Subarrays

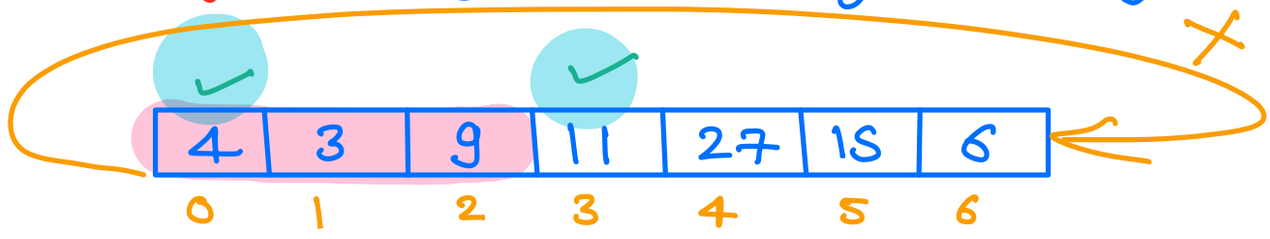
Vinay Neekhra

Senior Instructor & Mentor

Reachable in Scaler Lounge

"Life is a series of subarrays;
it's all about how you slice and dice your experiences"

§ Subarray : Contiguous part of an array



(0,2)
subarray
[4,3,9]

elements 4, 3, 9 ✓

4, 11 ✗ (not a subarray because 4,11 are not neighbor)
↓
[index difference should be 1]

27, 15 ✓

6 ✓

4, 6 ✗

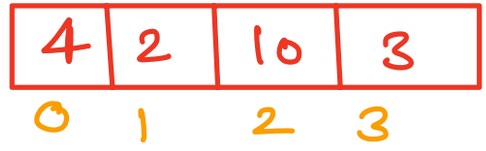
4, 3, 9, 11, 27, 15, 6 ✓

6, 4 ✗

uniquely represent a subarray:

start index, end index

Quiz: subarrays of



S = 0
[4]
[4,2]
[4,2,10]
[4,2,10,3]
#count = 4

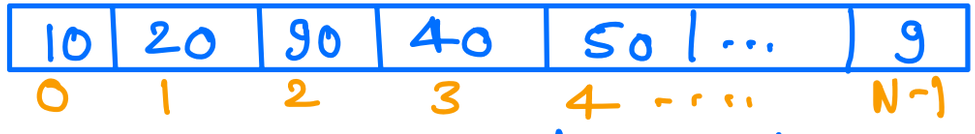
S = 1
[2]
[2,10]
[2,10,3]
#count = 3

S = 2
[10]
[10,3]
#count = 2

S = 3
[3]
#count = 1

#total subarrays = 4+3+2+1 = 10

§. Count of total subarrays in an array of size N.



Subarrays starting from index 0 → N
index 1 → N-1 +
index 2 → N-2 +
index N-1 → 1 +

count of subarrays = $\frac{N*(N+1)}{2}$

Doubt:

meaning of |A| $1 <= |A| <= 10^5$?

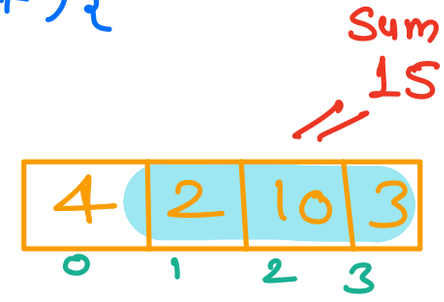
A - Array [Size of the array A]

Q. print subarray (starting index, end index)

```
printSubarray (int start, int end) {
    for ( i = start ; i <= end ; i++) {
        print ( A[i])
    }
}
```

Q. sum of subarray

```
int sumOfSubarray (int start, int end) {
    Sum = 0
    for (i = start ; i <= end ; i++) {
        sum += A[i]
    }
    return sum;
}
```

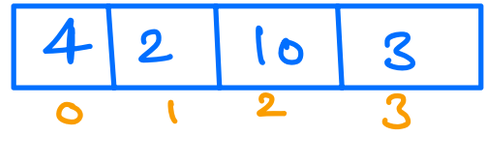


s = 1
e = 3

Sum ~~0~~ ~~2~~ ~~12~~ 15
i ~~1~~ ~~2~~ ~~3~~ 4

Q. print all possible subarrays.

subarrays of



s = 0	end	s = 1	e	s = 2	e	s = 3	e
[4]	0	[2]	1	[10]	2	[3]	3
[4,2]	1	[2,10]	2	[10,3]	3		

[4,2,10] 2 [2, 10,3] 3
[4,2, 10,3] 3

```
void printallSubarrays (A){
    for(i=0; i< N; i++){
        for(J= i ; J<N; J++){
            for (K= i; K<=J; K++){
                print (A[K])
            }
        }
    }
}
```

TC = O(N^3)
SC = O(1)

10	20	30
0	1	2

N = 3

i 0 1
J 0 1 2 3 1
K 0 1 0 1 2 0 1 2 3

$ 10 10 20 10 2030 ....

Q Find sum of each & every subarray

A =

3	2	-1	5
0	1	2	3

Prefix Sum

3	5	4	9
0	1	2	3

s	e	subarray	Sum
0	0	[3]	3
0	1	[3, 2]	5
0	2	—	4
0	3		9
1	1	[2]	2
1	2		1
1	3		6
2	2		-1
2	3		4
3	3		5

```
void printAllSubarrays(A){
    for(i=0; i<N; i++){
```

```
for (i = 0; i < N; i++) {
    for (j = i; j < N; j++) {
        int sum = 0
        for (k = i; k <= j; k++) {
            sum += A[k]
        }
        print (sum)
    }
}
```

T.C. $O(N^3)$
S.C $O(1)$

Better soln: use prefix sum

```
// create prefix Sum array
    pSum [N]

void printallSubarrays (A) {
    for(i = 0; i < N; i++) {
```

T.C = O(N)
S.C = O(N)

```
        for (J = i ; J < N ; J++)
            // i,J gives the boundry of the subarray
        if (i == 0)  sum = pSum [J]
        else         sum = pSum [J] - pSum [i-1]
        print (sum);
    }
}
```

A =

3	2	-1	5
0	1	2	3

Prefix Sum

3	5	4	9
0	1	2	3

i 0 1 2

J 0 1 2 3 1 2 3 2

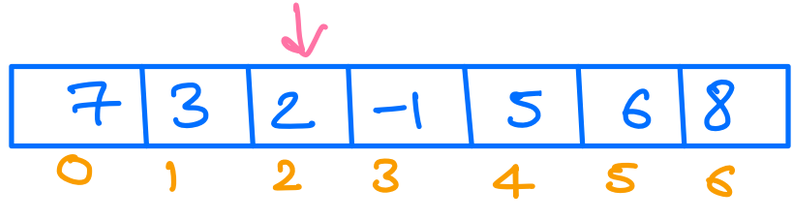
3 5 4 9 2 1 6 -1 ...

T.C = $O(N^2)$ [$O(N)$ ← prefix sum + $O(N^2)$ ← iterating subarrays]

S.C = $O(N)$ ← prefix sum array

subarrays ≈ $O(N^2)$

→ optimise the space:

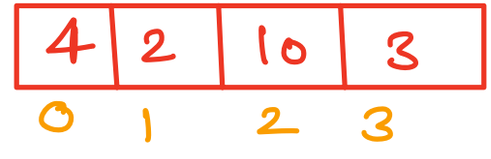


Obsⁿ

printing the sum of all subarrays starting from index 2

i	J	sum
2	2	A[2]
2	3	A[2] + A[3]
2	4	A[2] + A[3] + A[4]
2	5	A[2] + A[3] + A[4] + A[5]
2	6	A[2] + A[3] + A[4] + A[5] + A[6]

```
for (i = 0; i < N; i++) {
    sum = 0
    for (J = i; j < N; J++) {
```



i → 1

J 0 1 2 3 4 1 2 3

sum 0 4 6 16 19 0 2 12 15

```
        sum += A[j]
        print(sum)
    }
}
```

$ 4 6 16 19 2 12 15

T.C = $O(N^2)$
S.C = $O(1)$

most optimal complexity

Q. Find total sum of all subarray sums.

A =

3	2	-1	5
0	1	2	3

s	e	subarray	sum
0	0	[3]	3 +
0	1	[3, 2]	5 +
0	2	...	4 +
0	3	...	9 +
1	1	[2]	2 +

1	2
1	3
2	2
2	3
3	3

1	+
6	+
4	+
5	+

count of sum of all subarrays
= 38

```
total sum = 0
for (i = 0; i < N; i++) {
    sum = 0
    for (j = i; j < N; j++) {
        sum += A[j]
        total sum += sum
    }
}
print(totalsum)
```

T.C = $O(N^2)$
S.C = $O(1)$

↓

find a soln with better time complexity

10:47

→ better Soln:

H.W. Prefix Sum of prefix Sum ⇒ ans ?

Obsn: If we need to reduce time complexity from $O(N^2)$ ⇒ we can not visit each & every subarray

→ time complexity has to be equal or more than $O(N)$

A

−1	3	4
0	1	2

i	j	elements		Sum
0	0	A[0]	=	−1
0	1	A[0] + A[1]	=	2
0	2	A[0] + A[1] + A[2]	=	6
1	1	A[1] +	=	3
1	2	A[1] + A[2]	=	7

2 2

$A[2] = 4$

$3*A[0] + 4*A[1] + 3*A[2] = 21$

$-3 + 12 + 12$

Obⁿ : total count is sum of no of occurance of each element in subarray]*element

A :

4	-1	2	3
0	1	2	3

i	j	elements					
0	0	4				→	4
0	1	4	-1			→	3
0	2	4	-1	2		→	5
0	3	4	-1	2	3		8
1	1		-1				-1
1	2		-1	2		↓	1
1	3		-1	2	3		4
2	2			2			2
2	3			2	3		5

3 3 3 3 ⇒ 34

total sum = 4*(4) + 6*(-1) + 6*(2) + 4*3

= 16 − 6 + 12 + 12 = 34

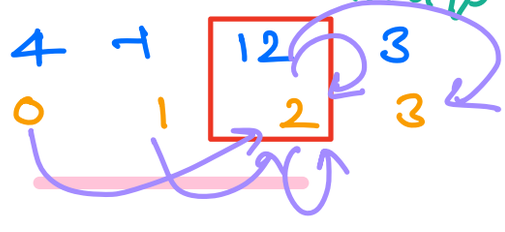
Obsⁿ: In how many subarrays the element is present

A :

4	-1	12	31
0	1	2	3

How many times 12 has come up in the subarrays

Total subarrays ⇒ 4*5/2 = 10

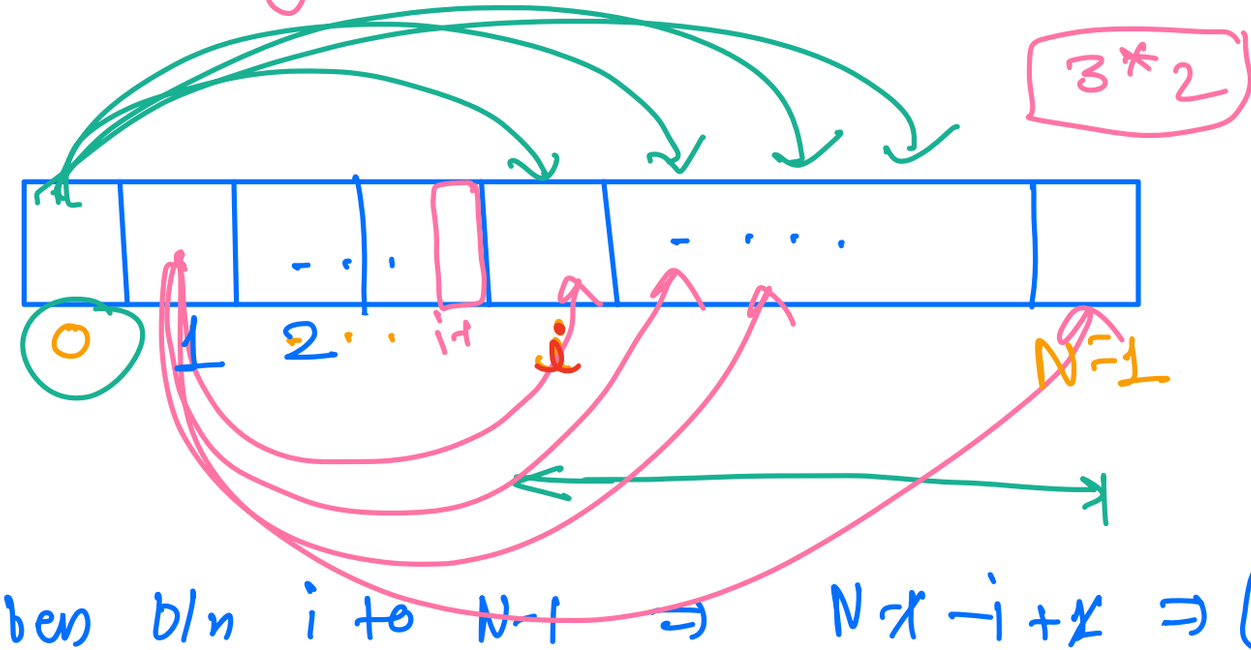


S 0 1 2
e 2 3

⇒ 6 times.

Obⁿ : starting index has to be from [0 to 2] ⇒ 3

: ending index has to be [2, 3] ⇒ 2



number b/n i to N-1 ⇒ N-1-i+1 ⇒ (N-i)

Obsⁿ:

① starting from 0th index we have (N-i) subarrays which would include ith element.

② starting from 1th index ⇒ (N-i)

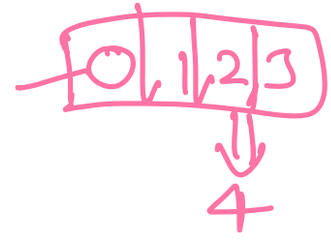
2^{nd} index $\Rightarrow (N-i)$ +

⋮

$j-1^{th}$ index $= (N-i)$ +

i^{th} index $= (N-i)$ +

$(i+1)^{th}$ index $= 0$



from all the indices $= (i+1) * (N-i)$

Subarrays which contains i^{th} element

$\Rightarrow$ Contribution of A[i] in ~~total~~ subarray sum

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

```
for (i=0; i<N; i++) {
    Sum += (i+1)*(N-i)*A[i]
```

}

T.C. = O(N)
S.C = O(1)

Contribution Technique

Doubt session:

why modulo → ans > 10^9+7

10^9+7 → int
→ prime
→ $>10^9+7$

10^{10} → int ✗

$10^{\cancel{10}}$ → long, long long

ans → ans % (10^9+7)

$>10^9+7$ → $0 - 10^9+6$

2×10^9 → nil

(right ans) $_{\%Mod}$ = (wrong ans) %. Mod

↓

taking modulo with a prime no.

⇒ least probability of collisions.

Fermat's theorem

$a^{p-1} \% p = 1$

10^9+11 — prime ? ✓ %

10^9+7 — prime

int

10^9+x

↓

1000000007

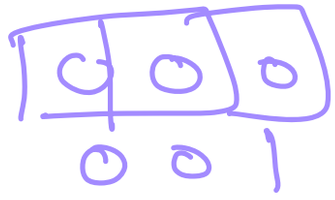
int → long int

(0 - 1 ∞)

[ ]

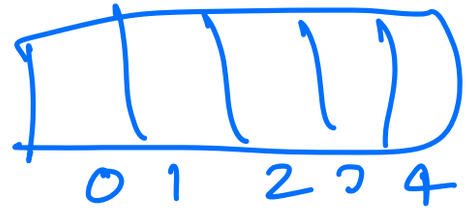
10^1+9

⇒ 19 ↑ 2 digit

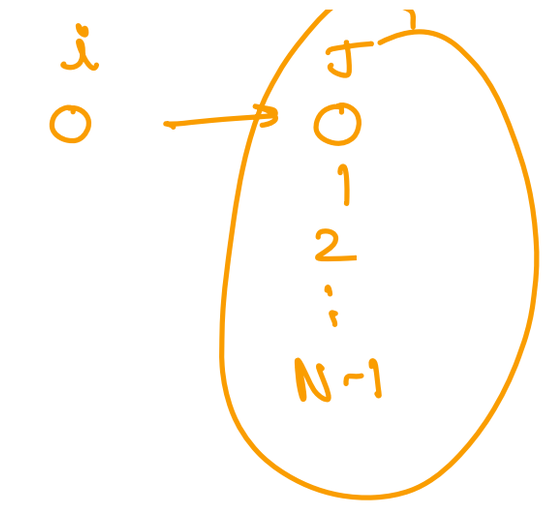


Amazing subarrays

```
void printAllSubarrays(A){
    for(i=0; i<N; i++){
        for(j=i; j<N; j++){
            int sum=0
            for(k=i; k<=j; k++){
                sum += A[k]
            }
            print(sum)
        }
    }
}
```



#iteration table



iterations

1

2 +

3 +

N

N*(N+1)/2 iteration

1

2

N-1 ⇒ $N*\frac{(N-1)}{2}$

2

1

$= O(N^3)$