

# Kadane's Algorithm (max sum subarray)

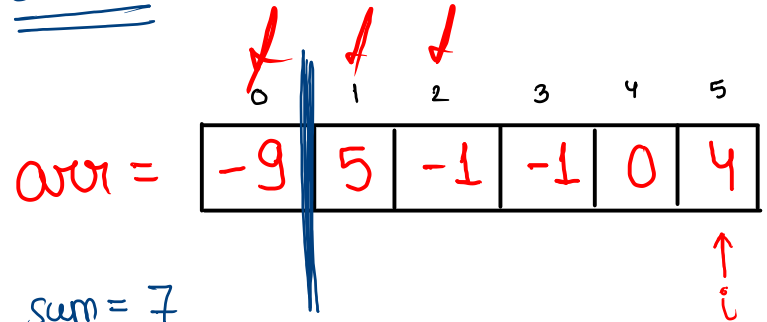
```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int[] arr = new int[n];
    for (int i = 0; i < n; i++) {
        arr[i] = scn.nextInt();
    }

    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;
}
```

$O(n)$



sumSoFar = ~~0~~ ~~-9~~ ~~5~~ ~~4~~ ~~3~~ ~~7~~  
maxSum = ~~-9~~ ~~5~~ ~~7~~

# Maximum Product Subarray 2

(Very Imp)

$$n = 5$$

$$\text{arr} = \begin{array}{|c|c|c|c|c|} \hline 2 & 3 & -2 & 4 & -1 \\ \hline \end{array}$$

$$\underline{\underline{\text{ans} = 48}}$$

Observation:-

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

Ex:-

$$n = 5$$

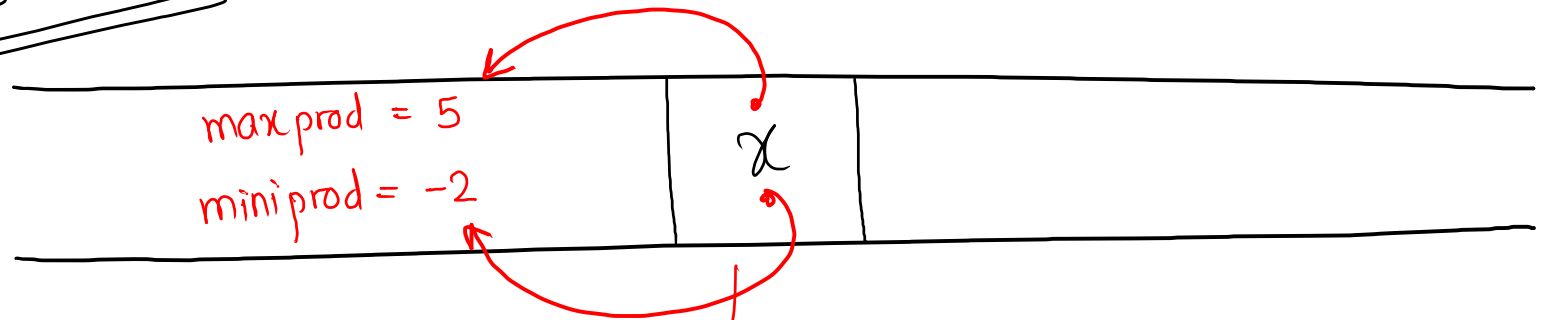
$$\text{arr} = \begin{array}{|c|c|c|c|c|} \hline 2 & 3 & -2 & 4 & -1 \\ \hline \end{array}$$

$$\underline{\underline{\text{ans} = 48}}$$

$$\text{maxProd} = 6 \times -1 = -6$$

$$\text{miniProd} = -48 \times -1 = 48$$

observation



Note:-

multiply  $x$   
with  $\text{maxProd}$   
and  $\text{miniprod}$ .

$x = 2 \rightarrow +ve :-$   $\text{maxProd} = 10$   
 $\text{miniprod} = -4$

$x = -2 \rightarrow -ve :-$   $\text{maxProd} = -10$   
 $\text{miniprod} = 4$

$x = 0 \rightarrow \text{zero} :-$  wall (reset elements)

$\text{maxProd} = 0$ $\text{minProd} = 0$	$x$	
--	-----	--

$x > 0$  ,  $\text{maxProd}$  ↑ing &  $\text{minProd}$  ↓ing

$x < 0$  ,  $\text{maxProd}$  ↓ing &  $\text{minProd}$  ↑ing

$x = 0$  ,  $\text{maxProd} = 0$  &  $\text{minProd} = 0$   
(wall) so reset

---

$$\text{maxProd} = \max(\text{maxProd} * x, \text{minProd} * x, x);$$

$$\text{minProd} = \min(\text{maxProd} * x, \text{minProd} * x, x);$$

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();
    }

    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

$$\text{maxProd} = 1$$

$$\text{minProd} = 1$$

---

$$i=0, \text{maxProd} = (3*1, 3*1, 3) = 3$$

$$\text{minProd} = (3*1, 3*1, 3) = 3$$

$$i=1, \text{maxProd} = (1*3, 1*3, 1) = 3$$

$$\text{minProd} = (1*3, 1*3, 1) = 1$$

$$i=2, \text{maxProd} = (-1*3, -1*1, -1) = -1$$

$$\text{minProd} = (-1*3, -1*1, -1) = -3$$

$$i=3, \text{maxProd} = (6*-1, 6*-3, 6) = 6$$

$$\text{minProd} = (6*-1, 6*-3, 6) = -18$$

$$\text{largestProd} = \cancel{\infty} \neq 6$$

# ⇒ Two Pointers

(used to store the location of something)

## GKSTR32 Reverse\_Array

arr =

0	1	2	3	4
1	2	3	4	5

arr =

0	1	2	3	4
5	4	3	2	1

update  
same  
array

arr =

<sup>0</sup> 5	<sup>1</sup> 4	<sup>2</sup>	<sup>3</sup> 2	<sup>4</sup> 1
<del>1</del>	<del>2</del>	3	<del>4</del>	<del>5</del>

↑ ↑  
i j

pseudo  
code

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



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();
    }
    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

$$\underline{\underline{n=5}}$$

$$\text{arr} = \left[ \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ x_1 & x_2 & x_3 & x_4 & x_5 & y_1 & y_2 & y_3 & y_4 & y_5 \end{array} \right]$$

Diagram illustrating the initial array structure. The array is divided into two segments: the first segment contains elements  $x_1, x_2, x_3, x_4, x_5$  (indices 0 to 4), and the second segment contains elements  $y_1, y_2, y_3, y_4, y_5$  (indices 5 to 9). Red arrows point to the starting indices of these segments, labeled  $i$  and  $j$  respectively.

$$\text{ans} = \left[ \begin{array}{ccccccccc} x_1 & y_1 & x_2 & y_2 & x_3 & y_3 & x_4 & y_4 & x_5 & y_5 \end{array} \right]$$

Diagram illustrating the interleaved array structure. The array is divided into five pairs of elements:  $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)$ . Red brackets group each pair. A red arrow points to the starting index of the first pair, labeled  $k$ .