→ Subarray ($O(N^2)$) quadratic

→ Kadane's Algo ($O(N)$) linear

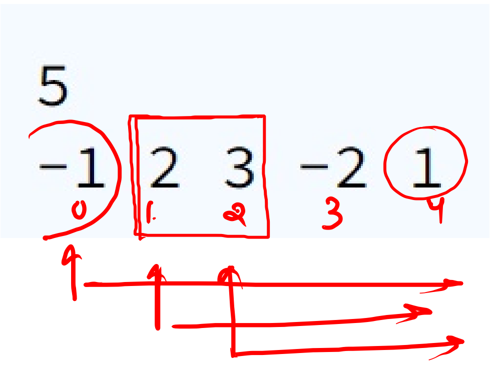
↳ used for finding "maxi sum subarray"

→ Subarray ($O(N^2)$) quadratic

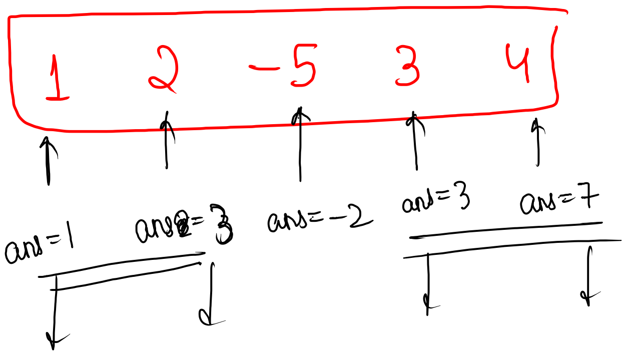
→ Kadane's Algo ($O(N)$) linear

↳ used for finding "maxi sum subarray"

dry run



```
5
-1 2 3 -2 1
```



-1 = -1

-1 2 = 1

-1 2 3 = 4

-1 2 3 -2 = 2

-1 2 3 -2 1 = 3

2 = 2

2 3 = 5

2 3 -2 = 3

2 3 -2 1 = 4

3 = 3

3 -2 = 1

3 -2 1 = 2

-2 = -2

-2 1 = -1

1 = 1

-1, 2, 3

↓

a =

0 1 2 3 4

ans = 0 , maxAns = ~~-∞~~ ~~-1~~ ~~2~~ $\boxed{5}$

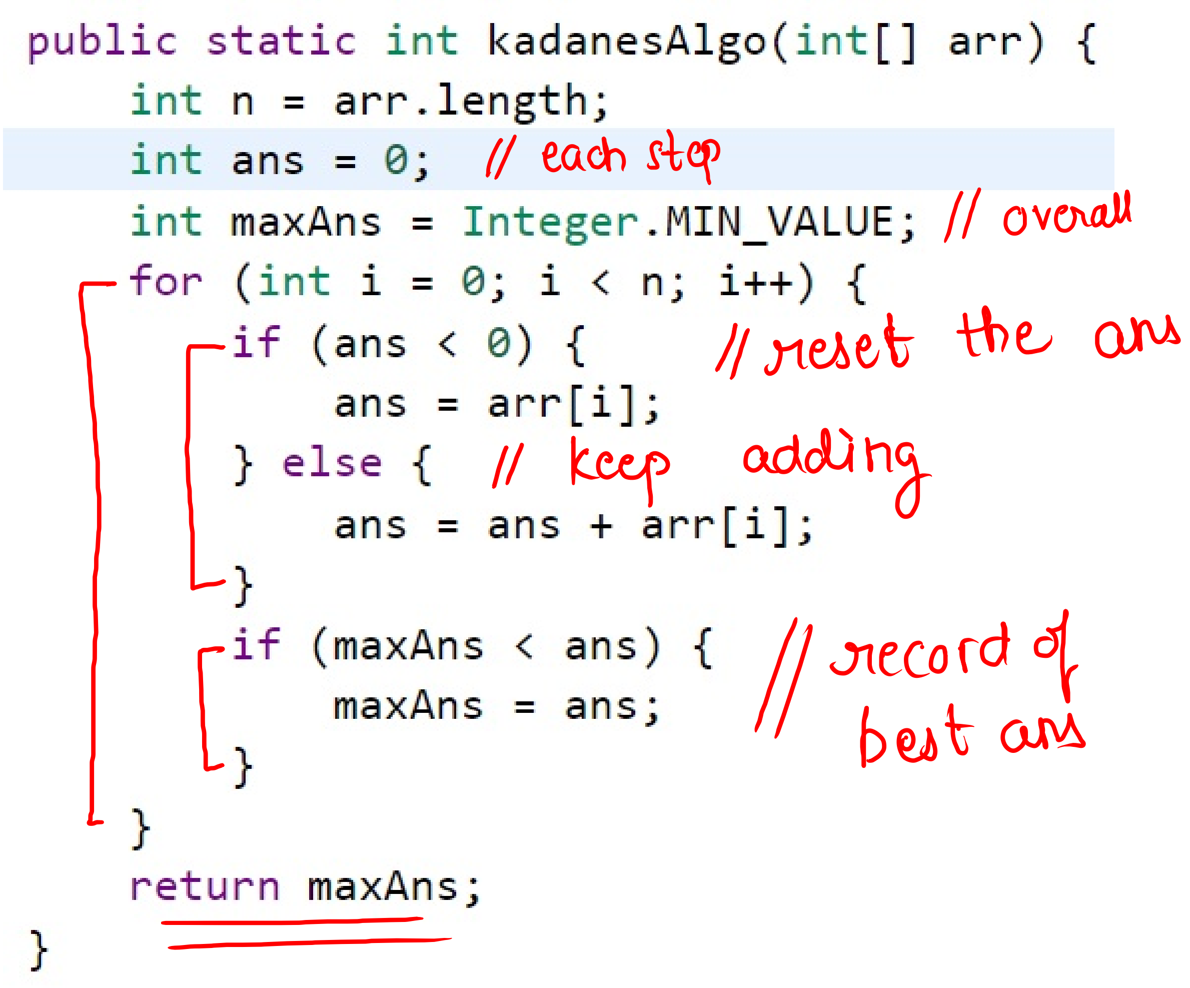
i = 0 , ans = -1

i = 1 , ans = 2

i = 2 , ans = 5

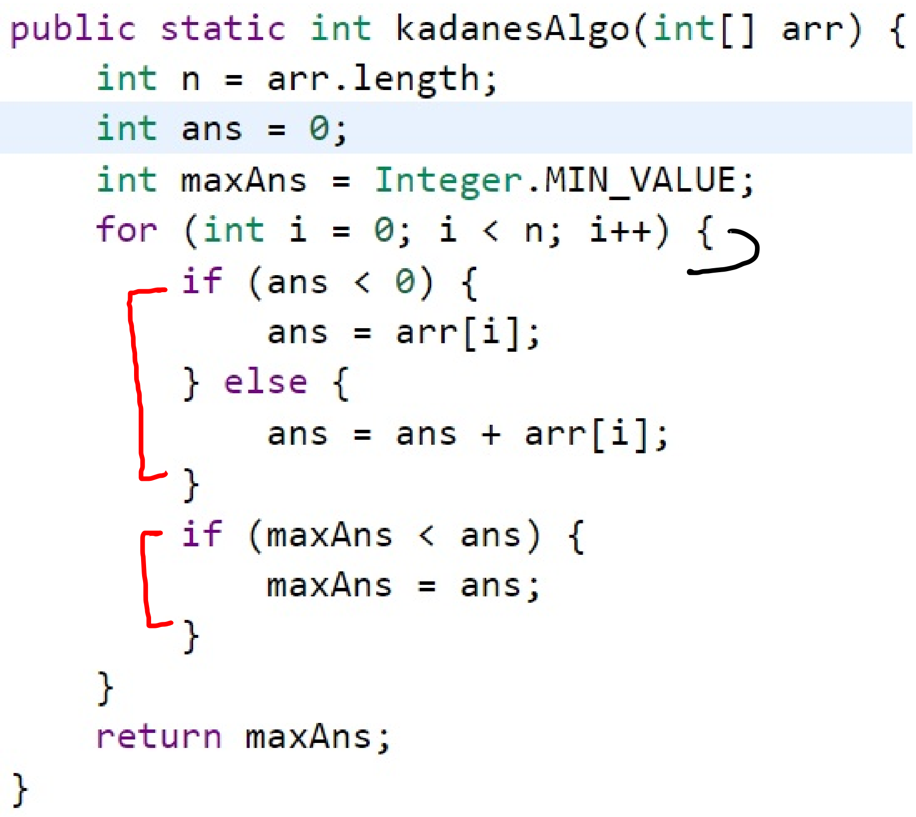
i = 3 , ans = 3

i = 4 , ans = 4



```
public static int kadanesAlgo(int[] arr) {
    int n = arr.length;
    int ans = 0;  // each step
    int maxAns = Integer.MIN_VALUE; // overall
    for (int i = 0; i < n; i++) {
        if (ans < 0) {          // reset the ans
            ans = arr[i];
        } else {  // keep  adding
            ans = ans + arr[i];
        }
        if (maxAns < ans) {   // record of
            maxAns = ans;     //   best ans
        }
    }
    return maxAns;
}
```

-1 2 , 3



```java
public static int kadanesAlgo(int[] arr) {
    int n = arr.length;
    int ans = 0;
    int maxAns = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {
        if (ans < 0) {
            ans = arr[i];
        } else {
            ans = ans + arr[i];
        }
        if (maxAns < ans) {
            maxAns = ans;
        }
    }
    return maxAns;
}
```

-2	4	3	-5	10
0	1	2	3	4

ans = 0 , maxAns = ~~-∞~~ ~~-2~~ ~~4~~ ~~7~~ 12

i=0, ans = -2

i=1, ans = 4

i=2, ans = 7

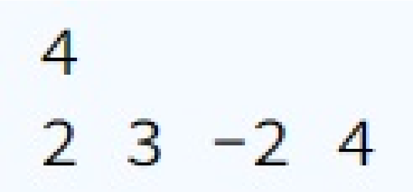
i=3, ans = 2

i=4, ans = 12

Maximum Product Subarray 2

M.M Imp

dry run



```
4
2 3 -2 4
```

(2) 2

(6) 2 3

(-12) 2 3 -2

(-48) 2 3 -2 4

(3) 3

(-6) 3 -2

(-24) 3 -2 4

(-2) -2

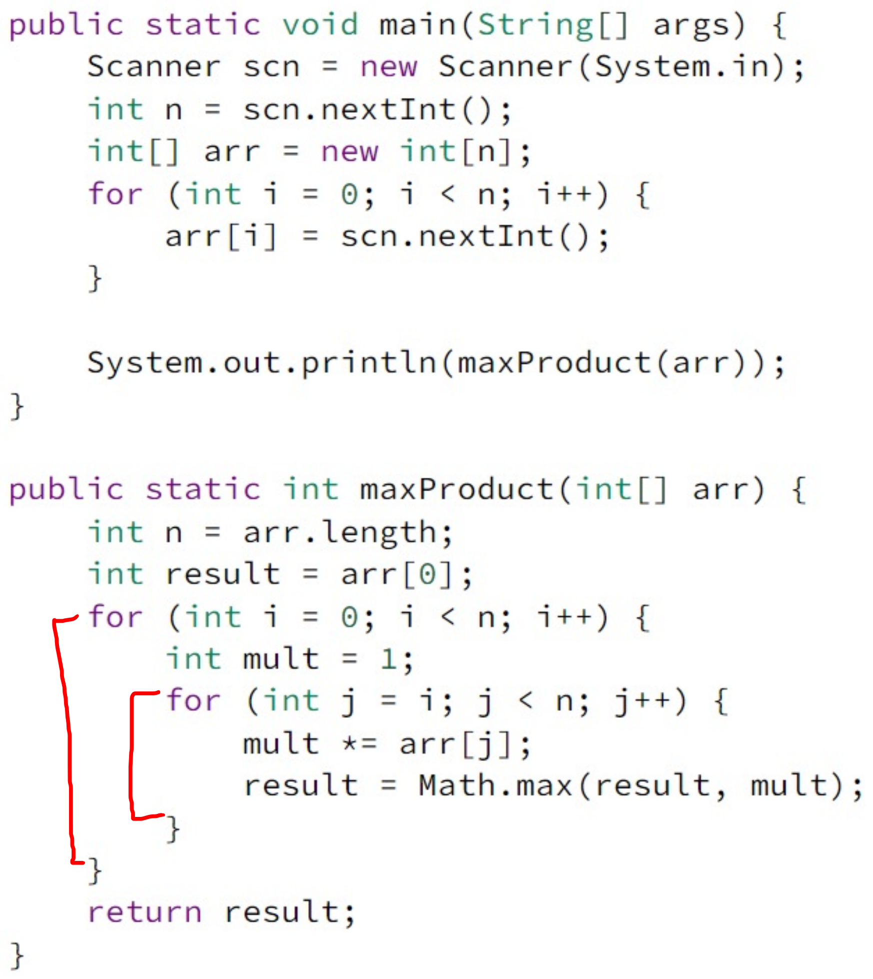
(-8) -2 4

(4) 4

Note:- +ve no. are always profitable

0, -ve no. will be profitable sometimes and non-pro. as well

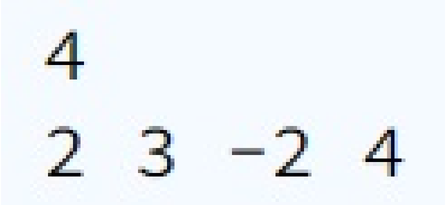
→ Simple approach with N^2 complexity



```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();
    }

    System.out.println(maxProduct(arr));
}

public static int maxProduct(int[] arr) {
    int n = arr.length;
    int result = arr[0];
    for (int i = 0; i < n; i++) {
        int mult = 1;
        for (int j = i; j < n; j++) {
            mult *= arr[j];
            result = Math.max(result, mult);
        }
    }
    return result;
}
```



```
4
2 3 -2 4
```

result = ~~2~~ 6

i=0 , j=0	mult = 2
j=1	mult = 6
j=2	mult = -12
j=3	mult = -48
i=1 , j=1	mult = 3
j=2	mult = -6
j=3	mult = -24
i=2 j=2	mult = -2
j=3	mult = -8
i=3 , j=3	mult = 4

arr	2	3	-2	4

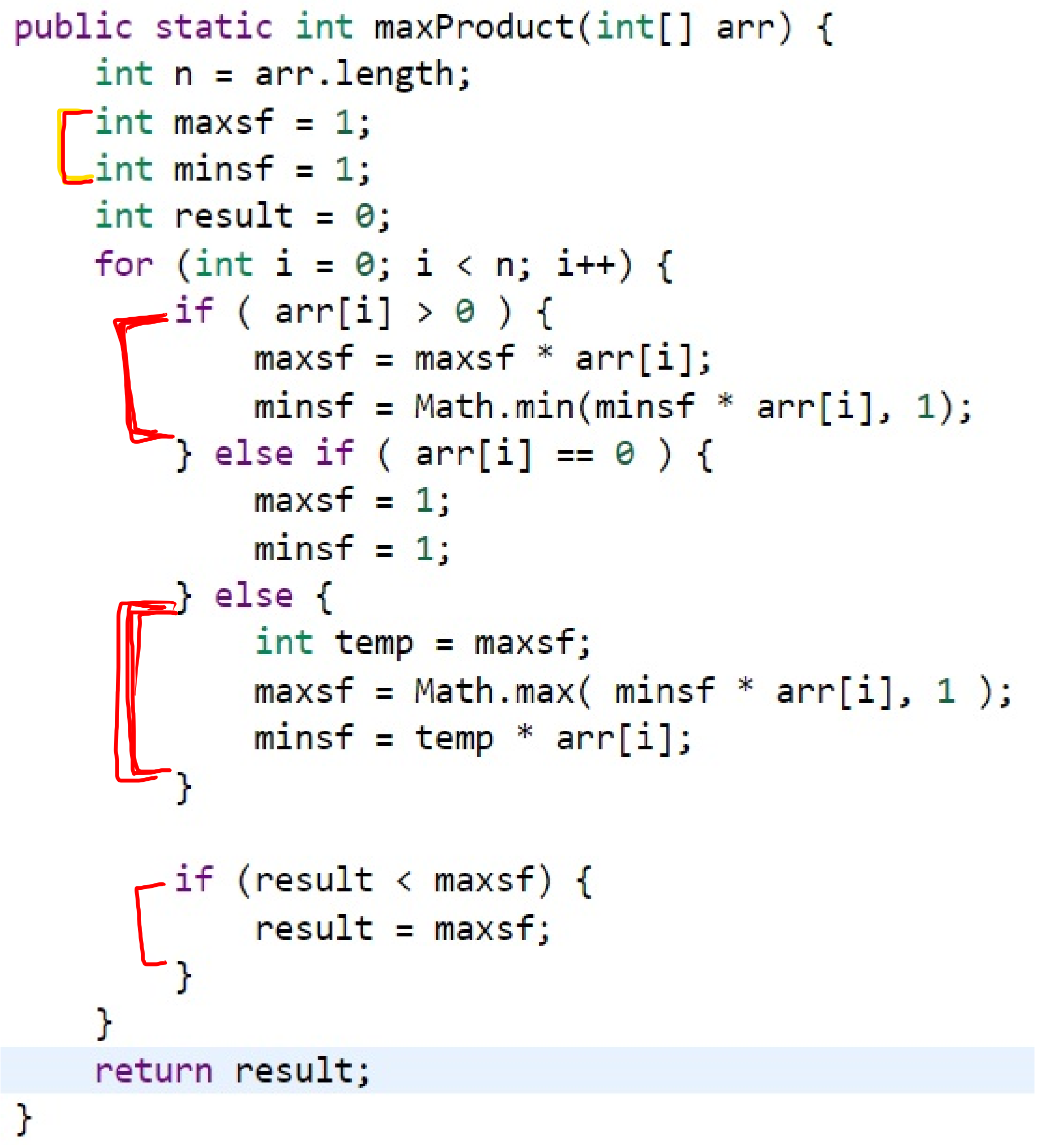
Save maxi value as well as mini val at each point of time.

maxi value so far = 1

mini vale so far = 1

maxsf = max(mvsf * arr[i]);

$$\text{minsf} = \text{mini}\left(\underbrace{\text{minsf} * \text{arr}[i]}_{-ve}, \underline{\underline{1}} \right);$$



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

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

arr	2	3	-2	4
	0	1	2	3

maxsf = 1
minsf = 1

result = ~~0~~ ~~2~~ 6

i = 0 , maxsf = 2
minsf = (2, 1) = 1

i = 1 , maxsf = 6
minsf = (3, 1) = 1

i = 2 , maxsf = (1 * -2 , 1) = 1
minsf = -12

i = 3 , maxsf = 4
minsf = -48