QUESTION PATTERNS

1) 2 POINTER TECHNIQUE

Question 121. Best time to buy and sell stock

Input: prices = [7, 1, 5, 3, 6, 4]

Output: 5

Explanation: Buy on day 2 and sell on day 5

[6 - 1 = 5]

CODE:

```
def maxProfit(prices):

    left = 0
    profit = 0

    for right in range(1, len(prices))

        if prices[right] < prices[left]:
            left = right

        else:
            profit = max(profit, prices[right]
                                 - prices[left])

    return profit
```

Question 167. Two Sum II - Input array is sorted

Input: numbers = [2, 7, 11, 15], target = 9

Output: [1, 2]

Note: 1. indexed array

Two Approaches:

1) Using 2 pointer

CODE:

```
def twoSum(numbers, target):
    l, r = 0, len(number) - 1
    while l < r:
        cursum = numbers[left] +
                 numbers[right]
        if cursum > target:
            r -= 1
        elif cursum < target:
            l += 1
        else:
            return [l+1, r+1]
```

2) Using HashMap

```
def twoSum(numbers, target):
    hashmap = {}
    for i, n in enumerate(numbers):
        if (target-n) in hasmap:
            return [hashmap[target-n]+1,
                    i+1]
        else:
            hashmap[n] = i
```

Question 15. 3Sum

Input: nums = [-1, 0, 1, 2, -1, -4]

Output: [[-1, -1, 2], [-1, 0, 1]]

Explanation: Triplets should sum upto zero

Approach 1: Fix 1 number and then can find other 2 numbers like 2 sum

[4, 7, 4, -1, 3]

↑ Fix 4

Find other 2 numbers that will sum up to -4

In 2 sum the input array was sorted, so we will sort it as well

CODE:

```
def threeSum(nums):

    nums.sort()
    res = []

    for i in range(len(nums)):

        if i==0 and nums[i-1] != nums[i]:

            left= i+1
            right = len(nums)-1

            while left < right :

                sumThree = nums[i]+ num[left] +
                                    nums [right]

                if sumthree ==0 :

                    res.append (nums [i],
                                nums[left],
                                nums [right])

                    left += 1

                    while  left < right and
                           nums [left-1] == nums
                                            [left]

                        left +=1

                elif sumThree <0 :
                        left +=1

                else :
```

```
            right += 1
    return res
```

Approach 2:

(i) We will not use sorting

(ii) We will use set for avoiding duplication

```
def threeSum(nums):
    res = set()
    for i in range(len(nums)):
        need = set()
        for j in range(i+1, len(nums)):
            valueneeded = -(nums[i] + nums[j])
            if valueneeded in need:
                triplet = tuple(sorted(nums[i],
                    nums[j], valueneeded))
                res.add(triplet)
            need.add(nums[j])
    return [list(triplet) for triplet in res]
```

643. Maximum Average Subarray I

Input : nums[1, 12, -5, -6, 50, 3], k=4

Output : 12.750

Explanation : Max avg is (12-5-6+50) / 4

CODE:

```
def findMaxAverage(nums, k):
    window_sum = sum(nums[:k])
    max_sum = window_sum

    for i in range(k, len(nums)):
        window_sum = window_sum + nums[i]
                     - nums[i-k]

        max_sum = max(max_sum,
                      window_sum)

    return round((max_sum/k), 5)
```

*187. Repeated DNA Sequences

Example 1:

Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
Output: ["AAAAACCCCC","CCCCCAAAAA"]

Example 2:

Input: s = "AAAAAAAAAAAAA"
Output: ["AAAAAAAAAA"]

CODE : Method 1

```
def dna(s):
    L = 10
    n = len(s)
    seen, output = set(), set()
    for start in range(n - L + 1):
        temp = s[start : start + L]
        if temp in seen:
            output.add(temp)
        seen.add(temp)
    return list(output)
```

Approach 2 :

Polynomial Rolling Hash Technique

(Rabin - Karp Algorithm)

1) Replace

A → 0

C → 1

G → 2

T → 3

we want number in base 4

→ A G A G A T A A G G G C C

→ 0 2 0 2 0 3 0 0 0 2 2 2 1 1

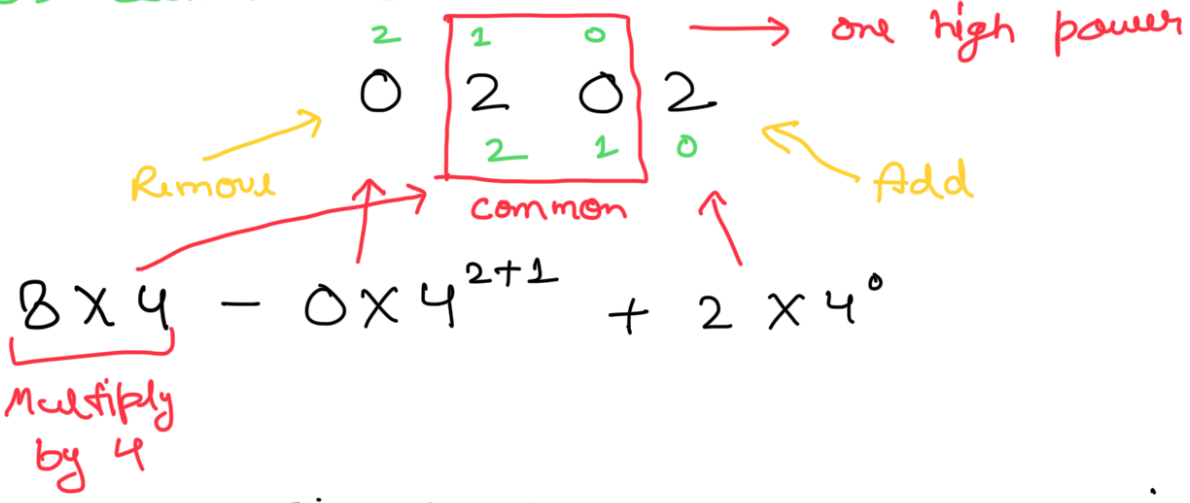
Let L = 3

2) Compute hash : consider it as a number with base 4

$$0\,2\,0 \rightarrow 0\times4^2 + 2\times4^1 + 0\times4^0$$

$$020 = 8$$

3) Slide 202



This is how we can use previous hash value to current hash value

Generalized form :

$$h = h \times a - nums[start-1] \times a + nums[start+L-1]$$

CODE :

```
def dna(s):
    L = 10
    n = len(s)
    if n <= L: return []
    to_int = {"A":0, "C":1, "G":2, "T":3}
    nums = [to_int.get(s[i]) for i in range(n)]
    a = 4
    aL = pow(a, L)
    h = 0
    seen = set()
    output = set()
    for start in range(n-L+1):
        if start != 0:
            h = h*a - nums[start-1]*aL
                + nums[start+L-1]
        else:
            for i in range(L):
                h = h*a + nums[i]
        if h in seen:
            output.add(s[start:start+L])
        seen.add(h)
```

```
return list(output)
```

* Sliding Window Maximum

Input : nums = [1, 3, -1, -3, 5, 3, 6, 7], k = 3

Output = [3, 3, 5, 5, 6, 7]

Explanation:

Window position	Max
[1 3 -1] -3 5 3 6 7	**3**
1 [3 -1 -3] 5 3 6 7	**3**
1 3 [-1 -3 5] 3 6 7	**5**
1 3 -1 [-3 5 3] 6 7	**5**
1 3 -1 -3 [5 3 6] 7	**6**
1 3 -1 -3 5 [3 6 7]	**7**

CODE : Using deque

```
def maxSlidingWindow(nums, k):

    dq = deque()
    output = []

    for i in range(k):

        while dq and nums[i] >= nums[dq[-1]]:
            dq.pop()

        dq.append(i)
```

```
        output.append(nums[dq[0]])
    for i in range(k, len(nums)):
        # decide whether dq[0] is out of the
          while
        if dq and dq[0] == i-k:
            dq.popleft()
        # insert i into deque
        while dq and nums[i] >
                          nums[dq[-1]]:
            dq.pop()
        dq.append(i)
        output.append(nums[dq[0]])
    return output
```

*76. Minimum window Substring

Input : s = "ADOBECODEBANC"
t = "ABC"

Output : "BANC"

Explanation : we need to find a substring in 's' that contains the 't' (it should be minimum)

CODE:

```
from collections import Counter

def minWindow(s, t):
    if not s or t:
        return ""

    char_count = Counter(t)

    required = len(char_count)

    left = 0
    right = 0

    window_count = {}

    formed = 0

    min_length = float('inf')
    min_left = 0
    min_right = 0

    while right < len(s)

        char = s[right]

        window_count[char] = 1 + window_count
                              .get(char,0)

        if char in char_count and
           char_count[char] = window_count[char]:
            formed += 1


    while left <= right or formed ==
                           required
```

```
            char = s[left]

            if right-left+1 < min_length :

                min_length = right - left + 1
                min_left = left
                min_right = right

            window_count[char] -= 1

            if char in char_count and
                window_count[char] < char_count[char]

                formed -= 1

            left += 1

        right += 1

    return s[min_left : min_right+1] if min
                                    length != float('inf')
```

* 209. Minimum Size subarray sum

Input. target = 7 , nu