70. Min Cost Climbing Stairs

Because we could only climb one or two steps, we will let current cost memorize min(n-1, n-2) + current, then in the end, we will get the minimum of cost[-1], cost[-2]. TC is O(n), SC is O(1)

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        for i in range(2, len(cost)):
          cost[i] = min(cost[i - 1], cost[i - 2]) + cost[i]
        return min(cost[-1], cost[-2])
```

303. Range Sum Query - Immutable

Use an array to store all sum of all previous number, then range number is memo[j + 1] - memo[i]. TC is O(n), O(1), SC is O(n)

```
class NumArray:

    def __init__(self, nums: List[int]):
        self.memo = [0]
        cur_sum = 0
        for i in nums:
          cur_sum += i
          self.memo.append(cur_sum)


    def sumRange(self, i: int, j: int) -> int:
        return self.memo[j + 1] - self.memo[i]
```

53. Maximum Subarray

We will use the greedy algorithm to get our current maximum and minimum of previous accumulating value. In the end, cur_max is what we want. TC is O(n), SC is O(1)

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        cur_max, cur_min = nums[0], min(0, 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
```

121. Best Time to Buy and Sell Stock

This question is similar to the above one. Basically, we only need to memorize the minimum number of the previous ones. Then always memorize the largest one by computing cur_price - cur_min. TC is O(n), SC is O(1)

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices:
          return 0
        cur_min = prices[0]
        cur_max = 0
        for price in prices[1:]:
          cur_max = max(price - cur_min, cur_max)
          cur_min = min(price, cur_min)
        return cur_max
```

198. House Robber
We could use greedy algo to solve this question. We will record the previous 3 positions' maximum sum, cur_max = max(max1, max2) + num, then reassign these three variables and repeat these processes. In the end, we could get what we want. TC is O(n), SC is O(1)

```
class Solution:
    def rob(self, nums: List[int]) -> int:
      if not nums:
        return 0
      if len(nums) < 3:
        return max(nums)

      max1, max2, max3 = 0, nums[0], nums[1]
      for num in nums[2:]:
        max1, max2, max3 = max2, max3, max(max1, max2) + num
      return max(max3, max2)
```