CS 590A: Algorithms  
Spring 2024: Homework 4 - Binary Search Tree and Red-Black Tree

|  |  |
| --- | --- |
| n | Time |
| 10000 | 2 |
| 25000 | 9 |
| 50000 | 16 |
| 75000 | 21 |
| 100000 | 35 |

|  |  |
| --- | --- |
| n | Duplicates |
| 10000 | 0 |
| 25000 | 0 |
| 50000 | 1 |
| 75000 | 2 |
| 100000 | 3 |

|  |  |
| --- | --- |
| n | Time |
| 10000 | 172 |
| 25000 | 1064 |
| 50000 | 4519 |
| 75000 | 9604 |
| 100000 | 17066 |

|  |  |
| --- | --- |
| n | Duplicates |
| 10000 | 0 |
| 25000 | 0 |
| 50000 | 0 |
| 75000 | 0 |
| 100000 | 0 |

|  |  |
| --- | --- |
| n | Time |
| 10000 | 185 |
| 25000 | 1178 |
| 50000 | 4622 |
| 75000 | 11575 |
| 100000 | 18495 |

|  |  |
| --- | --- |
| n | Duplicates |
| 10000 | 0 |
| 25000 | 0 |
| 50000 | 0 |
| 75000 | 0 |
| 100000 | 0 |

|  |  |
| --- | --- |
| n | Time |
| 50000 | 22 |
| 100000 | 38 |
| 250000 | 143 |
| 500000 | 391 |
| 1000000 | 844 |
| 2500000 | 2620 |
| 5000000 | 5828 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | Duplicates | Case 1 | Case 2 | Case 3 | Left rotate | Right rotate |
| 50000 | 3 | 25680 | 9739 | 19473 | 14598 | 14614 |
| 100000 | 4 | 51422 | 19493 | 38858 | 29135 | 29216 |
| 250000 | 11 | 128315 | 48640 | 97483 | 73117 | 73006 |
| 500000 | 65 | 256581 | 96676 | 193858 | 145195 | 145339 |
| 1000000 | 226 | 513424 | 194231 | 388077 | 291462 | 290846 |
| 2500000 | 1456 | 1281960 | 485475 | 969903 | 727358 | 728020 |
| 5000000 | 5783 | 2564380 | 971091 | 1940886 | 1455497 | 1456480 |

|  |  |
| --- | --- |
| n | Time |
| 50000 | 12 |
| 100000 | 26 |
| 250000 | 92 |
| 500000 | 164 |
| 1000000 | 349 |
| 2500000 | 1068 |
| 5000000 | 1696 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | Duplicates | Case 1 | Case 2 | Case 3 | Left rotate | Right rotate |
| 50000 | 0 | 49966 | 0 | 49971 | 0 | 49971 |
| 100000 | 0 | 99964 | 0 | 99969 | 0 | 99969 |
| 250000 | 0 | 249961 | 0 | 249967 | 0 | 249967 |
| 500000 | 0 | 499959 | 0 | 499965 | 0 | 499965 |
| 1000000 | 0 | 999957 | 0 | 999963 | 0 | 999963 |
| 2500000 | 0 | 2499952 | 0 | 2499960 | 0 | 2499960 |
| 5000000 | 0 | 4999950 | 0 | 4999958 | 0 | 4999958 |

|  |  |
| --- | --- |
| n | Time |
| 50000 | 15 |
| 100000 | 36 |
| 250000 | 90 |
| 500000 | 180 |
| 1000000 | 395 |
| 2500000 | 1052 |
| 5000000 | 1820 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| n | Duplicates | Case 1 | Case 2 | Case 3 | Left rotate | Right rotate |
| 50000 | 0 | 49966 | 0 | 49971 | 49971 | 0 |
| 100000 | 0 | 99964 | 0 | 99969 | 99969 | 0 |
| 250000 | 0 | 249961 | 0 | 249967 | 249967 | 0 |
| 500000 | 0 | 499959 | 0 | 499965 | 499965 | 0 |
| 1000000 | 0 | 999957 | 0 | 999963 | 999963 | 0 |
| 2500000 | 0 | 2499952 | 0 | 2499960 | 2499960 | 0 |
| 5000000 | 0 | 4999950 | 0 | 4999958 | 4999958 | 0 |

Randomly Sorted Arrays:

Both BST and RBT exhibit similar performance trends, where the time taken increases as the number of elements (n) increases. This is because both structures require traversal and adjustment operations that scale with the size of the tree. Despite RBT being balanced, it still requires additional operations compared to BST due to the maintenance of color properties.

Ascending and Descending Orders:

For BST, ascending and descending orders represent the worst-case scenarios. In both cases, the tree becomes unbalanced, leading to an inefficient in-order tree walk operation with a time complexity of O(n). This inefficiency arises because the tree essentially degenerates into a linked list.

In contrast, RBT maintains balance even in ascending and descending orders, ensuring that the in-order traversal operation remains efficient with a time complexity of O(log n). This is because RBT automatically balances itself during insertions, mitigating the issues of imbalance seen in BST.

Handling Duplicates:

Both BST and RBT show no duplicates in ascending and descending order scenarios. This is because duplicates are inherently eliminated in these orderings.

RBT Specific Observations:

In RBT, there are no values for Case 2 in ascending and descending orders. This is because Case 2 in RBT involves the uncle of the newly inserted node being red, which typically occurs in scenarios where the tree is not balanced. In ascending and descending orders, RBT remains balanced, thus avoiding Case 2.

Additionally, in RBT ascending order, there is no value for left rotate, and in descending order, there is no value for right rotate. This indicates that the rotations required to maintain balance predominantly involve one direction, either left or right, depending on the order of insertion.