Todays Content

1. Sliding Window
2. Kadane's

Q8: # No: of Subarrays of len = k

arr[6] = { $a_0$ $a_1$ $a_2$ $a_3$ $a_4$ $a_5$ }

k = 4 ans = 3

arr[7] = { $a_0$ $a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ }

k = 3 ans = 5

arr[8] = { $a_0$ $a_1$ $a_2$ $a_3$ $a_4$ $a_5$ $a_6$ $a_7$ }

k = 5 ans = 4

Q # No: of subarrays of len = k in arr[N] = N - k + 1

28. Given ar[N] return Max Subarray Sums with len = k

Constraints: Consider subarray of len = k

$1 <= N <= 10^5$

$1 <= k <= N$

$-10^6 <= ar[i] <= 10^6$

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| Ex: ar[10] : { | -3 | 4 | -2 | 5 | 3 | -2 | 8 | 2 | -1 | 4 } |

k = 5

| s | e | sum |
|---|---|---|
| [0 | 4] | 7 |
| [1 | 5] | 8 |
| [2 | 6] | 12 |
| [3 | 7] | 16 |
| [4 | 8] | 10 |
| [5 | 9] | 11 |

return Max = 16

Idea: Generate all subarrays of len = k.
Iterate & calculate sum & get overall max.

Estimated TC: $\underbrace{O(N-k+1)}_{\text{\#count of subarrays of len = k}} * \underbrace{O(k)}_{\text{Iterate in subarray}} \approx O(N^2)$ = TLE

k: Smallest = 1 TC: $O(N-1+1) * O(1) = O(N)$

k: largest = N TC: $O(N-N+1) * O(N) = O(N)$

k: Middle = N/2 TC: $O(N-N/2+1) * O(N/2) \approx O(N^2)$

$\approx O(N/2) * O(N/2)$

```
long subarraylen(vector<int> &ar, int N){
    int s=0, e=k-1;
    long mSum=0;
    while(e < N){ // e in arr[].
        long total=0;
        for(int i=s; i<=e; i++){
            total = total + ar[i]
        }
        if(total > mSum){
            mSum = total;
        }
        s++; e++
    }
    return mSum;
}
```

Dry Run:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Ex: ar[8] : { | -3 | 4 | -2 | 5 | 3 | -2 | 8 | 2 } |

k = 4

| s | e | tsum | mSum=0 | |
|---|---|---|---|---|
| 0 | 3 | 4 | 4 | s++ e++ |
| 1 | 4 | 10 | 10 | s++ e++ |
| 2 | 5 | 4 | 10 | s++ e++ |
| 3 | 6 | 14 | 14 | s++ e++ |
| 4 | 7 | 11 | 14 | s++ e++ |
| 5 | 8 > N-1 | Stop process & return mSum; | | |

# Idea: Generate all subarrays of len = k.

calculate sum using pf[] & get overall max.

TC: O(N + (N-K+1)*1)   SC: O(N) → because pf[]

TC: O(N)  SC: O(N)

```
long subarraylen(vector<int> &arr, int N){
    long pf[N], sum = 0;
    for(int i=0; i<N; i++){
        sum = sum + arr[i];
        pf[i] = sum;
    }

    int s=0, e=k-1;
    long mSum = 0;
    while(e < N){ // e in arr[].
        long total=0; // Subarray sum from [s..e]
        if(s==0){ // [0..e]
            total = pf[e]
        }
        else{
            total = pf[e] - pf[s-1]
        }

        if(total > mSum){
            mSum = total;
        }

        s++; e++
    }
    return mSum;
}
```

Idea3: Sliding Window: Fixed length subarrays.

Note: If subarray size is fixed, think in above approach.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[10] = | 3 | 4 | -2 | 5 | 3 | -2 | 8 | 2 | 1 | 4 |

k=6

s e

[0 5] sum = 11 : 1st subarray iterate & calculate

→ // apply sliding    sub    add

[1 6] sum = sum - arr[0] + arr[6] = 11 - 3 + 8 = 16

[2 7] sum = sum - arr[1] + arr[7] = 16 - 4 + 2 = 14

[3 8] sum = sum - arr[2] + arr[8] = 14 - (-2) + 1 = 17

[4 9] sum = sum - arr[3] + arr[9] = 17 - 5 + 4 = 16

s-1 [s..e] sum = sum - arr[s-1] + arr[e]

return max = 17

Note1: For 1st subarray we iterate & calculate

Note2: For remaining subarrays we slide & calculate.

int subarray(vector<int> &arr, int k){ TC: O(N) SC: O(1)

```
    long sum = 0, msum = INT_MIN;
    for(int i=0; i<k; i++){        → k iterations
        sum = sum + arr[i];
    }

    if(sum > msum){
        msum = sum;
    }
    int s=1, e=k;
    while(e<N){                    → N-k iterations
        sum = sum - arr[s-1] + arr[e];
        if(sum > msum){ msum = sum;}
        s++; e++;
    }
    return msum;
}
```

: N iterations

# Max Subarray Sum:

{Continuous part of an array}

Given ar[N] return max subarray sum.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Ex1: ar[ ] = | -2 | 3 | 4 | -1 | 5 | -10 | 7 |

Ex1: ar[ ] = { -2 3 4 -1 5 -10 7 } ans =

Ex2: ar[ ] = { -3 4 6 8 -10 2 7 } ans =

Q1: ar[ ] = { 4 5 2 1 6 } ans =

Q2: ar[ ] = { -4 -3 -6 -9 -2 } ans = -

**Idea 1:**

# Optimisation: Kadane's Algo: Max Subarray Sum.

**Case 1:** If all the elements in the array are positive:

| | 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|---|
| arr[ ] = { | 4 | 2 | 1 | 6 | 7 | } ans = |

**Case 2:** If all the elements in the array are negative

| | 0 | 1 | 2 | 3 | 4 | |
|---|---|---|---|---|---|---|
| arr[ ] = { | -4 | -8 | -3 | -10 | -5 | } ans = |

**Case 3:** If positives are present in between.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| arr[ ] = { | -3 | -5 | -3 | 4 | 3 | 2 | 10 | -5 | -7 | } ans = |

**Case 4:** Say we have max subarray sum

4.1 arr[ ] = {-3 -5 -3 $\boxed{\phantom{xxxxxxxxxx}}$ -5 -7}

4.2 arr[ ] = {-3 5 -3 $\boxed{\phantom{xxxxxxxxxx}}$ -5 -7}

Idea:

Trace:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| arr[ ] = { | 3 | 4 | -6 | 8 | -10 | 12 | 10 | -3 | 7 | -5 | } |

Sum =

0 1 2 3 4 5 6

ar[ ] = { -2 3 4 -1 5 -10 7 }

sum = 0

max =

int maxSubKadane's(int ar[]){ TC: O SC: O( )

}