Assuming n is odd

1. Θ((n-1)/2). By the provided algorithm, the “smaller than middle” keys will always be inserted as a left node; and “larger than middle” keys will always be inserted as a right node. In other words, the left and right subtree of the root node will each be a degenerate-shaped tree.

2. O((n-1)/2) or O(h) if h == height of the tree. The new key may be inserted as a child of any node other than the root node; we may have to traverse the entire height of the tree and insert the key at bottom level.

3. Θ(n) time, Θ(n) memory. Use a recursive algorithm for the right subtree, when reached the bottom of the tree, print the value of each node as we go up the recursive stack. Then print the value of the root node, finally go down the left subtree, print each value of the node as we go down. The bottom of left subtree is the last value to print. In total, we traversed the left subtree 1 time and right subtree 2 times, resulting a time complexity of Θ(1.5n) which simplifies to Θ(n); the entire right subtree is stored in recursive call stack when we reached the bottom, resulting in a space complexity of Θ(0.5n) which simplifies to Θ(n).