Circular Buffer Splay Tree Sort Complexity Proof

Shyamal Suhana Chandra

1 Theorem: Circular Buffer Splay Tree Sort Complexity

Statement: Sorting a Circular Buffer Splay Tree with n nodes takes $O(n)$ time for both ascending and descending order, regardless of comparison mode (lexicographic, numeric, or semantic).

2 Proof

The sort operation performs an in-order traversal of the tree:

1. Visit all nodes: $O(n)$
2. Comparison per node: $O(1)$ (for all modes)
3. Build result vector: $O(n)$

2.1 In-Order Traversal

In-order traversal visits each node exactly once:

$$T(n) = T(k) + T(n - k - 1) + O(1)$$

where k is the size of the left subtree.
Solving the recurrence:

$$T(n) = O(n)$$

2.2 Comparison Modes

2.2.1 Lexicographic Mode

- String comparison: $O(\min(|a|, |b|))$ where $|a|, |b|$ are string lengths
- For fixed-size keys: $O(1)$
- For variable-size keys: $O(k)$ where k is average key length
- Total: $O(n \cdot k)$ where k is key length

2.2.2 Numeric Mode

- Numeric comparison: $O(1)$
- Total: $O(n)$

2.2.3 Semantic Mode

- Custom comparison: Depends on comparator
- Assuming $O(1)$ comparator: $O(n)$
- With $O(k)$ comparator: $O(n \cdot k)$

2.3 Sort Order

2.3.1 Ascending Order

Traverse: left $\to$ node $\to$ right

$$\text{Time} = O(n)$$

2.3.2 Descending Order

Traverse: right $\to$ node $\to$ left

$$\text{Time} = O(n)$$

The order only affects traversal direction, not complexity.

2.4 Total Complexity

For fixed-size keys or $O(1)$ comparators:

$$\begin{aligned}
T(n) &= \text{Traversal} + \text{Comparisons} + \text{Result building} && (1)\\
&= O(n) + O(n) + O(n) && (2)\\
&= O(n) && (3)
\end{aligned}$$

For variable-size keys with lexicographic comparison:

$$T(n) = O(n \cdot k)$$

where k is the average key length.

Conclusion: Circular Buffer Splay Tree sort has $O(n)$ time complexity for fixed-size keys, and $O(n \cdot k)$ for variable-size keys where k is the key length.

3 Space Complexity

The sort operation requires:

- Result vector: $O(n)$
- Recursion stack: $O(h) = O(\log n)$ average, $O(n)$ worst case
- Total: $O(n)$

4 Comparison with Other Sorting Methods

- **Heap Sort**: $O(n \log n)$ - slower but in-place
- **Quick Sort**: $O(n \log n)$ average - faster but not stable
- **CBS Tree Sort**: $O(n)$ - fastest for already-built tree

The advantage of CBS Tree Sort is that the tree is already maintained, so sorting is just a traversal operation.