You are given a **0-indexed** integer array `nums`.

Return ***the maximum value over all triplets of indices*** `(i, j, k)` *such that* `i < j < k`. If all such triplets have a negative value, return `0`.

The **value of a triplet of indices** `(i, j, k)` is equal to `(nums[i] - nums[j]) * nums[k]`.

Example :- nums = [12, 6, 1, 2, 7]
i j K

Output :- 77

(12 - 1) * 7

77.

Thought Process

Brute Force

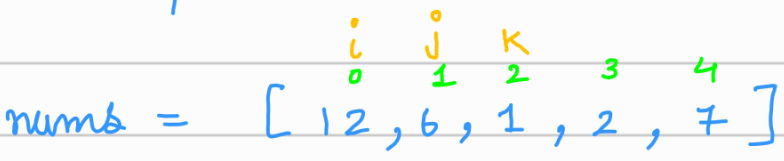
... i < j < K

(i) (j) (K)

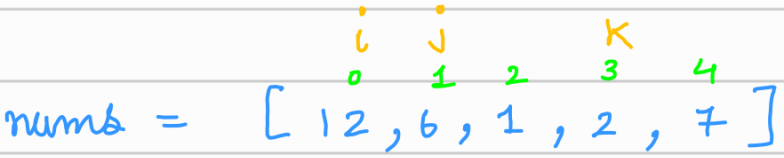
nums = [12, 6, 1, 2, 7]
0 1 2 3 4

we will take 3 for loops for i, j, K each.
i ftr will start from index 0, j will start from after i &
(j will start from i + 1)
K will start from after i & j since i < j < K.
(K will start from j + 1)

Ex: the ptrs will move like this i < j < K

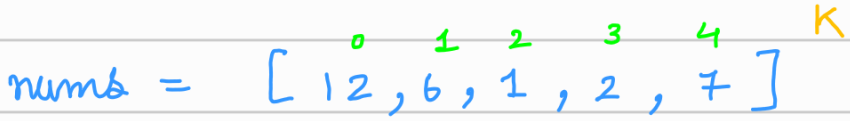


(12, 6, 1)

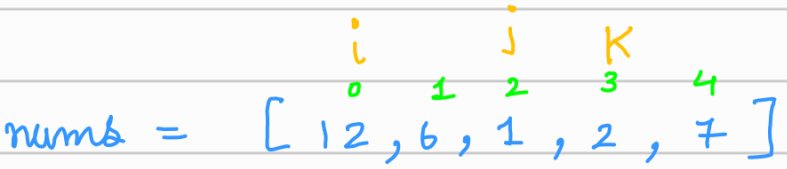


(12, 6, 2)

(12, 6, 7)



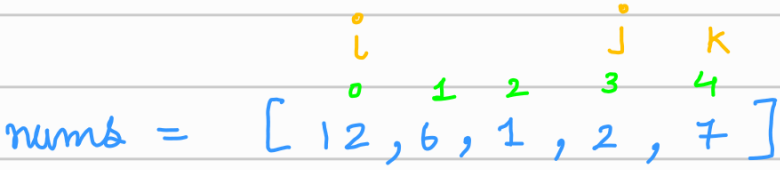
once K gets out of bounds j will move.



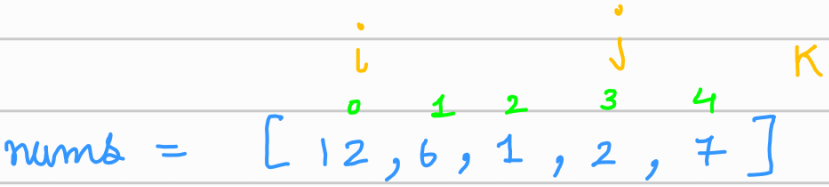
(12, 1, 2)

(12, 1, 7)

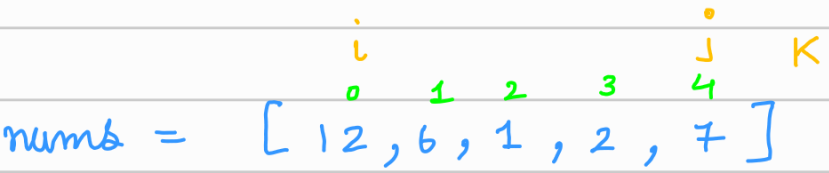
again K oob j will move.



(12, 2, 7)



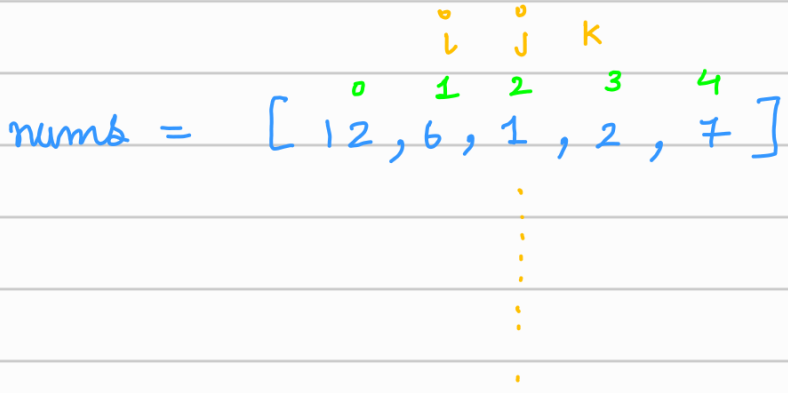
again K oob j will move.



when j is here

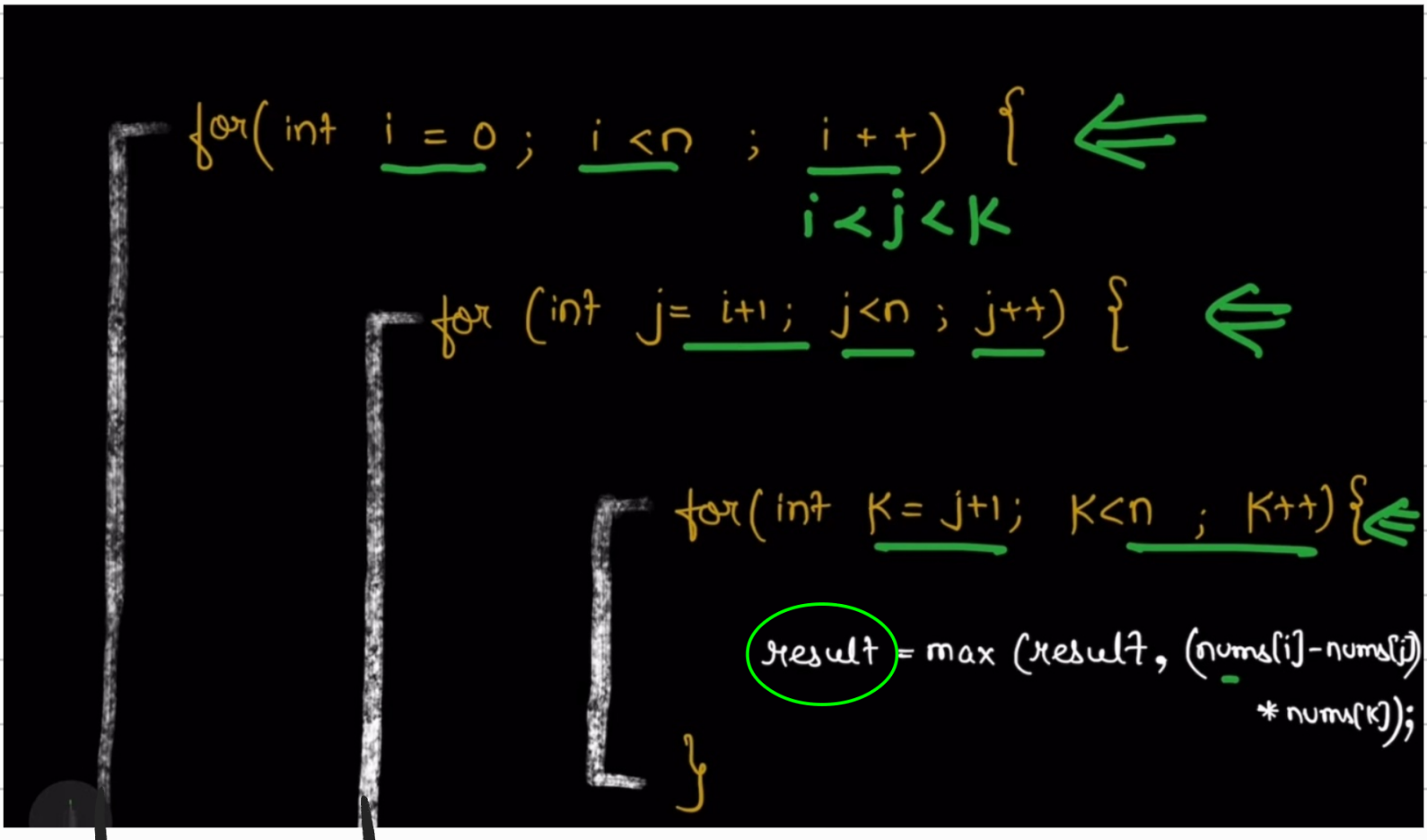
K will be oob that means j can move only till here.

once j & K completes i moves.
& it will repeat again like before.



The max possible combination we get acc to formula given in question will be our answer.

for ex: (12 - 1) * 7 = 77



return result ;

TC = O(n³)

SC = O(1).

Slight Improvement

$$(nums[i] - nums[j]) * nums[k]$$

maximise

$$nums = [12, 6, 1, 2, 7]$$ (indices 0, 1, 2, 3, 4)

we would want nums[k] to be bigger since its getting multiplied and we want max.

Similarly, we would want nums[i] to be larger since its getting subtracted by nums[j] so getting a big value won't affect the no by that much after subtraction & also since we want max we want to reduce the effect of subtraction. [we want to maximise the diff]

∴ for any given j we want to find the possible largest i & k.

$$nums = [12, 6, 1, 2, 7]$$ (indices 0, 1, 2, 3, 4)

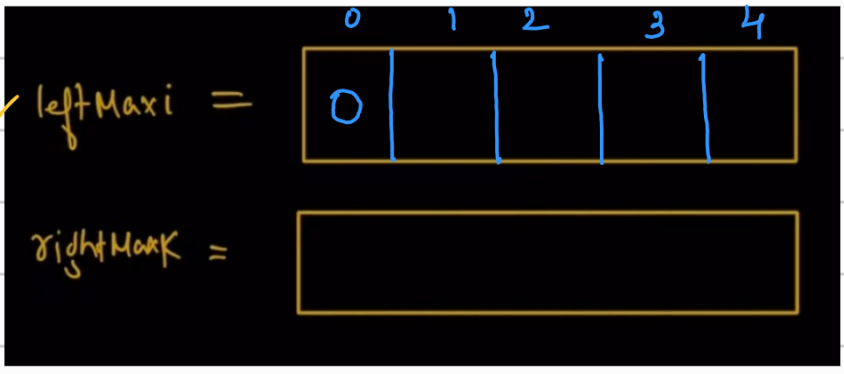
j

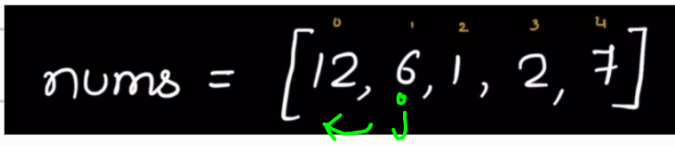
maximum i

maximum k

from j's left side we will find max i & from j's right side we will find max k. [since $i < j < k$]

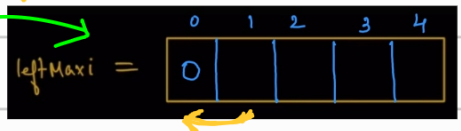
∴ for every given j we can preprocess and find leftmaxi & rightmaxK.

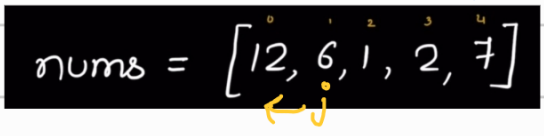




there is no element at left hand side of index 0
∴ take leftmaxi[0] = 0 .

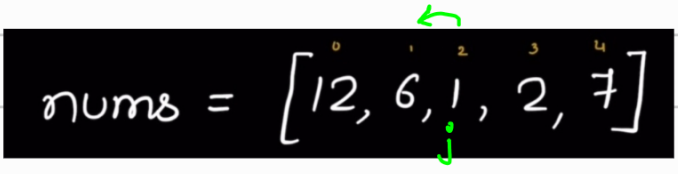
now let's say j is at index 1, iske left hand side mein max kaun hoga? hum already jitna max dekhte aayein hai ya to voh max hoga. ya to abhi iss j ke left hand side jo element hai



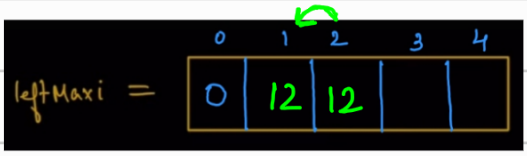


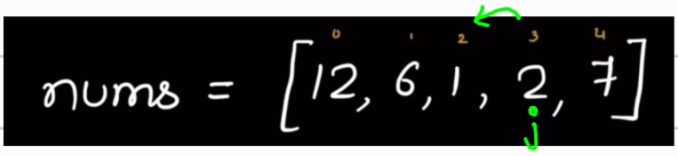
leftmaxi[j] = max (leftmax[j-1], nums[j-1]);

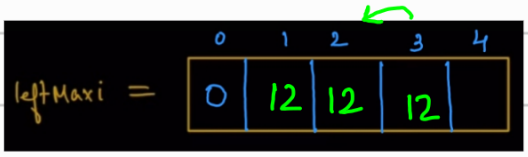
Ex: (0, 12) ⇒ 12 .



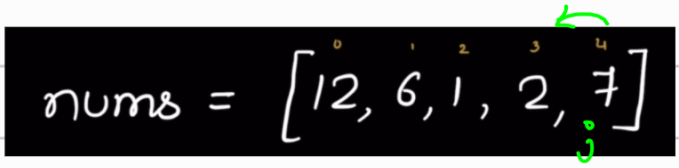
(12, 6) ⇒ 12 .

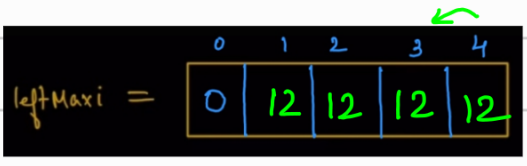






(1, 12) ⇒ 12

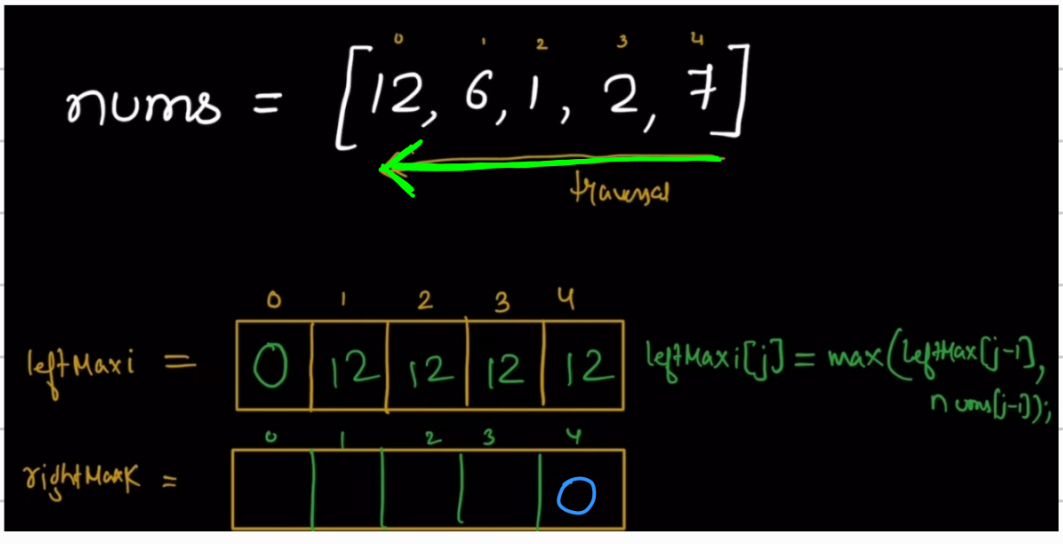




(2, 12) ⇒ 12

Similarly, we will find for right hand side.

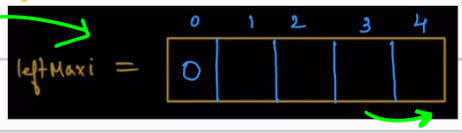
since we need to find right hand side max.
∴ we will do traversal from right to left.

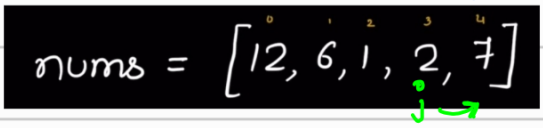


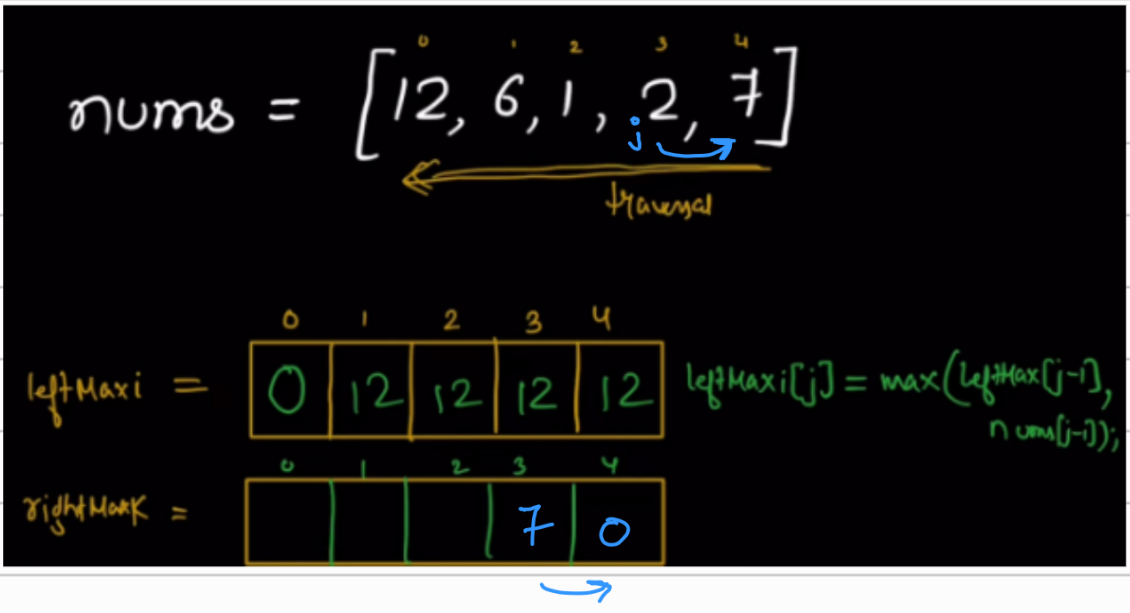
for last index i.e 4 there is no right side element ∴ put 0.

Similarly,

now let's say j is at index 3 , iske right hand side mein max kaun hoga ? hum already jitna max dekhte aayein hai ya to voh max hoga . ya to abhi iss j ke right hand side jo element hai

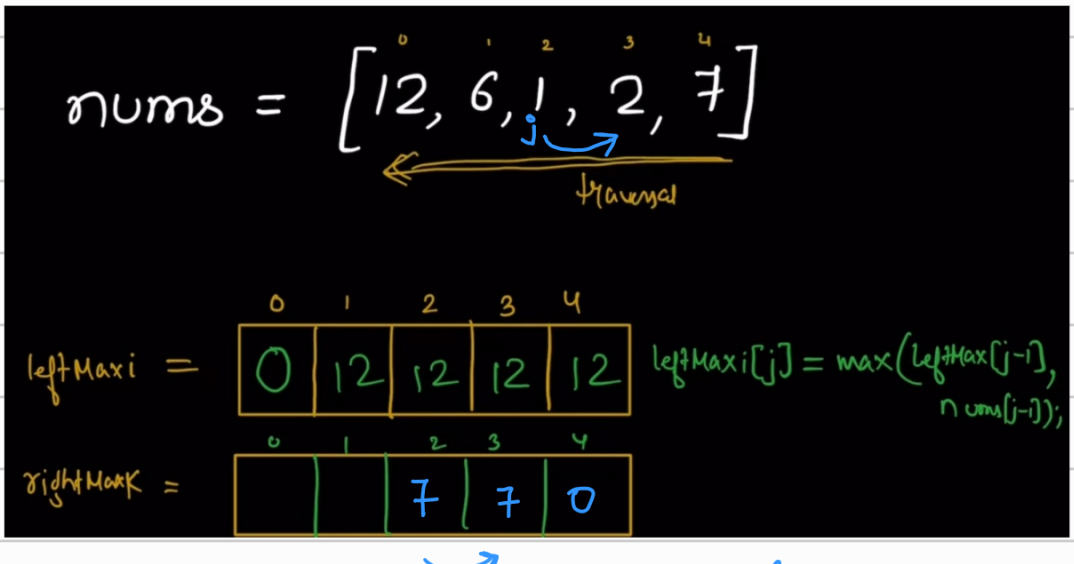




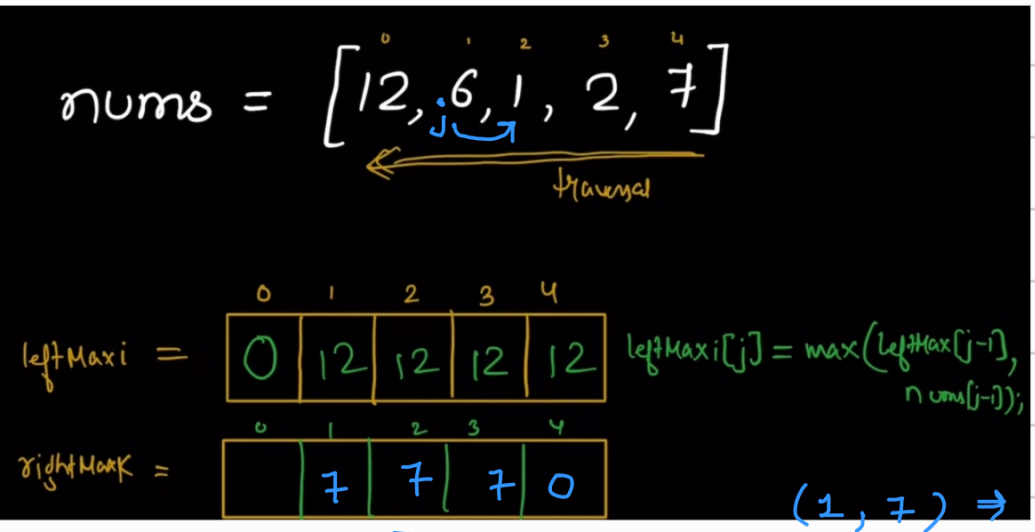


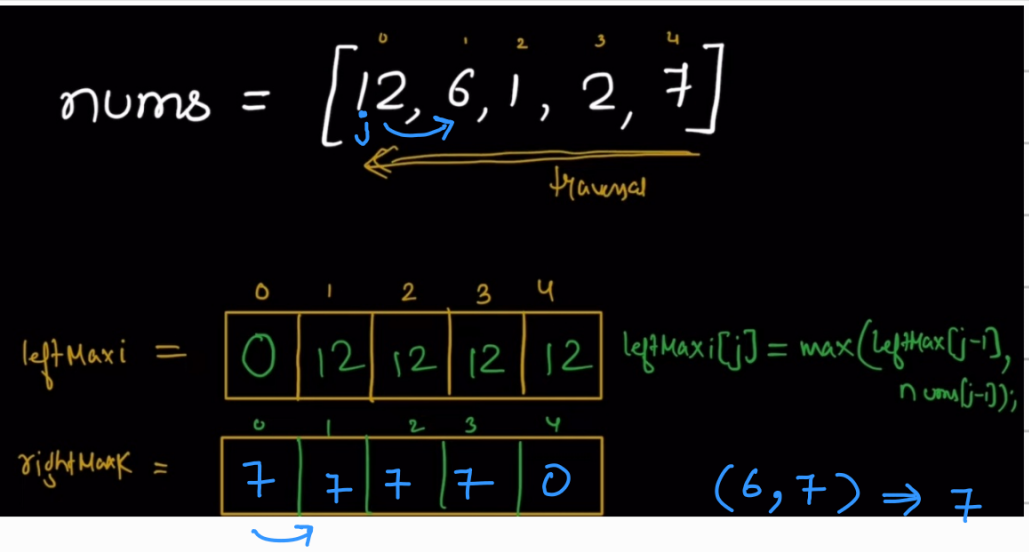
rightmaxK [j] = max [rightmaxK [j+1], nums [j+1]] ;

(0, 7) ⇒ 7 .



(2, 7) ⇒ 7






```
for ( int j = 1 ; j < n ; j++ ) {

    result = max ( result, ( leftmaxi[j] - nums[j] )
                                * rightmaxK[j] );

}

return result ;
```

TC : O(n)
SC : O(n)

Avoiding extra Space ...

$$(nums[i] - nums[j]) * nums[k]$$

maximize diff.

nums = [12, 6, 1, 2, 7] (indices 0, 1, 2, 3, 4)

In Approach-2, we saw we need to maximise diff by making nums[i] larger so that subtraction with nums[j] won't have that much effect.

∴ we need maxDiff = 0 ;

maxi = 0 .

↳ to maximize diff we need largest possible nums[i] .

result = 0 ;

result = max (result , maxDiff * nums[k]) ;

nums = [12, 6, 1, 2, 7]

(indices: 0, 1, 2, 3, 4)

K → 12

maxDiff = 0 ;

maxi = 0 ; ∴ 12

result = 0

result = max(result, maxDiff * nums[K]) ;

maxDiff = (nums[i] - nums[j]

↓

we want this max .

∴ maxDiff = (maxi - nums[K])

maxi = max (maxi , nums[K]) ;

0 - 12 = -12

already maxdiff = 0 > -12

∴ no need to update .

(0 , 12) ⇒ 12

update maxi .

Similarly do it for all .

TC = O(n)

SC = O(1)

Approach - 2 Code

```cpp
class Solution {
public:
    long long maximumTripletValue(vector<int>& nums) {
        int n = nums.size();

        vector<int> leftMaxi(n);
        vector<int> rightMaxk(n);

        for(int j = 1; j < n; j++) {
            leftMaxi[j] = max(leftMaxi[j-1], nums[j-1]);
        }

        for(int j = n-2; j >= 0; j--) {
            rightMaxk[j] = max(rightMaxk[j+1], nums[j+1]);
        }

        long long result = 0;
        for(int j = 1; j < n; j++) {
            result = max(result, (long long)(leftMaxi[j] - nums[j])*rightMaxk[j]);
        }

        return result;

    }
};
```

Approach - 3 Code

```cpp
class Solution {
public:
    long long maximumTripletValue(vector<int>& nums) {
        int n = nums.size();

        long long result = 0;

        long long maxDiff = 0;
        long long maxi = 0;

        for(int k = 0; k < n; k++) {
            result = max(result, maxDiff * nums[k]);
            maxDiff = max(maxDiff, maxi - nums[k]);
            maxi = max(maxi, (long long)nums[k]);
        }

        return result;

    }
};
```