108. Convert Sorted Array to Binary Search Tree

We will use recursion to construct this BST. TC is O(n).

```
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
        if not nums:
          return None
        length = len(nums)
        if length == 1:
          return TreeNode(nums[0])
        left_nums = nums[:length // 2]
        right_nums = nums[length // 2 + 1:]
        root = TreeNode(nums[length // 2])
        root.left = self.sortedArrayToBST(left_nums)
        root.right = self.sortedArrayToBST(right_nums)
        return root
```

501. Find Mode in Binary Search Tree

Using inorder traversal. TC is O(n), SC is O(1)

```
class Solution:
    def findMode(self, root: TreeNode) -> List[int]:
        self.count = 0
        self.result = []
        self.max = 0
        self.prev = None
        if not root:
          return self.result
        def inorderTraverse(node):
          if not node:
            return
          inorderTraverse(node.left)
          if node.val == self.prev or self.prev == None:
            self.count += 1
          else:
            if self.count > self.max:
              self.max = self.count
              self.result = [self.prev]
            elif self.count == self.max:
              self.max = self.count
              self.result.append(self.prev)
            self.count = 1
          self.prev = node.val
          inorderTraverse(node.right)
        inorderTraverse(root)
        if self.count > self.max:
          self.max = self.count
          self.result = [self.prev]
        elif self.count == self.max:
          self.max = self.count
          self.result.append(self.prev)
        return self.result
```

239. Sliding Window Maximum

We will use a deque to manage our number, if len(deque) == k, we will popleft(), if the pop number is max, we will find a new cur_max from deque, then we will compare the new number with cur_max, if it’s larger or equal to cur_max, we will clear deque, and let cur_max = num, and append num to memo. TC is O(kn)

```
from collections import deque
class Solution:
    def maxSlidingWindow(self, nums: List[int], k: int) -> List[int]:
        if not nums:
          return []
        if k == 1:
          return nums
        memo = deque(nums[:k])
        cur_max = max(memo)
        result = [cur_max]
        for num in nums[:k]:
          if num >= cur_max:
            cur_max = num
            memo.clear()
          memo.append(num)
        for num in nums[k:]:
          if len(memo) == k:
            temp = memo.popleft()
            if temp == cur_max:
              cur_max = max(memo)
          if num >= cur_max:
            cur_max = num
            memo.clear()
          memo.append(num)
          result.append(cur_max)
        return result
```

4. Maximum Subarray

We will accumulate all numbers from left to right, so every sub sum from j - 1 to i is nums[i] - nums[j], so we only need to maintain our cur_min and cur_max in every iteration. In the end, we will return cur_max

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        cur_min = min(nums[0], 0)
        cur_max = nums[0]
        for i in range(1, len(nums)):
          nums[i] += nums[i - 1]
          cur_max = max(cur_max, nums[i] - cur_min)
          cur_min = min(cur_min, nums[i])
        return cur_max
```

5. Maximum Product Subarray

We will always remember cur_min and cur_max. And in each iteration, we will do result = (result, cur_max * num, cur_min * num). TC is O(n)

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        cur_min = min(1, nums[0])
        cur_max = max(1, nums[0])
        result = nums[0]
        for num in nums[1:]:
          num1, num2 = num * cur_min, num * cur_max
          result = max(result, num1, num2)
          cur_min = min(1, num1, num2)
          cur_max = max(1, num1, num2)
        return result
```