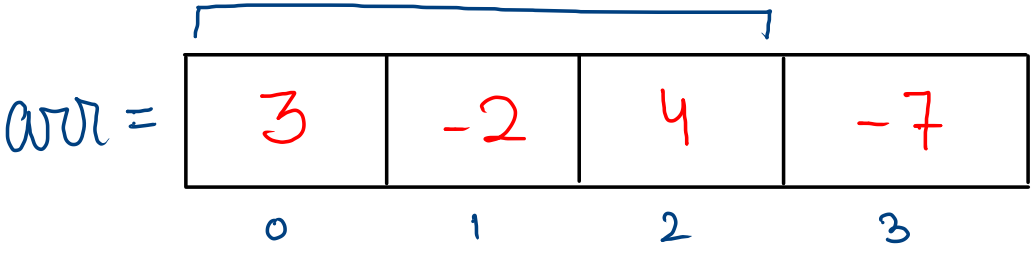
⇒ Kadane's Algorithm (Imp)

↳ used to find "maximum sum subarray" in linear time // O(N)
↳ size of array

3	-2	4	-7

(3) 3
(1) 3 -2
✓ (5) 3 -2 4
(-2) 3 -2 4 -7
(-2) -2
(2) -2 4
(-5) -2 4 -7
(4) 4
(-3) 4 -7
(-7) -7



maxSum = ~~$-\infty$~~ ~~3~~ 5

sumSoFar = ~~0~~ ~~3~~ ~~1~~ ~~5~~ -2

	element
i = 0 →	3
i = 1 →	-2
i = 2 →	4
i = 3 →	-7

```java
public static int kadanesAlgo(int[] arr, int n) {
    int maxSum = Integer.MIN_VALUE;
    int sumsf = 0;
    for (int i = 0; i < n; i++) {
        if (sumsf < 0) {
            sumsf = arr[i];
        } else {
            sumsf = sumsf + arr[i];
        }
        if ( sumsf > maxSum ) {
            maxSum = sumsf;
        }
    }
    return maxSum;
}
```

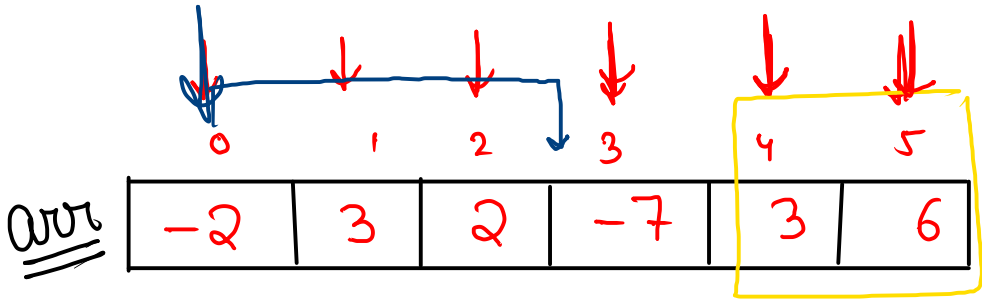
a

b

c

sumsf = Math.max(arr[i], sumsf + arr[i])

maxSum = max(maxSum, sumsf);



maxSum = ~~-∞~~ ~~-2~~ ~~3~~ 9

sumsf = ~~∅~~ ~~-2~~ ~~3~~ ~~5~~ ~~-2~~ ~~3~~ 9

Maximum Product Subarray 2

(V.V. Imp)

arr =

2	3	-2	4

ans = 6

↳ (2) 2

(1) 2 3

(-12) 2 3 -2

(-48) 2 3 -2 4

(3) 3

(-6) 3 -2

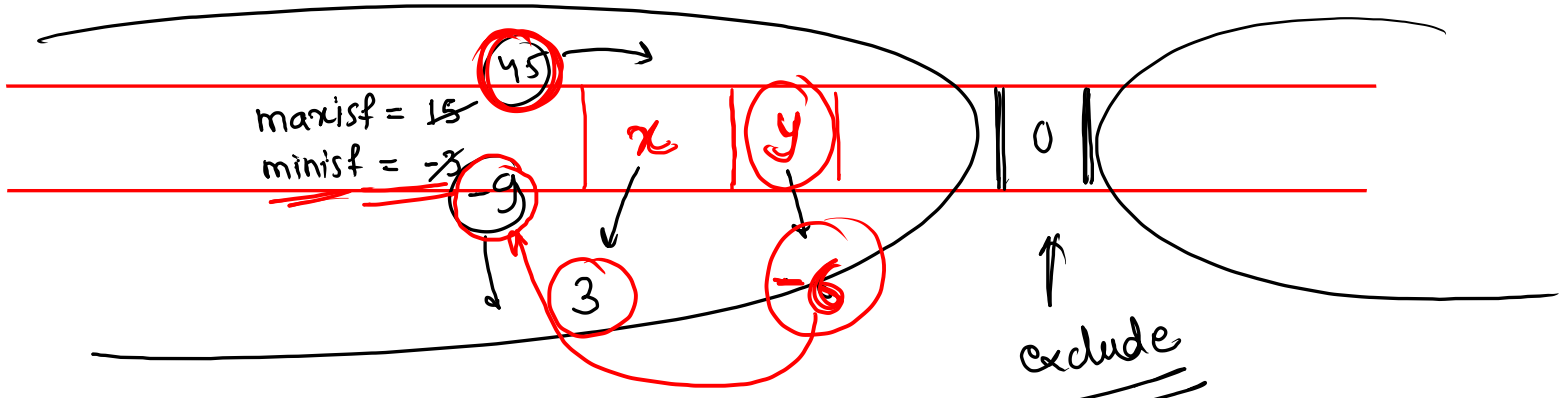
(-24) 3 -2 4

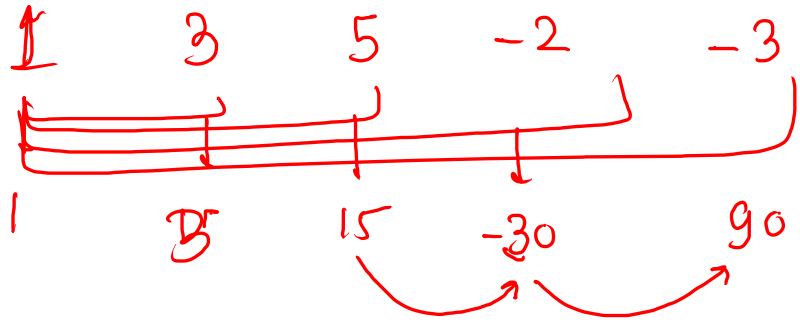
(-2) -2

(-8) -2 4

(4) 4

arr





minisf = 45 * -6 = -ve

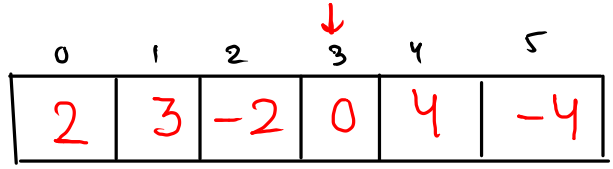
```java
public static int kadanesAlgo(int[] arr, int n) {
    int maxisf = 1;
    int minisf = 1;
    int result = 0;
    for (int i = 0; i < n; i++) {
        if ( arr[i] > 0 ) {
            maxisf = Math.max(maxisf, maxisf * arr[i]);
            minisf = Math.min( minisf * arr[i], 1 );
        } else if ( arr[i] == 0 ) {
            maxisf = 1;
            minisf = 1;
        } else {
            int temp = maxisf;
            maxisf = Math.max( minisf * arr[i], 1 );
            minisf = temp * arr[i];
        }

        if ( result < maxisf ) {
            result = maxisf;
        }
    }
    return result;
}
```

a

b

c



currEle = 2 3 -2 0 4 -4

maxsf = 1 2 6 1 1 4 1

minsf = 1 -12 1 -16

result = 0 1 2 6

temp = 6 4