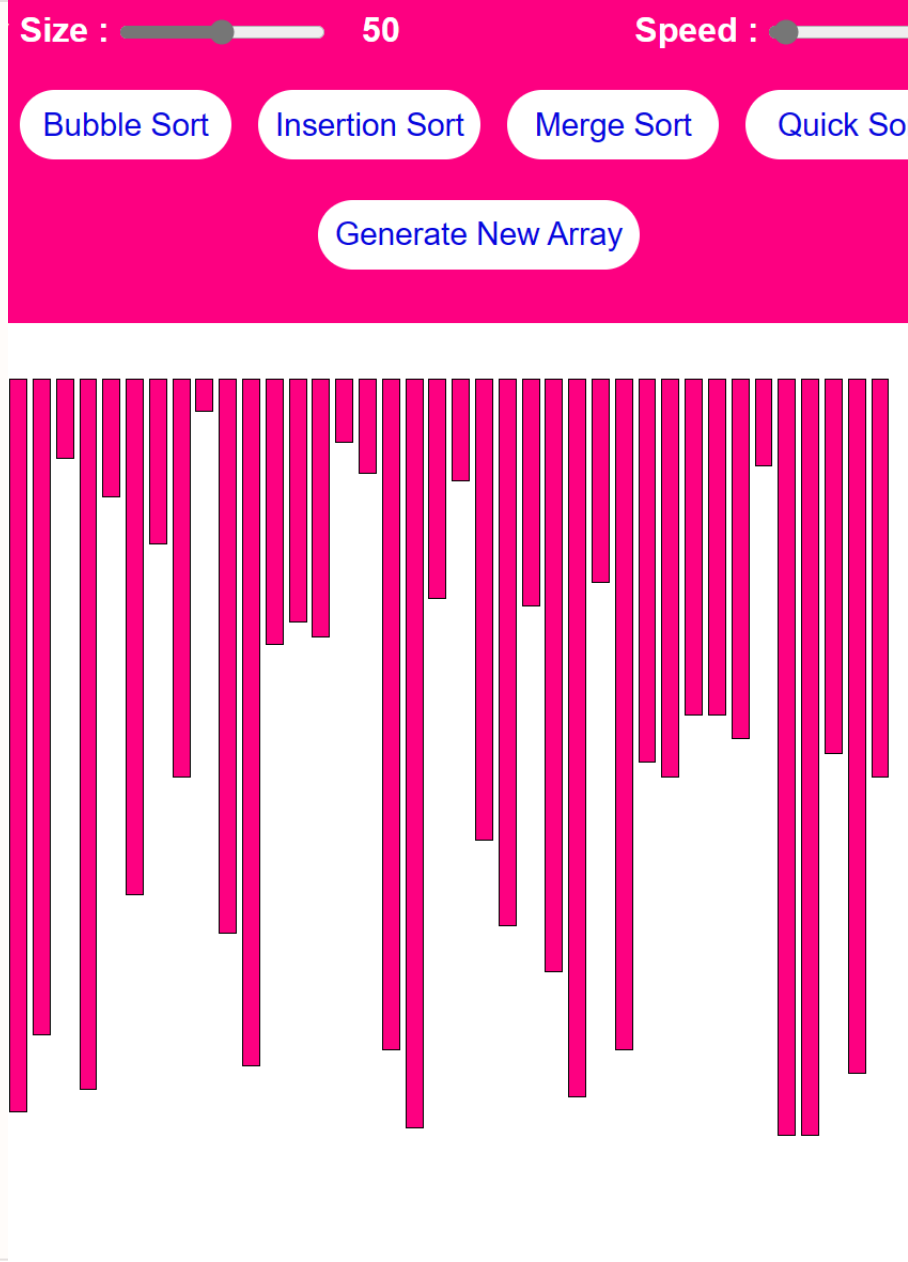
Project By-

Manas Tiwari (211165)

Manav Gautam(211140)

Introduction

This Sorting Visualizer is a website allows you to see how different sorting algorithms work in a visual way. It provides a graphical representation of how elements are rearranged and sorted, helping users understand the inner workings of sorting algorithms more intuitively. Sorting algorithms are fundamental to computer science and play a crucial role in various applications where data needs to be organized in a specific order. However, understanding how these algorithms work and comparing their efficiency can be challenging when dealing with large data sets or complex algorithms.



How Sorting Visualizer Works

Input Array

The Sorting Visualizer takes an input array of random numbers.

Sorting Algorithms

The input array is sorted using different sorting algorithms.

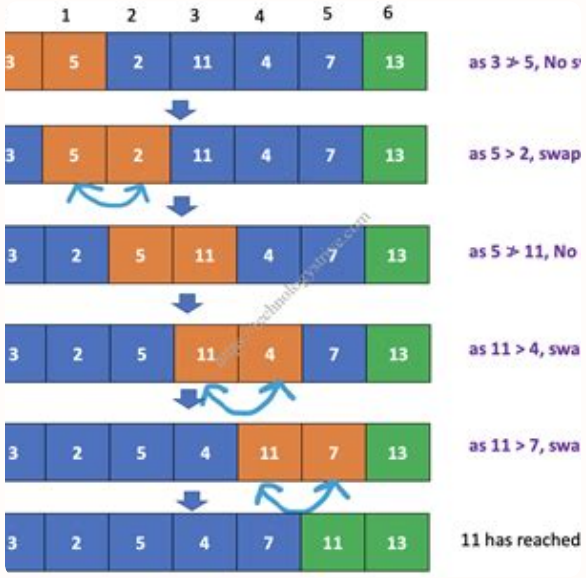
Real-Time Visualization

As the algorithms sort the array, each step is visualized in real-time.

Sorted Output Array

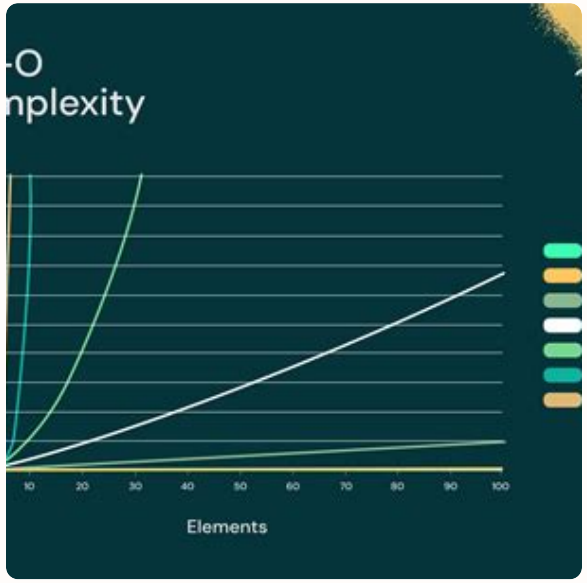
The end result is a sorted output array.

Bubble Sort



Description

Bubble sort compares adjacent elements and swaps them if they are not in the correct order.



Time and Space Complexity

Time complexity is O(n^2) while space complexity is O(1).

Source Code:

```
async function BubbleSort() {
    let delay = Disable_The_Input();
    let container = document.getElementById("container");

    for (let i = 0; i < bars.length - 1; i++) {
        let has_swap = false;
        for (let j = 0; j < bars.length - i - 1; j++) {
            let curr_id = bars[j].split('id="')[1].split('"')[0];
            let nxt_ele = bars[j + 1].split('id="')[1].split('"')[0];

            document.getElementById(curr_id).style.backgroundColor = selected;
            let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
            beep(100, sound, delay)
            document.getElementById(nxt_ele).style.backgroundColor = chng;
            await Sleep(delay / 2);
            let a = parseInt(bars[j].split(/[:%]/)[1]);
            let b = parseInt(bars[j + 1].split(/[:%]/)[1]);
            if (a > b) {
                has_swap = true;

                let t = bars[j];
                bars[j] = bars[j + 1];
                bars[j + 1] = t;

                container.innerHTML = bars.join('');
            }
            document.getElementById(curr_id).style.backgroundColor = selected;
            document.getElementById(nxt_ele).style.backgroundColor = chng;
            await Sleep(delay / 2.0);
            document.getElementById(curr_id).style.backgroundColor = def;
            document.getElementById(nxt_ele).style.backgroundColor = def;
        }
        if (has_swap == false) break;
    }
    Finished_Sorting();
}
```

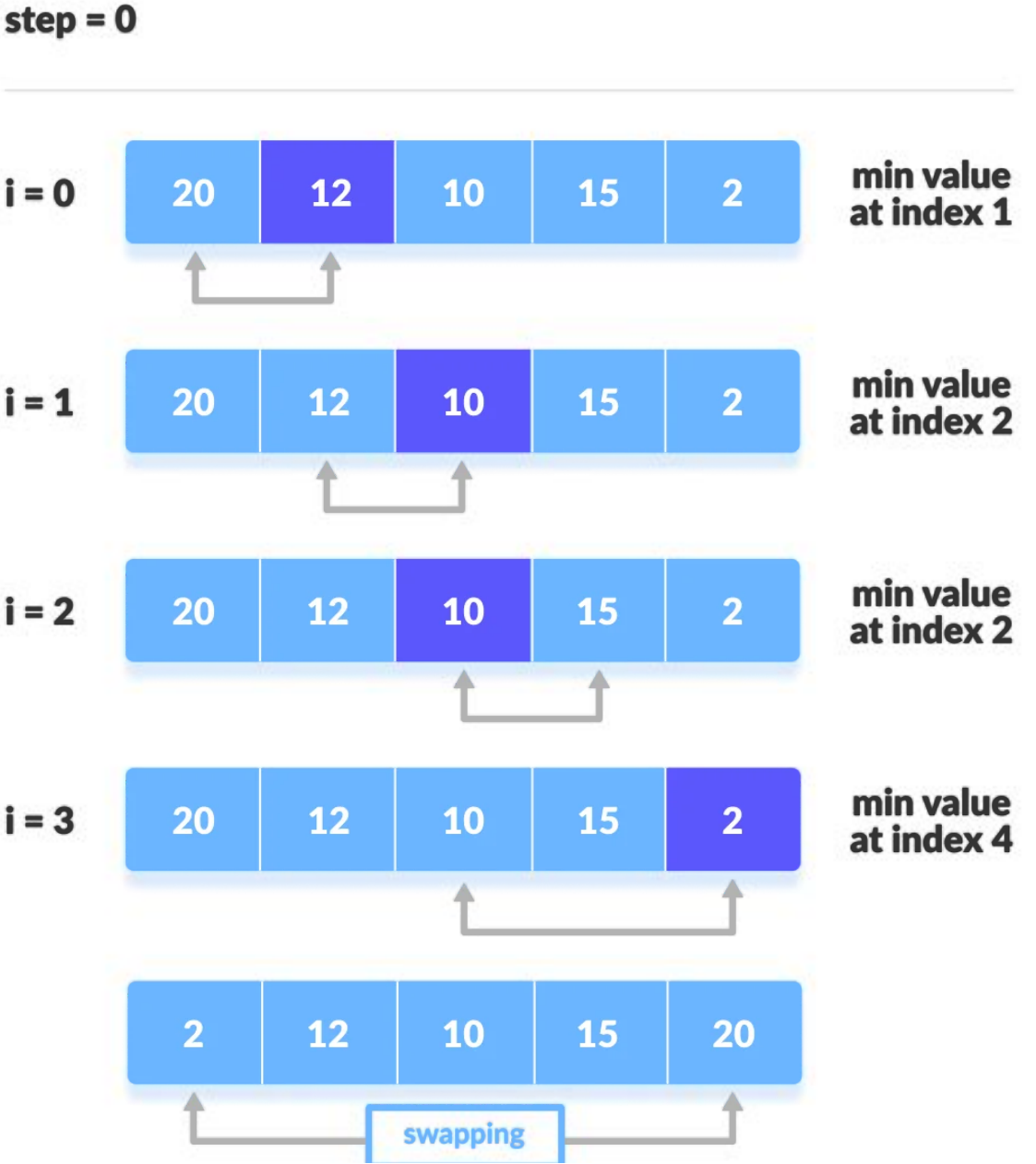
Selection Sort

1 Description

Selection Sort selects the smallest element from unsorted part, swaps it with the first element, and repeats the process with the remaining unsorted part.

2 Time and Space Complexity

Time complexity is O(n^2) while space complexity is O(1).



Source Code:

```
async function SelectionSort() {
    let delay = Disable_The_Input();

    let container = document.getElementById("container");
    for (let i = 0; i < bars.length; i++) {
        let mn_ind = i;
        let curr_id = bars[i].split('id="')[1].split('"')[0];
        document.getElementById(curr_id).style.backgroundColor = selected;
        let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
        beep(100, sound, delay)
        for (let j = i + 1; j < bars.length; j++) {
            let nxt_ele = bars[j].split('id="')[1].split('"')[0];
            document.getElementById(nxt_ele).style.backgroundColor = chng;
            let a = parseInt(bars[mn_ind].split(/[:%]/)[1]);
            let b = parseInt(bars[j].split(/[:%]/)[1]);
            if (a > b) mn_ind = j;
            await Sleep(delay / 5.0);
            document.getElementById(nxt_ele).style.backgroundColor = def;
        }

        let nxt_ele = bars[mn_ind].split('id="')[1].split('"')[0];
        document.getElementById(nxt_ele).style.backgroundColor = selected;
        await Sleep(2 * delay / 5.0);

        let tmp = bars[mn_ind];
        bars[mn_ind] = bars[i];
        bars[i] = tmp;

        container.innerHTML = bars.join('');
        await Sleep(2 * delay / 5.0);
        document.getElementById(curr_id).style.backgroundColor = def;
        document.getElementById(nxt_ele).style.backgroundColor = def;
    }
    Finished_Sorting();
}
```

Insertion Sort

1

Description

Insertion Sort picks an element and inserts it in its correct position in the already sorted part by shifting larger elements to the right.

2

Time Complexity

Time complexity is O(n^2) in the worst case and O(n) in the best case.

3

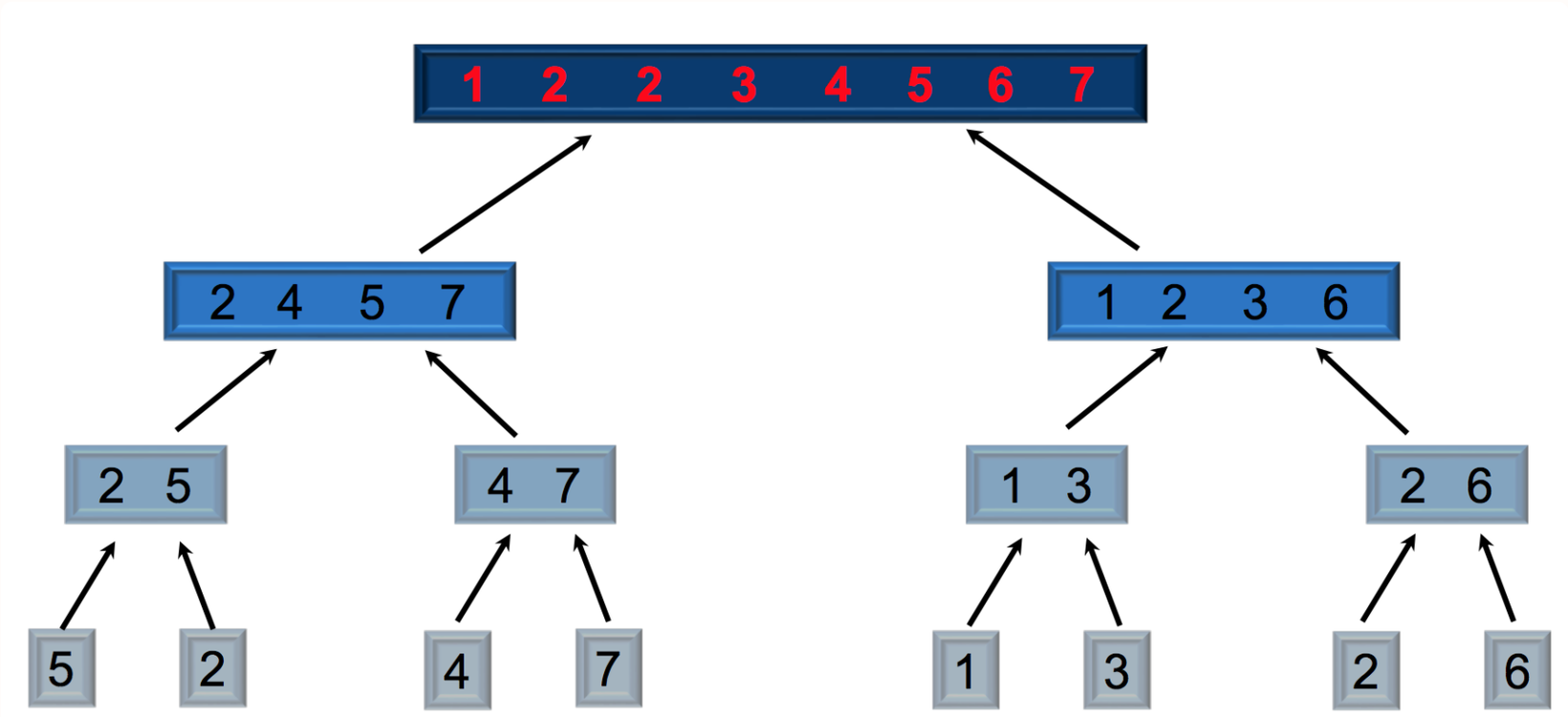
Space Complexity

Space complexity is O(1).

Source Code:

```
async function InsertionSort() {
    let delay = Disable_The_Input();
    let container = document.getElementById("container");
    for (let i = 1; i < bars.length; i++) {
        let j = i - 1;
        let key = bars[i];
        let curr_id = key.split('id="')[1].split('"')[0];
        let nxt_ele = bars[j].split('id="')[1].split('"')[0];
        document.getElementById(curr_id).style.backgroundColor = selected;
        let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
        beep(100, sound, delay)
        while (j >= 0 && parseInt(bars[j].split(/[:%]/)[1]) > parseInt(key.split(/[:%]/)[1])) {
            document.getElementById(nxt_ele).style.backgroundColor = def;
            nxt_ele = bars[j].split('id="')[1].split('"')[0];
            document.getElementById(nxt_ele).style.backgroundColor = chng;
            await Sleep(delay);
            bars[j + 1] = bars[j];
            j--;
        }

        bars[j + 1] = key;
        container.innerHTML = bars.join('');
        document.getElementById(curr_id).style.backgroundColor = selected;
        document.getElementById(nxt_ele).style.backgroundColor = chng;
        await Sleep(delay * 3.0 / 5);
        document.getElementById(curr_id).style.backgroundColor = def;
        document.getElementById(nxt_ele).style.backgroundColor = def;
    }
    Finished_Sorting();
}
```

Merge Sort



Description

Merge Sort divides the input array in half, recursively sorts each half, and then merges the sorted halves back together.

Time Complexity

Time complexity is O(n log n).

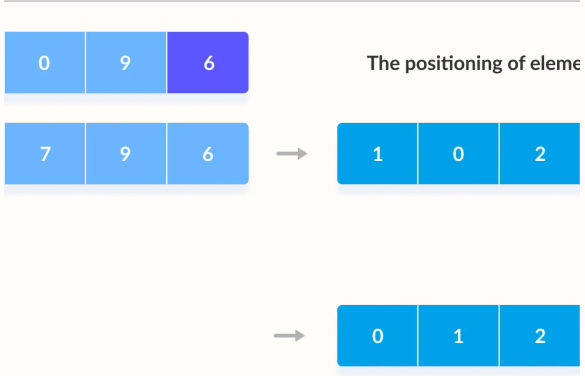
Space Complexity

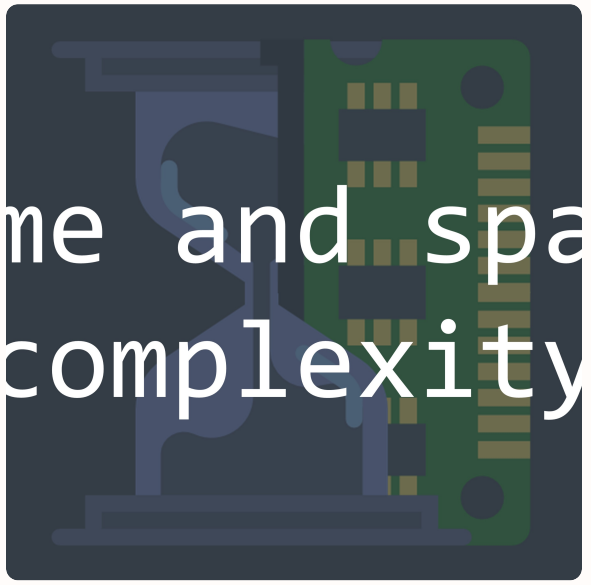
Space complexity is O(n).

Source Code:

```
function Slide_down(l, r) {
    let temp = bars[r];
    for (let i = r - 1; i >= l; i--) {
        bars[i + 1] = bars[i];
    }
    bars[l] = temp;
}


async function merge(l, m, r, d) {
    let y = l;
    let i = l;
    let j = m + 1;

    while (i < j && j <= r) {
        let curr_id = bars[j].split('id="')[1].split('"')[0];
        let nxt_ele = bars[i].split('id="')[1].split('"')[0];
        document.getElementById(curr_id).style.backgroundColor = selected;
        document.getElementById(nxt_ele).style.backgroundColor = chng;
        let a = parseInt(bars[j].split(/[:%]/)[1]);
        let b = parseInt(bars[i].split(/[:%]/)[1]);

        if (a > b) i++;
        else {
            Slide_down(i, j);
            i++; j++;
        }
        await Sleep(d / 2.0);
        container.innerHTML = bars.join('');
        document.getElementById(curr_id).style.backgroundColor = selected;
        document.getElementById(nxt_ele).style.backgroundColor = chng;
        let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
        beep(100, sound, d)
        await Sleep(d / 2.0);
        document.getElementById(curr_id).style.backgroundColor = def;
        document.getElementById(nxt_ele).style.backgroundColor = def;
        sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
        beep(100, sound, d)
    }
}
```

Quick Sort





Description

Quick Sort selects a pivot element and partitions the array around it by placing smaller elements to its left and larger elements to its right.

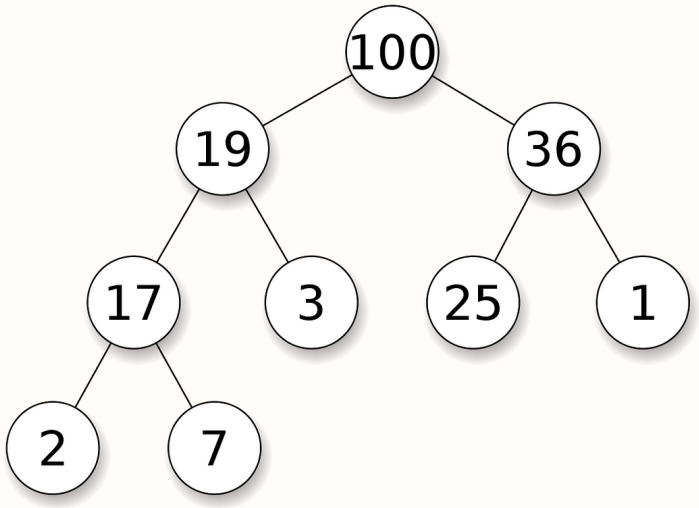
Time and Space Complexity

Time complexity is O(n log n) average case and O(n^2) worst case, while space complexity is O(log n) in the best case and O(n) in the worst case.

Source Code:

```
async function Partition(l, r, d) {
    let i = l - 1;
    let j = l;
    let id = bars[r].split('id="')[1].split('"')[0];
    document.getElementById(id).style.backgroundColor = selected;
    for (j = l; j < r; j++) {
        let a = parseInt(bars[j].split(/[:%]/)[1]);
        let b = parseInt(bars[r].split(/[:%]/)[1]);
        if (a < b) {
            i++;
            let curr_id = bars[i].split('id="')[1].split('"')[0];
            let nxt_ele = bars[j].split('id="')[1].split('"')[0];
            document.getElementById(curr_id).style.backgroundColor = chng;
            document.getElementById(nxt_ele).style.backgroundColor = chng;

            let temp = bars[i];
            bars[i] = bars[j];
            bars[j] = temp;

            await Sleep(d / 3.0);
            container.innerHTML = bars.join('');
            document.getElementById(curr_id).style.backgroundColor = chng;
            document.getElementById(nxt_ele).style.backgroundColor = chng;
            document.getElementById(id).style.backgroundColor = selected;
            let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
            beep(100, sound, d)
            await Sleep(d / 3.0)
            document.getElementById(curr_id).style.backgroundColor = def;
            document.getElementById(nxt_ele).style.backgroundColor = def;
        }
    }

    let temp = bars[i + 1];
    bars[i + 1] = bars[r];
    bars[r] = temp;

    container.innerHTML = bars.join(' ');
    document.getElementById(id).style.backgroundColor = selected;
    await Sleep(d / 3.0);
    document.getElementById(id).style.backgroundColor = def;
```

Heap Sort: A Fast In-Place Sorting Algorithm



Description

Heap Sort is a comparison-based algorithm that uses a binary heap data structure to sort elements. It operates in place and has a time complexity of O(nlogn).

Time and Space Complexity

Heap Sort has a time complexity of O(nlogn) and a space complexity of O(1), making it an efficient algorithm for large data sets.

Source Code:

```
async function Heapfiy(n, i, d) {
    let largest = i;
    let l = 2 * i + 1; // lft
    let r = 2 * i + 2; // rgt
    let curr_id = bars[i].split('id="')[1].split('"')[0];
    let nxt_ele;
    let id3;

    document.getElementById(curr_id).style.backgroundColor = selected;
    if (r < n) {
        id3 = bars[r].split('id="')[1].split('"')[0];
        document.getElementById(id3).style.backgroundColor = chng;
    }
    if (l < n) {
        nxt_ele = bars[l].split('id="')[1].split('"')[0];
        document.getElementById(nxt_ele).style.backgroundColor = chng;
    }
    await Sleep(d / 3.0)
    if (l < n && parseInt(bars[l].split(/[:%]/)[1]) > parseInt(bars[largest].split(/[:%]/)[1]))
        largest = l;
    if (r < n && parseInt(bars[r].split(/[:%]/)[1]) > parseInt(bars[largest].split(/[:%]/)[1]))
        largest = r;

    if (largest != i) {
        let t = bars[i]; bars[i] = bars[largest]; bars[largest] = t;
        container.innerHTML = bars.join(' ');
        document.getElementById(curr_id).style.backgroundColor = selected;
        let sound = MapRange(document.getElementById(curr_id).style.height.split('%')[0], 2, 100, 500, 1000);
        beep(100, sound, d)
        if (r < n) document.getElementById(id3).style.backgroundColor = chng;
        if (l < n) document.getElementById(nxt_ele).style.backgroundColor = chng;
        await Sleep(d / 3.0)
        container.innerHTML = bars.join(' ');
        await Heapfiy(n, largest, d);
    }
    container.innerHTML = bars.join(' ');
}
```

Why Sorting Visualizer is Useful

1

Hands-On Learning

Sorting Visualizer provides a visual way to learn how sorting algorithms work.

2

Develop Critical Thinking

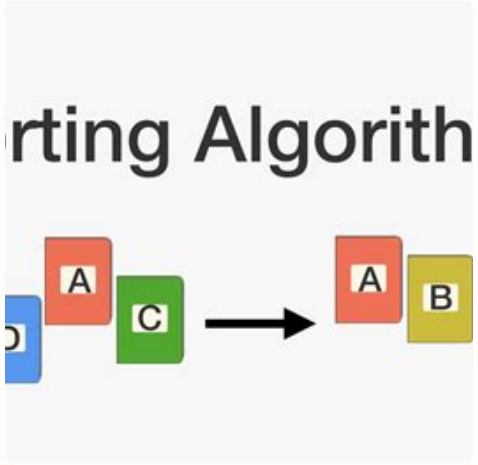
By observing the behavior of different algorithms, users can develop critical thinking and logical reasoning skills.

3

Helps to Understand Algorithm Complexity

Sorting Visualizer makes it easy to understand the time and space complexities of different sorting algorithms.

Future Inclusions Possible



Additional Sorting Algorithms

There is potential to add more sorting algorithms to the tool.

Additional Features

New features that could be added include more customizable settings and more detailed visualizations.

Conclusion

In conclusion, the sorting visualizer presented today provides an interactive and visual representation of various sorting algorithms. By using this tool, users can gain a deeper understanding of how different sorting techniques work and their respective performance characteristics.

Throughout the presentation, we explored popular sorting algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. We discussed their working principles, advantages, disadvantages, and analyzed their time and space complexity. By understanding these algorithms we have come broadened our horizons and gained insight allowing us to have a better understanding of which sorting technique to use based on specific requirements and constraints.