1-O(b) - The loop iterates b times, so the runtime is directly proportional to b.

2-O(log b) - This uses a recursive divide and conquer approach, halving the problem size each call. So the runtime is logarithmic in b.

3-O(1) - The runtime is constant, not dependent on the inputs.

4-O(log a) - This uses a basic binary search approach, doubling the guess each iteration. So the runtime is logarithmic in a.

5-O(log n) - This also uses a binary search approach, halving the search space each call. So the runtime is logarithmic in n.

6-O(sqrt(n)) - The loop iterates up to the square root of n times. So the runtime is directly proportional to the square root of n.

7-O(n) - In the worst case, the tree is completely unbalanced, resembling a linked list. So we would have to traverse all n nodes.

8-O(n) - Again in the worst case, we would have to traverse all n nodes. Without order, we have no way to prune the search space.

9-O(n) - The runtime is directly proportional to the length of the array, n, since we have to copy each element.

10-O(log n) - This uses repeated division by 10 to isolate the digits, so the runtime is logarithmic in n.