Todays Content

1. Kadane's
2. Sliding Window

2Q: # No:of Subarrays of len : k

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

k = 4, ans = 3

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

k = 3 ans = 5

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

k = 5 ans = 4

Given ar[N] no:of subarrays of len : k

ans = N - k + 1

1Q Given ar[N] return Max Subarray Sums of len = k

Consider subarray of len = k

Constraints:

$1 <= N <= 10^5$

$1 <= K <= N$

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

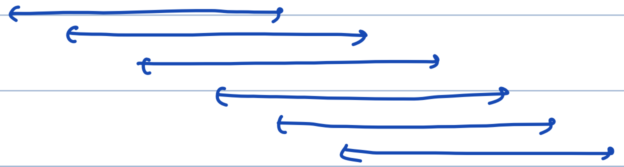
sum = $\{-10^{11}$ ______ $10^{11}\}$ = long

$ar[10^5] = \{-10^6 -10^6 .. -10^6\}$ $ar[10^5] = \{10^6\ 10^6 .. 10^6\}$

0 1 2 3 4 5 6 7 8 9 10

Eg: ar[10] : {-3 4 -2 5 3 -2 8 2 -1 4}

k = 5



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

Idea: Generate all subarrays of len = k:

Iterate & calculate sum & get overall max.

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

$k=1$ $O(N-1+1)*O(1) = O(N*1) = O(N)$

$k=N$ $O(N-N+1)*O(N) = O(1*N) = O(N)$

$k=N/2$ $O(N-N/2+1)*O(N/2) = O(N/2 * N/2) = O(N^2)$

Code:

```
                0   1   2   3   4   5   6   7   8
Ex: ar[9] : { -3   4  -2   5   3  -2   8   2  -1
                                    s           e
    k=5
```

s e sum

[0 4] iterate from s..e & get sum = 7; s++; e++;

[1 5] iterate from s..e & get sum = 8; s++; e++;

[2 6] iterate from s..e & get sum = 12; s++; e++;

[3 7] iterate from s..e & get sum = 16; s++; e++;

[4 8] iterate from s..e & get sum = 10; s++; e++;

[5 9]: If e outside stop process & return max sum; 16.

```
long maxSum(vector<int> &ar){
    int N = ar.size();
    long max = INT_MIN;                          a    b    b-a+1
    int s=0, e=k-1;             # 1st subarray : [0.. k-1]; k-1-0+1 = k;
    while(e<N){
        long sum=0;
        for(int i=s; i<=e; i++){    ⟶ 1. Iterate & calculate sum
            sum=sum+ar[i];
        }
        if(sum>max){               ⟶ 2. Cmp with max
            max=sum;
        }
        s++; e++;                  ⟶ 3. Goto next subarray.
    }
    return max;
}
```

Idea2: For every subarray of len=k, calculate sum using pf() & get overall ma

TC: O(N + (N-K+1) * O(1)) = O(N + N-K+1) ≈ O(N)   SC = O(N)

pf[]

Code:

	0	1	2	3	4	5	6	7	8
Ex: ar[9] :	-3	4	-2	5	3	-2	8	2	-1
pf[9] :	-3	1	-1	4	7	5	13	15	14

k = 5   0

s e sum:

[0 4] = pf[4] = 7

[1 5] = pf[5] - pf[0] = 5-(-3) = 8

[2 6] = pf[6] - pf[1] = 13 - 1 = 12

[3 7] = pf[7] - pf[2] = 15-(-1) = 16

[4 8] = pf[8] - pf[3] = 14 - 4 = 10

[5 9] : *

```
sum[s..e] : if(s==0){ // [0..e]
                pf[e]
            }
            else{
                pf[e] - pf[s-1]
            }
```

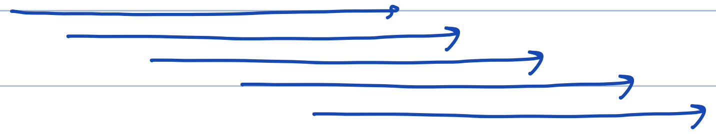
```
long maxSum(vector<int> &arr){ TC: O(N) SC: O(N)
    int N = arr.size();
    long pf[N], sum=0;
    for(int i=0; i<N; i++){
        sum = sum + arr[i]; // 1. update sum
        pf[i] = sum;
    }

    long max = INT_MIN;                                      a    b    b-a+1
    int s=0, e=K-1;                 # 1st subarray : [0.. K-1]; K-1-0+1 = K;
    while(e<N){
        long sum=0;
        if(s==0){                   ——> 1. #Sum [s..e] get using pf[]
            sum = pf[e];
        }
        else{
            sum = pf[e] - pf[s-1];
        }
        if(sum > max){              ——> 2. Cmp with max
            max = sum;
        }
        s++; e++;                   ——> 3. Goto next subarray.
    }
    return max;
}
```

Idea3: Sliding Window: Fixed length subarrays.

0 1 2 3 4 5 6 7 8 9

ar[10] = {3 4 -2 5 3 -2 8 2 1 4}



k=6

s e

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

\# For remaining windows calculate using sliding window.

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

[2 7] sum = sum - ar[1] + ar(7) = 16 - 4 + 2 = 14

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

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

[5 10] Stop & return max sum = 17

s-1 | s s+1 s+2 ...... e-1 e

[s,,e] sum = sum - ar(s-1) + ar[e]

\# Remove ar(s-1)

\# Adding ar[e]

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

    # Step1: For 1st window iterate & get it.   1st subarray [0.. K-1]
    long sum=0, max=INT_MIN;                    2nd subarray [1.. k]
    for(int i=0; i<k; i++){   #[0..k-1] = k iterations
        sum= sum+ ar[i];
    }

    if(sum>max){
        max=sum;
    }

    # Step2: For remaining subarrays, slide & calculate sum.
    int s=1, e=k;
    while(e<N){ ——————————→ # (N-k) iterations
        #get sum from [s.. e] using sliding.
        sum= sum - ar[s-1] + arr[e];
        if(sum > max){
            max = sum;
        }
        s++; e++;
    }
    return max;
}
```

Max Subarray Sum:

Given ar[N] return max subarray sum. → {Continuous part of an array}

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

0 1 2 3 4 5 6
Ex2: ar[ ] = { -3 4 6 8 -10 2 7 } ans =

0 1 2 3 4
Q1: ar[ ] = { 4 5 2 1 6 } ans =

0 1 2 3 4
Q2: ar[ ] = { -4 -3 -6 -9 -2 } ans = -

Idea 1:

Optimisation: Kadane's Algo: Max Subarray Sum.

Case1: If all the elements in the array are positive. :

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

Case2: If all the elements in the array are negative

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

Case3: 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 =

Case4: Say we have max subarray sum

4.1 arr[ ] = {-3 -5 -3 🟩 -5 -7}

4.2 arr[ ] = {-3 5 -3 🟩 -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()
|
}
```