Maximum Sum Subarrays (Kadane's Algorithm)

arr = [10, 5, 17, 9, 80]

Assumption is largest subarray will give max sum if that do not have negative sum in between.

```
int sum=0;
for(int i=0;i<n;i++){
    sum+=arr[i];
    if(sum>maxSum){
      maxSum=sum;
    }
}
```

Problem arises when we have below array

arr: [10, 5, -17, 9, 80]

Sum of 0 to n-1

10+5 = 15

15-17 = -2 → sum<0, sum = 0.

-2+9+80 = 87

9+80 = 89 → this is maxSum

To resolve this issue, we make sum as 0, when it is negative because if we continue adding elements of array to -ve sum, sum will decrease.

```
if (sum<0){
   sum=0;
}
```

final code

```
int sum=0; int maxsum=Integer.MIN_VALUE;
for(int i=0;i<n;i++){
  sum+=arr[i];
  if(sum>maxsum){
    maxsum=sum;
  }
  if (sum<0){
     sum=0;
  }
}
```

Maximum Product Subarray 2

n = 4

arr = [2, 3, -2, 4]

We can Kadane's Algorithm

Ex1: arr = [2, 4, 6, 5]

which subarray gives max product

↳ 2×4×6×5 = 240

Ex2: arr = [2, 4, -6, 5]

Subarray → 2, 4, -6, 5 ⇒ 2×4×-6×5 = -240

Product → -240 → is it maxProduct

↓

No

[2, 4] = 8

Ex3. arr = [2, 4, -6, -5]

[2, 4, -6, -5] will give maxProduct

= 240

[2, 4, -6, -5]

maxProd → (8)

[2, 4, -6, -5]

prod = 2 × 4 = 8 ← maxProd → (8)

2 × 4 × -6 = (-48) ← minProd. → (2)

when we try to multiply -5,
then we will consider minProd.

-48 × -5 = 240

```
int minProd = arr[0]; // 2
int maxProd = arr[0]; // 2
int result = arr[0]; // 2
for (int i=1; i<n; i++) {
    if (arr[i]<0) {
        swap(maxProd, minProd);
    }
    maxProd = Math.max( arr[i], maxProd * arr[i]);
    or
    if( maxProd* arr[i] > maxProd ){
        maxProd = arr[i];

    }
    minProd = Math.min( arr[i], minProd * arr[i]);
    result = Math.max (result, maxProd);
```

→ before -6.

minProduct = 2

product = 8.

minProduct

maxProduct = 8.

minProduct = 8

maxProduct = 2.

maxProduct = Math.max(-6, 2X-6)
= 2.

minProduct = Math.min(-6, 8X-6)
= -48.

At, i = 1, arr[1] = 4.

result = Math.max(result, maxProd)

result = 8.

Now, When -6 comes

maxProd = 2

minProd = -48

result = Math.max(result, maxProd)
= Math.max(8, 2)
= 8.

When We are at -5.

maxProd = 2

minProd = -48

result = 8

minProd = -1

result = 8

-5 < 0

maxProd = -48

minProd = 2

maxProd = Math.max(-5, -48 * -5)

= Math.max(-5, 240)

= 240

minProd = Math.min(-5, 2 * -5)

= -10

result = Math.max(8, 240)

result = 240

---

arr = [5, -2, -3, 4]

Dry Run

minProd = 5 = arr[0]

maxProd = 5 = arr[0]

result = 5 = arr[0]

i = 1.

-2 < 0.

swap (minProd & maxProd)

minProd = 5.

maxProd = 5

maxProd = Math.max(-2, maxProd * arr[i]);

= Math.max(-2, 5 * -2)

maxProd = Math.max(-2, 5 --)

= Math.max(-2, 5)

maxProd = 5

minProd = Math.min(arr[i], minProd * arr[i]);

= Math.min(5, 5 * -2)

minProd = -10, result = Math.max(5, 5)

= 5

↳ It is actual max product or greatest product

i = 2. arr[i] = -3

arr[i] < 0

swap(maxProd, minProd)

{ maxProd = -10

minProd = 5

maxProd = Math.max(-3, maxProd * arr[i])

= Math.max(-3, -10 * -3)

= Math.max(-3, 30) = 30

maxProd = 30

minProd = Math.min(-3, 5 * -3).

= -15

result = Math.max(result, maxProd)

= Math.max(5, 30)

= 30

i = 3. arr[i] = 4.

maxProd = 30

minProd = -15

= Math.max(4, 30 * 4);

minProd = -15

maxProd = Math.max(4, 30*4);

= 120

minProd = Math.min(4, -15*4)

= -60

result = Math.max(maxProd, result)

= Math.max(120, 30)

result = 120;

Code

```
int minProd = arr[0];
int maxProd = arr[0];
int result = arr[0];
for(int i=1; i<n; i++){
    if(arr[i]<0){
    int temp = maxProd;
    maxProd = minProd;
    minProd = temp;
    }
maxProd = Math.max(arr[i], maxProd*arr[i]);
minProd = Math.min(arr[i], minProd*arr[i]);
result = Math.max(result, maxProd);
```

}
S.O.println(result);


```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        /* Enter your code here. Read input from STDIN. Print output to
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int arr[] = new int[n];
        for(int i=0;i<n;i++){
            arr[i] = sc.nextInt();
        }

        int minProd = arr[0];
        int maxProd = arr[0];
        int result = arr[0];
        for(int i=1;i<n;i++){
            if(arr[i]<0){
                int temp = maxProd;
                maxProd = minProd;
                minProd = temp;
            }
            maxProd = Math.max(arr[i],maxProd*arr[i]);
            minProd = Math.min(arr[i],minProd*arr[i]);
            result = Math.max(maxProd,result);
        }
        System.out.println(result);
    }
}
```

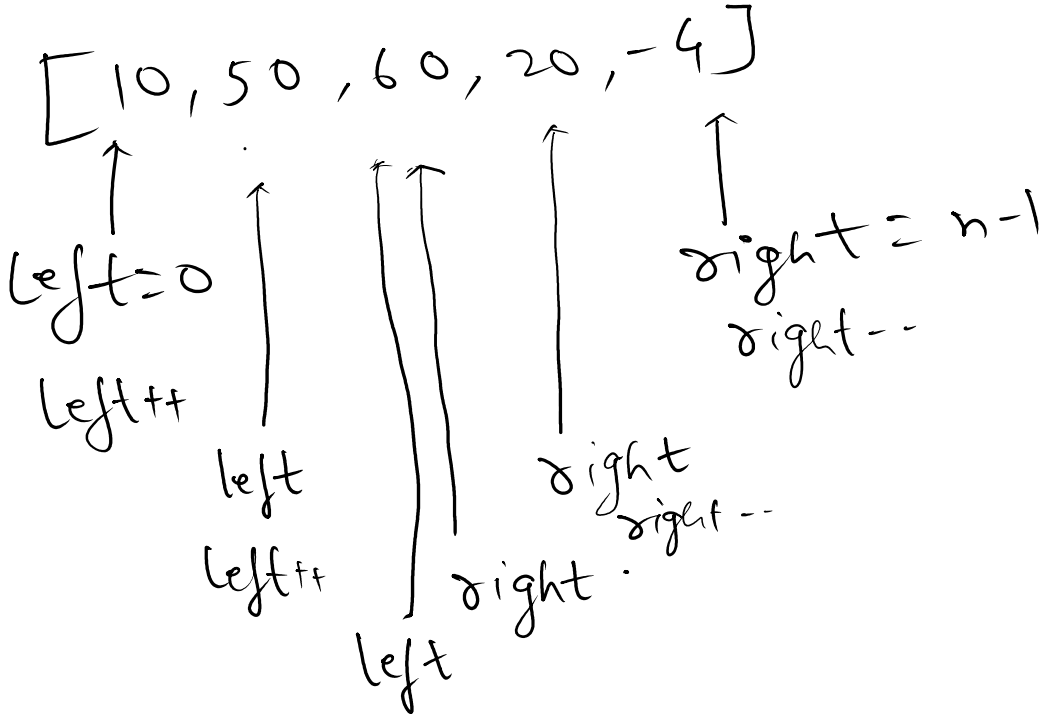
Two Pointers

arr = [10, 50, 60, 20, -4]

i = 0 to n-1 → This is without two pointer

When using two pointer

We keep one pointer at beginning and another at end.



It reduces the time complexity to O(n/2)

O(n/2) is better than O(n)

Reverse Array

n = 5

arr = [2 4 8 3 5]

Reverse using Brute Force Approach

Loop will run from n-1 to 0

Time Complexity → O(n)

We can reduce T.C. to O(n/2) by two pointers

```
int left = 0
int right = n-1;
```

[2, 4, 8, 3, 5]

left ↑ right ↑

left → [5, 4, 8, 3, 2] ← right

```
left++
right--
swap(arr[left], arr[right])
```

[5, 3, 8, 4, 2] → Reversed Array

Code

```
int left = 0;
int right = n-1;
```

```
int right = n-1;
while (left < right) {
    int temp = arr[left];
    arr[left] = arr[right];
    arr[right] = temp;
    left++;
    right--;
}
```