222. Count Complete Tree Nodes
O(logn * logn)

```
class Solution:
    def getHeight(self, node):
        return self.getHeight(node.left) + 1 if node else 0

    def countNodes(self, root: TreeNode) -> int:
        if root:
            h = self.getHeight(root)
            if h == self.getHeight(root.right) + 1:
                return self.countNodes(root.right) + (1 << (h - 1))
            else:
                return self.countNodes(root.left) + (1 << (h - 2))
        else:
            return 0
```

416. Partition Equal Subset Sum
O(n**2)

```
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        total = sum(nums)
        if total % 2 == 1:
            return False
        total = total // 2
        length = len(nums)
        memo = [[False for j in range(total + 1)] for i in range(length + 1)]
        memo[0][0] = True
        for i in range(1, length + 1):
            memo[i][0] = True
        for j in range(1, total + 1):
            memo[0][j] = False
        for i in range(1, length + 1):
            for j in range(1, total + 1):
                memo[i][j] = memo[i - 1][j]
                if j - nums[i - 1] >= 0:
                    memo[i][j] = memo[i][j] or memo[i - 1][j - nums[i - 1]]
            if memo[i][total]:
                return True
        return memo[length][total]
```

Space saving version:

```
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        total = sum(nums)
```

```
        if total % 2 == 1:
            return False
        total = total // 2
        length = len(nums)
        memo = [False for _ in range(total + 1)]
        memo[0] = True
        for num in nums:
            for i in range(total, -1, -1):
                if i >= num:
                    memo[i] = memo[i] or memo[i - num]
            if memo[total]:
                return True
        return memo[total]
```

445. Add Two Numbers II
O(n), using two stack.

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
        stack1, stack2 = [], []
        carry = 0
        head = ListNode(0)
        while l1:
            stack1.append(l1.val)
            l1 = l1.next

        while l2:
            stack2.append(l2.val)
            l2 = l2.next

        while stack1 and stack2:
            carry, rest = divmod(stack1.pop() + stack2.pop() + carry, 10)
            temp = head.next
            head.next = ListNode(rest)
            head.next.next = temp
        stack1 = stack1 or stack2
        while stack1:
            carry, rest = divmod(stack1.pop() + carry, 10)
```

```
            temp = head.next
            head.next = ListNode(rest)
            head.next.next = temp
        if carry > 0:
            temp = head.next
            head.next = ListNode(carry)
            head.next.next = temp
        return head.next
```

173. Binary Search Tree Iterator

```
class BSTIterator:

    def __init__(self, root: TreeNode):
        self.stack = []
        while root:
            self.stack.append(root)
            root = root.left

    def next(self) -> int:
        """
        @return the next smallest number
        """
        node = self.stack.pop()
        ret = node.val
        if node.right:
            node = node.right
            while node:
                self.stack.append(node)
                node = node.left
        return ret

    def hasNext(self) -> bool:
        """
        @return whether we have a next smallest number
        """
        return len(self.stack) > 0
```