**PRACTICAL 4**

**TASK 1 :**

**CODE :**

#include <stdio.h>

#include <limits.h>

// Global Declaration to find the exact array

int global\_max\_sum = INT\_MIN;

int global\_start = 0;

int global\_end = 0;

// Finding Max

int max(int a, int b) {

return (a > b) ? a : b;

}

// Return max of three integers but less than or equal to constraint

int max\_of\_three\_constrained(int a, int b, int c, int constraint) {

int res = INT\_MIN;

if (a <= constraint && a > res) res = a;

if (b <= constraint && b > res) res = b;

if (c <= constraint && c > res) res = c;

return res == INT\_MIN ? 0 : res;

}

// Display Function

void Displaysubarray(int arr[], int low, int high) {

printf("[");

for (int i = low; i <= high; i++) {

printf("%d ", arr[i]);

}

printf("]");

}

// Max For Crossing.

int maxCrossingsum(int arr[], int low, int mid, int high, int constraint) {

// Left Side

int sum = 0;

int left\_sum = INT\_MIN;

int left\_index = mid; // global

for (int i = mid; i >= low; i--) {

sum += arr[i];

if (sum <= constraint && sum > left\_sum) {

left\_sum = sum;

left\_index = i; // Keeping Track of left part

}

}

// Right Side

sum = 0;

int right\_sum = INT\_MIN;

int right\_index = mid + 1; // global

for (int i = mid + 1; i <= high; i++) {

sum += arr[i];

if (sum <= constraint && sum > right\_sum) {

right\_sum = sum;

right\_index = i; // Keeping Track of right part

}

}

// Cross

int cross\_sum = (left\_sum == INT\_MIN || right\_sum == INT\_MIN) ? INT\_MIN : left\_sum + right\_sum;

// Update globals if crossing sum is more valid

if (cross\_sum <= constraint && cross\_sum > global\_max\_sum) {

global\_max\_sum = cross\_sum;

global\_start = left\_index; // Update global start

global\_end = right\_index; // Update global end

}

if (cross\_sum <= constraint && cross\_sum > INT\_MIN) return cross\_sum;

// Return the best valid sum among left\_sum and right\_sum

if (left\_sum == INT\_MIN) {

if (right\_sum <= constraint && right\_sum > global\_max\_sum) {

global\_max\_sum = right\_sum;

global\_start = mid + 1;

global\_end = right\_index;

}

return (right\_sum == INT\_MIN) ? 0 : right\_sum;

}

if (right\_sum == INT\_MIN) {

if (left\_sum <= constraint && left\_sum > global\_max\_sum) {

global\_max\_sum = left\_sum;

global\_start = left\_index;

global\_end = mid;

}

return left\_sum;

}

// Compare and update globals for left and right sums

if (left\_sum > right\_sum) {

if (left\_sum <= constraint && left\_sum > global\_max\_sum) {

global\_max\_sum = left\_sum;

global\_start = left\_index;

global\_end = mid;

}

return left\_sum;

}

else

{

if (right\_sum <= constraint && right\_sum > global\_max\_sum) {

global\_max\_sum = right\_sum;

global\_start = mid + 1;

global\_end = right\_index;

}

return right\_sum;

}

}

// Recusive Division for each max.

int maxsumRecursive(int arr[], int low, int high, int constraint) {

if (low > high) return 0;

if (low == high) {

if (arr[low] <= constraint && arr[low] > global\_max\_sum) {

global\_max\_sum = arr[low];

global\_start = low; // Track single element start

global\_end = low; // Track single element end

}

return (arr[low] <= constraint) ? arr[low] : 0;

}

int mid = low + (high - low) / 2;

printf("\n--- Dividing Array ");

Displaysubarray(arr, low, high);

printf(" ---\n");

printf("First subarray: ");

Displaysubarray(arr, low, mid);

printf("\nSecond subarray: ");

Displaysubarray(arr, mid + 1, high);

printf("\n");

int left\_max\_sum = maxsumRecursive(arr, low, mid, constraint);

int right\_max\_sum = maxsumRecursive(arr, mid + 1, high, constraint);

// Calling function maxCrossingsum

int cross\_max\_sum = maxCrossingsum(arr, low, mid, high, constraint);

if (left\_max\_sum <= constraint && left\_max\_sum > global\_max\_sum) {

global\_max\_sum = left\_max\_sum;

// The subarray indices are updated during recursive calls

}

if (right\_max\_sum <= constraint && right\_max\_sum > global\_max\_sum) {

global\_max\_sum = right\_max\_sum;

// Subarray indices updated in recursive calls

}

printf("\n--- Dividing Array : ");

Displaysubarray(arr, low, high);

printf(" ---\n");

printf("Max sum in Left half : %d\n", left\_max\_sum);

printf("Max sum in Right half : %d\n", right\_max\_sum);

printf("Max sum for crossing subarray : %d\n", cross\_max\_sum);

return max\_of\_three\_constrained(left\_max\_sum, right\_max\_sum, cross\_max\_sum, constraint);

}

int main() {

int oriarr[100], size, constraint;

printf("Enter array size: ");

scanf("%d", &size);

if(size<0){

printf("Empty Array ! No Subarrays !");

return 0;

}

printf("Enter array elements:\n");

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

printf("Element %d: ", i);

scanf("%d", &oriarr[i]);

}

printf("Enter constraint value: ");

scanf("%d", &constraint);

printf("Original Array : ");

Displaysubarray(oriarr, 0, size - 1);

printf("\n------------------------------------------------\n");

printf("\n");

int maxsum = maxsumRecursive(oriarr, 0, size - 1, constraint);

if (maxsum == 0 || global\_max\_sum == INT\_MIN ) {

printf("------------------------------------------------\n");

printf("No Valid subarray sum <= %d found.\n", constraint);

printf("\n------------------------------------------------\n");

}

else {

printf("\n------------------------------------------------\n");

printf("Maximum subarray sum <= %d is %d\n", constraint, global\_max\_sum);

printf("From Array/Subarray : ");

Displaysubarray(oriarr, global\_start, global\_end); // Displays max subarray

printf("\n------------------------------------------------\n");

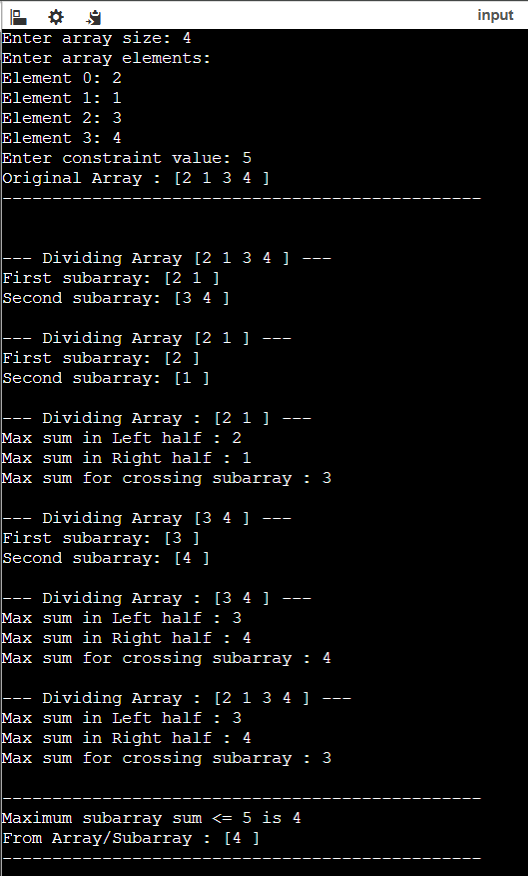
printf("\n");

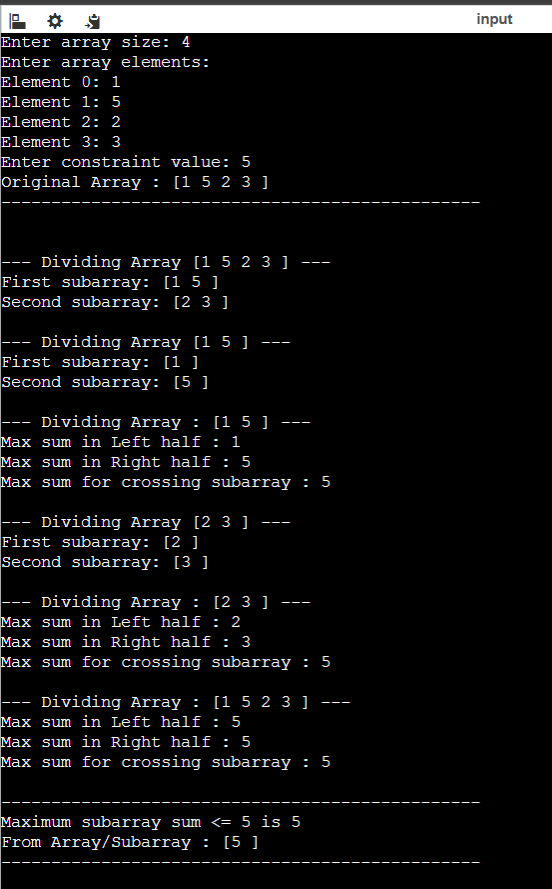
}

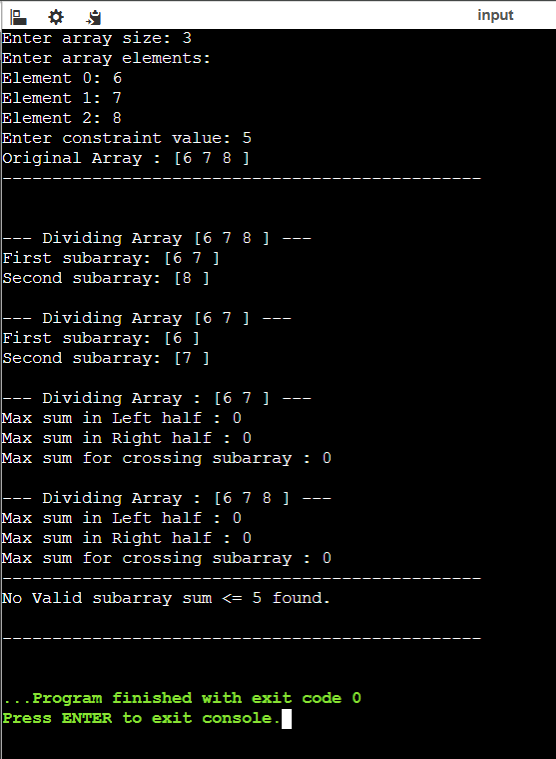
return 0;

}

**OUTPUT :**

****

****

****

**TASK 2 :**

**CODE :**

**// To find max crossing sum**

**int maxCrossingSum(int\* nums, int low, int mid, int high) {**

**// For Left**

**int sum = 0;**

**int left\_sum = INT\_MIN;**

**for (int i = mid; i >= low; i--) {**

**sum += nums[i];**

**if (sum > left\_sum){**

**left\_sum = sum;**

**}**

**}**

**// For Right**

**sum = 0;**

**int right\_sum = INT\_MIN;**

**for (int i = mid + 1; i <= high; i++) {**

**sum += nums[i];**

**if (sum > right\_sum) right\_sum = sum;**

**}**

**return left\_sum + right\_sum;**

**}**

**// Recursively Divide**

**int maxSubArraySum(int\* nums, int low, int high) {**

**if (low == high) {**

**// Base case: one element**

**return nums[low];**

**}**

**int mid = low + (high - low) / 2;**

**int left\_max = maxSubArraySum(nums, low, mid);**

**int right\_max = maxSubArraySum(nums, mid + 1, high);**

**int cross\_max = maxCrossingSum(nums, low, mid, high);**

**// Return maximum of the three**

**if (left\_max >= right\_max && left\_max >= cross\_max)**

**return left\_max;**

**else if (right\_max >= left\_max && right\_max >= cross\_max)**

**return right\_max;**

**else**

**return cross\_max;**

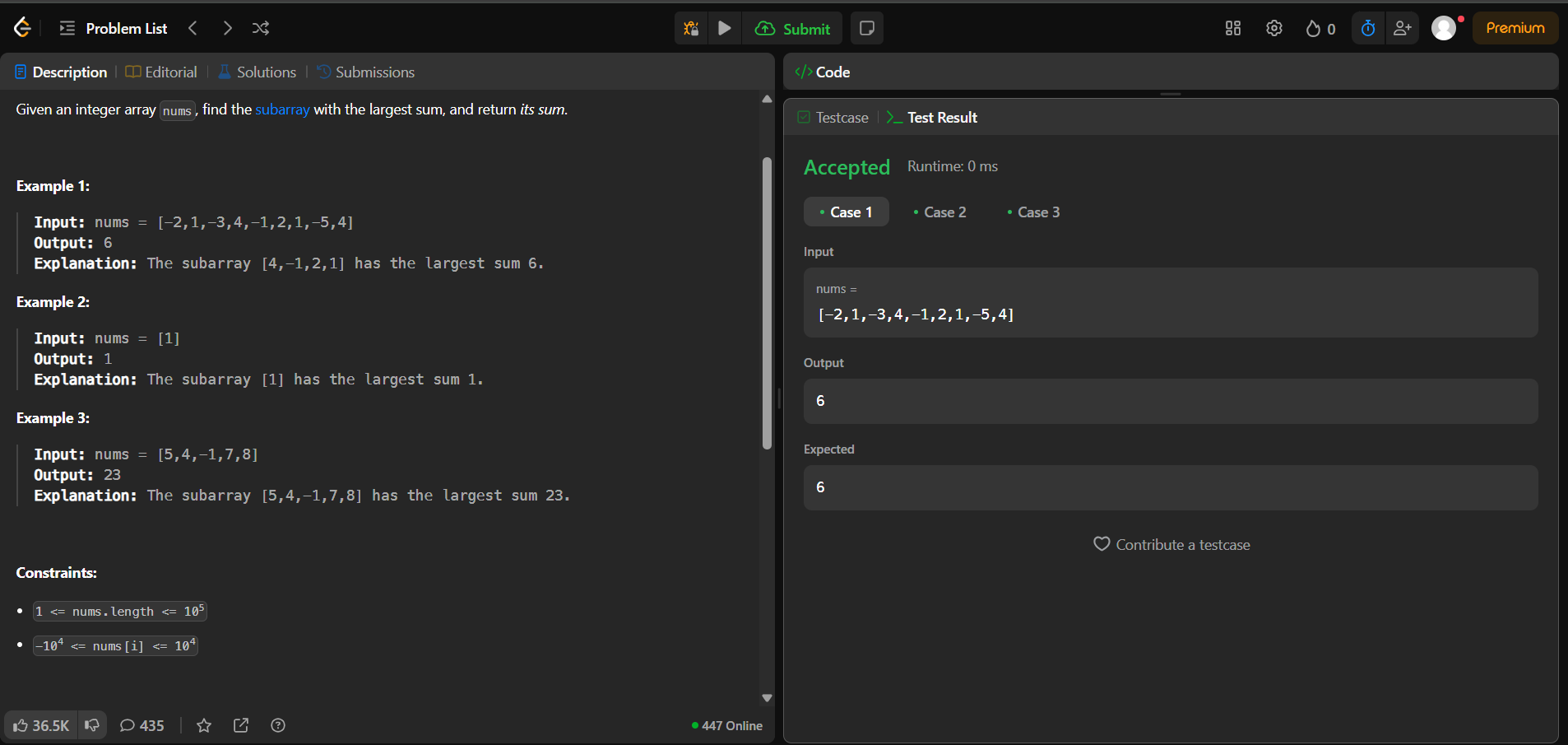
**}**

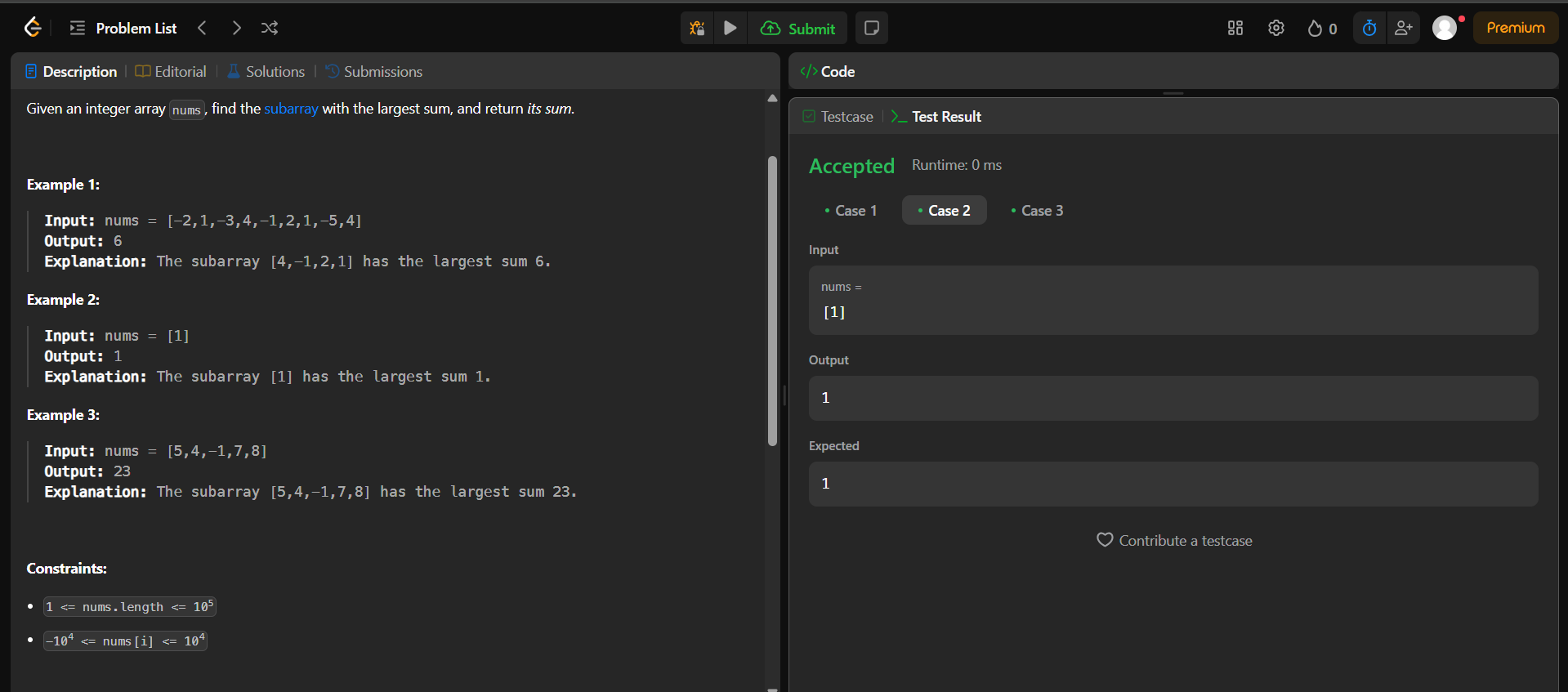
**int maxSubArray(int\* nums, int numsSize) {**

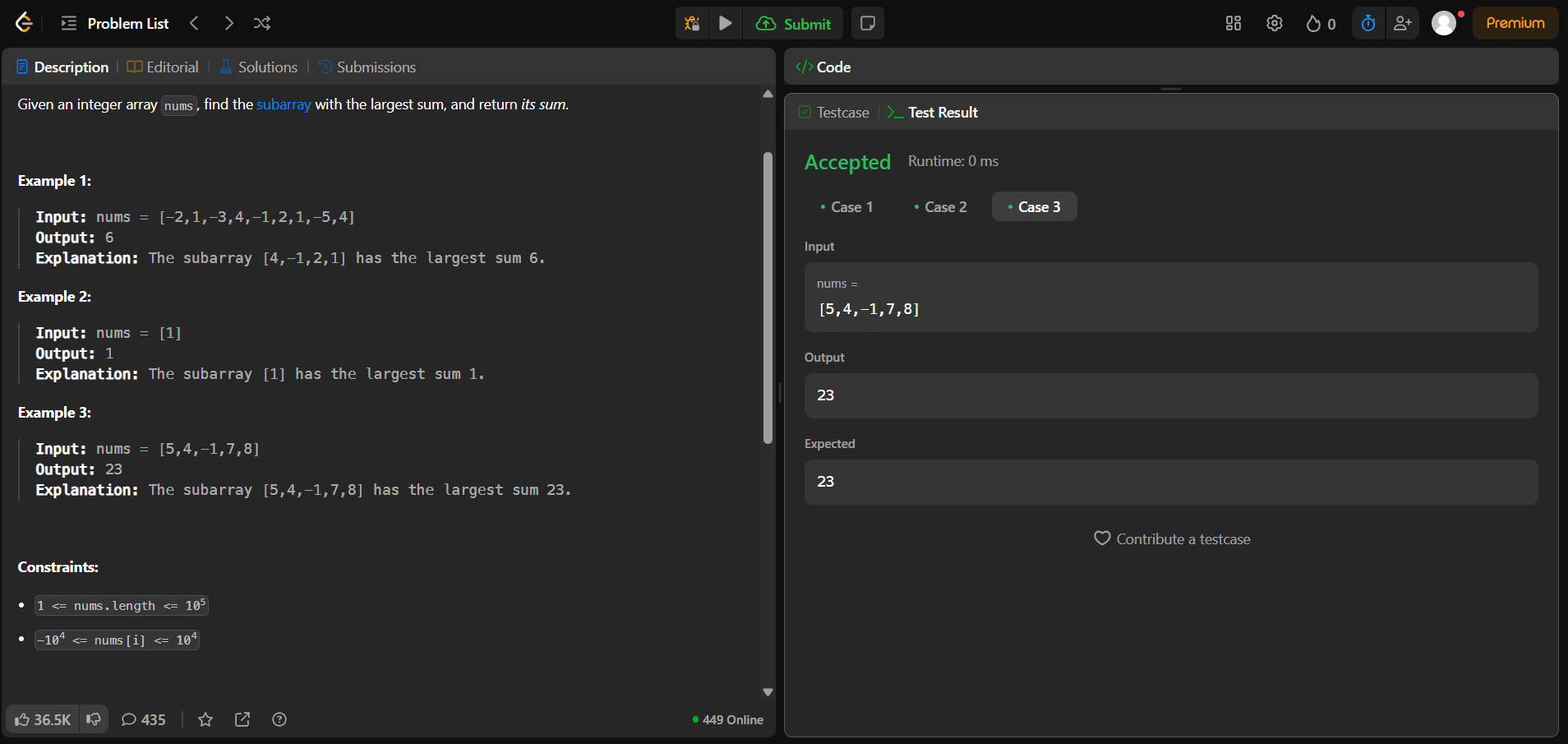
**return maxSubArraySum(nums, 0, numsSize - 1);**

**}**

**SCREENSHOT :**

****

****

****