450. Delete Node in a BST
We will replace our node with smallest node on the right branch and replace them then recursively delete the node we replaced. TC is O(logn), SC is O(logn)

```
class Solution:
    def deleteNode(self, root: TreeNode, key: int) -> TreeNode:
        if not root:
            return root
        if root.val < key:
            root.right = self.deleteNode(root.right, key)
        elif root.val > key:
            root.left = self.deleteNode(root.left, key)
        else:
            if not root.right:
                return root.left

            node = self.findMin(root.right)
            root.val = node.val
            root.right = self.deleteNode(root.right, node.val)
        return root

    def findMin(self, node):
        if not node:
            return None
        while node.left:
            node = node.left
        return node
```

1. Two Sum
We will use hashmap to record previous number and once there is another matched pair, we will return true, or we will return false. TC is O(n), SC is O(n)

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        memo = {}
        for i, num in enumerate(nums):
            if target - num in memo:
                return [memo[target - num], i]
            else:
                memo[num] = i
```

560. Subarray Sum Equals K
We could use pre-sum to solve this question. TC is O(n), SC is O(n)

```
from collections import defaultdict
class Solution:
```

```
    def subarraySum(self, nums: List[int], k: int) -> int:
        pre_sum = defaultdict(int)
        pre_sum[0] = 1
        cur_sum = 0
        result = 0
        for num in nums:
            cur_sum += num
            result += pre_sum[cur_sum - k]
            pre_sum[cur_sum] += 1

        return result
```

218. The Skyline Problem

We will use minheap to get maximum height cord and compare with the current node. If not equal, there will be a point. TC is O(N^2), SC is O(n)

```
from heapq import *
class Buildpoint:
    def __init__(self, start, height, enter):
        self.start = start
        self.hi = height
        self.enter = enter

    def __lt__(self, other):
        if self.start == other.start:
            if self.enter == 0 and other.enter == 0:
                return self.hi > other.hi
            elif self.enter == 1 and other.enter == 1:
                return self.hi < other.hi
            else:
                if self.enter == 0 and other.enter == 1:
                    return True
                else:
                    return False
        else:
            return self.start < other.start

    def __eq__(self, other):
        return self.start == other.start and self.hi == other.hi and self.enter == other.enter



class Solution:
    def getSkyline(self, buildings: List[List[int]]) -> List[List[int]]:
        nodes = []
```

```
        height = []
        result = []
        for l, r, h in buildings:
            nodes.append(Buildpoint(l, h, 0))
            nodes.append(Buildpoint(r, h, 1))

        nodes.sort()
        for node in nodes:
            if node.enter == 0:
                if not height or node.hi > -height[0]:
                    result.append([node.start, node.hi])
                heappush(height, -node.hi)
            else:
                height.remove(-node.hi)
                heapify(height)
                if not height:
                    result.append([node.start, 0])
                elif node.hi > -height[0]:
                    result.append([node.start, -height[0]])


        return result
```

2. Add Two Numbers

```
# 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:
        carry = 0
        dummy = ListNode(0)
        head = dummy
        while l1 and l2:
            cur, mod = divmod(l1.val + l2.val + carry, 10)
            carry = cur
            head.next = ListNode(mod)
            head = head.next
            l1 = l1.next
            l2 = l2.next
```

```
while l1:
    cur, mod = divmod(l1.val + carry, 10)
    carry = cur
    head.next = ListNode(mod)
    head = head.next
    l1 = l1.next
while l2:
    cur, mod = divmod(l2.val + carry, 10)
    carry = cur
    head.next = ListNode(mod)
    head = head.next
    l2 = l2.next
if carry:
     head.next = ListNode(carry)
return dummy.next
```