Tree Sort

Introduction

Tree Sort is a sorting algorithm that builds a binary search tree from the elements of the array and then traverses the tree in-order to retrieve the sorted elements.

Logic

1. Insert each element of the array into a binary search tree.
2. Perform an in-order traversal of the binary search tree to retrieve the sorted elements.

Implementation

- **TreeNode:** Represents a node in the binary search tree.
- **insert(root, value):** Inserts a value into the binary search tree.
- **inOrderTraversal(root, result):** Performs an in-order traversal of the tree, collecting the sorted values.
- **treeSort(arr):** Initializes an empty tree, inserts each element from the array into the tree, and retrieves the sorted values through in-order traversal.

JavaScript implementation of Tree Sort: Tree Sort.

Complexity

- **Time Complexity:**
  - Best Case: O(n log n) (balanced tree)
  - Average Case: O(n log n)
  - Worst Case: O(n^2) (skewed tree)
- **Space Complexity:**
  - O(n) (for the binary search tree)

Advantages

- Can be efficient for partially sorted lists.
- Stable sort (maintains the relative order of equal elements).

Considerations

- The time complexity depends on the shape of the binary search tree.
- May have suboptimal performance for already sorted or reverse-sorted lists.

Disadvantages/Limitations

- The worst-case time complexity is O(n^2) for a skewed tree.
- Requires additional space for the tree structure.

Edge Cases

- Performance can vary based on the initial order of the list and the construction of the binary search tree.

External Links

To delve further into Tree Sort, check out these resources: - Tree Sort - Wikipedia - Sorting Algorithms: Tree Sort - GeeksforGeeks - Tree Sort - Brilliant.org