|  |  |  |  |
| --- | --- | --- | --- |
|  | BinarySearchTree | AVLTree | Lazy AVLTree |
| (Pre removal) |  |  |  |
| Avg Depth | 10.3274 | 7.37522 | 7.37522 |
| Avg Depth/log2(n) | 1.12966 | 0.806731 | 0.806731 |
| Recursive Insert Calls | 10686 | 8368 | 8368 |
| Recursive Remove Calls | 4389 | 3817 | 3377 |
| Recursive Contains Calls | 4212 | 3377 | 3377 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | BinarySearchTree | AVLTree | Lazy AVLTree |
| Avg Depth | 9.75987 | 6.47368 | 7.37522 |
| Avg Depth/log2(n) | 1.18331 | 0.784886 | 0.806731 |
| Recursive Contains Calls | 4376 | 3874 | 3377 |

Caption: What we can grasp from this table is essentially what we know regarding the complexity. In a worst and average case scenario, AVLTree is O(log n), while BinarySearchTree is O(log n) in an average case, but O(n) in it’s worst case. We see this due to the recursive calls being higher for BinarySearchTree than AVLTree than BinarySearchTree. LazyAVLTree has the same computation, with the remove call being less complicated due to simply using a flag, which allows it essentially to replicate the same demands as Contain (and the same code as well).

However, the avg depth for the Lazy AVLTree doesn’t change since it doesn’t actually delete nodes, which leads to AVLTree becoming smaller, and overall easier to traverse because of it using it’s more complicated remove function.