⇒ **Subarray** (sub-part of an array)

↳ **Substring**

arr = [5, 3, 2, 7]

properties :-

- → subarray is always continuous
- → subarray always built in forward direction.

arr = [5, 3, 2, 7]

subarrays :-

[5, 3] ✓

[5, 2, 7] ✗

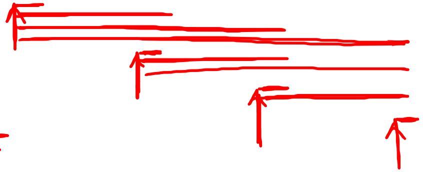
[5, 2, 3] ✗

[2, 3] ✗

[2] ✓

[5, 3, 2, 7] ✓

$$arr = [\overset{0}{5}, \overset{1}{3}, \overset{2}{2}, \overset{3}{7}] \qquad n = 4$$



subarrays

(i, j)

i :- 0 to $(n-1)$

j :- i to $(n-1)$

create loops for subarrays

indexes	
(0, 0)	[5]
(0, 1)	[5, 3]
(0, 2)	[5, 3, 2]
(0, 3)	[5, 3, 2, 7]
(1, 1)	[3]
(1, 2)	[3, 2]
(1, 3)	[3, 2, 7]
(2, 2)	[2]
(2, 3)	[2, 7]
(3, 3)	[7]

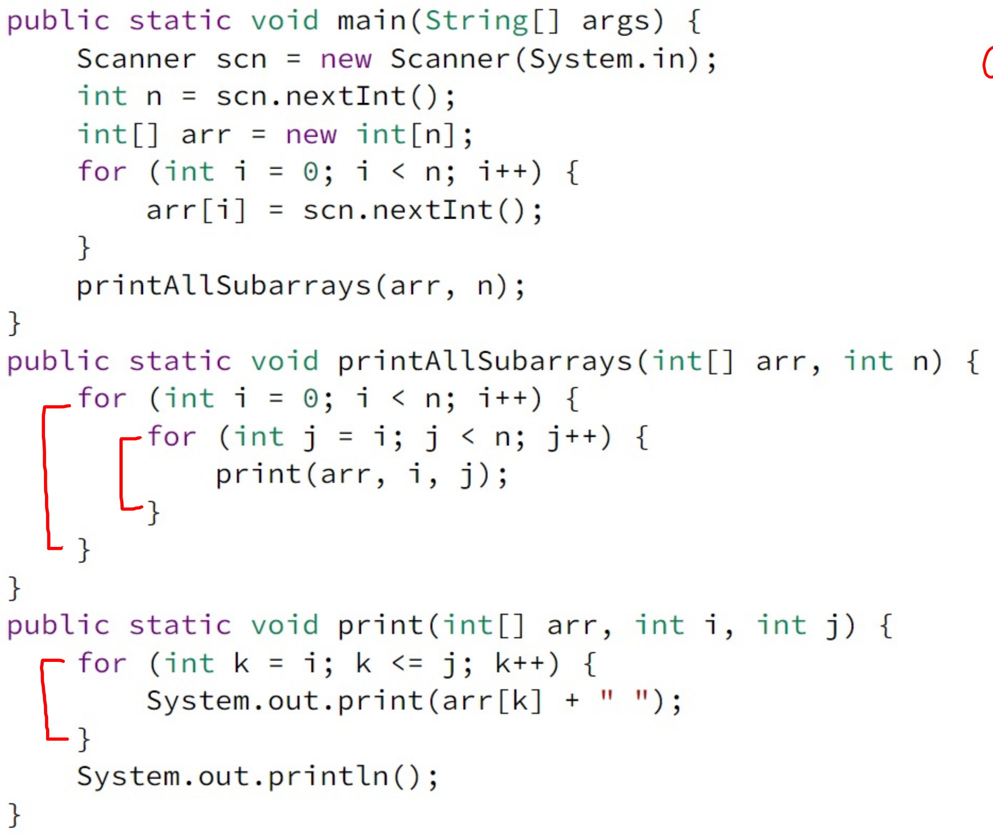
psudo code

```
for( int i=0 ; i<n ; i++){
    for(int j=i ; j<n ; j++){
        print( arr, i, j);
    }
}
```

print

```
for(int k = i ; k<=j ; k++){
    Sys.( arr[k]);
}
```

Code



```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    printAllSubarrays(arr, n);
}
public static void printAllSubarrays(int[] arr, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            print(arr, i, j);
        }
    }
}
public static void print(int[] arr, int i, int j) {
    for (int k = i; k <= j; k++) {
        System.out.print(arr[k] + " ");
    }
    System.out.println();
}
```

arr [1, 2, 3]
0 1 2

i=0, j=0   1
j=1   1 2
j=2   1 2 3
j=3 X

i=1, j=1   2
j=2   2 3
j=3 X

i=2, j=2   3
j=3 X

i=3 X

Sum Equals Zero

$arr = [\underset{0}{5}, \underset{1}{-2}, \underset{2}{3}, \underset{3}{-1}, \underset{4}{4}]$

all subarrays

sum

5 → 5

3 → 5 -2

6 → 5 -2 3

5 → 5 -2 3 -1

9 → 5 -2 3 -1 4

-2 → -2

1 → -2 3

return true 0 → $\boxed{-2 \quad 3 \quad -1}$

-2 3 -1 4

3

3 -1

3 -1 4

-1

-1 4

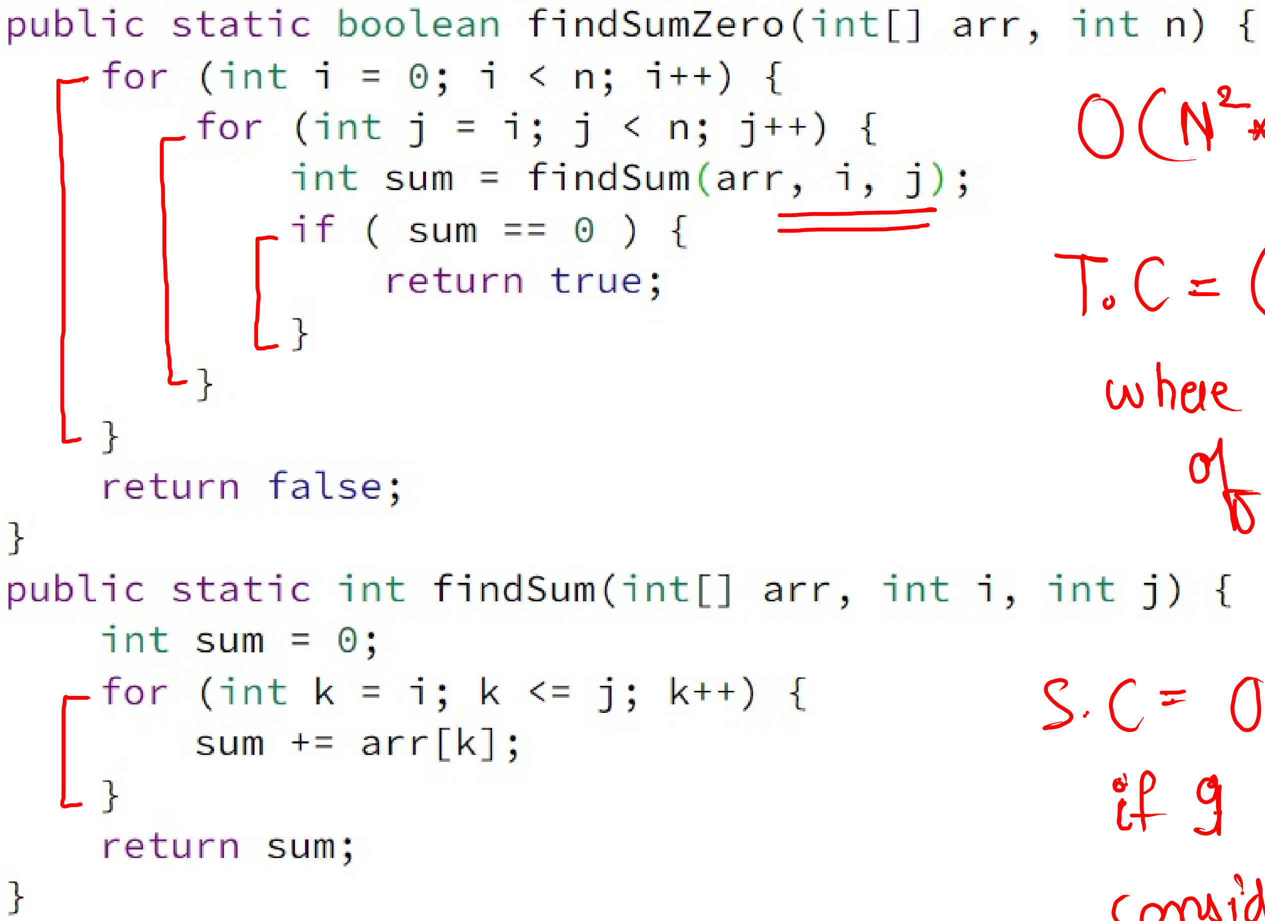
4

curr

(i, j)

$$sum = -2 + 3 + (-1)$$
$$= 0$$

Code



```
public static boolean findSumZero(int[] arr, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            int sum = findSum(arr, i, j);
            if ( sum == 0 ) {
                return true;
            }
        }
    }
    return false;
}
public static int findSum(int[] arr, int i, int j) {
    int sum = 0;
    for (int k = i; k <= j; k++) {
        sum += arr[k];
    }
    return sum;
}
```

$O(N^2 * N)$

T.C = $O(N^3)$

where N is size of array

S.C = $O(1)$

if I will not consider given i/p

Max Subarray 2

(find subarray with max sum)
in linear time

⇒ Whenever we want to find maximum sum subarray in O(N) then always we use Kadane's algo

Imp

$$arr = [\underset{0}{3}, \underset{1}{-20}, \underset{2}{4}, \underset{\substack{3 \\ \uparrow \\ i}}{7}]$$

max sum of a subarray :- $[4, 7] = 11$

$maxSum = \cancel{-\infty} \; \cancel{3} \; \cancel{4} \; 11$

$sum\_so\_far = \cancel{0} \; \cancel{3} \; \cancel{-17} \; \cancel{4} \; 11$

```
    if (sum_so_far < 0) {
a       sum_so_far = arr[i];
    } else {
b       sum_so_far += arr[i];
    }
```

```
    if( sum_so_far > maxSum){
        maxSum = sum_so_far;
    }
```

Ex:-

$$arr = [\,\overset{0}{-9},\ \underbrace{\overset{1}{5},\ \overset{2}{-1},\ \overset{3}{-1},\ \overset{4}{0},\ \overset{5}{\underset{\underset{i}{\uparrow}}{4}}}\,]$$

max sum = ~~$-\infty$~~ ~~-9~~ ~~5~~ 7

sum_so_far = ~~0~~ ~~-9~~ ~~5~~ ~~4~~ ~~3~~ ~~3~~ 7

Imp Note:-

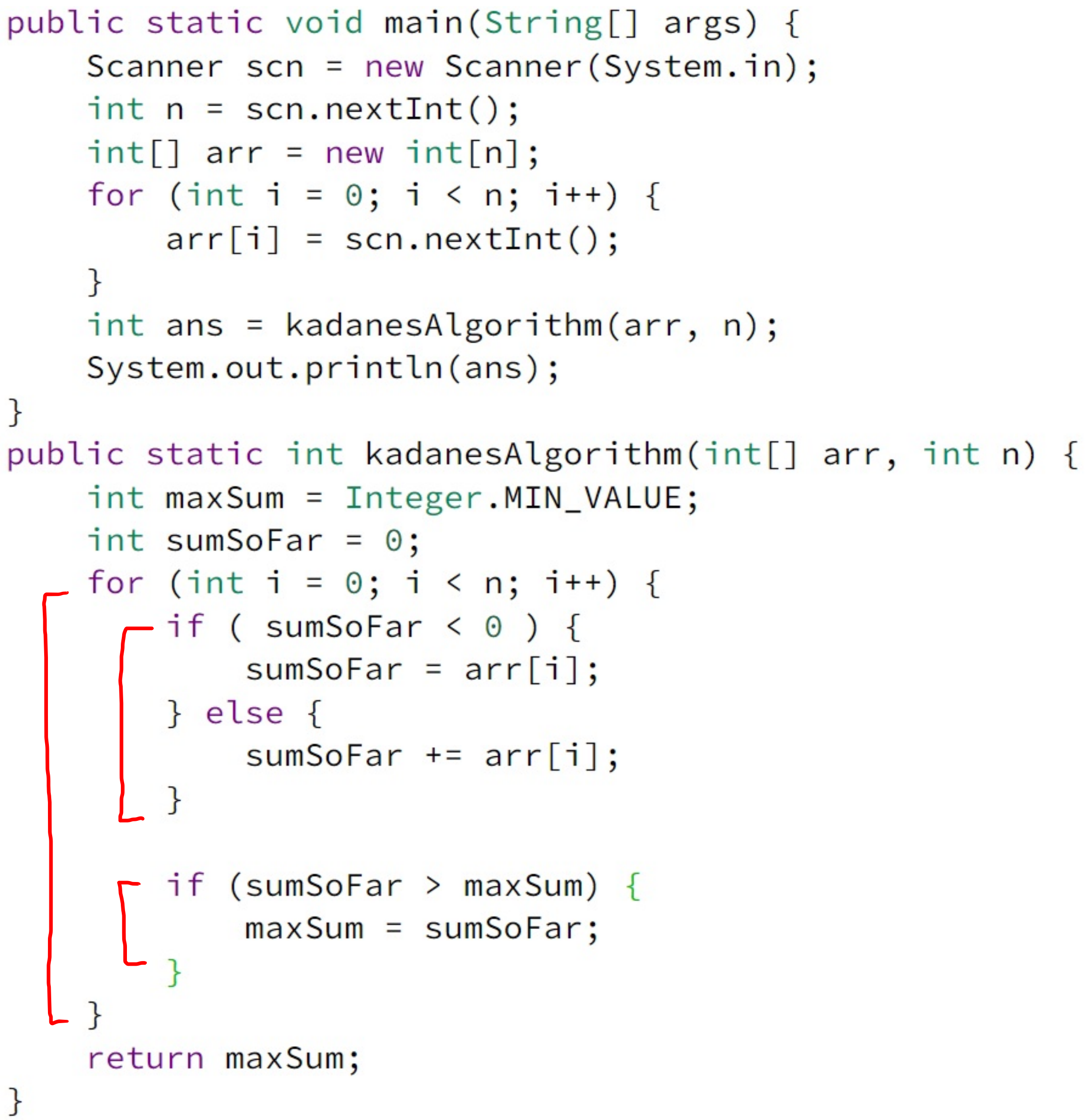
we are checking sum-so-far not current element

```
  if (sum_so_far < 0) {
a       sum_so_far = arr[i];   ← wall
  } else {
b       sum_so_far += arr[i];
  }

  if ( sum_so_far > max Sum) {
        max Sum = sum_so_far;
  }
```

code

V. Imp



```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }
    int ans = kadanesAlgorithm(arr, n);
    System.out.println(ans);
}
public static int kadanesAlgorithm(int[] arr, int n) {
    int maxSum = Integer.MIN_VALUE;
    int sumSoFar = 0;
    for (int i = 0; i < n; i++) {
        if ( sumSoFar < 0 ) {
            sumSoFar = arr[i];
        } else {
            sumSoFar += arr[i];
        }

        if (sumSoFar > maxSum) {
            maxSum = sumSoFar;
        }
    }
    return maxSum;
}
```