Subarray content

- → Subarray Basics
- → Printing Subarray
- → Generating all Subarrays
- → Printing all Subarray sums
  - Approach1 Approach2
  - Max Subarray sum
  - Sum of all Subarray Sums
    - Approach2

[Max subarray sum with len = k.] → Next Session

## Subarray Basics

- Continous part of an array is called Subarray
- Single Element is an subarray
- Complete array is Subarray
- Empty array [] not subarray.

Ex:

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

Indices : [3 4 5 7 8] : 6 is missing, not continous

Indices : [4 5 6 7 8] : subarray. [4 - 8] → all indices from 4 to 8

↳ len : 8-4+1 = 5

Indices : [2 - 6] : Subarray : 2 3 4 5 6

↳ len = 6-2+1 = 5

Indices : [s - e] = len = e-s+1

// Print the given subarray

```
printsub(ar[], s, e) {
    for(i = s; i <= e; i++) {
        print(ar[i])
    }
}
```

TC : O(N)

SC : O(1)

// How many subarrays?

6, 9, 12, 8, 7, 10, 11, 5, 16, 15,

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Eg1: ar[4] = | -1 | 0 | 3 | 4 |

→ 4+3+2+1 = 10

Subarrays start at ind = 0

[0 0] : {-1}
[0 1] : {-1 0}
[0 2] : {-1 0 3}
[0 3] : {-1 0 3 4}

4

Subarray start ind = 1

[1 1] : {0}
[1 2] : {0 3}
[1 3] : {0 3 4}

3

Subarray start ind = 2

[2 2] : {3}
[2 3] : {3 4}

2

Subarray start ind = 3

[3 3] : {4}

1

Given N Elements, How many subarrays?

ar[N] : { 0 1 2 3 4 .. i i+1 ... N-2 N-1 }

Start ind : 0

[0 0]
[0 1]
[0 2]
[0 3]
:
[0 N-1]

N

Start ind = 1

[1 1]
[1 2]
[1 3]
:
[1 N-1]

N-1

Start ind = 2

[2 2]
[2 3]
[2 4]
:
[2 N-1]

N-2

Start ind = i

[i i]
[i i+1]
[i i+2]
:
[i N-1]

N-i

Subarr at ind N-1

[N-1, N-1] = 1

Total subarrays

N + N-1 + N-2 .. 1

1 + 2 + 3 + ... N

$\frac{(N)(N+1)}{2}$

// Print all Subarray

```
↦ start of subarray
for(i = 0; i < N; i++){
    ↦ end of subarray.
    for(j = i; j < N; j++){
        for(k = i; k <= j; k++){
            print(ar[k])
        }
        print('\n') // newline
    }
}
```

N = 4: All Subarrays

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| ar[4]: | 2 | -1 | 3 | 1 |

[0 0] : {2}
[0 1] : {2, -1}
[0 2] : {2, -1, 3}
[0 3] : {2, -1, 3, 1}
[1 1] : {-1}
[1 2] : {-1, 3}
[1 3] : {-1, 3, 1}
[2 2] : {3}
[2 3] : {3, 1}
[3 3] : {1}

TC: $O(N^3)$ SC: $O(1)$

// Subarrays * [For each subarray we need to iterate printing]

$O(N^2)$ $O(N)$

overall TC: $O(N^3)$ SC: $O(1)$

28) Given N array element print all subarray sums

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| ar[4] : | 2 | -1 | 3 | 1 |

[0 0] : 2
[0 1] : 1
[0 2] : 4
[0 3] : 5
[1 1] : -1
[1 2] : 2
[1 3] : 3
[2 2] : 3
[2 3] : 4
[3 3] : 1

```
↦ start of subarray
i = 0; i < N; i++){
    ↦ end of subarray
    j = i; j < N; j++){
        // [i j]
        sum = 0
        k = i; k <= j; k++){
            sum = sum + ar[k]
        }
        print(sum)
    }
}
```

**Approach : 1**

TC : $O(N^3)$ SC : $O(1)$

**Approach : 2**

Un prefix sum to optimize

TC : $O(N + N^2 * O(1))$

Pf() getting all subarray sums

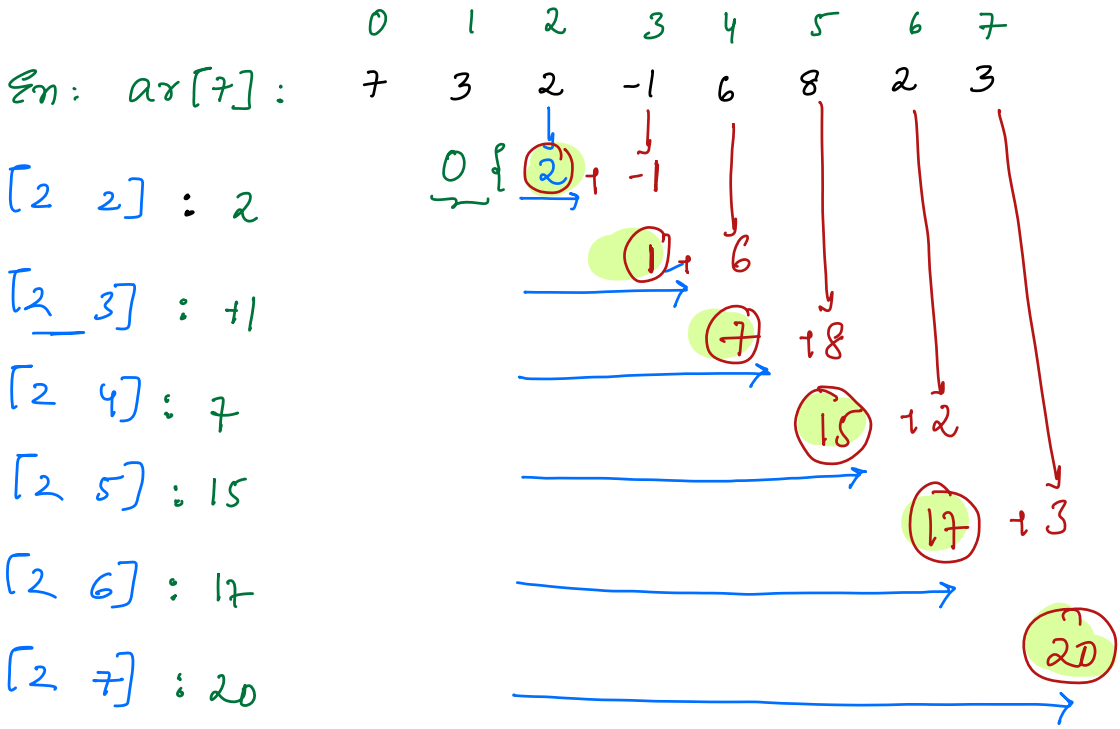
SC : $O(N)$

**Approach : 3**

By using carry forward

TC : $O(N^2)$ SC : $O(1)$

Printing all subarray sums starting at index 2




```
sum = 0
for (j = 2; j < N; j++) {
    sum = sum + ar[j]
    print(sum)
}
```

Printing all subarray sums starting at index 2

// Printing all subarray sums using carry forward.

```
for(i = 0; i < N; i++) {  -> i -> [0, N-i]
    sum = 0
    for(j = i; j < N; j++) {
        sum = sum + ar[j]
        print(sum)
    }
}
```

Print all subarray sums starting at index i

TC: $O(N^2)$ SC: $O(1)$

ar[]: 2 4 5 7

i = 0 :

| | j = 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| sum = 0 | 2 | 6 | 11 | 18 |
| | [0 0] | [0 1] | [0 2] | [0 3] |

print(2) print(6) print(11) print(18)

// Given N array element print max subarray sum

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| ar[4]: | 2 | -1 | 3 | 1 |

ans = 5

[0_0] : 2
[0_1] : 1
[0_2] : 4
[0_3] : 5
[1_1] : -1
[1_2] : 2
[1_3] : 3
[2_2] : 3
[2_3] : 4
[3_3] : 1

```
maxSum = ar[0] / INT_MIN
for(i = 0; i < N; i++) {
    sum = 0
    for(j = i; j < N; j++) {
        sum = sum + ar[j]
        maxSum = max(maxSum, sum)
    }
}
```

TC: O(N^2)  SC: O(1)

// To get max subarray sum → TC: O(N)  SC: O(1)

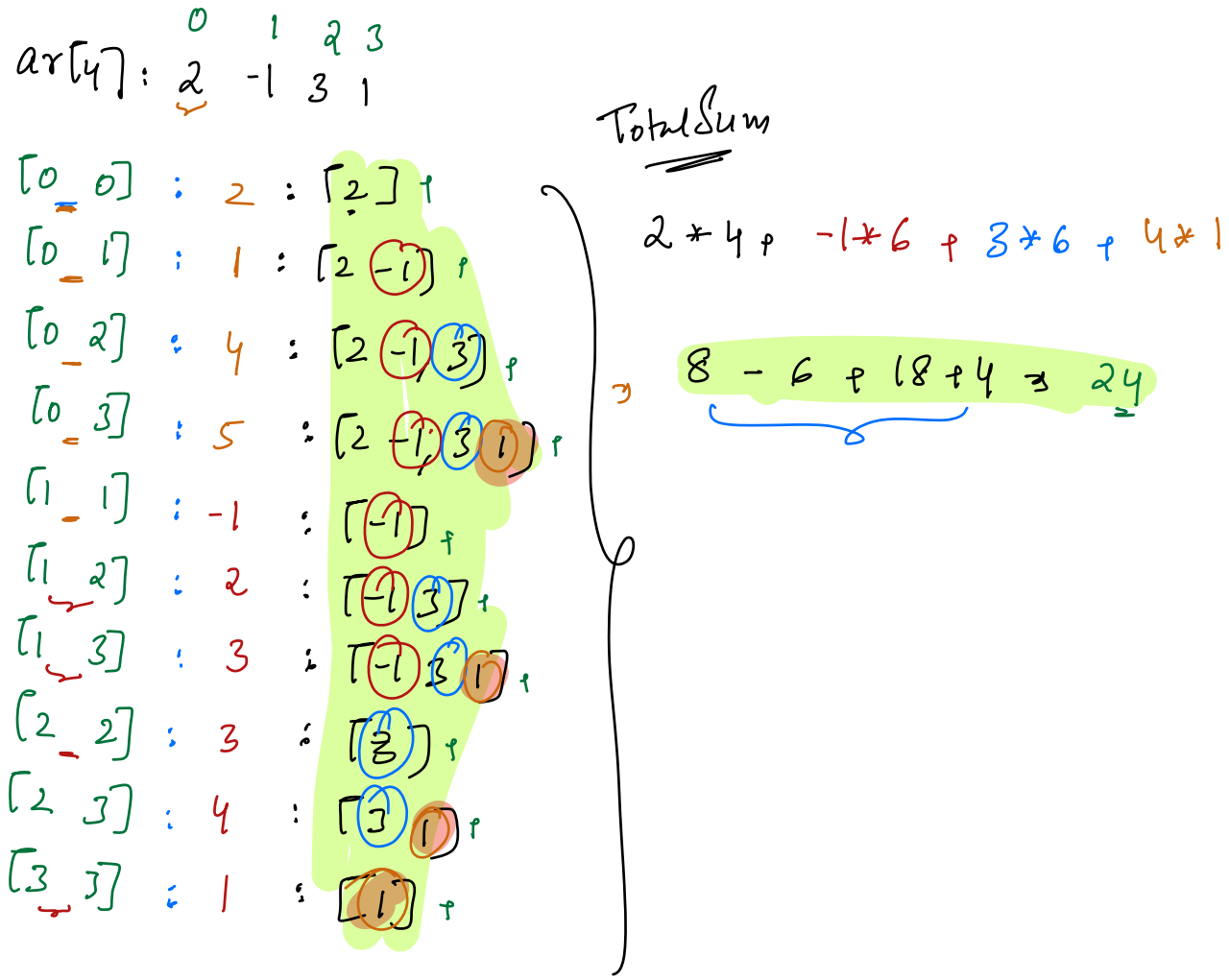
3@) Given N array element return **Sum of All Subarray Sums**

ar[4]: 0 1 2 3 → { 2 -1 3 1 } return 24

[0_0] : 2
[0_1] : 1
[0_2] : 4
[0_3] : 5
[1_1] : -1
[1_2] : 2
[1_3] : 3
[2_2] : 3
[2_3] : 4
[3_3] : 1

```
// totalSum = 0
for(i = 0; i < N; i++)
    sum = 0
    for(j = i; j < N; j++){
        sum = sum + ar[j]
        // every sum is indicating a subarray sum.
        totalSum = totalSum + sum
    }
return totalSum;
```

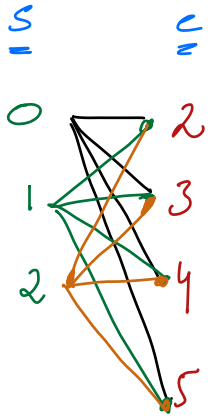
TC ⇒ O(N²)  SC ⇒ O(1)


0 1 2 3
ar[4] : 2 -1 3 1
[0 0] : 2 : [2]
[0 1] : 1 : [2 -1]
[0 2] : 4 : [2 -1 3]
[0 3] : 5 : [2 -1 3 1]
[1 1] : -1 : [-1]
[1 2] : 2 : [-1 3]
[1 3] : 3 : [-1 3 1]
[2 2] : 3 : [3]
[2 3] : 4 : [3 1]
[3 3] : 1 : [1]
TotalSum
2*4 + -1*6 + 3*6 + 4*1
8 - 6 + 18 + 4 = 24

// In how many subarrays index 2 is present

// Given ar[6].

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| -2 | 3 | 1 | 2 | 5 | 7 |




12 pairs

We can start : 3

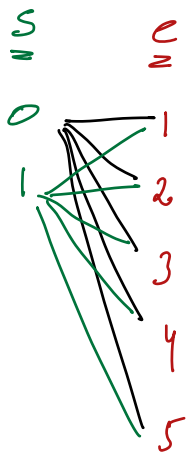
We can end = 4

} => 12 pairs

// In how many subarrays index 1 is present

// Given ar[6].

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| -2 | 3 | 1 | 2 | 5 | 7 |




10 pairs

We can start = 2

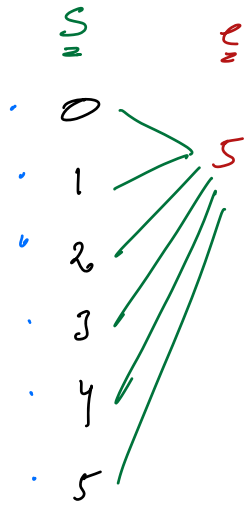
We can end = 5

} 10 pairs

// In how many subarrays index 5 is present

// Given ar[6]:

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| -2 | 3 | 1 | 2 | 5 | 7 |




} We can start = 6
We can end = 1 } 6 pairs

// Given N array elements, in how subarrays index i is present

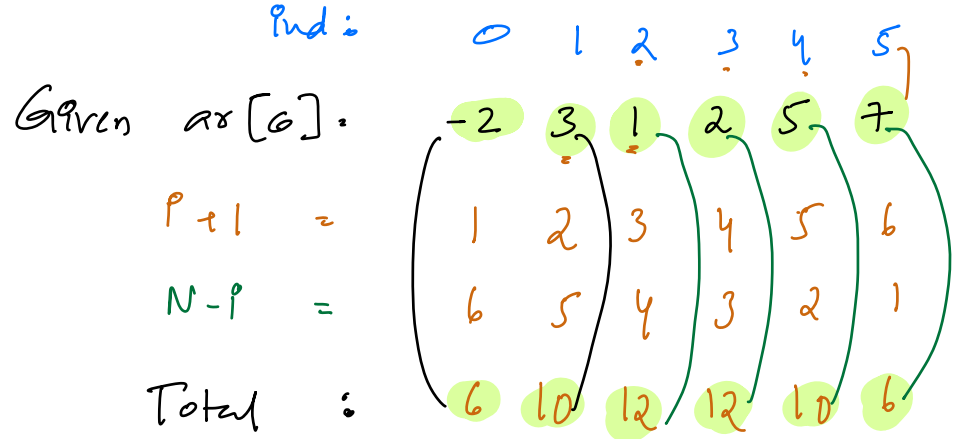
| Index: | 0 | 1 | 2 | .. | i-1 | i | i+1 | i+2 | .. | N-1 |
|---|---|---|---|---|---|---|---|---|---|---|

S: [0, i] → i+1 Elements
E: [i, N-1] → N-1-i+1
→ N-i Elements

} Total subarrays in which i is present
→ (i+1)(N-i)

N = 6

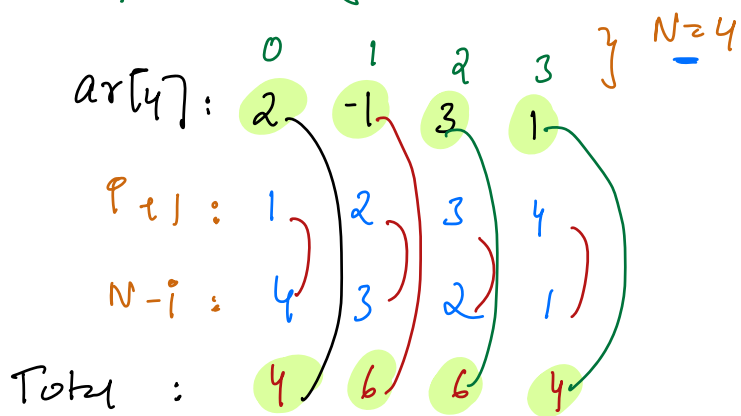
// Sum of all subarray sums.




Final sum : -12 + 30 + 12 + 24 + 50 + 42 = 146

// Sum of subarray sums

(i+1)(N-i)




Final sum : 8 - 6 + 18 + 4 = 24

// Contribution Technique → Contribution of each array element in final ans.

// Pseudocode :

→ TC : O(N) SC : O(1)

```
sum = 0

for(i = 0; i < N; i++){
    s = i+1
    e = N-i
    t = (i+1)(N-i)        // in how many subarray index i is present
    sum = sum + t * ar[i]
}

return sum.
```

// Double dimensional array ... N

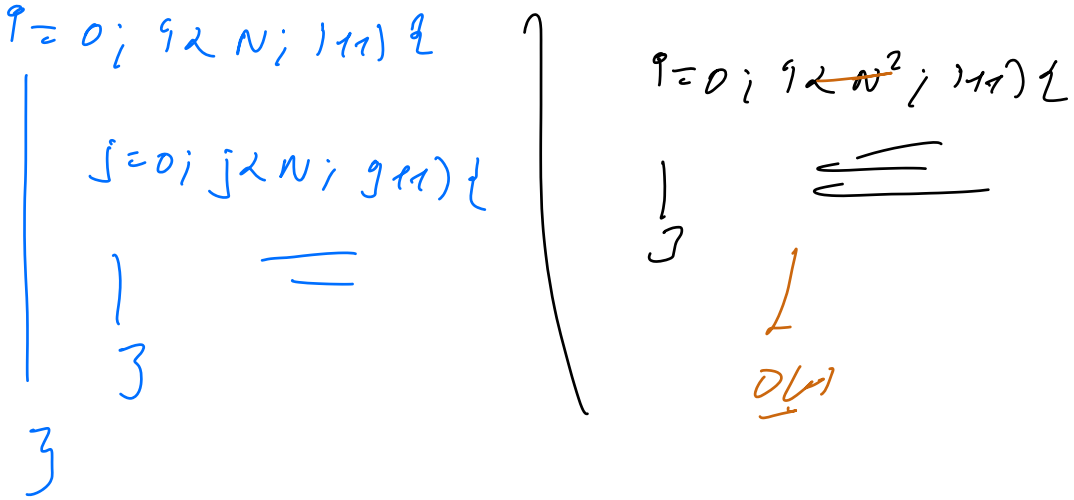
```
list<string> l;
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        l.add(i + " " + j);
    }
}
```

index: 0 1 2

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | (0 0) | (2, 2) | | |
| 1 | (0 1) | (2, 2) | | |
| 2 | (0 2) | (2, 2) | | |
| 3 | (0, 3) | | | |
| | (1, 1) | | | |
| | (1, 2) | | | |
| | (1, 3) | | | |

N2

```
for(i=0; i<l.size(); i++){
    s = l[i][0]
    e = l[i][2]
    for(k=s; k<=e; k++){
        print(k)
    }
    println()
}
```


for(i=0; i<N; i++){
for(j=0; j<N; j++){
}
}
for(i=0; i<N^2; i++){
}
O(n)