

4 types

① Constant window

Que- Given an array and consist of 4 element consecutively, find the max sum

problem of constant window $k = 4$.

example

$[-1, 2, 3, 3, 4, 5, -1]$

$[-1, 2, 3, 3, 4, 5, -1]$

$[-1, 2, 3, 3, 4, 5, -1]$

$[-1, 2, 3, 3, 4, 5, -1]$

max sum = 0

left = 0, right = k-1

```
int windowSum = 0;
for (int i = 0; i < k; i++) {
    windowSum += arr[i];
}
```

int maxSum = windowSum;

while (right < n-1)

{

sum = sum - arr[left]

left++;

right++;

sum = sum + arr[right]

maxSum = max(maxSum, sum);

}

By using the for loop :

```
int maxSumInWindow(int[] arr, int k) {
    int n = arr.length;
    if (n < k) {
        System.out.println("Window size is larger than array size!");
        return -1;
    }

    // Compute the initial window sum
    int windowSum = 0;
    for (int i = 0; i < k; i++) {
        windowSum += arr[i];
    }

    // Initialize max sum with the first window sum
    int maxSum = windowSum;

    // Slide the window across the array
    for (int i = k; i < n; i++) {
        // Slide the window: Remove the first element and add the next element
        windowSum += arr[i] - arr[i - k];

        // Update max sum if the new window sum is greater
        maxSum = Math.max(maxSum, windowSum);
    }

    return maxSum;
}
```

2ND type of Problem

2. Largest Subarray / substring where <Condition> (most of the interview ..)

Example

longest subarray with sum <= k

Array arr={2,5,1,7,10} k=14

Sol subarray is { 5,1,7} with max sum 14

3 way to solve this type of Question

1. Brute force
2. Better
3. Optimal

Brute force

For Generate the all subArray

```
for (i=0 --> n-1){
    for (j=i --> n-1){
        // ...
    }
}
```

Solution 1

brute force

```
maxlength=0
for (int i=0; i<n; i++){
    sum=0;
    for (int j=i; j<n; j++){
        sum=sum+arr[j];
        if (sum<=k)
            maxlength = max(maxlength, sum-j+i);
        else if (sum>k) break;
    }
}
```

Array arr={2,5,1,7,10} k=14
Dry Run.

i=0
2, 5, 1, 7, 10
sum=2
sum=7
sum=8
sum=15 (break)
then i=1

2, 5, 1, 7, 10
sum=5
sum=6
sum=13
sum=23 (else break)

i=2
2, 5, 1, 7, 10
sum=1
sum=8
sum=19 (else break)

i=3
2, 5, 1, 7, 10
sum=3
sum=10

i=4
2, 5, 1, 7, 10
sum=2

Two pointer and sliding window

- Expand --> r,
- Shrink --> l

Sol2

better

l=0, r=0, sum=0, maxl=0

While (r<n)

{

// Adding the new element
sum=sum+arr[r];

// shrink the window .. If violate the condition

```
while (sum>k)
{
    sum=sum-arr[l];
    l=l+1;
}
```

// valid condition

```
if (sum<=k)
    maxl = max(maxl, r-l+1)
```

// necessary condition

```
r=r+1;
}
```

TC-O(N+N), SC-O(N) ||

Some tips

- 1st write the valid condition
- Above the valid condition ... write down the invalid condition
- Note : invalid condition is write ... after the writing the valid condition and above it ...

- At last don't forget to increment

4.Type of the problem
..... shortest /minimum window <condition >

- 1. Get window which is valid ..
- 2. Then shrink ... it ..
- 3. Shrink 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++) {
        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 $\geq k$:
 - 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

