53. Maximum Subarray

Maximum Subarray

```
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        maxSub=nums[0]
        curSum=0

        for i in nums:
            if curSum<0:
                curSum=0
            curSum+=i

            maxSub=max(maxSub,curSum)
        return maxSub
```

70. Climbing Stairs

Climbing Stairs

```
## RC ##
        ## APPROACH 1 : RECURSION ##
        # must do approach to understand concept.
        # Recursion has :
        # 1. base condition (to exit loop) i.e i>n => return 0  or i==n return 1
        # 2. recursive call : climb(i+1,n) + climb(i+2,n)

        ## APPROACH 2 : DP ##
        #   1. top can be reached from (N-1)th Step or (N-2)th Step i.e ===> dp[N-1] + dp[N-
2]
        #   2. base case :                                              No of ways
        #   0 steps     ===>    0 step                                      0
        #   1 steps     ===>    1 step                                      1
        #   2 steps     ===>    (1 + 1 steps) or (2 steps)                  2

        #   FINDING DP PATTERN
        #   3 steps     ===>    (1+1+1) (1+2) (2+1) (3)                     3
        #   4 steps     ===>    (1+1+1+1) (1+2+1) (2+1+1) (1+1+2) (2+2)     5   (pattern foun
d n-1 + n-2 )

        if(n==1): return 1
        dp = [0] * (n+1)
        dp[1] = 1
        dp[2] = 2
        for i in range(3,n+1):
            dp[i] = dp[i-1] + dp[i-2]
        return dp[-1]                           # last position will have solution.
```

121. Best Time to Buy and Sell Stock

Best Time to Buy and Sell Stock

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices:
            return 0

        profit=0
        buy_stock=prices[0]

        for i in range(len(prices)):
            # update the buy_stock if there's
            # any smaller value is present in the list

            if buy_stock>prices[i]:
                buy_stock=prices[i]

            # Now we've buy_stock.
            # Calculate the max of price diff and profit made till now

            profit=max((prices[i]-buy_stock,profit))

        return profit
```

392. Is Subsequence

Is Subsequence

```
class Solution:
    def isSubsequence(self, s: str, t: str) -> bool:
        i=0
        j=0
        while(j<len(t) and i<len(s)):
            if(s[i] == t[j]):
                i+=1
                j+=1
            else:
                j+=1
        if(i==len(s)):
            return True
        return False
```

746. Min Cost Climbing Stairs

Min Cost Climbing Stairs

```
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:

        # ==== 0. check input
        if not cost:
            return 0

        # ==== 1. create an array to memorize the results, and think about how you are going
to
        # use/define it.
        # Here, I want this array to help me to memorize the "min cost" at the i-th step
        dp = [0] * len(cost)

        # ==== 2. Next, think about what are your first 3 cases (in most of cases) until you
find the
        # pattern, which means "what do your dp[0], dp[1], dp[2]... look like?"
        # Let's start from dp[0]. My dp[0] would equal to cost[0] because I have no choice.
        dp[0] = cost[0]

        # My dp[1] would equal to cost[1]. I've wrote down "dp[1] = min(cost[0] + cost[1], co
st[1])",
        # but I found it is nonsense because in this problem, taking 2 costs will always high
er than taking
        # 1 cost, which means cost[1] will always smaller than cost[0]+cost[1]. I mention thi
s because I
        # want to let you know that you probably understand more about the relationships and
the
        # problem itself when you are solving it.
        if len(cost) >= 2:
            dp[1] = cost[1]

        # Next, I try to write down my dp[2]. It was like:
        # dp[2] = cost[2] + min(dp[0], dp[1])
        # We found the pattern!!!!!!!!!!!!!
        # dp[i] would be "cost[i] + min(dp[i-2], dp[i-1])". The cost at the stairs plus the m
in of previous
        # one and two stairs (you could only come from the previous two stairs, and let's pic
k up the min
        # one.)
        # ==== 3. Once you found the pattern, let loop to help you!
        # We start from 2 because we already know the dp[0] and dp[1]. Also, the truth is: we
are not
        # able to caculate dp[0] and dp[1]. However, from dp[2], we can caculate the results.
        for i in range(2, len(cost)):
            dp[i] = cost[i] + min(dp[i-1], dp[i-2])

        # ==== 4. Think again what you are trying to find in your dp array.
        # To finish the stairs journey in this problem, there are 2 ways to be the last step
before we finish
        # the stairs. The last step might come from both last two stairs. So, we want to know
the min of
        # the costs of last 2 stairs from our dp array.
        return min(dp[-1], dp[-2])
```

1025. Divisor Game

Divisor Game

```
class Solution:
    def divisorGame(self, N: int) -> bool:
        if N%2==0:
            return True
        else:
            return False
```