**LC#152:MAXIMUM PRODUCT SUBARRAY:**

**Given an integer array nums, find a subarray that has the largest product, and return *the product*.**

**The test cases are generated so that the answer will fit in a 32-bit integer.**

**1)Input: nums = [2,3,-2,4]** Output: **:**6 **Explanation: [2,3] has the largest product 6.**

**2)Input:** nums **= [-2,0,-1] , Output: 0 , Explanation: The result cannot be 2, because [-2,-1] is not a subarray.**

**APP1:BRUTE O(N^3)**

**->loop through three loops**

**-> I = outer,-> j = inner , -> k = traversing**

**-> find the product and update as when are we getting the values**

class Solution {

    public int maxProduct(int[] nums) {

        int n = nums.length;

        int res = Integer.MIN\_VALUE;//**since value from -2^31 to 2^31 use the minimum**

        for(int i = 0 ; i<n;i++)

        {

            for(int j = i;j<n;j++)

            {

                int pro = 1;

                for(int k = i; k<=j;k++)

                {

                    pro = pro\*nums[k];

                    res = Math.max(pro , res);

                               }

            }

        }

        return res;

    }}

**APP2: BRUTE (O(N^2) passed but not optimal**

* **Loop only two I and j calculate product and update max**
* **If pro becomes 0 turn it to 1 and continue**

**CODE:**

class Solution {

    public int maxProduct(int[] nums) {

        int n = nums.length;

        int pro=1, res = Integer.MIN\_VALUE;

        for(int i = 0 ; i<n;i++)

        {   pro =1;

            for(int j = i; j<n;j++)

            {

                pro = pro\*nums[j];

                res = Math.max(res,pro);

                if(pro==0)

                    pro=1;

            }

        }

        return res;

    }

}

**APP3: using pre and suff as pointers TC:O(N) , SC :O(1)**

* **Compute pre from start to I**
* **Compute suff from last to i-1**
* **Compare and update the max value**

class Solution {

    public int maxProduct(int[] nums) {

        int pre = 1,suff = 1;

        int res = Integer.MIN\_VALUE;

        for(int i = 0 ;i < nums.length ;i++)

        {

            if(pre==0)

                pre=1;

            if(suff==0)

                suff=1;

            pre = pre\*nums[i];

            suff = suff\*nums[nums.length-i-1];//reverse

            res = Math.max(res , Math.max(pre,suff));

            }

        return res;

    }

}