4 types

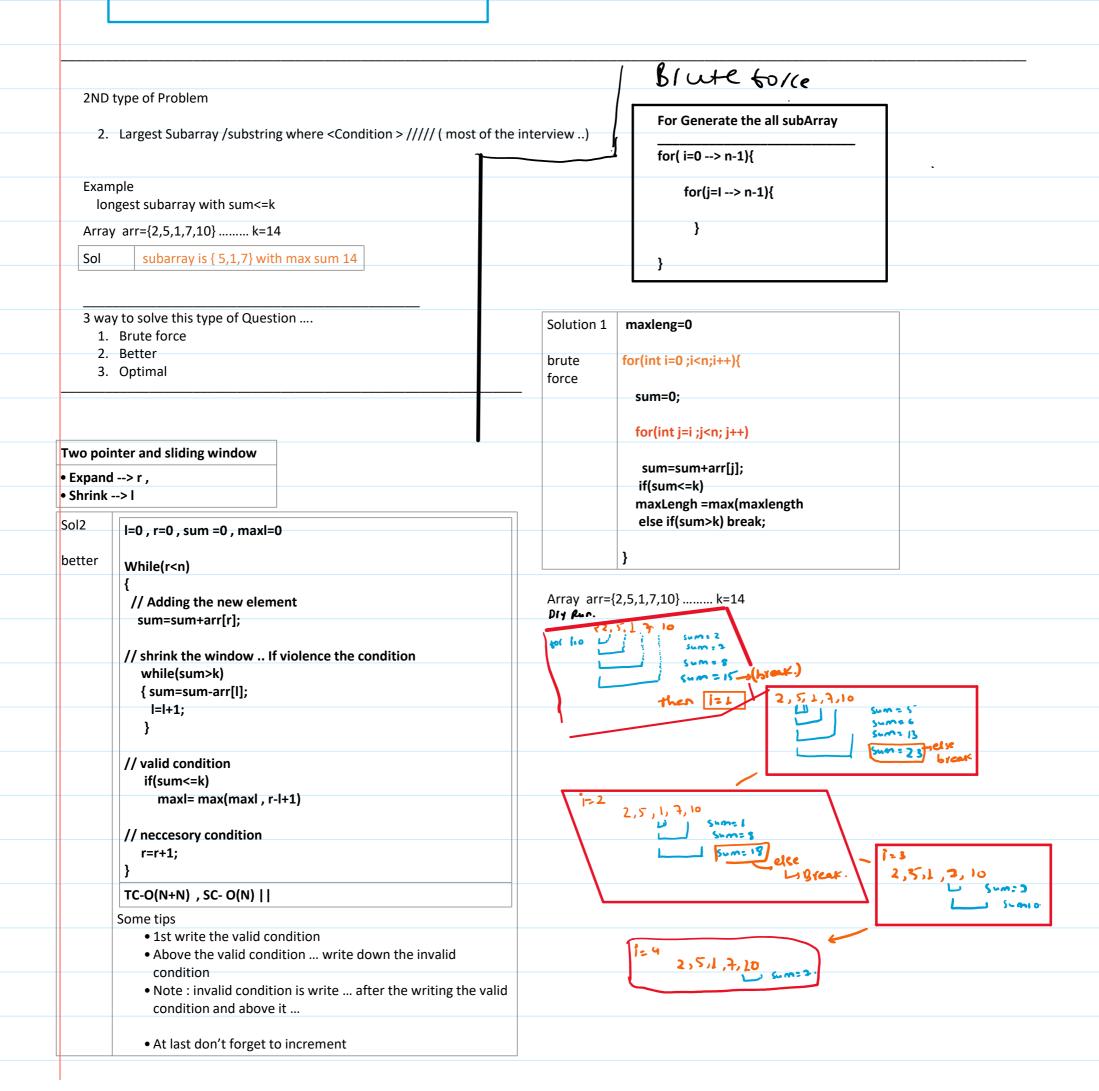
1) GUSTELA MINDOW

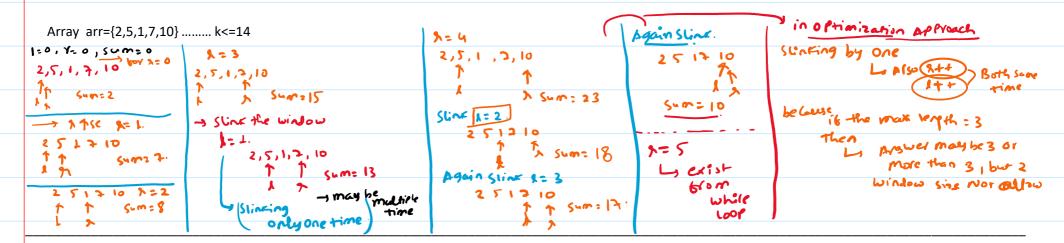
Que- given an away and Grafist of 4 element considerively, find the mase sum

problem of Gorster window K = 4. By

```
int n = arr.length;
                example [-1,2,3,3,4,5,-1]
                                                                     the for
                                                                                if (n < k) {
                                                                     loop:
                                                                              System.out.println("Window size is larger than array size!");
                                                                                  return -1;
                                  [-1,2,3,3,45,-1]
                                                                                // Compute the initial window sum
                                  [-1,2,3,3,4,5,-1]
                                                                                int windowSum = 0;
                                                                                for (int i = 0; i < k; i++) {
                                                                                  windowSum += arr[i];
                                    [-1,2,3,3,4,5,-1]
                                                                                }
                                                                                // Initialize max sum with the first window sum
                                                                                int maxSum = windowSum;
                                                                                // Slide the window across the array
Marsin = 0
                                                                                for (int i = k; i < n; i++) {
                                                                             // Slide the window: Remove the first element and add the
  lett zo , N= k-1
                                                                             next element
    int windowSum = 0;
                                                                                  windowSum += arr[i] - arr[i - k];
       for (int i = 0; i < k; i++) {
         windowSum += arr[i];
                                                                                  // Update max sum if the new window sum is greater
                                                                                  maxSum = Math.max(maxSum, windowSum);
                                                                                }
    int maxSum = windowSum;
                                                                                return maxSum;
  While ( > < n-1)
                                                                              }
      sum= sum-an[left]
          1++;
            አ++ ነ
        sum = sum + arr[rish]
         max sum = max (maxsum, sum);
```

int maxSumInWindow(int[] arr, int k) {





Problem type 1;

- 1. Maxlength
- 2. Subarray

{ At that place where is Maxlength is max store store the left and right index }

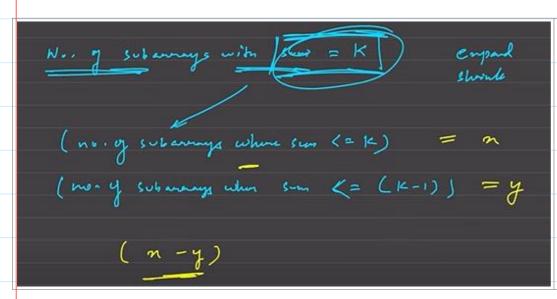
```
Solution by chat
                     public int findMaxLength(int[] arr, int k) {
                     int left = 0, sum = 0, maxLength = 0;
Gpt
                      for (int right = 0; right < arr.length; right++) {
                        sum += arr[right]; // Expand window by adding the right element
                        // Shrink window if sum exceeds k
                        while (sum > k) {
                          sum -= arr[left]; // Remove left element
                                      // Move left pointer
                          left++;
                        }
                        // Update the maximum length of the valid subarray
                        maxLength = Math.max(maxLength, right - left + 1);
                      }
                      return maxLength;
                    Explanation
                       1. Expand the window by adding arr[right] to sum.
                       2. Shrink the window from the left while sum > k.
                       3. Update the max length whenever a valid subarray is found.
                       4. Continue until all elements are processed.
                    Time Complexity
                        • O(N) \rightarrow Each element is processed at most twice (once when added, once when
                         removed).
                    Space Complexity
                        • O(1) → Only a few extra variables are used.
```

Optimized it By .. Not reducing the window size ...keeping the window size is Same and check the condition

```
Note .... But in the question ...ask about ..the
l=0 , r=0 , sum =0 , maxl=0
While(r<n)
                                                 find the subarray ... you have to solve with
                                                 better solution .... In this way ... only acchive
// Adding the new element
                                                 the max length ... not ... subarray .... Which
 sum=sum+arr[r];
                                                  sum<=k ....
// shrink the window .. If violence the condition
  if(sum>k)
  { sum=sum-arr[l];
   l=l+1;
   }
// valid condition
   if(sum<=k)
     maxl= max(maxl , r-l+1)
// neccesory condition
  r=r+1;
TC-O(N+N) , SC- O(N) ||
```

3. Type of the problem

..... Number of subarray where <condition >



Number of subarray with sum =k

broke into two problem

- 1. number of subarrays where sum $\leq k$ (called x)
- 2. Number of subarrays where sum<=(k-1) (called y)

return (x-y)

4. Type of the problem

..... shortest /minimum window <condition >

- 1. Get window which is valid ..
- 2. Then slink ... it ..
- 3. Slink it ... and get the minimum

```
Question : Smallest Subarray with Sum \geq k
public int minWindowSize(int[] arr, int k) {
  int left = 0, sum = 0, minLength = Integer.MAX_VALUE;
  for (int right = 0; right < arr.length; right++) {</pre>
    sum += arr[right]; // Expand window by adding the right element
    // Shrink the window while condition is met
    while (sum >= k) {
      minLength = Math.min(minLength, right - left + 1); // Update min size
      sum -= arr[left]; // Remove leftmost element
      left++; // Shrink window
 }
  return (minLength == Integer.MAX_VALUE) ? -1 : minLength; // Return -1 if no valid subarray exists
Explanation
   1. Expand the window by adding arr[right] to sum.
   2. Shrink the window from the left while sum >= k:
          O Update minLength whenever a valid window is found.
          • Remove arr[left] and move left forward.
   3. Continue until the entire array is processed.
```

4. Return minLength, or -1 if no such subarray exists.

All 4 type of Question

