JavaScript

🚀 Master Sorting Algorithms: Simplify Complexity!

Bubble Sort

Bubble Sort repeatedly compares adjacent elements and swaps them if they’re in the wrong order. It's simple but not the fastest. Think of it as the "lazy river" of sorting algorithms.

⚙️ **Time Complexity:** $O(n^2)$
✨ **Stability:** Stable
📈 **Best Use Case:** When simplicity matters more than speed.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Bubble Sort

```
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 (arr[j] > arr[j + 1]) {
                // Swap elements
                [arr[j], arr[j + 1]] =
                    [arr[j + 1], arr[j]];
            }
        }
    }
    return arr;
}

console.log(bubbleSort([5, 3, 8, 4, 2]));
// [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Selection Sort

Selection Sort finds the smallest element in the array and places it in its correct position. It’s like picking the smallest rock from a pile and lining them up one by one.

⚙️ **Time Complexity:** $O(n^2)$
✨ **Stability:** Not Stable
📈 **Best Use Case:** When memory usage needs to be minimal.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Selection Sort

```javascript
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;
            }
        }
        // Swap the found minimum element with the first element
        [arr[i], arr[minIndex]] =
            [arr[minIndex], arr[i]];
    }
    return arr;
}

console.log(selectionSort([5, 3, 8, 4, 2])); // [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Insertion Sort

Insertion Sort builds the sorted list one element at a time by placing each element in its correct position. It’s like sorting a deck of cards by hand.

⚙️ **Time Complexity:** $O(n^2)$
✨ **Stability:** Stable
📈 **Best Use Case:** Small datasets or when data is nearly sorted.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Insertion Sort

```
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;
    }
    return arr;
}

console.log(insertionSort([5, 3, 8, 4, 2]));
// [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Merge Sort

Merge Sort splits the array into halves, sorts each half, and then merges them back together. It’s like conquering a big task by breaking it into smaller, more manageable tasks.

⚙️ **Time Complexity:** O(n log n)
✨ **Stability:** Stable
📈 **Best Use Case:** Sorting large datasets efficiently.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Merge Sort

```
function mergeSort(arr) {
    if (arr.length <= 1) return arr;

    const mid = Math.floor(arr.length / 2);
    const left = mergeSort(arr.slice(0, mid));
    const right = mergeSort(arr.slice(mid));

    return merge(left, right);
}

function merge(left, right) {
    const result = [];
    while (left.length && right.length) {
        if (left[0] < right[0]) {
            result.push(left.shift());
        } else {
            result.push(right.shift());
        }
    }
    return [...result, ...left, ...right];
}

console.log(mergeSort([5, 3, 8, 4, 2]));
// [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Quick Sort

Quick Sort selects a pivot element and partitions the array into elements smaller and larger than the pivot. It’s like dividing a room into smaller groups and sorting each group individually.

⚙️ **Time Complexity:** O(n log n) (average) / O(n^2) (worst)
✨ **Stability:** Not Stable
📈 **Best Use Case:** When you need speed and don’t mind extra space.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Quick Sort

```
function quickSort(arr) {
    if (arr.length <= 1) return arr;

    const pivot = arr[arr.length - 1];
    const left = [];
    const right = [];

    for (let i = 0; i < arr.length - 1; i++) {
        if (arr[i] < pivot) {
            left.push(arr[i]);
        } else {
            right.push(arr[i]);
        }
    }

    return [...quickSort(left), pivot, ...quickSort(right)];
}
```

```
console.log(quickSort([5, 3, 8, 4, 2]));
// [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Heap Sort

Heap Sort organizes the array into a heap structure (a binary tree) and repeatedly extracts the root to sort the array. It’s like building a pyramid and removing blocks one by one.

⚙️ **Time Complexity:** O(n log n)
✨ **Stability:** Not Stable
📈 **Best Use Case:** When you need consistent performance and low memory usage.

Haroon Iqbal
@haroon-iqbal-dev

Implementation of Heap Sort

```
function heapSort(arr) {
    const 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) {
            [arr[i], arr[largest]] = [arr[largest], arr[i]];
            heapify(arr, n, largest);
        }
    };
    let n = arr.length;
    for (let i = Math.floor(n / 2) - 1; i >= 0; i--) heapify(arr, n, i);
    for (let i = n - 1; i > 0; i--) {
        [arr[0], arr[i]] = [arr[i], arr[0]];
        heapify(arr, i, 0);
    }
    return arr;}
console.log(heapSort([5, 3, 8, 4, 2]));
// [2, 3, 4, 5, 8]
```

Haroon Iqbal
@haroon-iqbal-dev

Did you find it use Useful?

Leave a comment

Haroon Iqbal

@haroon-iqbal-dev