Kadane's Algorithm (max sum subarray)

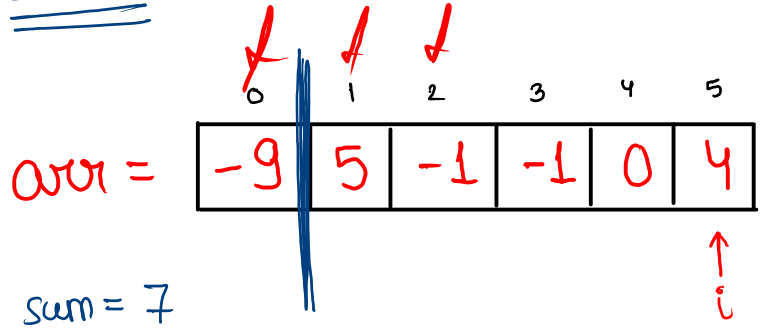
O(n)

```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(maxSumSubarray(arr, n));
}
public static int maxSumSubarray(int[] arr, int n) {
    int sumSoFar = 0;
    int maxSum = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {

        if ( sumSoFar < 0 ) {
            sumSoFar = arr[i];
        } else {
            sumSoFar += arr[i];
        }

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



sum = 7

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

MaxSum = ~~-∞~~ ~~-9~~ ~~5~~ 7

Maximum Product Subarray 2

(Very Imp)

n = 5

arr =

2	3	-2	4	-1

ans = 48

Observation:- Here, we need to store largest and smallest possible answers as well.

Ex:-

n = 5

arr =

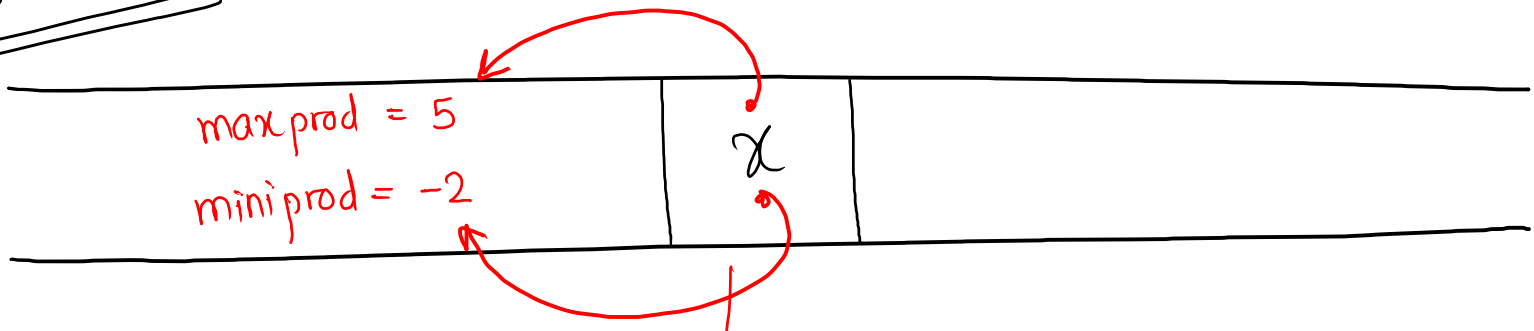
2	3	-2	4	-1

ans = 48

maxProd = 6 * -1 = -6

miniProd = -48 * -1 = 48

observation



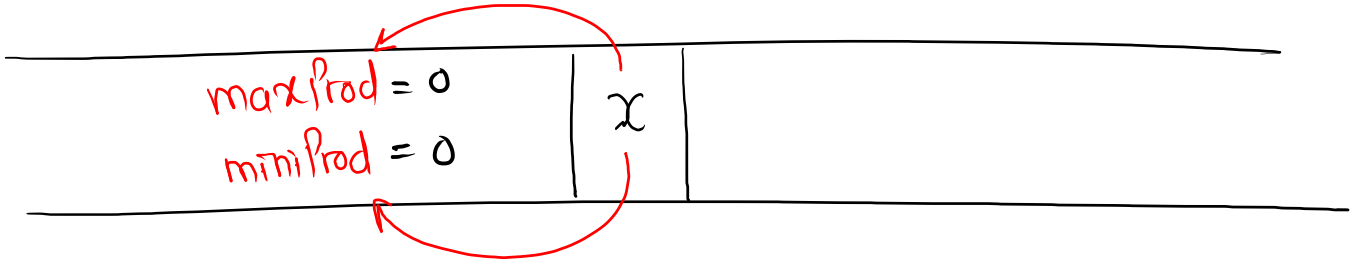
Note:-

multiply x with maxProd and miniProd.

$x = 2$ → +ve :- maxProd = 10, miniProd = −4

$x = -2$ → −ve :- maxProd = −10, miniProd = 4

$x = 0$ → zero :- wall (reset elements)



$x > 0$, maxProd ↑ing & miniProd ↓ing

$x < 0$, maxProd ↓ing & miniProd ↑ing

$x = 0$, maxProd = 0 & miniProd = 0
(wall) so reset

```
maxProd = max( maxProd * x , miniProd * x , x );
miniProd = min( maxProd * x , miniProd * x , x );
```

Code

```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(maxProdSubarray(arr, n));
}
public static int maxProdSubarray(int[] arr, int n) {
    int maxProd = 1;
    int minProd = 1;
    int largestProduct = Integer.MIN_VALUE;
    for (int i = 0; i < n; i++) {
        int curr = arr[i];
        int temp = maxProd;
        maxProd = Math.max(curr, Math.max( maxProd * curr, minProd * curr ));
        minProd = Math.min(curr, Math.min( temp * curr, minProd * curr ));

        largestProduct = Math.max( largestProduct, maxProd );
    }
    return largestProduct;
}
```

dry run

arr

3	1	-1	6

maxProd = 1

minProd = 1

---

i=0, maxProd = (3*1, 3*1, 3) = 3

minProd = (3*1, 3*1, 3) = 3

largestProd = ~~-∞~~ ~~3~~ 6

i=1, maxProd = (1*3, 1*3, 1) = 3

minProd = (1*3, 1*3, 1) = 1

i=2, maxProd = (-1*3, -1*1, -1) = -1

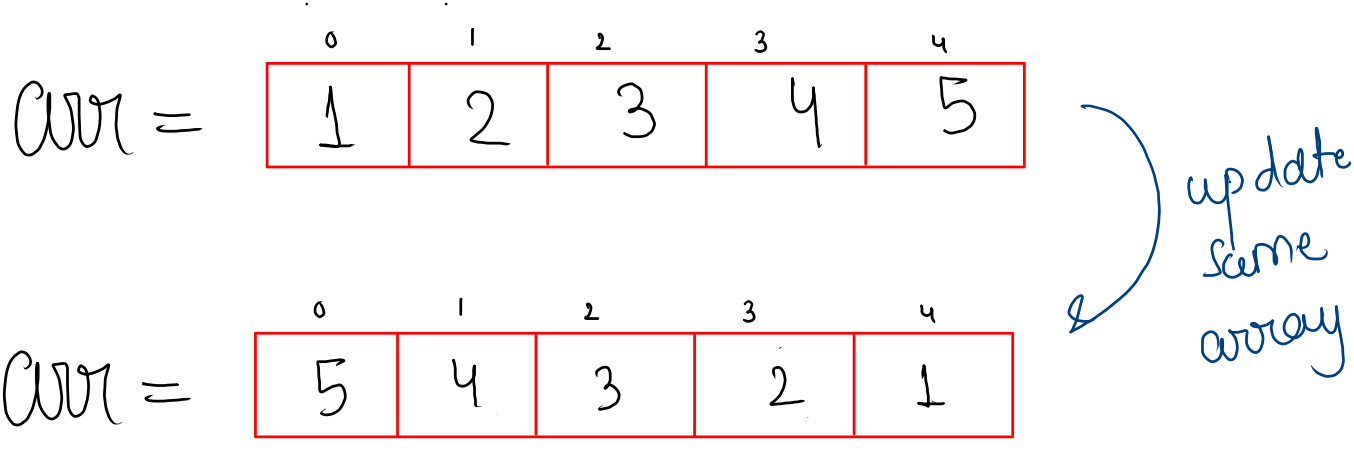
minProd = (-1*3, -1*1, -1) = -3

i=3, maxProd = (6*-1, 6*-3, 6) = 6

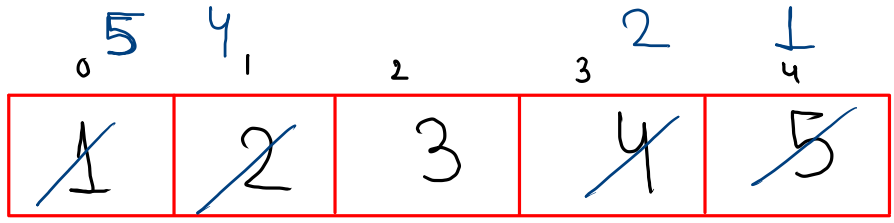
minProd = (6*-1, 6*-3, 6) = -18

⇒ Two Pointers (used to store the location of something)

GKSTR32 Reverse_Array



arr =



psudo code

1) declare i = 0
2) declare j = n-1;
3) loop until (i < j)
   3.1) swap (i, j)
   i++
   j--

Code

```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();
    }
    reverseArray(arr, n);
}
public static void reverseArray(int[] arr, int n) {
    int i = 0;
    int j = n - 1;
    while ( i < j ) {
        swap(arr, i, j);
        i++;
        j--;
    }

    // print
    for (int k = 0; k < n; k++) {
        System.out.println(arr[k]);
    }
}
public static void swap(int[] arr, int x, int y) {
    int temp = arr[x];
    arr[x] = arr[y];
    arr[y] = temp;
}
```

$T.C = O(n)$

where, n is size of array

$S.C = O(1)$

Interleaving x and y Elements