7. Reverse Integer
We will reverse our integer one digit by one digit until it's out of the bound. Or we will return the reversed number.

```
class Solution:
    def reverse(self, x: int) -> int:
        rev = 0
        if x > 0:
            mark = 1
        else:
            mark = -1
            x = -x

        while x:
            rev = rev * 10 + x % 10
            x = x // 10
            if mark * rev > 2 ** 31 - 1 or mark * rev < - 2 ** 31:
                return 0
        return mark * rev
```

138. Copy List with Random Pointer
We will copy every node first and store them in a node map, which associated new node with original node. TC is O(n), SC is O(n)

```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        node_map = {}
        node_map[None] = None
        head_mem = head
        while head:
            node = Node(head.val, None, None)
            node_map[head] = node
            head = head.next
        head = head_mem
        while head:
            node_map[head].next = node_map[head.next]
            node_map[head].random = node_map[head.random]
            head = head.next
        return node_map[head_mem]
```

69. Sqrt(x)
For this question, we will use binary search to get the result. TC is O(logx), SC is O(1)

```
class Solution:
    def mySqrt(self, x: int) -> int:
        left, right = 0, x
        while left < right:
            mid = (left + right) // 2
            if mid ** 2 == x:
                return mid
            elif mid ** 2 < x:
                left = mid + 1
            else:
                right = mid
        return left if left ** 2 <= x else left - 1
```

230. Kth Smallest Element in a BST
We will use in-order traversal to traverse all nodes in the tree. And we also subtract k by 1 every time we traverse a node. Until k == 0, we will return that number.

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */
/**
 * @param {TreeNode} root
 * @param {number} k
 * @return {number}
 */
let g_k;
var kthSmallest = function(root, k) {
    if (k == 0) {
        return root.val;
    }
    g_k = k;
    return traverse(root);
};

const traverse = (node) => {
    if (!node) {
        return null;
    }

    if (node.left) {
        const tmp = traverse(node.left);
        if (tmp) {
            return tmp;
        }
    }
    g_k -= 1;
    if (g_k === 0) {
        return node.val;
    }
    if (node.right) {
        const tmp = traverse(node.right);
        if (tmp) {
            return tmp;
        }
    }
    return null;
}
```

230. Kth Smallest Element in a BST

We could traverse our tree iteratively. TC is O(k), SC is O(k)

```
var kthSmallest = function(root, k) {
    const stack = [];
    let node = root;
    while (stack.length > 0 || node) {
        while (node) {
            stack.push(node);
            node = node.left;
        }
        node = stack.pop()
        k -= 1;
        if (k === 0) {
            return node.val;
        }
        node = node.right;
    }

};
```