LEETCODE BLIND -75

1) TWO SUM (1)

Example 1:

Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
Example 2:

Input: nums = [3,2,4], target = 6
Output: [1,2]
Example 3:

Input: nums = [3,3], target = 6
Output: [0,1]

Soln:

```
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dict={}
        for i,n in enumerate(nums):
            diff=target-n
            if diff in dict:
                return [dict[diff],i]
            dict[n]=i
```

2) **Best Time to Buy and Sell Stock(121)**

Example 1:
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.
Example 2:
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

```
    def maxProfit(self, prices: List[int]) -> int:
        mini=prices[0]
        maxi=0
        for price in prices:
            mini=min(mini,price)
            maxi=max(maxi,price-mini)
        return maxi
```

3) Contains Duplicate

Example 1:
Input: nums = [1,2,3,1]
Output: true
Example 2:
Input: nums = [1,2,3,4]
Output: false
Example 3:
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true

Soln :

```
    def containsDuplicate(self, nums: List[int]) -> bool:
        s={}
        for i in nums:
            if i in s:
                return True
            else:
                s[i]=1
        return False
```

4) Product of Array Except Self

Example 1:

Input: nums = [1,2,3,4]
Output: [24,12,8,6]
Example 2:

Input: nums = [-1,1,0,-3,3]
Output: [0,0,9,0,0]

Solution:

```
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        prefix=1
        res=[1]*len(nums)
        for i in range(len(nums)):
            res[i]=prefix
            prefix=prefix*nums[i]
        postfix=1
        for i in range(len(nums)-1,-1,-1):
            res[i]*=postfix
            postfix*=nums[i]
        return res
```

5) Maximum Subarray

Example 1:
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.
Example 2:
Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.

Example 3:
Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

Soln

```
    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
```

6) Maximum Product Subarray

Example 1:
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
Example 2:
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

```
    def maxProduct(self, nums: List[int]) -> int:
        res=max(nums)
        curmin,curmax=1,1
        for i in nums:
            if i==0:
                curmin,curmax=1,1
                continue
            tmp=curmin*i
            curmin=min(i*curmax,i*curmin,i)
            curmax=max(tmp,curmax*i,i)
            res=max(curmax,res)
        return res
```

7) Find Minimum in Rotated Sorted Array

Note: You must write an algorithm that runs in *O(log n) time.*

```
Example 1:

Input: nums = [3,4,5,1,2]
Output: 1
Explanation: The original array was [1,2,3,4,5] rotated 3 times.
Example 2:

Input: nums = [4,5,6,7,0,1,2]
Output: 0
Explanation: The original array was [0,1,2,4,5,6,7] and it was
rotated 4 times.
```

```
    def findMin(self, nums: List[int]) -> int:
            l,r=0,len(nums)-1
            res=nums[0]
            while l<=r:
                if nums[l]<nums[r]:
                    res=min(res,nums[l])
                    break
                m=(l+r)//2
                res=min(nums[m],res)
                if nums[m]>=nums[l]:
                    l=m+1
                else:
                    r=m-1
            return res
```

8) **3Sum**

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
Example 2:

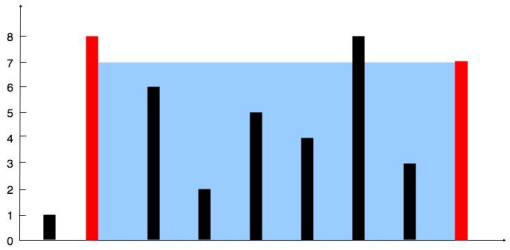
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.

```
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        res=[]
        nums.sort()
        for i,n in enumerate(nums):
            if i>0 and nums[i-1]==n:
                continue
            l,r=i+1,len(nums)-1
            while l<r:

                threesum=n+nums[l]+nums[r]
                if threesum>0:
                    r-=1
                elif threesum<0:
                    l+=1
                else:
                    res.append([n,nums[l],nums[r]])
                    l+=1
                    while nums[l]==nums[l-1] and l<r:
                        l+=1
        return res
```

9) **Container With Most Water (11)**

Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.



Soln:

```
l=0
r=len(height)-1
maxa=0
while l<r:
    cura=min(height[l],height[r])*(r-l)
    maxa=max(maxa,cura)
    if height[l]<height[r]:
        l+=1
    else:
        r-=1

return maxa
```

10) **Number of 1 Bits(191)**

Example 1:

Input: n = 11

Output: 3

Explanation:

The input binary string 1011 has a total of three set bits.

Example 2:

Input: n = 128

Output: 1

Explanation:

The input binary string 10000000 has a total of one set bit.

Soln

```
count = 0
while n:
    count += n%2
    n >>= 1
return count
```

11) **Counting Bits**

Example 1:

Input: n = 2
Output: [0,1,1]
Explanation:
0 --> 0
1 --> 1
2 --> 10
Example 2:

```
Input: n = 5
Output: [0,1,1,2,1,2]
Explanation:
0 --> 0
1 --> 1
2 --> 10
3 --> 11
4 --> 100
5 --> 101
Soln
```

```python
    def countBits(self, n: int) -> List[int]:
        dp=[0]*(n+1)
        offset=1
        for i in range(1,n+1):
            if offset*2==i:
                offset=i
            dp[i]=1+dp[i-offset]
        return dp
```

12 Climbing Stairs

Example 1:

Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
1. 1 step + 1 step
2. 2 steps
Example 2:

Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step
Soln:

```python
    def climbStairs(self, n: int) -> int:
        dp=[0]*(n+1)
        dp[1]=1
        dp[0]=1
        for i in range(2,n+1):
            dp[i]=dp[i-1]+dp[i-2]
        return dp[n]
=====================================================
        one,two=1,1
        for i in range(n-1):
            tmp=one
            one=one+two
            two=tmp
        return one
```

13 Coin Change

Example 1:

Input: coins = [1,2,5], amount = 11
Output: 3

Explanation: 11 = 5 + 5 + 1
Example 2:

Input: coins = [2], amount = 3
Output: -1
Example 3:

Input: coins = [1], amount = 0
Output: 0
Soln:

```
    def coinChange(self, coins: List[int], amount: int) -> int:

        dp=[amount+1]*(amount+1)
        dp[0]=0
        for i in range(1,amount+1):
            for j in coins:
                if i-j>=0:
                    dp[i]=min(dp[i],1+dp[i-j])
        return dp[amount] if dp[amount]!=amount+1 else -1
```

14 Longest Increasing Subsequence(300)

Example 1:

Input: nums = [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
Example 2:

Input: nums = [0,1,0,3,2,3]
Output: 4
Example 3:

Input: nums = [7,7,7,7,7,7,7]
Output: 1
Soln:

```
    def lengthOfLIS(self, nums: List[int]) -> int:
        n=len(nums)
        dp=[1]*n
        for i in range(n-1,-1,-1):
            for j in range(i+1,n):
                if nums[i]<nums[j]:
                    dp[i]=max(dp[i],1+dp[j])
        return max(dp)
```

15 Longest Common Subsequence

Example 1:
Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.
Example 2:
Input: text1 = "abc", text2 = "abc"
Output: 3
Explanation: The longest common subsequence is "abc" and its length is 3.
Example 3:

Input: text1 = "abc", text2 = "def"
Output: 0

Soln:

```
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        m=len(text1)
        n=len(text2)
        dp=[[0]*(n+1)  for _ in range(m+1) ]
        for i in range(len(text1)-1,-1,-1):
            for j in range(len(text2)-1, -1,-1):
                if text1[i]==text2[j]:
                    dp[i][j]=1+dp[i+1][j+1]
                else:
                    dp[i][j]=max(dp[i][j+1],dp[i+1][j])
        return dp[0][0]
```

16) **Word Break**

Example 1:
Input: s = "leetcode", wordDict = ["leet","code"]
Output: true
Explanation: Return true because "leetcode" can be segmented as "leet code".
Example 2:
Input: s = "applepenapple", wordDict = ["apple","pen"]
Output: true
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".
Note that you are allowed to reuse a dictionary word.
Example 3:
Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]
Output: false

```
    def wordBreak(self, s: str, wordDict: List[str]) -> bool:
        n=len(s)
        dp=[False]*(n+1)
        dp[n]=True
        for i in range(n-1,-1,-1):
            for w in wordDict:
                if i+len(w)<=n and s[i:i+len(w)]==w:
                    dp[i]=dp[i+len(w)]
                if dp[i]:
                    break
        return dp[0]
```

17) **Combination Sum**

Example 1:

Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
Example 2:

Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
Example 3:

Input: candidates = [2], target = 1
Output: []

Soln:

```
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        res=[]
        def dfs(i,cur,total):
            if total==target:
                res.append(cur.copy())
                return
            if i>=len(candidates) or total>target:
                return
            cur.append(candidates[i])
            dfs(i,cur,sum(cur))
            cur.pop()
            dfs(i+1,cur,total)
        dfs(0,[],0)
        return res
```

18) **House Robber**

Example 1:

Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
Example 2:

Input: nums = [2,7,9,3,1]
Output: 12
Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).
Total amount you can rob = 2 + 9 + 1 = 12.
Soln:

```
    def rob(self, nums: List[int]) -> int:
        rob1,rob2=0,0
        for n in nums:
            tmp=max(n+rob1,rob2)
            rob1=rob2
            rob2=tmp
        return rob2
```

19 **House Robber II**

Example 1:
Input: nums = [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.
Example 2:
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.
Soln:

```
    def rob(self, nums: List[int]) -> int:
        return max(self.helper(nums[1:]),self.helper(nums[:-1]))
    def helper(self,nums):
        rob1,rob2=0,0
        for i in nums:
            tmp=max(i+rob1,rob2)
            rob1=rob2
            rob2=tmp
        return rob2
```