958. Check Completeness of a Binary Tree

```
class Solution:
    def isCompleteTree(self, root: TreeNode) -> bool:
        if not root:
            return True
        stack, i = [root], 0
        while stack[i]:
            stack.append(stack[i].left)
            stack.append(stack[i].right)
            i += 1
        return not any(stack[i + 1:])
```

29. Divide Two Integers

```
class Solution:
    def divide(self, dividend: int, divisor: int) -> int:
        mark = 1
        result = 0
        if dividend * divisor < 0:
            mark = -1
        dividend = abs(dividend)
        divisor = abs(divisor)
        while dividend - divisor >= 0:
            s = divisor
            i = 1
            while (s << 1) < dividend:
                s = s << 1
                i = i << 1
            result += i
            dividend -= s
        result = result * mark
        if result > 0x7FFFFFFF:
            return 0x7FFFFFFF
        return result
```

Dfs:

```
class Solution:
    def __init__(self):
        self.height = 0
        self.leastheight = False

    def isCompleteTree(self, root: TreeNode, h=0) -> bool:
        if not root:
            if self.height == 0:
```

```
            self.height = h
        elif h == self.height - 1 and not self.leastheight:
            self.leastheight = True
            self.height = h
        else:
            return h == self.height
        return True
    return self.isCompleteTree(root.left, h + 1) and self.isCompleteTree(root.right, h + 1)
```

270. Closest Binary Search Tree Value

```
class Solution:
    def closestValue(self, root: TreeNode, target: float) -> int:
        self.largest_diff = abs(root.val - target)
        self.cur_closest = root.val
        def traverse(node):
            if not node:
                return
            if abs(node.val - target) < self.largest_diff:
                self.largest_diff = abs(node.val - target)
                self.cur_closest = node.val

            if target < node.val:
                traverse(node.left)
            else:
                traverse(node.right)
        if root.val < target:
            traverse(root.right)
        else:
            traverse(root.left)
        return self.cur_closest
```

13. Roman to Integer

```
class Solution:
    def romanToInt(self, s: str) -> int:
        dic = {"I":1, "V":5, "X":10, "L":50, "C":100, "D":500, "M":1000}
        result = 0
        for i in range(len(s) - 1):
            if dic[s[i]] < dic[s[i + 1]]:
                result -= dic[s[i]]
            else:
                result += dic[s[i]]
```

```
result += dic[s[-1]]
return result
```