Max subarray sum

Total of all subarray sums

Max subarray sum of length = k

1. Given an array of integers, find sum of each possible subarray of array and maintain maximum sum

A = [1, 2, 3]

max subarray sum = 6

[1] = 1
[1,2] = 3
[1,2,3] = 6

[2] = 2
[2,3] 5
[3] = 3

A = [4, -1]

[4] = (4)
[4 -1] = 3
[-1] = -1

Brute force : Explore all possible subarrays

2 loops → to fix starting and ending indx

3rd loop → to iterate and find sum

sum [0 0]
↑

```
int ans = -∞ → INT_MIN / arr[0]

for (s = 0 ; s < n ; s++) {
    for (e = s ; e < n ; e++) {
        // s e
        int sum = 0
        for (k = s ; k <= e ; k++) {
            sum = sum + arr[k];
        }
        ans = max (ans, sum)
    }
}
```

TC : $O(N^3)$

SC : $O(1)$

How many subarrays ? $\frac{N(N+1)}{2} \simeq N^2$

Approach 2 : Use prefix sum

1) First create prefix sum
2) Go to all subarrays, get sum using pf[ ]

To Do Create a prefix array → N

int ans = -∞ → INT_MIN / arr[0]

```
for (s = 0 ; s < n ; s++) {                    ] → N²
    for (e = s ; e < n ; e++) {
        // s e
        int sum = 0
        if (s == 0)   sum = pf[e]
        else          sum = pf[e] - pf[s-1]
        ans = max(ans, sum)
    }
}
```

TC : O(N²)
SC : O(N)

[s → e] = pf[e] - pf[s-1]

Can we do it in TC : O(N²), SC : O(1) without modifying given array ?

A = [-4, 1, 3, 2] (0, 1, 2, 3)

s	e	k	sum
0	0	0 → 0	A[0]
0	1	0 → 1	A[0] + A[1]

0 2 $0 \to 2$ A[0] + A[1] + A[2]

↓ A[3]

0 3 $0 \to 3$ A[0] + A[1] + A[2] + A[3]

- Carry forward the sum

  Every time e changes, we can just add arr[e] to the sum.

maxsum = $-\infty$

s = 0

curSum = 0

A = [−4 , 1 , 3 , 2]
0 1 2 3

e		curSum	Value	maxSum
0	[0 0]	+ A[0]	−4	−4
1	[0 1]	+ A[1]	−3	−3
2	[0 2]	+ A[2]	0	0
3	[0 3]	+ A[3]	2	2

+3, +2

s = 1

curSum = 0

e		curSum	value	maxSum
1	[1 1]	+ A[1]	1	2
2	[1 2]	+ A[2]	4	4
3	[1 3]	+ A[3]	6	6

+3, +2

s = 2
cursum = 0

A = [-4 , 1 , 3 , 2]
0 1 2 3

e	cursum	value	maxsum
2 [2,2]	+ A[2]	3	6
3 [2,3]	+ A[3]	5	6

+2

s = 3
cursum = 0

e	cursum	value	maxsum
3 [3,3]	+ A[3]	2	6

```
int ans = -∞ → INT/MIN/arr[0]
for (s = 0 ; s < n ; s++) {
    int cursum = 0
    for (e = s ; e < n ; e++) {
        // s e
        cursum += A[e]
        ans = max (ans, cursum)
    }
}
```

s ↑ e

s e+1
↓
+arr[e+1]

TC: O(N²)
SC: O(1)

Carry forward sum

→ Google, FB

2. Given an array of integers, find total sum of all possible subarrays.

[1 2 3] gtotal = 1+3+6+2+5+3 = 20

[1] = 1
[1,2] = 3
[1,2,3] = 6

[2] = 2
[2,3] 5
[3] = 3

gtotal of all subarray sum

```
int total = 0
for (s = 0 ; s < n ; s++) {
    int currsum = 0
    for (e = s ; e < n ; e++) {
        // s e
        currsum += A[e]
        total += currsum
    }
}
return total
```

TC: O(N²)
SC: O(1)

[1 2 3] (0 1 2)

s	e	currsum		gtotal
0	0	0+1	①	1
0	1	1+1+2 (new)	③	1+3

0 2 4+3 1+3+6

cursum = cursum + cursum + A[c]

new sum

10:13

[1,2,3] gtotal = 1+3+6+2+5+3 = 20

[1] = 1
[1,2] = 3
[1,2,3] = 6

[2] = 2
[2,3] 5
[3] = 3

[1, 2, 3]

Instead of going to each subarr, find an alternative

Think about contribution of each element in gtotal

(3) 1 → [1] [1,2,3] [1,2]
(4) 2 → [1,2] [1,2,3] [2] [2,3]
(3) 3 → [1,2,3] [2,3] [3]

ans = 1*3 + 2*4 + 3*3
= 3 + 8 + 9 = 20

	0	1	2	3	4	5
A =	3	-2	4	-1	2	6

In how many subarrays, index 1 (-2) is present?

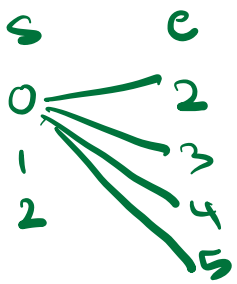
s	e
0	1
1	2
	3
	4
	5

ans = 10

0,1 [3 -2]
0,2 [3 -2 4]
0,3 [3 -2 4 -1]
0,4 [3 -2 4 -1 2]
0,5 [3 -2 4 -1 2 6]
1,1 [-2]
1,2 [-2 4]
1,3 [-2 4 -1]
1,4 [2 4 -1 2]
1,5 [-2 4 -1 2 6]

	0	1	2	3	4	5
A =	3	-2	4	-1	2	6

In how many subarrays, index **2** is present?



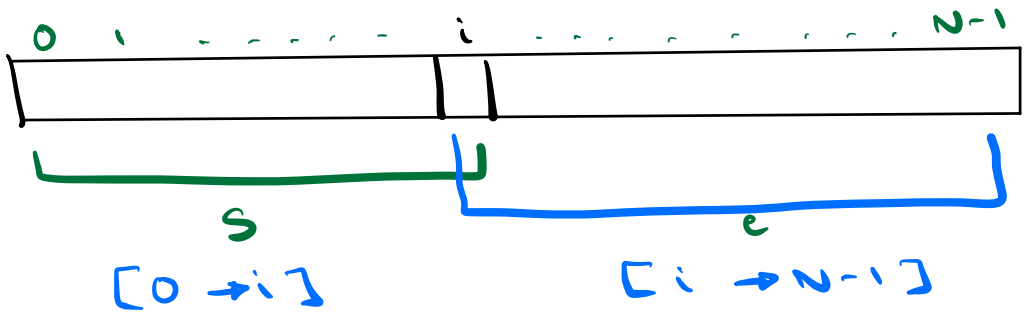
No. of s x No. of e

Total count = 3 x 4
= 12

Generalized Calculation

arr[N]

In how many subarrays, index i present?



cnt of s = i - 0 + 1

= (i+1)

cnt of e = N - x - i + x

= (N-i)

Total subarray cnt = (i+1) * (N-i)

Contribution of ith elem = A[i] * (i+1) * (N-i)

```
int gtotal = 0
for(i = 0 ; i < N ; i++) {
    gtotal += A[i] * (i+1) * (N-i)
}
return gtotal
```

TC : O(N)

SC : O(1)

[1 2 3] N = 3

i	No. of subarrays	Contribution	gtotal
0	(0+1) * (3-0) = 3	1 * 3 = 3	3
1	(1+1) * (3-1) = 4	2 * 4 = 8	11
2	(2+1) * (3-2) = 3	3 * 3 = 9	20

Q. No. of subarrays of len = k

[0 1 2 3 4] N = 5 k = 3

s	e
0	2
1	3
2	4

For a particular s, there's a unique e bcoz size of subarr is fixed

s	e
0	k-1
1	k
2	k+1
3	k+2
⋮	⋮
N-k	N-1

[s e] len = e - s + 1

k = e - 0 + 1

k - 1 = e

k = e - s + 1

k + s - 1 = e

$[0 \rightarrow N-K]$

$N-K-0+1$

$[a \quad b]$

$b-a+1$

cnt of $s = N-K+1$

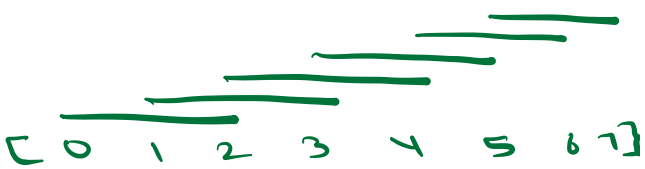
No. of subarrays of len $K = N-K+1$

$N=7, K=4$

No. of subarrays of len $4 = 7-4+1 = 4$

Q. Given an array of N, print start and end index of all subarrays of len K

$N=8 \quad K=3$



Output

s	e
0	2
1	3
2	4
3	5
4	6
5	7

e $[K-1 \quad N-1]$

$K-1 \quad K \quad K+1 \quad \ldots \quad N-1$

```
s = 0   e = K-1
while (e < n) {
    print (s, e)
    s++
    e++
}
```

$K \rightarrow [1 \; N]$

TC: $O(N-K+1)$

$N - N/2 + 1 = N/2$

Q. Given an array of N de
print max subarray sum for
subarrays with length = k

N = 8 , k = 5

0	1	2	3	4	5	6	7
-3	4	-2	5	3	-2	8	2

7

8

12

16

ans = 16

BF : Consider all subarrays and get sum

```
int ans = INT_MIN / -∞
s = 0   e = k-1
while (e < n) {
    sum = 0
    for (i = s ; i ≤ e ; i++)
        sum += arr[i]
    ans = max (ans, sum)
    s++
    e++
}
```

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

k = 1 : (N-1+1)*1 = N

k = N/2 : (N-N/2+1)N/2 = N²/4

k = N : (N-N+1)*N = N

[1 N]

O(N²)

SC : O(1)

Approach 2 : Use prefix [] to find sum

To DO create prefix array

```
int ans = INT_MIN / -∞
s = 0   e = k-1
while (e < n) {
    sum = 0
    if (s == 0)  sum = prefix[e]
    else  sum = prefix[e] - prefix[s-1]
    ans = max (ans, sum)
    s++
    e++
}
```

TC: O(N + N-K+1) (N-K+1 → N/2)

↓

O(N)

SC: O(N)

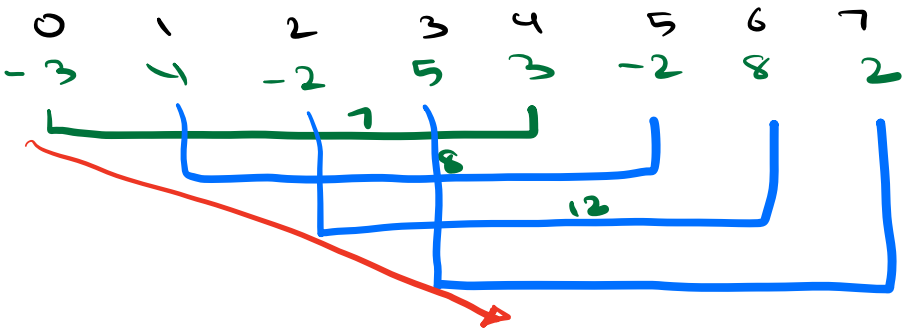
↓

O(1) Modify original arr

Approach 3 :

-3

Sliding window



k = 5

s	e	sum	
0	4	7	
1	5	sum - A[0] + A[5]	7 - (-3) - 2 = 8
2	6	sum - A[1] + A[6]	8 - 4 + 8 = 12
3	7	sum - A[2] + A[7]	12 - (-2) + 2 = 16

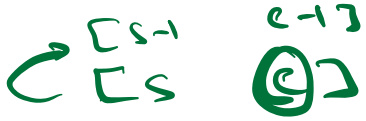
ans = 16

Window → subarray of fixed size

```
int ans = -∞ / INT_MIN
s = 0   e = k-1
int sum = 0
for (i = s ; i <= e ; i++) {
    sum += arr[i]
}
```

k

```
ans = max(ans, sum);
s++   e++
```



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

```
while (e < N) <     →  N - K
|
|   sum = sum + arr[e]  - arr[s-1])
|   ans = max (ans, sum);
|   s++   e++
}
```

N - K + K

TC : O(N)
SC : O(1)

---

1. Subarray → cont part of array
   s and e

2. No. of subarrays $= \frac{N * (N+1)}{2}$

3. Print all subarrays   TC : $O(N^3)$

4. Compute every subarray's sum

   PF ↙ ↘ carry forward

   PF: TC : $O(N^2)$, SC : $O(N)$

   Carry forward: TC : $O(N^2)$, SC : $O(1)$

5. window / subarray size is fixed → sliding window

6. total of all subarray sum

element → contribution →

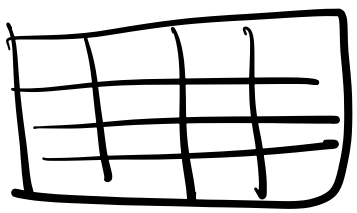
A[i] * (i+1) * (N-i)

2D matrices [ ]

[ ]

[ ]

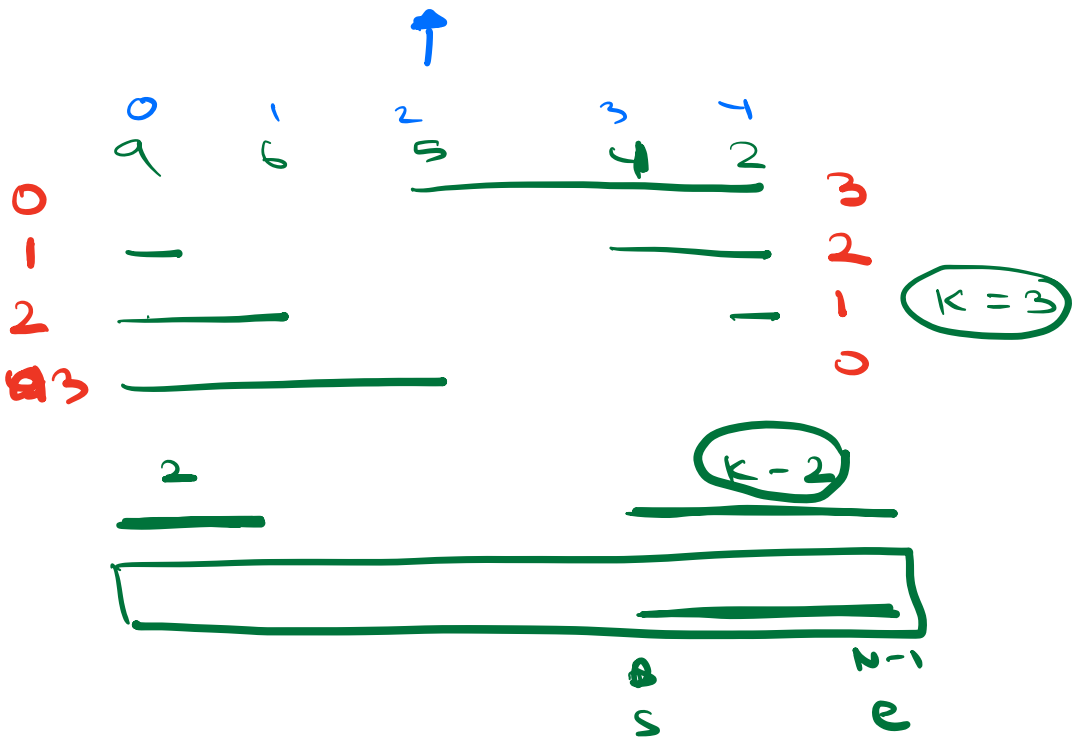
[ ]

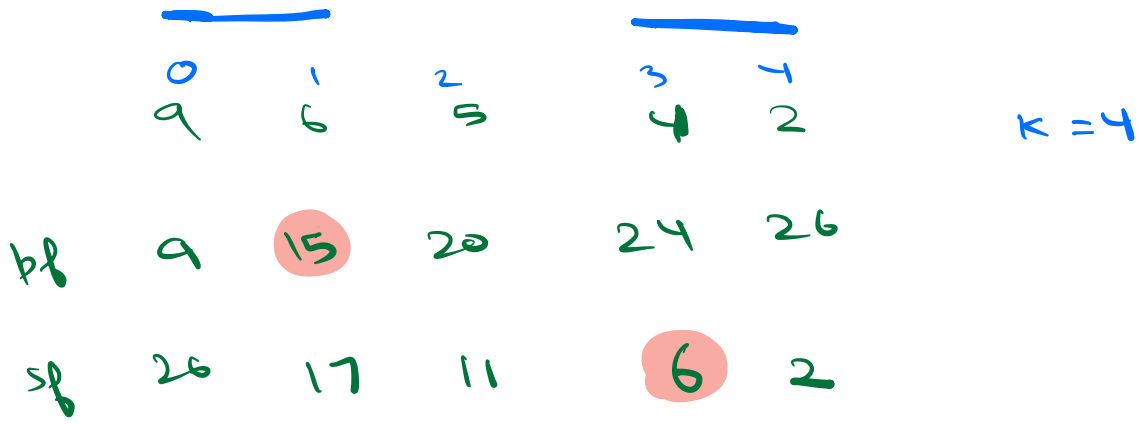


[a b]

↓

b - a + 1





cnt

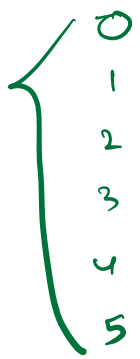
ans[cnt] = ans[k]

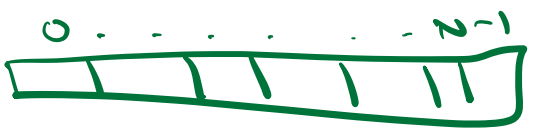
[1, 2, 3]

rows → 6

[len] [size]

list<int> ans[len]





bigint

int
long
float
double
bigint

long
sum += (long) A[i] * (i + 1) * N;
int int int

int

int a = 10^9;