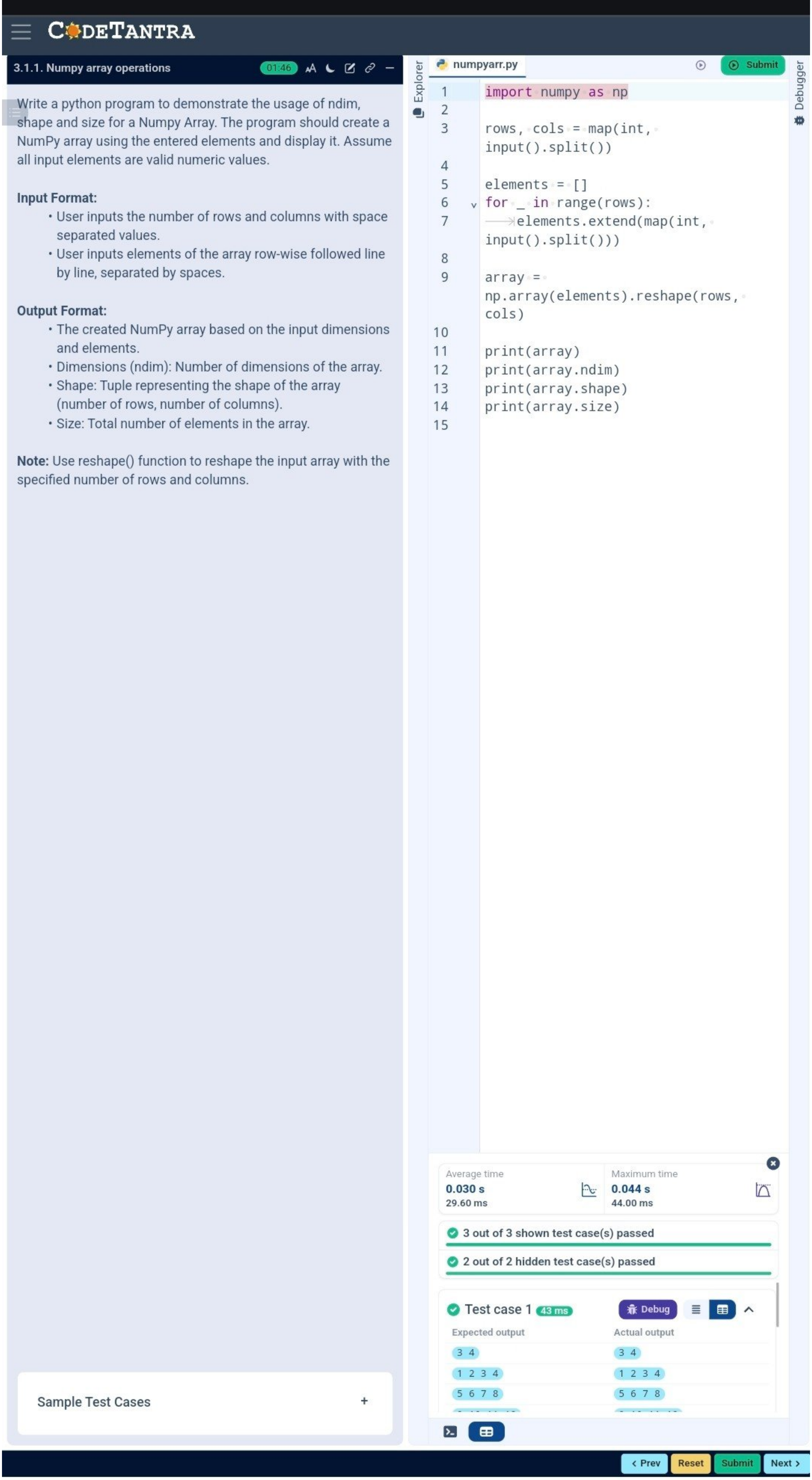


3.1.1. Numpy array operations

Write a python program to demonstrate the usage of ndim, shape and size for a Numpy Array. The program should create a NumPy array using the entered elements and display it. Assume all input elements are valid numeric values.

Input Format:

- User inputs the number of rows and columns with space separated values.
- User inputs elements of the array row-wise followed line by line, separated by spaces.

Output Format:

- The created NumPy array based on the input dimensions and elements.
- Dimensions (ndim): Number of dimensions of the array.
- Shape: Tuple representing the shape of the array (number of rows, number of columns).
- Size: Total number of elements in the array.

Note: Use reshape() function to reshape the input array with the specified number of rows and columns.

```python
import numpy as np

rows, cols = map(int, input().split())

elements = []
for _ in range(rows):
	elements.extend(map(int, input().split()))

array = np.array(elements).reshape(rows, cols)

print(array)
print(array.ndim)
print(array.shape)
print(array.size)

```

Average time
0.030 s
29.60 ms

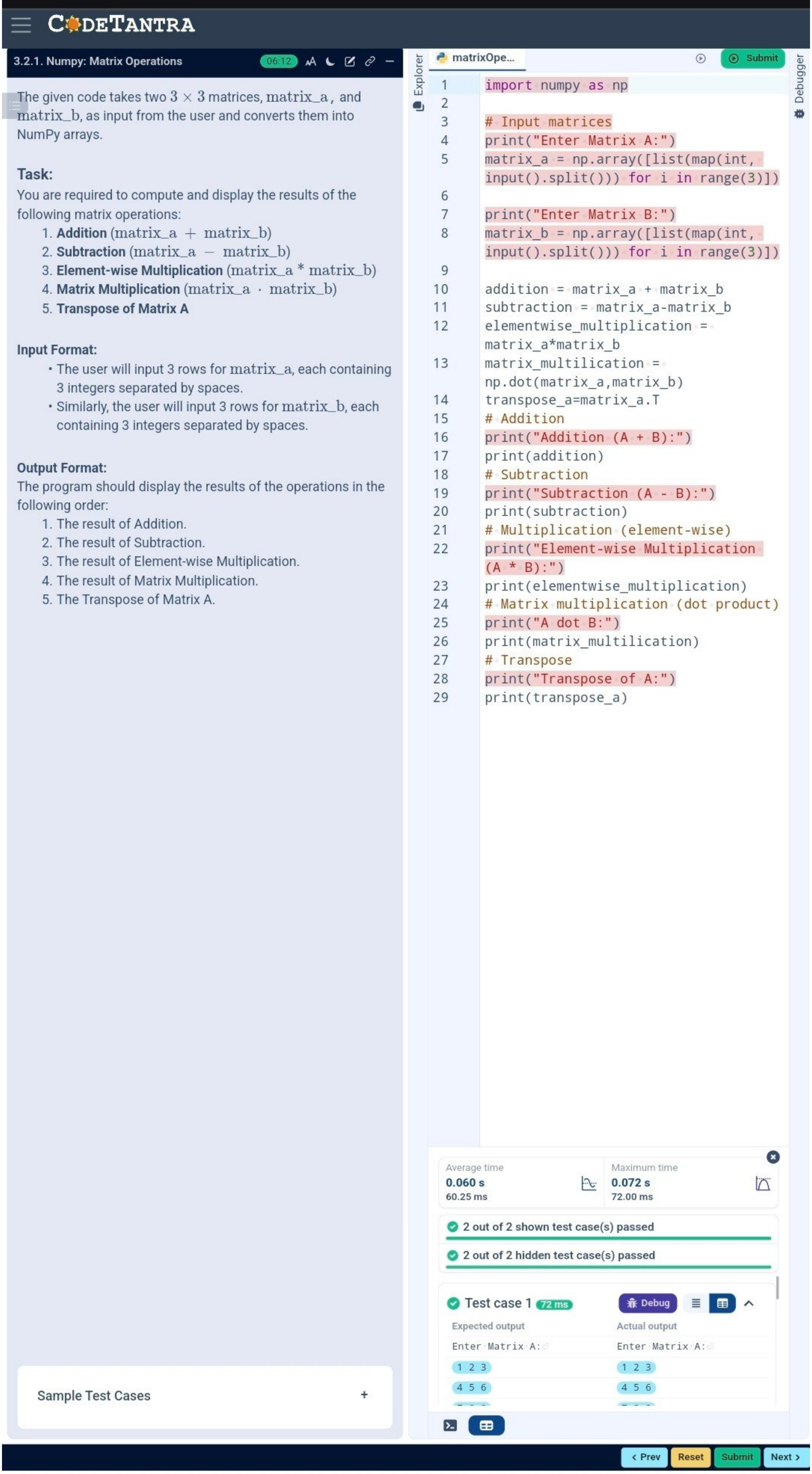
Maximum time
0.044 s
44.00 ms

✓ 3 out of 3 shown test case(s) passed

✓ 2 out of 2 hidden test case(s) passed

✓ Test case 1 43 ms

Expected output	Actual output
3 4	3 4
1 2 3 4	1 2 3 4
5 6 7 8	5 6 7 8



3.2.1. Numpy: Matrix Operations

The given code takes two 3×3 matrices, matrix_a, and matrix_b, as input from the user and converts them into NumPy arrays.

Task:

You are required to compute and display the results of the following matrix operations:

1. **Addition** (matrix_a + matrix_b)
2. **Subtraction** (matrix_a − matrix_b)
3. **Element-wise Multiplication** (matrix_a * matrix_b)
4. **Matrix Multiplication** (matrix_a · matrix_b)
5. **Transpose of Matrix A**

Input Format:

- The user will input 3 rows for matrix_a, each containing 3 integers separated by spaces.
- Similarly, the user will input 3 rows for matrix_b, each containing 3 integers separated by spaces.

Output Format:

The program should display the results of the operations in the following order:

1. The result of Addition.
2. The result of Subtraction.
3. The result of Element-wise Multiplication.
4. The result of Matrix Multiplication.
5. The Transpose of Matrix A.

Sample Test Cases +

```python
import numpy as np

# Input matrices
print("Enter Matrix A:")
matrix_a = np.array([list(map(int, input().split())) for i in range(3)])

print("Enter Matrix B:")
matrix_b = np.array([list(map(int, input().split())) for i in range(3)])

addition = matrix_a + matrix_b
subtraction = matrix_a-matrix_b
elementwise_multiplication = matrix_a*matrix_b
matrix_multilication = np.dot(matrix_a,matrix_b)
transpose_a=matrix_a.T
# Addition
print("Addition (A + B):")
print(addition)
# Subtraction
print("Subtraction (A - B):")
print(subtraction)
# Multiplication (element-wise)
print("Element-wise Multiplication (A * B):")
print(elementwise_multiplication)
# Matrix multiplication (dot product)
print("A dot B:")
print(matrix_multilication)
# Transpose
print("Transpose of A:")
print(transpose_a)
```

Average time
0.060 s
60.25 ms

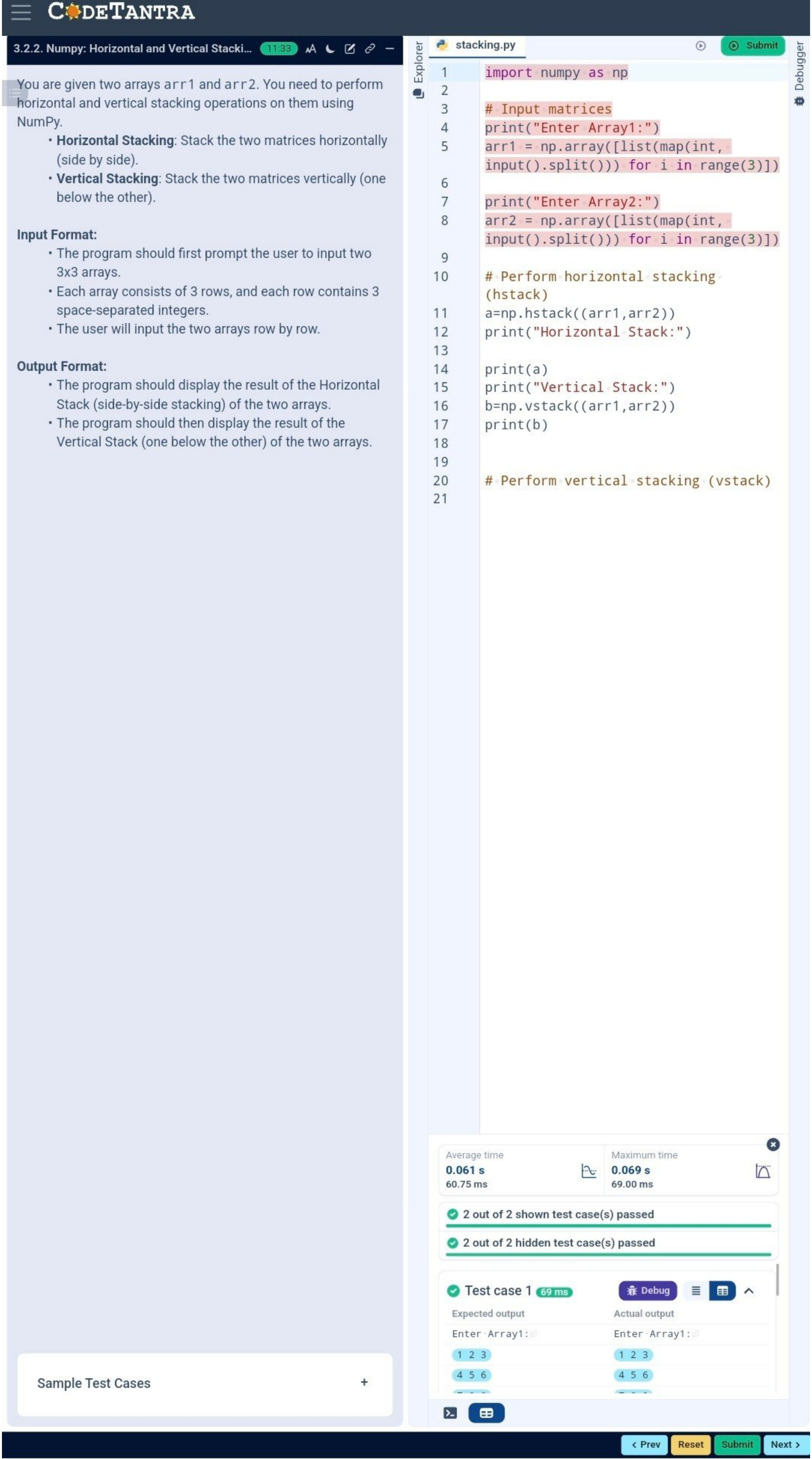
Maximum time
0.072 s
72.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 72 ms

Expected output	Actual output
Enter Matrix A:	Enter Matrix A:
1 2 3	1 2 3
4 5 6	4 5 6



You are given two arrays `arr1` and `arr2`. You need to perform horizontal and vertical stacking operations on them using NumPy.

- **Horizontal Stacking**: Stack the two matrices horizontally (side by side).
- **Vertical Stacking**: Stack the two matrices vertically (one below the other).

Input Format:

- The program should first prompt the user to input two 3x3 arrays.
- Each array consists of 3 rows, and each row contains 3 space-separated integers.
- The user will input the two arrays row by row.

Output Format:

- The program should display the result of the Horizontal Stack (side-by-side stacking) of the two arrays.
- The program should then display the result of the Vertical Stack (one below the other) of the two arrays.

Sample Test Cases +

```python
import numpy as np

# Input matrices
print("Enter Array1:")
arr1 = np.array([list(map(int, input().split())) for i in range(3)])

print("Enter Array2:")
arr2 = np.array([list(map(int, input().split())) for i in range(3)])

# Perform horizontal stacking (hstack)
a=np.hstack((arr1,arr2))
print("Horizontal Stack:")

print(a)
print("Vertical Stack:")
b=np.vstack((arr1,arr2))
print(b)


# Perform vertical stacking (vstack)

```

Average time
0.061 s
60.75 ms

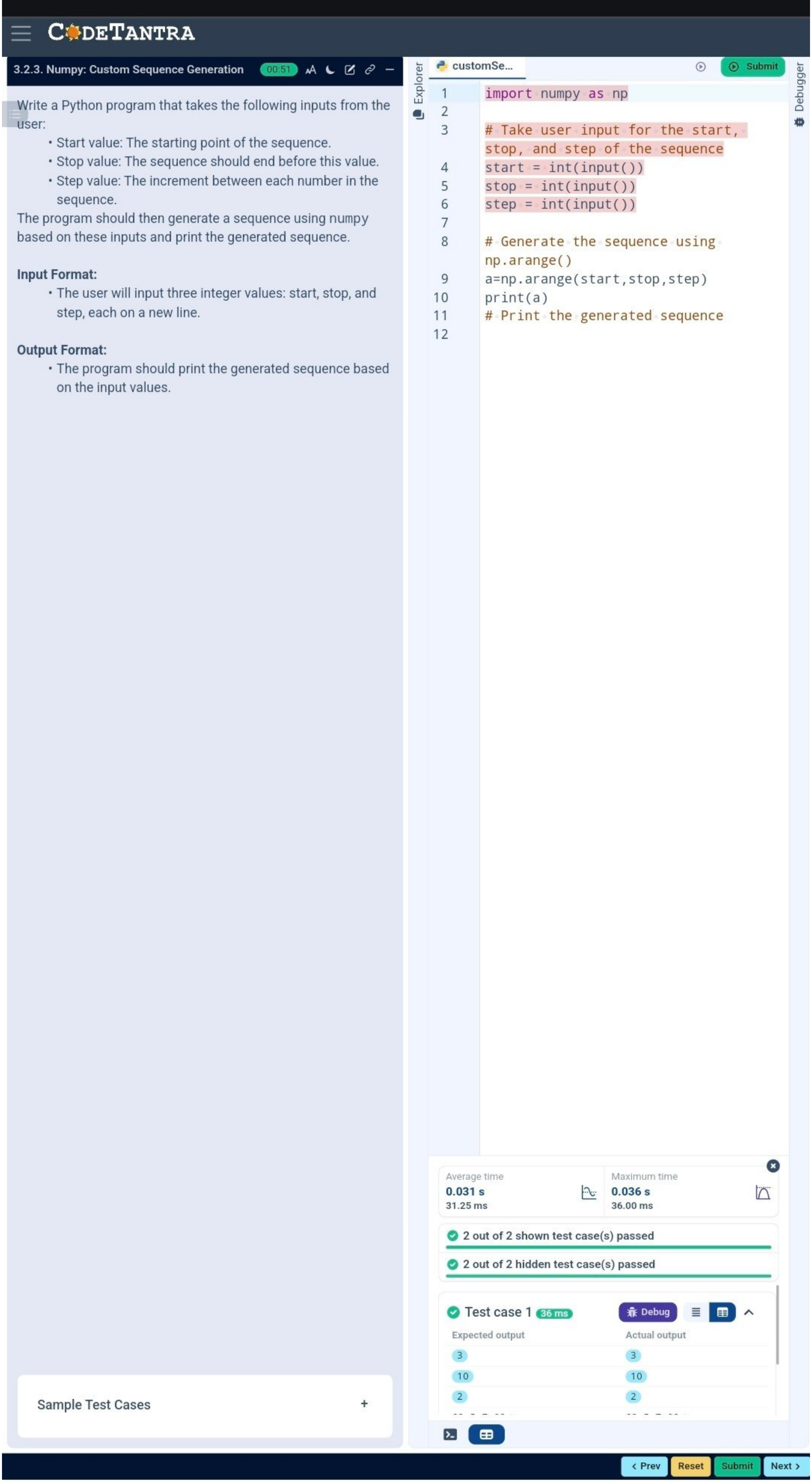
Maximum time
0.069 s
69.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 69 ms

Expected output	Actual output
Enter Array1:	Enter Array1:
1 2 3	1 2 3
4 5 6	4 5 6



3.2.3. Numpy: Custom Sequence Generation

Write a Python program that takes the following inputs from the user:

- Start value: The starting point of the sequence.
- Stop value: The sequence should end before this value.
- Step value: The increment between each number in the sequence.

The program should then generate a sequence using numpy based on these inputs and print the generated sequence.

Input Format:

- The user will input three integer values: start, stop, and step, each on a new line.

Output Format:

- The program should print the generated sequence based on the input values.

Sample Test Cases +

```python
import numpy as np

# Take user input for the start, stop, and step of the sequence
start = int(input())
stop = int(input())
step = int(input())

# Generate the sequence using np.arange()
a=np.arange(start,stop,step)
print(a)
# Print the generated sequence

```

Average time
0.031 s
31.25 ms

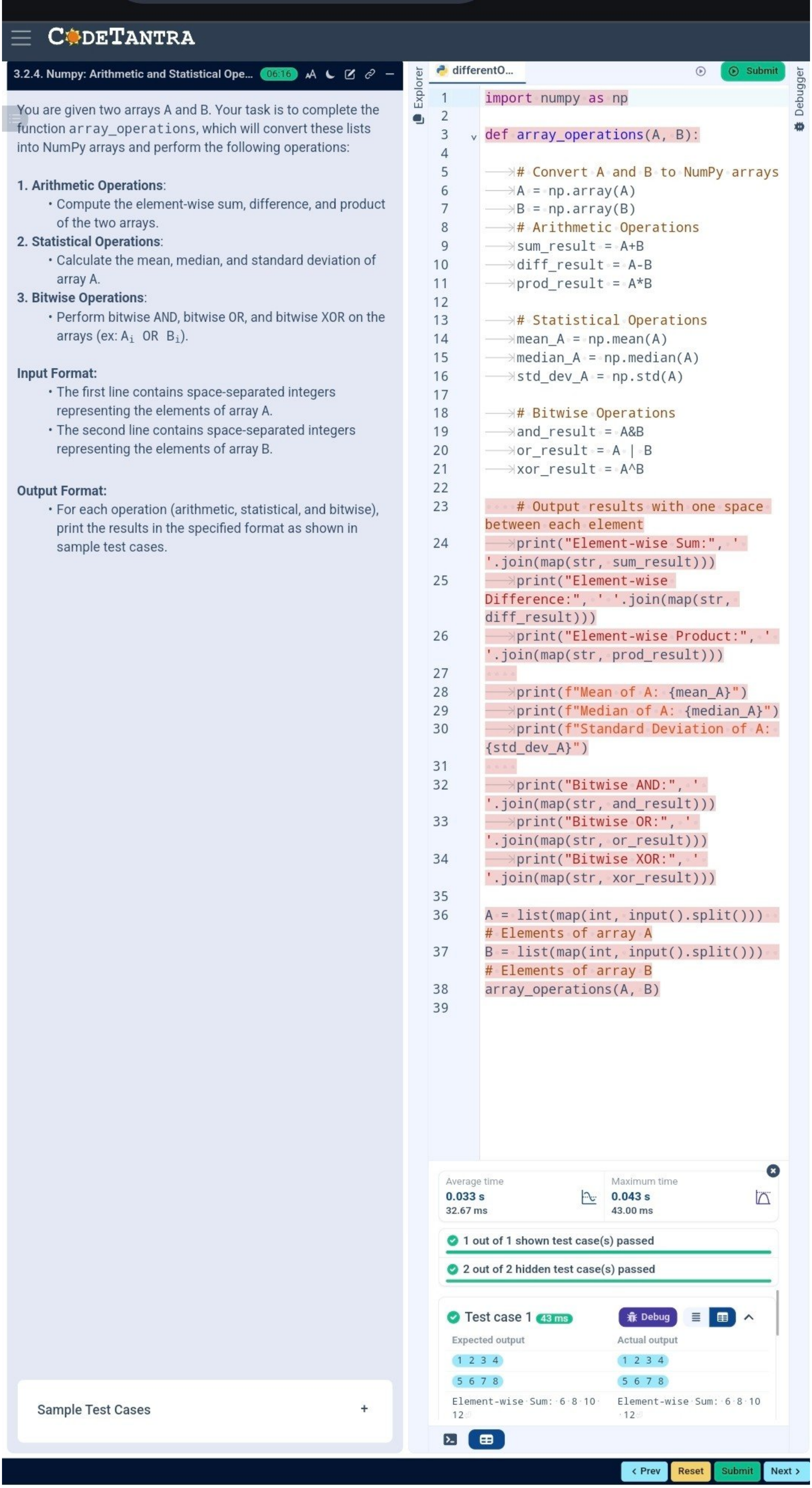
Maximum time
0.036 s
36.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 36 ms

Expected output	Actual output
3	3
10	10
2	2



You are given two arrays A and B. Your task is to complete the function `array_operations`, which will convert these lists into NumPy arrays and perform the following operations:

1. **Arithmetic Operations**:
    - Compute the element-wise sum, difference, and product of the two arrays.
2. **Statistical Operations**:
    - Calculate the mean, median, and standard deviation of array A.
3. **Bitwise Operations**:
    - Perform bitwise AND, bitwise OR, and bitwise XOR on the arrays (ex: A_i OR B_i).

Input Format:

- The first line contains space-separated integers representing the elements of array A.
- The second line contains space-separated integers representing the elements of array B.

Output Format:

- For each operation (arithmetic, statistical, and bitwise), print the results in the specified format as shown in sample test cases.

```python
import numpy as np

def array_operations(A, B):

    # Convert A and B to NumPy arrays
    A = np.array(A)
    B = np.array(B)
    # Arithmetic Operations
    sum_result = A+B
    diff_result = A-B
    prod_result = A*B

    # Statistical Operations
    mean_A = np.mean(A)
    median_A = np.median(A)
    std_dev_A = np.std(A)

    # Bitwise Operations
    and_result = A&B
    or_result = A | B
    xor_result = A^B

    # Output results with one space between each element
    print("Element-wise Sum:", ' '.join(map(str, sum_result)))
    print("Element-wise Difference:", ' '.join(map(str, diff_result)))
    print("Element-wise Product:", ' '.join(map(str, prod_result)))
    
    print(f"Mean of A: {mean_A}")
    print(f"Median of A: {median_A}")
    print(f"Standard Deviation of A: {std_dev_A}")
    
    print("Bitwise AND:", ' '.join(map(str, and_result)))
    print("Bitwise OR:", ' '.join(map(str, or_result)))
    print("Bitwise XOR:", ' '.join(map(str, xor_result)))

A = list(map(int, input().split()))  # Elements of array A
B = list(map(int, input().split()))  # Elements of array B
array_operations(A, B)

```

Average time
0.033 s
32.67 ms

Maximum time
0.043 s
43.00 ms

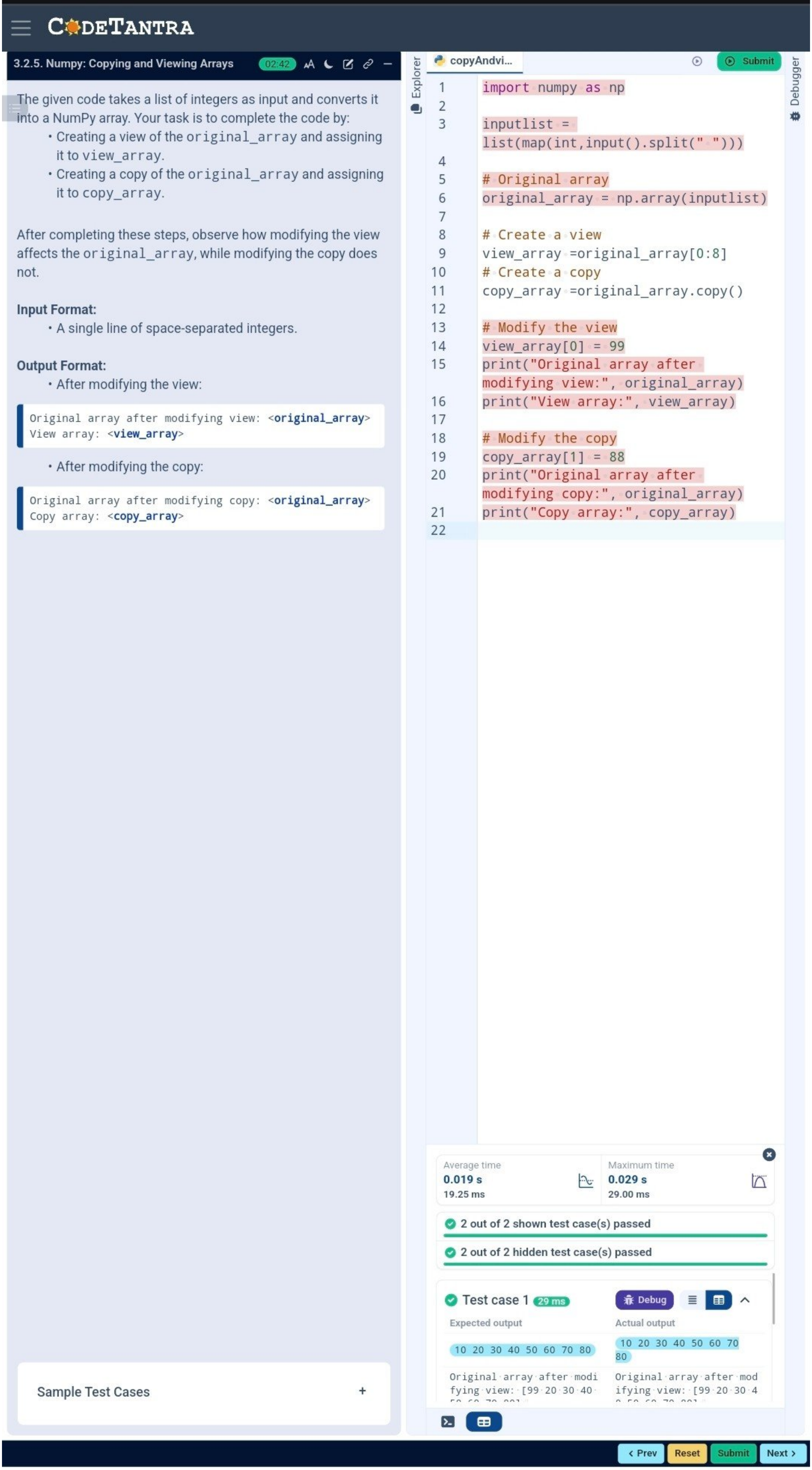
✓ 1 out of 1 shown test case(s) passed

✓ 2 out of 2 hidden test case(s) passed

✓ Test case 1 43 ms

Expected output	Actual output
1 2 3 4	1 2 3 4
5 6 7 8	5 6 7 8
Element-wise Sum: 6 8 10 12	Element-wise Sum: 6 8 10 12

Sample Test Cases +



3.2.5. Numpy: Copying and Viewing Arrays

The given code takes a list of integers as input and converts it into a NumPy array. Your task is to complete the code by:

- Creating a view of the `original_array` and assigning it to `view_array`.
- Creating a copy of the `original_array` and assigning it to `copy_array`.

After completing these steps, observe how modifying the view affects the `original_array`, while modifying the copy does not.

Input Format:

- A single line of space-separated integers.

Output Format:

- After modifying the view:

```
Original array after modifying view: <original_array>
View array: <view_array>
```

- After modifying the copy:

```
Original array after modifying copy: <original_array>
Copy array: <copy_array>
```

Sample Test Cases +

```python
import numpy as np

inputlist = list(map(int,input().split(" ")))

# Original array
original_array = np.array(inputlist)

# Create a view
view_array =original_array[0:8]
# Create a copy
copy_array =original_array.copy()

# Modify the view
view_array[0] = 99
print("Original array after modifying view:", original_array)
print("View array:", view_array)

# Modify the copy
copy_array[1] = 88
print("Original array after modifying copy:", original_array)
print("Copy array:", copy_array)

```

Average time
0.019 s
19.25 ms

Maximum time
0.029 s
29.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 29 ms

Expected output

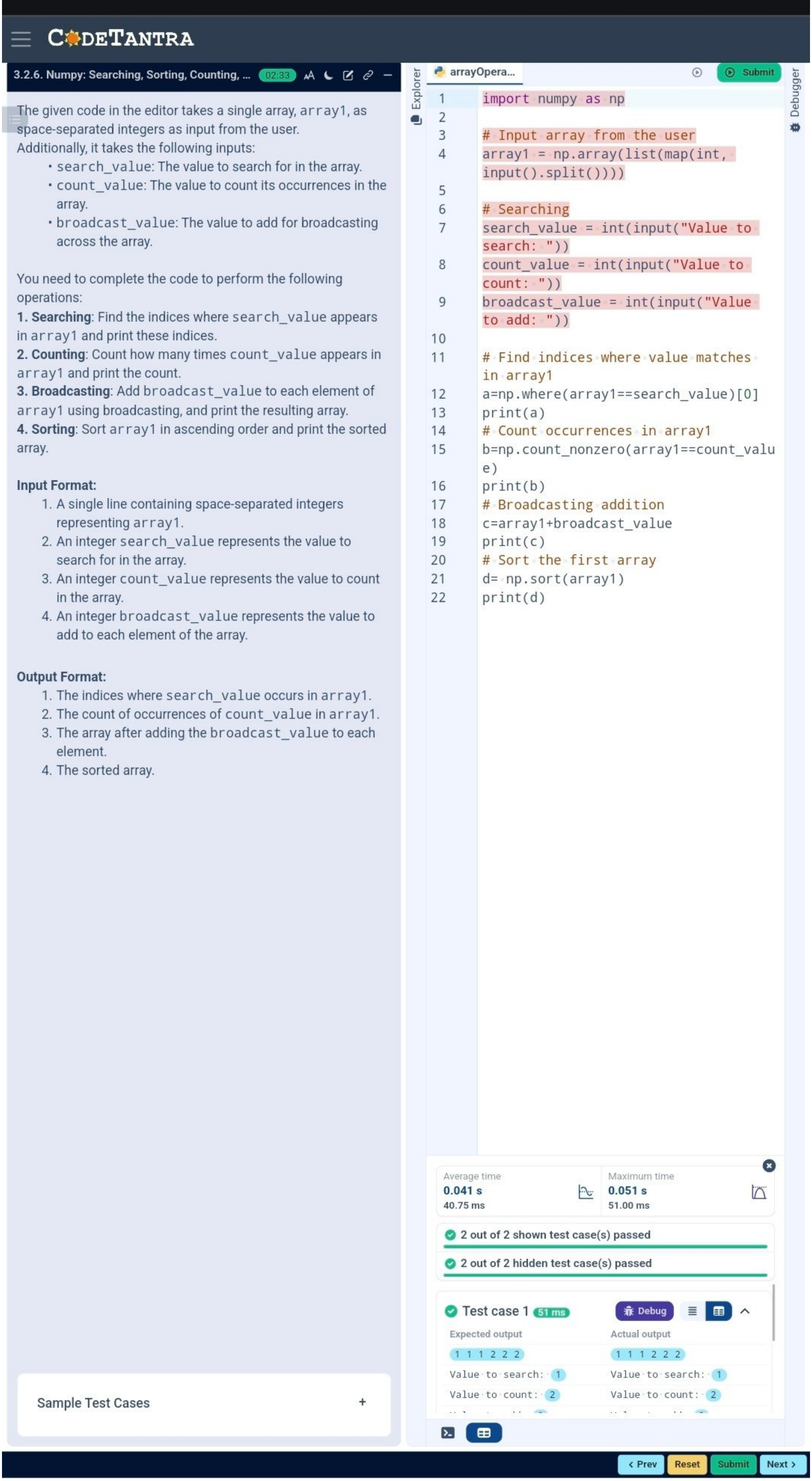
```
10 20 30 40 50 60 70 80

Original array after modi
fying view: [99 20 30 40
```

Actual output

```
10 20 30 40 50 60 70
80

Original array after mod
ifying view: [99 20 30 4
```



The given code in the editor takes a single array, `array1`, as space-separated integers as input from the user.
Additionally, it takes the following inputs:

- `search_value`: The value to search for in the array.
- `count_value`: The value to count its occurrences in the array.
- `broadcast_value`: The value to add for broadcasting across the array.

You need to complete the code to perform the following operations:

1. Searching: Find the indices where `search_value` appears in `array1` and print these indices.
2. Counting: Count how many times `count_value` appears in `array1` and print the count.
3. Broadcasting: Add `broadcast_value` to each element of `array1` using broadcasting, and print the resulting array.
4. Sorting: Sort `array1` in ascending order and print the sorted array.

Input Format:

1. A single line containing space-separated integers representing `array1`.
2. An integer `search_value` represents the value to search for in the array.
3. An integer `count_value` represents the value to count in the array.
4. An integer `broadcast_value` represents the value to add to each element of the array.

Output Format:

1. The indices where `search_value` occurs in `array1`.
2. The count of occurrences of `count_value` in `array1`.
3. The array after adding the `broadcast_value` to each element.
4. The sorted array.

Sample Test Cases +

```python
import numpy as np

# Input array from the user
array1 = np.array(list(map(int, input().split())))

# Searching
search_value = int(input("Value to search: "))
count_value = int(input("Value to count: "))
broadcast_value = int(input("Value to add: "))

# Find indices where value matches in array1
a=np.where(array1==search_value)[0]
print(a)
# Count occurrences in array1
b=np.count_nonzero(array1==count_value)
print(b)
# Broadcasting addition
c=array1+broadcast_value
print(c)
# Sort the first array
d= np.sort(array1)
print(d)
```

Average time
0.041 s
40.75 ms

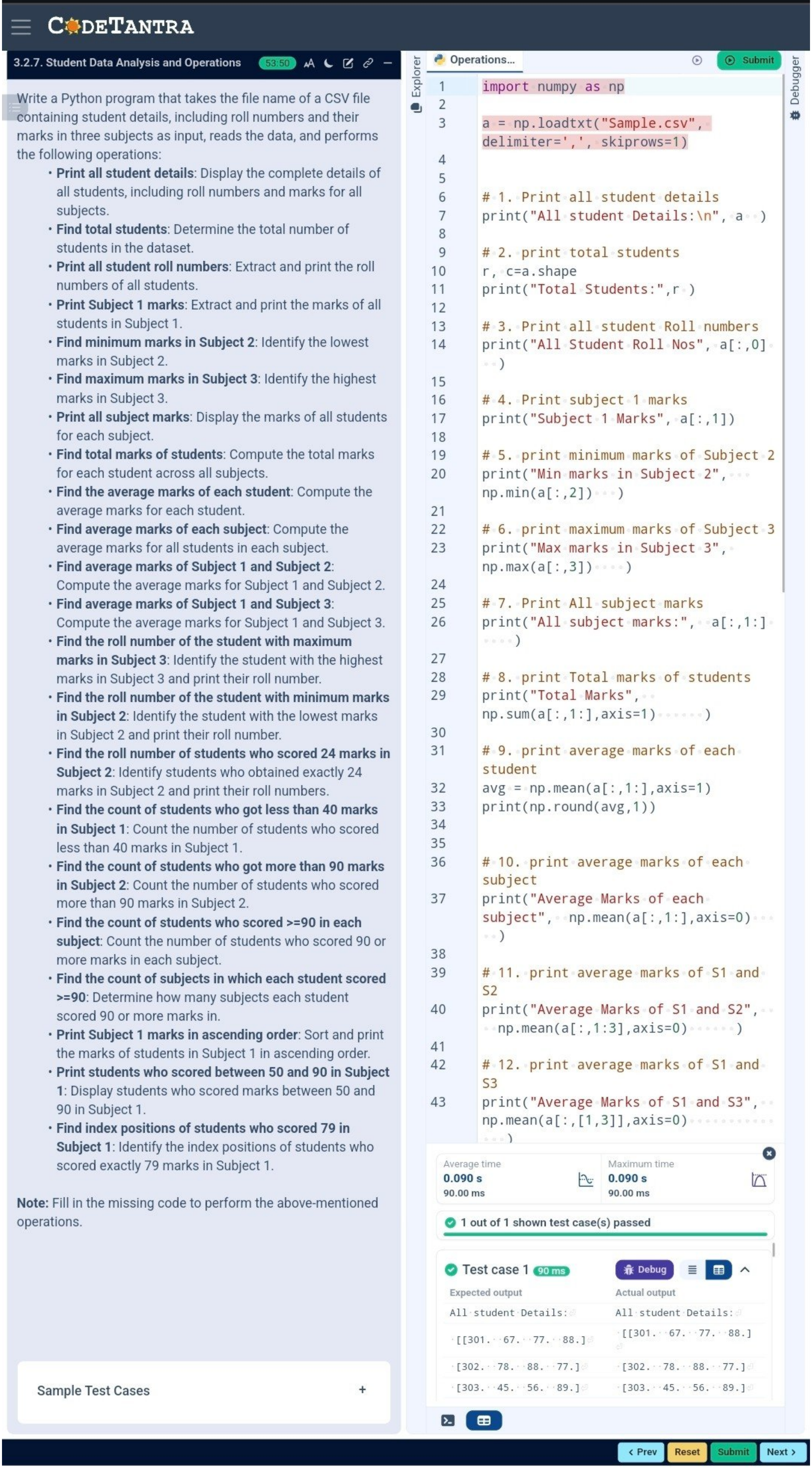
Maximum time
0.051 s
51.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 51 ms

Expected output	Actual output
1 1 1 2 2 2	1 1 1 2 2 2
Value to search: 1	Value to search: 1
Value to count: 2	Value to count: 2



3.2.7. Student Data Analysis and Operations

Write a Python program that takes the file name of a CSV file containing student details, including roll numbers and their marks in three subjects as input, reads the data, and performs the following operations:

- **Print all student details**: Display the complete details of all students, including roll numbers and marks for all subjects.
- **Find total students**: Determine the total number of students in the dataset.
- **Print all student roll numbers**: Extract and print the roll numbers of all students.
- **Print Subject 1 marks**: Extract and print the marks of all students in Subject 1.
- **Find minimum marks in Subject 2**: Identify the lowest marks in Subject 2.
- **Find maximum marks in Subject 3**: Identify the highest marks in Subject 3.
- **Print all subject marks**: Display the marks of all students for each subject.
- **Find total marks of students**: Compute the total marks for each student across all subjects.
- **Find the average marks of each student**: Compute the average marks for each student.
- **Find average marks of each subject**: Compute the average marks for all students in each subject.
- **Find average marks of Subject 1 and Subject 2**: Compute the average marks for Subject 1 and Subject 2.
- **Find average marks of Subject 1 and Subject 3**: Compute the average marks for Subject 1 and Subject 3.
- **Find the roll number of the student with maximum marks in Subject 3**: Identify the student with the highest marks in Subject 3 and print their roll number.
- **Find the roll number of the student with minimum marks in Subject 2**: Identify the student with the lowest marks in Subject 2 and print their roll number.
- **Find the roll number of students who scored 24 marks in Subject 2**: Identify students who obtained exactly 24 marks in Subject 2 and print their roll numbers.
- **Find the count of students who got less than 40 marks in Subject 1**: Count the number of students who scored less than 40 marks in Subject 1.
- **Find the count of students who got more than 90 marks in Subject 2**: Count the number of students who scored more than 90 marks in Subject 2.
- **Find the count of students who scored >=90 in each subject**: Count the number of students who scored 90 or more marks in each subject.
- **Find the count of subjects in which each student scored >=90**: Determine how many subjects each student scored 90 or more marks in.
- **Print Subject 1 marks in ascending order**: Sort and print the marks of students in Subject 1 in ascending order.
- **Print students who scored between 50 and 90 in Subject 1**: Display students who scored marks between 50 and 90 in Subject 1.
- **Find index positions of students who scored 79 in Subject 1**: Identify the index positions of students who scored exactly 79 marks in Subject 1.

Note: Fill in the missing code to perform the above-mentioned operations.

Sample Test Cases +

```python
import numpy as np

a = np.loadtxt("Sample.csv", delimiter=',', skiprows=1)


# 1. Print all student details
print("All student Details:\n", a  )

# 2. print total students
r, c=a.shape
print("Total Students:",r )

# 3. Print all student Roll numbers
print("All Student Roll Nos", a[:,0]  )

# 4. Print subject 1 marks
print("Subject 1 Marks", a[:,1])

# 5. print minimum marks of Subject 2
print("Min marks in Subject 2",   np.min(a[:,2])   )

# 6. print maximum marks of Subject 3
print("Max marks in Subject 3",  np.max(a[:,3])    )

# 7. Print All subject marks
print("All subject marks:",  a[:,1:]     )

# 8. print Total marks of students
print("Total Marks",  np.sum(a[:,1:],axis=1)      )

# 9. print average marks of each student
avg = np.mean(a[:,1:],axis=1)
print(np.round(avg,1))


# 10. print average marks of each subject
print("Average Marks of each subject",  np.mean(a[:,1:],axis=0)     )

# 11. print average marks of S1 and S2
print("Average Marks of S1 and S2",  np.mean(a[:,1:3],axis=0)      )

# 12. print average marks of S1 and S3
print("Average Marks of S1 and S3",  np.mean(a[:,[1,3]],axis=0)       )
```

Average time
0.090 s
90.00 ms

Maximum time
0.090 s
90.00 ms

1 out of 1 shown test case(s) passed

Test case 1 90 ms

Expected output	Actual output
All student Details:	All student Details:
[[301. 67. 77. 88.]	[[301. 67. 77. 88.]
[302. 78. 88. 77.]	[302. 78. 88. 77.]
[303. 45. 56. 89.]	[303. 45. 56. 89.]