**Maximum Product Subarray**

**Medium**

Given an array **Arr[]** that contains **N** integers (may be **positive**, **negative**or **zero**). Find the product of the maximum product subarray.

**Example 1:**

**Input:**

N = 5

Arr[] = {6, -3, -10, 0, 2}

**Output:** 180

**Explanation:** Subarray with maximum product

is [6, -3, -10] which gives product as 180.

**Example 2:**

**Input:**

N = 6

Arr[] = {2, 3, 4, 5, -1, 0}

**Output:** 120

**Explanation:** Subarray with maximum product

is [2, 3, 4, 5] which gives product as 120.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 ≤ N ≤ 500  
-102 ≤ Arri ≤ 102

**Company Tags**

[**Morgan Stanley**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Morgan%20Stanley) [**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon) [**Microsoft**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Microsoft) [**OYO Rooms**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=OYO%20Rooms) [**D-E-Shaw**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=D-E-Shaw) [**Google**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Google)

**Topic Tags**

[**Arrays**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Arrays) [**Dynamic Programming**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Dynamic%20Programming) [**Data Structures**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Data%20Structures) [**Algorithms**](https://practice.geeksforgeeks.org/explore/?category%5b%5d=Algorithms)

//{ Driver Code Starts

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br =

new BufferedReader(new InputStreamReader(System.in));

int tc = Integer.parseInt(br.readLine());

while (tc-- > 0) {

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

String[] inputLine = br.readLine().split(" ");

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(inputLine[i]);

}

System.out.println(new Solution().maxProduct(arr, n));

}

}

}

// } Driver Code Ends

class Solution {

// Function to find maximum product subarray

long maxProduct(int[] arr, int n) {

// code here

long max=Integer.MIN\_VALUE;

for(int i=0;i<n;i++){

long p=1;

for(int j=i;j<n;j++){

p=p\*arr[j];

if(p>max)

max=p;

}

}

return max;

}

}

// } Driver Code Ends

class Solution

{

//Function to return max value that can be put in knapsack of capacity W.

static int knapSack(int W, int wt[], int val[], int n)

{

//Arrays.sort(val);

int row=val[n-1];

int[][]dp=new int[n+1][W+1];

for(int i=1;i<=n;i++){

for(int j=1;j<=W;j++){

if(j>=wt[i-1]){

dp[i][j]=Math.max(dp[i-1][j-wt[i-1]]+val[i-1],dp[i-1][j]);

}

else{

dp[i][j]=dp[i-1][j];

}

}

}

return dp[n][W];

}

}