931230187,G2,S1,علي عز علي عليان

931230015,G1,S1,احمد ابوبكر حنفى عبدالخالق

931230030,G1,S1,أحمد صبري حمزة هيكل

the MaximumProduct Function

```
def MaximumProduct(nums, s):
```

- This defines a function MaximumProduct that takes a list of numbers nums and an integer s

```
max1 = max2 = max3 = float('-inf')
```

- It initializes three variables max1, max2, and max3 to negative infinity
- representing the three largest numbers

```
min1 = min2 = float('inf')
```

- It also initializes two variables min1 and min2 to positive infinity
- representing the two smallest numbers

Finding the Maximum and Minimum Values

```
if num > max1:
```

- Checking if num is greater than max1
- This line checks if the current number num is greater than max1
- Updating max3, max2, max1

```
max3, max2, max1 = max2, max1, num
```

- If num is greater than max1, it means num is the largest number
- The previous max1 becomes max2
- The previous max2 becomes max3
- num is assigned to max1

```
elif num > max2:
```

- Checking if num is greater than max2 but not greater than max1
- This line checks if num is greater than max2 but not greater than max1

```
max3, max2 = max2, num
```

- Updating max3, max2
- If num is greater than max2, it means num is the second largest number
- The previous max2 becomes max3
- num is assigned to max2
- Checking if num is greater than max3 but not greater than max2 or max1

```
elif num > max3:
```

- This line checks if num is greater than max3 but not greater than max2 or max1
- Updating max3

```
max3 = num
```

it calculates two possible products:

```
prod1 = max1 * max2 * max3
```

```
prod2 = max1 * min1 * min2
```

return prod1 is the product of the three largest numbers

```
if prod1 > prod2:
```

returns the larger of these two products

```
return prod1
```

returns the product of the largest number and the two smallest numbers

- If num is greater than max3, it means num is the third largest number
- num is assigned to max3
- prod2 is the product of the largest number and the two smallest numbers

```
while True:
```

- This loop asks the user for the size of the array s and ensures it is a positive integer

```
try:
    s = int(input("Enter size of array: "))
    if s > 0:
        break
    print("Please enter a positive number")
except ValueError:
    print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
```

- This initializes nums with zeros and fills it with user input

```
            break
        except ValueError:
```

- It ensures that each input is a valid floating-point number

```
            print("Please enter a valid number")
```

best case

```
f s <= 3:
```

- If s is less than or equal to 3
- it calculates the product of all numbers directly
- If s is greater than 3
- it calls MaximumProduct and prints the result

```
    maxp = 1
    for num in nums:
```

```
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

Pseudocode

Function MaximumProduct(nums, s):

Initialize max1, max2, max3 to negative infinity

Initialize min1, min2 to positive infinity

For each num in nums:

If num is greater than max1:

Update max3, max2, max1

Else If num is greater than max2:

Update max3, max2

Else If num is greater than max3:

Update max3

If num is less than min1:

Update min2, min1

Else If num is less than min2:

Update min2

prod1 = max1 * max2 * max3

prod2 = max1 * min2* min1

Return the greater of prod1 and prod2

While True:

Try to get input s as integer

If s is a positive number:

Break the loop

Else:

Print message to enter a positive number

If input is invalid:

Print message to enter a valid number

Initialize nums as a list of zeros with length s

For i from 0 to s-1:

While True:

Try to get nums[i] as floating point number

If input is valid:

Break the loop

Else:

Print message to enter a valid number

If s is less than or equal to 3:

Initialize maxp to 1

For each num in nums:

Multiply maxp by num

Print the maximum product maxp

Else:

Print the maximum product by calling MaximumProduct(nums, s)

the time complexity

```
def MaximumProduct(nums, s):
```

```
max1 = max2 = max3 = float('-inf')
```

```
min1 = min2 = float('inf')
```

Time Complexity: **O(1)**

Initializing variables is a **constant time operation**

```
for num in nums:
  if num > max1:
    max3, max2, max1 = max2, max1, num
  elif num > max2:
    max3, max2 = max2, num
  elif num > max3:
    max3 = num
  if num < min1:
    min2, min1 = min1, num
  elif num < min2:
    min2 = num
```

$$T(n) = \sum_{l=0}^{n-1} 1 = n$$

This summation means that we are summing the value 1 for each l from 0 to n-1
he total sum is n

Time Complexity: **O(n)**

```
prod1 = max1 * max2 * max3
prod2 = max1 * min1 * min2
if prod1 > prod2:
 prod1 = max1 * max2 * max3
prod2 = max1 * min1 * min1
if prod1 > prod2:
  return prod1
else:
```

```
    return prod2
```

Time Complexity: **O(1) constant time operation**

```
while True:
  try:
    s = int(input("Enter size of array: "))
    if s > 0:
      break
    print("Please enter a positive number")
  except ValueError:
    print("Please enter a valid number")
```

Time Complexity: **O(1) constant time operation**

```
nums = [0]*s
for i in range(s):
  while True:
    try:
      nums[i] += float(input(f"Enter element {i + 1}: "))
      break
    except ValueError:
      print("Please enter a valid number")
```

$$T(n) = \sum_{l=0}^{n-1} 1 = n$$

$$T(s) = s$$

Time Complexity: ***O*(s)**

```
if s <= 3: # best case
```

Time Complexity: **O(1)**

```
  maxp = 1
for num in nums:
  maxp *= num
  print(f"The maximum product is: {maxp:.1f}")
else: print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

For the case where $s \leq 3$:

$$T(s) = \sum_{i=0}^{s-1} 1 = s$$

For the case where $s > 3$:

$$T(n) = \sum_{i=0}^{n-1} 1 = n$$

Initialize maxp:

```
maxp=1
```

$$\sum_{i=0}^{s-1} (\text{maxp} \times = \text{nums}[i])$$

- If s is less than or equal to 3: ***O*(s)**
- If s is greater than 3: ***O*(n)**

Overall Time Complexity: ***O*(n)**

Median of Two Sorted Arrays

Function `recursive`

```
def recursive(arr, index=0):
  try:
    VC = arr[index]
    return 1 + recursive(arr, index + 1)
  except IndexError:
    return 0
```

recursive function recursive: Takes an array arr and an index index (default is 0)

- Try block:
  VC = arr[index]: Attempts to access the element at the current index
  return 1 + recursive(arr, index + 1):
  Returns 1 plus the result of the recursive call with the next index
- Except block:

return 0:
Returns 0 if an IndexError

Function `mg` (Merge Sort)

```
def mg(nums):
  n = recursive(nums)
  if n > 1:
    mid = n // 2
    l = nums[:mid]
    r = nums[mid:]
```

```
mg(l)
mg(r)
i = j = k = 0
lf = recursive(l)
lr = recursive(r)
while i < lf and j < lr:
  if l[i] < r[j]:
    nums[k] = l[i]
     i += 1
  else:
    nums[k] = r[j]
    j += 1
  k += 1
while i < lf:
  nums[k] = l[i]
  i += 1
  k += 1
while j < lr:
  nums[k] = r[j]
   j += 1
  k += 1
return nums
```

function `mg`:

- Takes a list nums.

- Recursive call to recursive(nums):

`n = recursive(nums)`:
gets the length of nums

- If n > 1 (to avoid sorting a single element list):
- Splits the list into two halves l and r
- Recursively calls mg on each half

Initializes indices i, j, k to 0.

- If and lr store the lengths of l and r

Merging process:

- Compares elements of l and r and places the smaller one into nums

Returns the sorted list nums

```
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n % 2 == 0:
        return (arr[mid-1] + arr[mid]) / 2
    else:
        return arr[mid]
```

function `middle`:

- Takes two lists nums1, nums2, and their sizes s1, s2

Merges and sorts both lists:

- arr = mg(nums1 + nums2):
- Merges and sorts nums1 and nums2

Finds the median:

- n = s1 + s2: Total number of elements
- mid = n // 2: Middle index

If even:
- Returns the average of the two middle elements

If odd:
- Returns the middle element

`while True:`

- This creates an infinite loop that will continue running until it encounters a `break` statement.
- This loop is used to repeatedly prompt the user for input until a valid input is provided

```
try:
    s1 = int(input("Enter size of array: "))
```

The `try` block attempts to execute the code within it In this case, it tries to read user input, convert it to an integer, and assign it to the variable `s1` If the input is not a valid integer, it will raise a `ValueError` and jump to the `except` block

```
if s1 > 0:
    break
```

- This checks if the input value `s1` is a positive number (greater than 0)
- If this condition is true, the `break` statement is executed, which exits the infinite loop
- This ensures that the loop continues only if the user input is not valid or not positive

`print("Please enter a positive number")`

- If the `if` condition (`s1 > 0`) is not met, this statement is executed, printing the message "Please enter a positive number" to inform the user that their input must be a positive number

```
except ValueError:
    print("Please enter a positive number")
```

- If the input cannot be converted to an integer a `ValueError` is raised
- The `except` block catches this error and prints "Please enter a positive number" to notify the user that their input was invalid and needs to be a valid number

```
nums1 = [0] * s1
```

```
nums2 = [0] * s2
```

- These two lines initialize two lists `nums1` and `nums2` each with a length of `s1` and `s2` respectively
- All elements in these lists are initially set to `0`

```
for i in range(s1):
```

```
  while True:
```

```
    try:
```

```
      nums1[i] += float(input(f"Enter element {i + 1} array1: "))
```

```
      nums2[i] += float(input(f"Enter element {i + 1} array2: "))
```

```
      break
```

```
    except ValueError:
```

```
      print("Please enter a valid number")
```

  - For Loop: `for i in range(s1)` iterates through each index `i` from `0` to `s1-1`
  - Nested While Loop: `while True` creates an infinite loop that will continue until the input is valid
  - Try Block:
    - `nums1[i] += float(input(f"Enter element {i + 1} array1: "))` prompts the user to enter a number for `nums1` at index `i`, converts it to a float, and adds it to the current value of `nums1[i]`
    - `nums2[i] += float(input(f"Enter element {i + 1} array2: "))` does the same for `nums2` at index `i`
  - Break: If both inputs are valid, `break` exits the while loop
  - Except Block: If the input is not a valid number, a `ValueError` is raised, and the program prints "Please enter a valid number"

```
print(f"The middle is: {middle(nums1, nums2, s1, s2)}")
```

  - This line calls the `middle` function with the filled `nums1` and `nums2` lists and their respective sizes `s1` and `s2`
  - The `middle` function merges and sorts these lists, then finds
- returns the median value.

Pseudocode

```
Function recursive(arr, index = 0)
    Try:
        VC = arr[index]
        Return 1 + recursive(arr, index + 1)
    Except IndexError:
        Return 0
End Function
Function mg(nums)
    n = recursive(nums)
    If n > 1 Then
        mid = n // 2
        l = nums[0:mid]
        r = nums[mid:n]
        Call mg(l)
        Call mg(r)
        i, j, k = 0, 0, 0
        lf = recursive(l)
        lr = recursive(r)
        While i < lf AND j < lr
            If l[i] < r[j] Then
                nums[k] = l[i]
                i += 1
            Else
                nums[k] = r[j]
                j += 1
            End If
            k += 1
        End While
        While i < lf
            nums[k] = l[i]
            i += 1
            k += 1
        End While
        While j < lr
            nums[k] = r[j]
            j += 1
            k += 1
        End While
    End If
    Return nums
End Function
Function middle(nums1, nums2, s1, s2)
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    If n % 2 == 0 Then
        Return (arr[mid-1] + arr[mid]) / 2
    Else
        Return arr[mid]
    End If
End Function
While True
    Try:
        s2 = Input("Enter size of array: ")
        s2 = Convert to Integer(s2)
        If s2 > 0 Then
            Break
        End If
    Return nums
End Function
Function middle(nums1, nums2, s1, s2)
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    If n % 2 == 0 Then
        Return (arr[mid-1] + arr[mid]) / 2
    Else
        Return arr[mid]
    End If
End Function
While True
    Try:
        s2 = Input("Enter size of array: ")
        s2 = Convert to Integer(s2)
        If s2 > 0 Then
            Break
        End If
        Print("Please enter a positive number")
    Except ValueError:
        Print("Please enter a positive number")
End While
```

the time complexity

```
def recursive(arr, index=0):
  try:
    VC = arr[index]
    return 1 + recursive(arr, index + 1)
  except IndexError:
    return 0
```

- This function recursively leght the number of elements in the array

$$T_{\text{recursive}}(n) = \sum_{i=0}^{n-1} 1 = n$$

the time complexity: **O(n)**

`mg` Function (Merge Sort)

```
def mg(nums):
  n = recursive(nums)
  if n > 1:
    mid = n // 2
    l = nums[:mid]
     r = nums[mid:]
    mg(l)
    mg(r)
    i = j = k = 0
    lf = recursive(l)
    lr = recursive(r)
    while i < lf and j < lr:
      if l[i] < r[j]:
        nums[k] = l[i]
        i += 1
      else:
        nums[k] = r[j]
         j += 1
      k += 1
```

```
    while i < lf:
        nums[k] = l[i]
        i += 1
        k += 1
    while j < lr:
        nums[k] = r[j]
        j += 1
        k += 1
    return nums
```

$$T_{\text{mg}}(n) = 2T_{\text{mg}}\left(\frac{n}{2}\right) + \sum_{i=0}^{n-1} 1 = 2T_{\text{mg}}\left(\frac{n}{2}\right) + n$$

$$T_{\text{merge}}(n) = \sum_{i=0}^{n-1} 1 = n$$

$$T_{\text{merge}}(n) = \sum_{i=0}^{n-1} 1 = n$$

the time complexity: **O(n)**

```
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
```

```
    mid = n // 2
    if n % 2 == 0:
        return (arr[mid - 1] + arr[mid]) / 2
    else:
        return arr[mid]
```

$$T_{\text{middle}}(n) = T_{\text{mg}}(s1 + s2) = (s1 + s2)\log(s1 + s2)$$

the time complexity: **O((s1+s2)log(s1+s2))**

```
while True:
    try:
        s1 = int(input("Enter size of array: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
while True:
    try:
        s2 = int(input("Enter size of array: "))
        if s2 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
```

the time complexity: **O(1)**

```
nums1 = [0] * s1
nums2 = [0] * s2
for i in range(s1):
  while True:
    try:
      nums1[i] += float(input(f"Enter element {i + 1} array1: "))
      nums2[i] += float(input(f"Enter element {i + 1} array2: "))
      break
    except ValueError:
      print("Please enter a valid number")
```

$$T_{\text{filling}}(s1+s2) = \sum_{i=0}^{s1-1} 1 + \sum_{j=0}^{s2-1} 1 = s1+s2$$

the time complexity: **O(s1+s2)**

Recursive Function:

$$T_{\text{recursive}}(n) = n$$

Merge Sort Function:

$$T_{\text{mg}}(n) = n\log n$$

Middle Function:

$$T_{\text{middle}}(s1+s2) = (s1+s2)\log(s1+s2)$$

$$O((s1+s2)\log(s1+s2)) \approx O(n\log n)$$

$$T_{\text{overall}} = O((s1 + s2)\log(s1 + s2))$$

$$O(n\log n)$$

where n=$s1$+$s2$

This shows that your entire algorithm has a time complexity of ***O(nlogn)***

CODE the MaximumProduct Function

```
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
```

```
            elif num < min2:
                min2 = num
            prod1 = max1 * max2 * max3
            prod2 = max1 * min1 * min2
            if prod1 > prod2:
                return prod1
            else:
                return prod2
    while True:
        try:
            s = int(input("Enter size of array: "))
            if s > 0:
                break
            print("Please enter a positive number")
        except ValueError:
            print("Please enter a valid number")
    nums = [0]*s
    for i in range(s):
        while True:
            try:
                nums[i] += float(input(f"Enter element {i + 1}: "))
                break
            except ValueError:
                print("Please enter a valid number")
    if s <= 3:  # best case
        maxp = 1
        for num in nums:
```

```
            maxp *= num
        print(f"The maximum product is: {maxp:.1f}")
    else:
        print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

CODE Median of Two Sorted Arrays

```
def recursive(arr, index=0):
    try:
        VC = arr[index]
        return 1 + recursive(arr, index + 1)
    except IndexError:
        return 0
def mg(nums):
    n = recursive(nums)
    if n > 1:
        mid = n // 2
        l = nums[:mid]
        r = nums[mid:]
        mg(l)
        mg(r)
        i = j = k = 0
        lf = recursive(l)
        lr = recursive(r)
        while i < lf and j < lr:
            if l[i] < r[j]:
```

```
            nums[k] = l[i]
            i += 1
        else:
            nums[k] = r[j]
            j += 1
        k += 1
    while i < lf:
        nums[k] = l[i]
        i += 1
        k += 1
    while j < lr:
        nums[k] = r[j]
        j += 1
        k += 1
    return nums
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n%2==0:
        return (arr[mid-1]+arr[mid])/2
    else:
        return arr[mid]
while True:
    try:
        s1 = int(input("Enter size of array1: "))
        if s1 > 0:
```

```
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
while True:
    try:
        s2 = int(input("Enter size of array2: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
nums1 = [0] * s1
for i in range(s1):
    while True:
        try:
            nums1[i] = float(input(f"Enter element {i + 1} for array1: "))
            break
        except ValueError:
            print("Please enter a valid number")
nums2 = [0] * s2
for i in range(s2):
    while True:
        try:
            nums2[i] = float(input(f"Enter element {i + 1} for array2: "))
            break
        except ValueError:
```

```
        except ValueError:
            print("Please enter a valid number")
print(f"The middle is: {middle(nums1, nums2, s1, s2)}")
```

$$C(n, k) = \frac{n!}{k! \times (n - k)!}$$

in my code

the MaximumProduct Function

k=3

n= size of array...(number of elements in array)

Certainly! Combinations can be calculated using factorials with the following formula:

- (n!) is the factorial of the total number of elements.
- (k!) is the factorial of the number of elements chosen.
- ((n-k)!) is the factorial of the number of elements remaining after the selection.

To analyze the space usage in the code Median of Two Sorted Arrays

1. Function `recursive(arr, index=0)`

- **Space Complexity**: O(n)
  - This function uses recursion to count the number of elements in the array `arr`
  - Each recursive call adds a new frame to the call stack. Thus, if the array has `n` elements, there will be `n` frames on the stack, leading to a space complexity of O(n)

2. Function `mg(nums)`

- **Space Complexity**: O(n)
  - This function implements the merge sort algorithm.
  - The merge sort algorithm inherently uses O(n) extra space due to the need to create temporary arrays `l` and `r` for splitting the array `nums`
  - Additionally, the recursion depth will be O(log n), but the primary space usage here is due to the temporary arrays created during the merge process

3. Function `middle(nums1, nums2, s1, s2)`

- **Space Complexity**: O(n + m)
  - This function concatenates two arrays `nums1` and `nums2` to form a single array, which is then sorted using the `mg` function
  - The space complexity is dominated by the merge sort operation, which is O(n + m), where `n` and `m` are the sizes of `nums1` and `nums2`, respectively

Overall Space Usage

- **input Handling**: The space used for storing `nums1` and `nums2` is directly proportional to their sizes, so O(n + m)
- **Merge Sort**: The merge sort operation is the most space-intensive part, requiring additional space for temporary arrays during the sorting process, resulting in an overall space complexity of O(n + m)
- **Function Calls**: The recursive calls in both `recursive` and `mg` functions contribute to the space complexity, but they are dominated by the auxiliary array space

To analyze the space usage in the code the MaximumProduct Function

Function `MaximumProduct(nums, s)`

- **Initialization**:
  - `max1`, `max2`, `max3` are initialized to negative infinity to track the three largest numbers
  - `min1`, `min2` are initialized to positive infinity to track the two smallest numbers (useful for negative numbers)

- **Iteration through the list**:
  - The function iterates through each number in `nums` to determine the top three maximum values (`max1`, `max2`, `max3`) and the two minimum values (`min1`, `min2`)

- **Product Calculation**:
  - If `s` (size of the array) is 4, it calculates two potential maximum products:
    - `prod1`: The product of the three largest numbers
    - `prod2`: The product of the largest number and the two smallest numbers (useful if the smallest numbers are negative)
  - If `s` is not 4, it calculates:
    - `prod1`: The product of the three largest numbers

    - `prod2` : The product of the two largest numbers and the smallest number
  - The function returns the greater of the two products

- ***Space Complexity***: O(1)
  - The function uses a constant amount of additional space, as it only maintains a few variables (`max1` , `max2` , `max3` , `min1` , `min2`) regardless of the input size. This results in a space complexity of O(1).

Input Handling and Main Logic

- The code prompts for the size of the array and then reads `s` numbers into the list `nums` .
- If the size `s` is 3 or less, it simply computes the product of all numbers as they are the only numbers available.
- If `s` is greater than 3, it uses the `MaximumProduct` function to determine the highest possible product based on the logic described.

Overall Analysis

- **Time Complexity**: O(n), where `n` is the size of the array, because the function iterates through the list once to determine the maximum and minimum values
- **Space Complexity**: O(1)

Algorithm	Best Case	Average Case	Worst Case	Space	Stable
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Quick Sort	O(n log n)	O(log n)	$O(n^2)$	O(log n)	No
Counting Sort	O(n + k)	O(n + k)	O(n + k)	O(k)	Yes

Merge Sort (used in my code **Median of Two Sorted Arrays**):

- Advantages: Consistent performance, & stable & ideal for lists
- Disadvantages: Requires extra O(n) space

Bubble Sort:

- Advantages: Easy to implement & stable
- Disadvantages: Very slow for large arrays

Quick Sort:

- Advantages: Fast in practice & less space usage
- Disadvantages: Not stable & worst case $O(n^2)$

Counting Sort:

- Advantages: Linear time for bounded integers (liner sort algorithm)
- Disadvantages: Requires large extra space if range is large

Could use Quick Sort for better space performance

Could use Counting Sort if values are bounded and known

Bubble Sort would not be suitable due to $O(n^2)$ time complexity

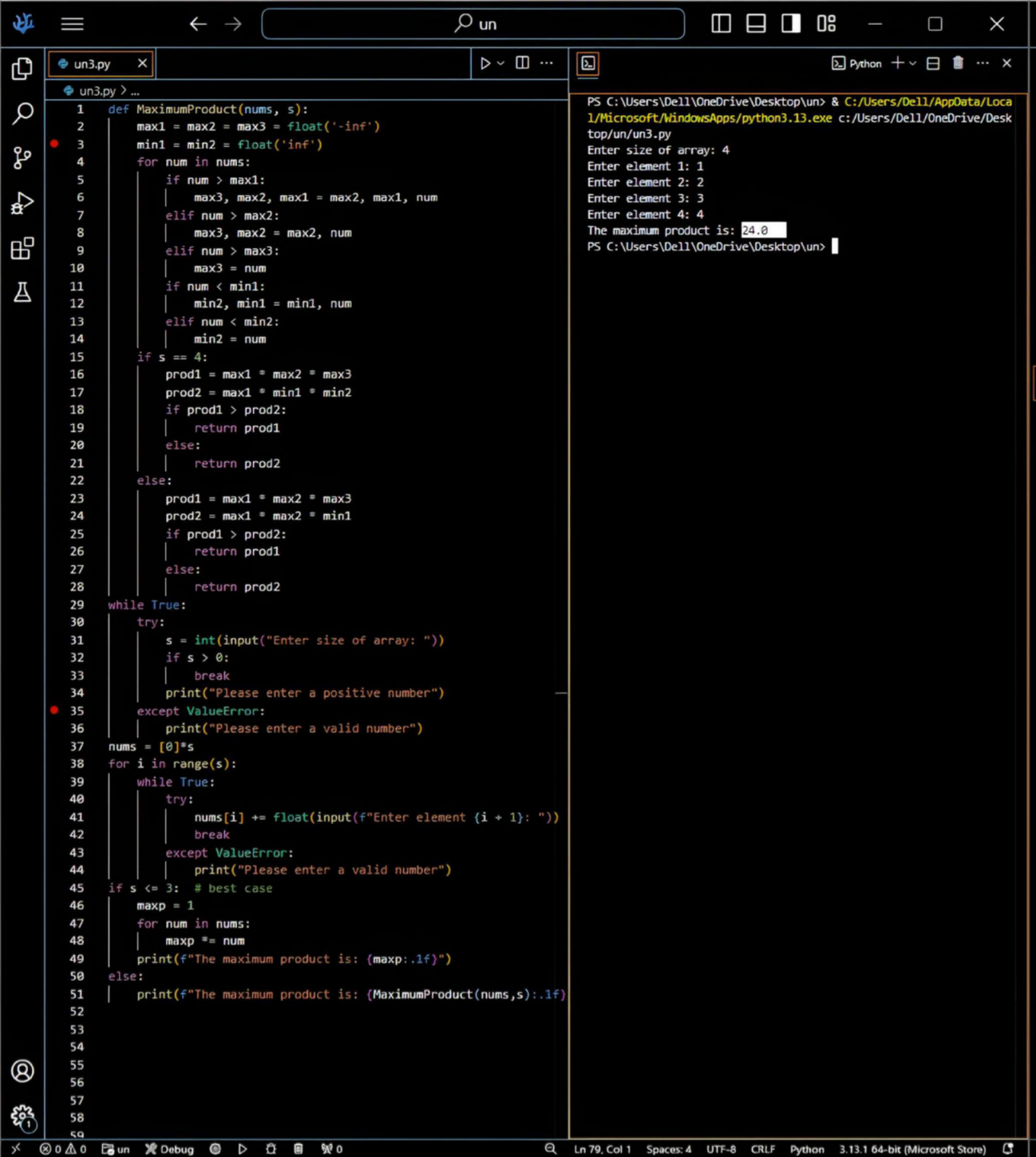
using Merge Sor t is actually a good choice because:

It guarantees O(n log n) performance in all cases

It's stable

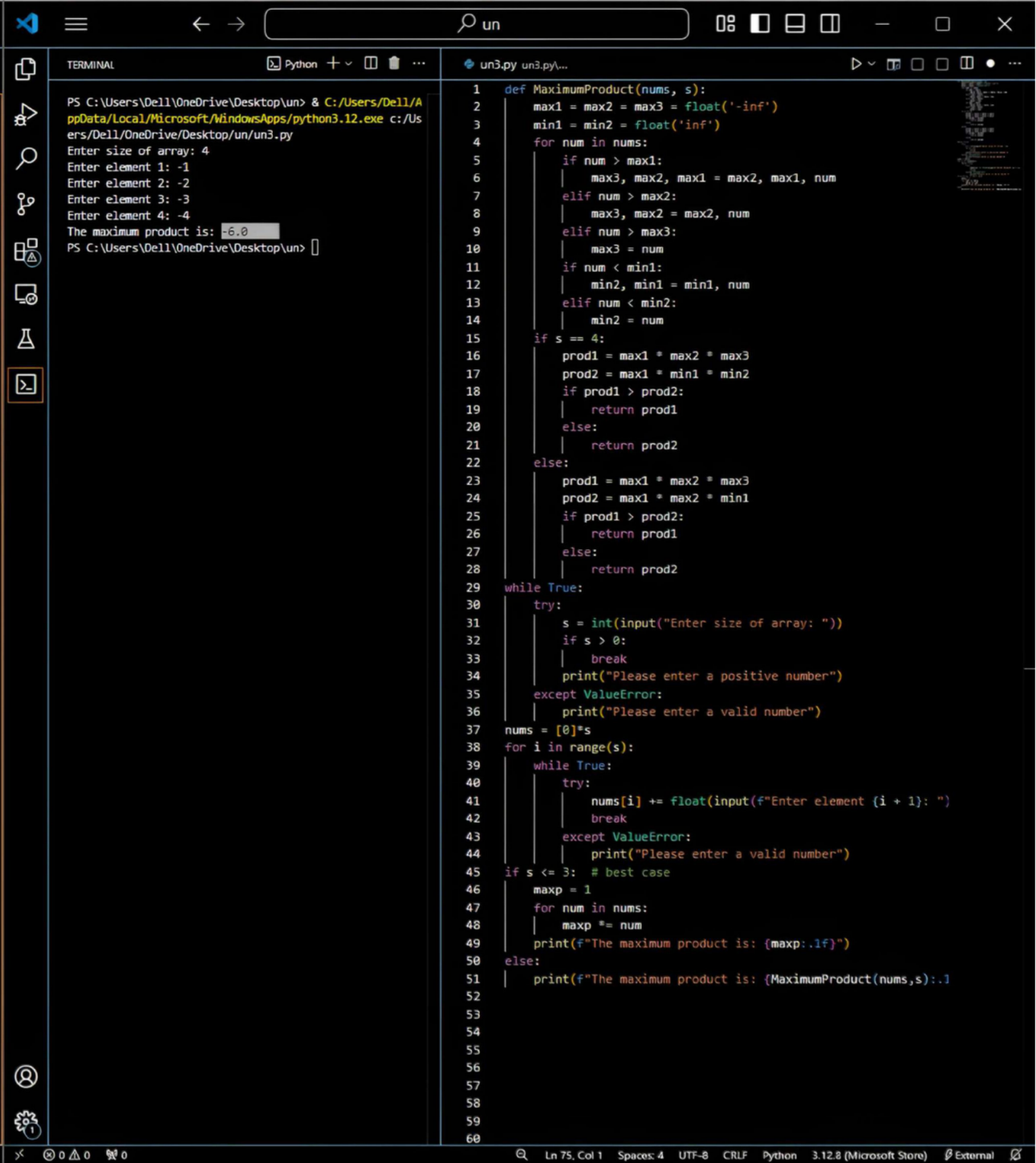
Works well with the median finding

Reliable for all input sizes



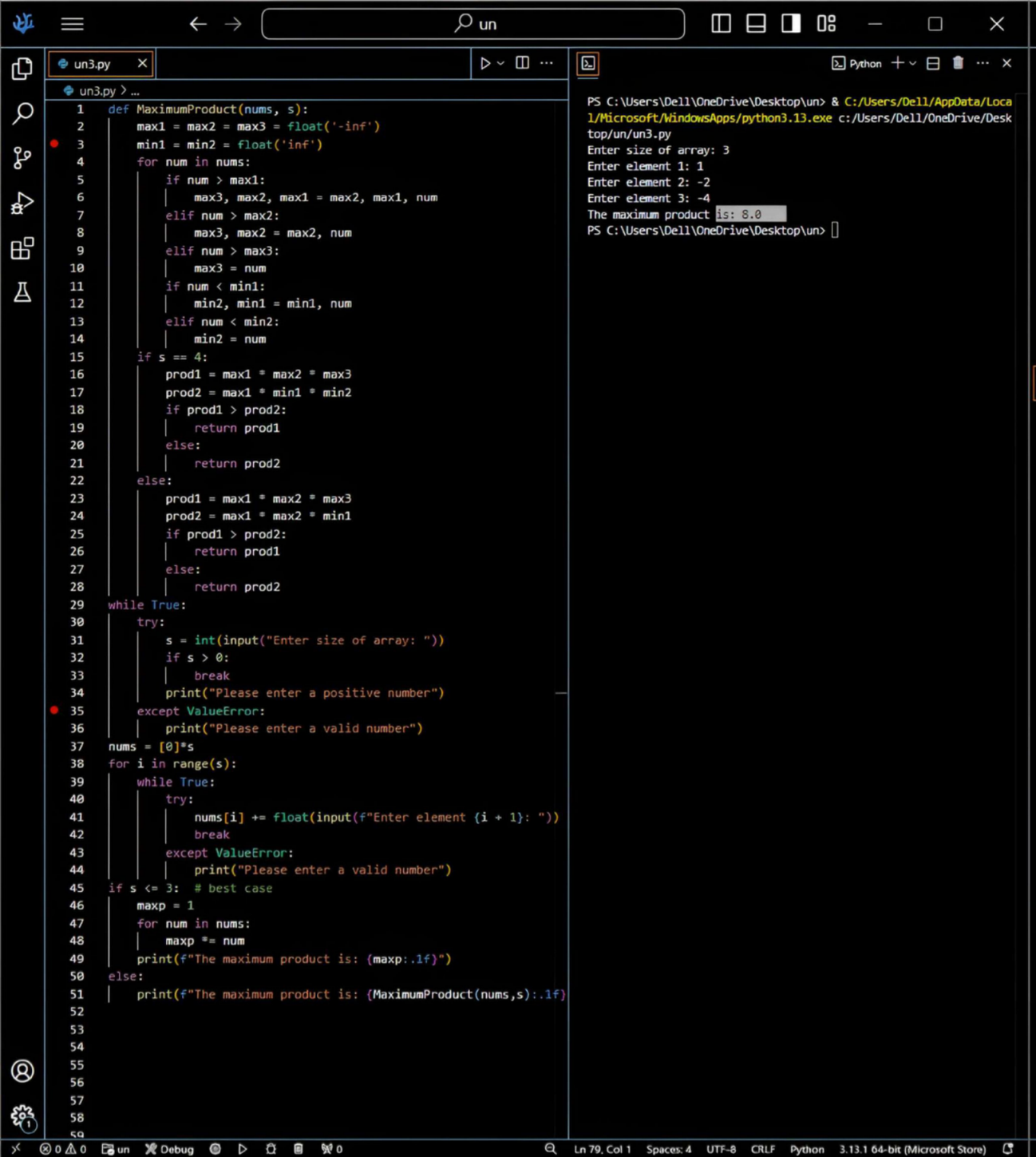
```python
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
        elif num < min2:
            min2 = num
    if s == 4:
        prod1 = max1 * max2 * max3
        prod2 = max1 * min1 * min2
        if prod1 > prod2:
            return prod1
        else:
            return prod2
    else:
        prod1 = max1 * max2 * max3
        prod2 = max1 * max2 * min1
        if prod1 > prod2:
            return prod1
        else:
            return prod2
while True:
    try:
        s = int(input("Enter size of array: "))
        if s > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
            break
        except ValueError:
            print("Please enter a valid number")
if s <= 3:  # best case
    maxp = 1
    for num in nums:
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 4
Enter element 1: 1
Enter element 2: 2
Enter element 3: 3
Enter element 4: 4
The maximum product is: 24.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```



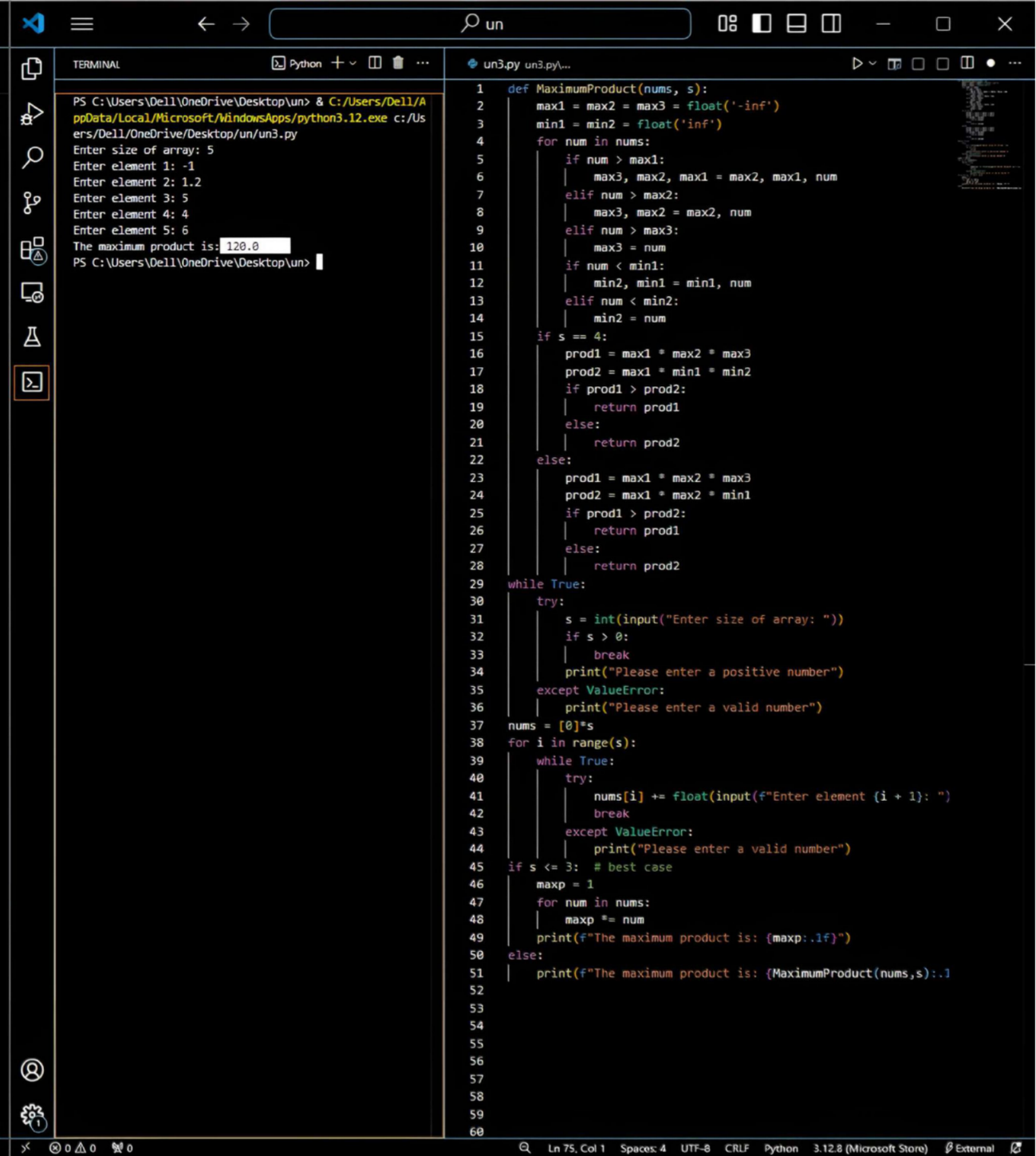
```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 4
Enter element 1: -1
Enter element 2: -2
Enter element 3: -3
Enter element 4: -4
The maximum product is: -6.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```

```python
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
        elif num < min2:
            min2 = num
    if s == 4:
        prod1 = max1 * max2 * max3
        prod2 = max1 * min1 * min2
        if prod1 > prod2:
            return prod1
        else:
            return prod2
    else:
        prod1 = max1 * max2 * max3
        prod2 = max1 * max2 * min1
        if prod1 > prod2:
            return prod1
        else:
            return prod2
while True:
    try:
        s = int(input("Enter size of array: "))
        if s > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
            break
        except ValueError:
            print("Please enter a valid number")
if s <= 3:  # best case
    maxp = 1
    for num in nums:
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1
```



```python
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
        elif num < min2:
            min2 = num
    if s == 4:
        prod1 = max1 * max2 * max3
        prod2 = max1 * min1 * min2
        if prod1 > prod2:
            return prod1
        else:
            return prod2
    else:
        prod1 = max1 * max2 * max3
        prod2 = max1 * max2 * min1
        if prod1 > prod2:
            return prod1
        else:
            return prod2
while True:
    try:
        s = int(input("Enter size of array: "))
        if s > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
            break
        except ValueError:
            print("Please enter a valid number")
if s <= 3:  # best case
    maxp = 1
    for num in nums:
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 3
Enter element 1: 1
Enter element 2: -2
Enter element 3: -4
The maximum product is: 8.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```



```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 5
Enter element 1: -1
Enter element 2: 1.2
Enter element 3: 5
Enter element 4: 4
Enter element 5: 6
The maximum product is: 120.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```

```python
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
        elif num < min2:
            min2 = num
    if s == 4:
        prod1 = max1 * max2 * max3
        prod2 = max1 * min1 * min2
        if prod1 > prod2:
            return prod1
        else:
            return prod2
    else:
        prod1 = max1 * max2 * max3
        prod2 = max1 * max2 * min1
        if prod1 > prod2:
            return prod1
        else:
            return prod2
while True:
    try:
        s = int(input("Enter size of array: "))
        if s > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
            break
        except ValueError:
            print("Please enter a valid number")
if s <= 3:  # best case
    maxp = 1
    for num in nums:
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1f}")
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 6
Enter element 1: -1
Enter element 2: -2
Enter element 3: -3
Enter element 4: -4
Enter element 5: -5
Enter element 6: -6
The maximum product is: -6.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un3.py
Enter size of array: 6
Enter element 1: 1.2
Enter element 2: 1.2
Enter element 3: 3.2
Enter element 4: 4.5
Enter element 5: 6.8
Enter element 6: 6.9
The maximum product is: 211.1
PS C:\Users\Dell\OneDrive\Desktop\un>
```

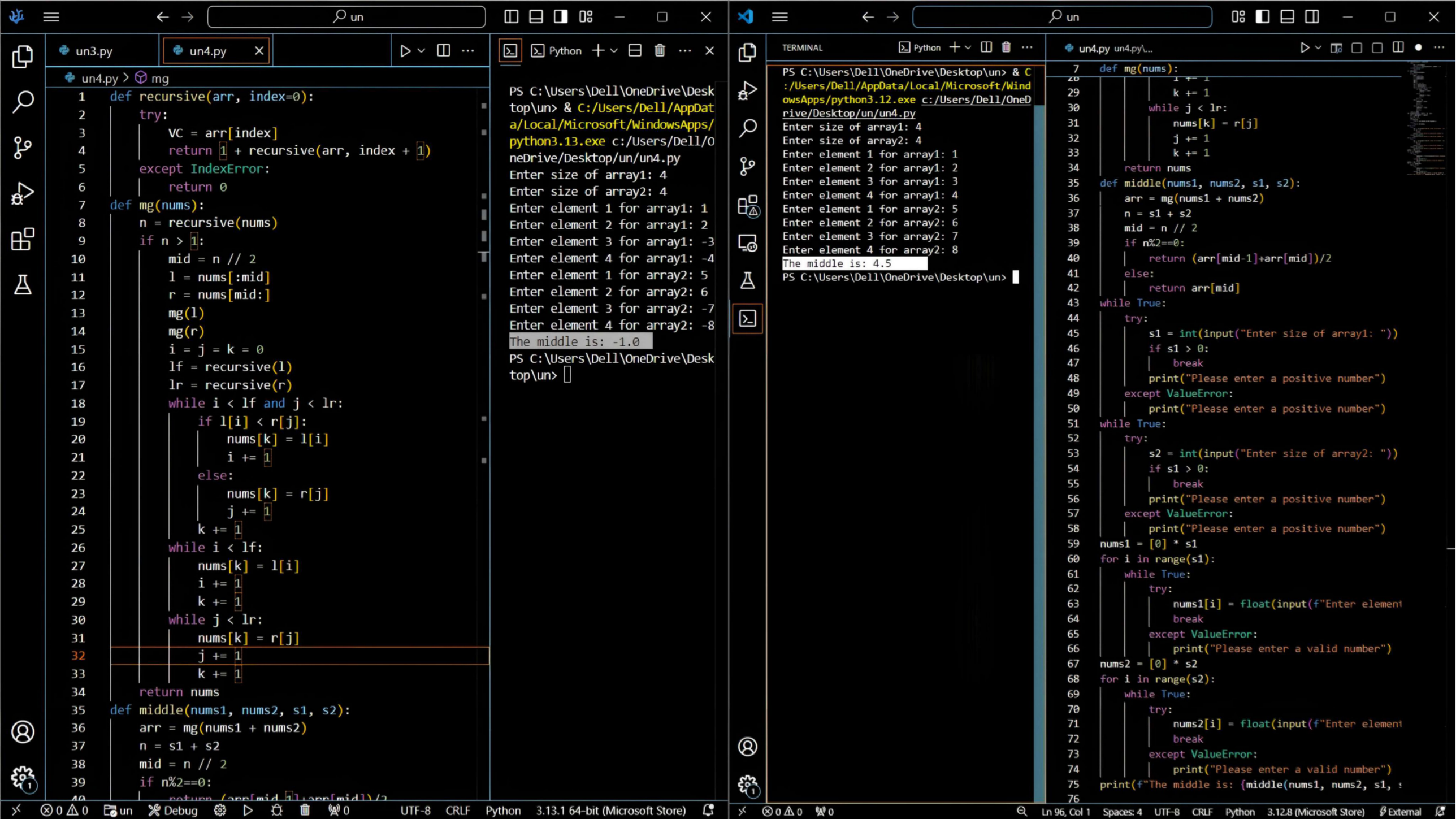

```python
def MaximumProduct(nums, s):
    max1 = max2 = max3 = float('-inf')
    min1 = min2 = float('inf')
    for num in nums:
        if num > max1:
            max3, max2, max1 = max2, max1, num
        elif num > max2:
            max3, max2 = max2, num
        elif num > max3:
            max3 = num
        if num < min1:
            min2, min1 = min1, num
        elif num < min2:
            min2 = num
    if s == 4:
        prod1 = max1 * max2 * max3
        prod2 = max1 * min1 * min2
        if prod1 > prod2:
            return prod1
        else:
            return prod2
    else:
        prod1 = max1 * max2 * max3
        prod2 = max1 * max2 * min1
        if prod1 > prod2:
            return prod1
        else:
            return prod2
while True:
    try:
        s = int(input("Enter size of array: "))
        if s > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a valid number")
nums = [0]*s
for i in range(s):
    while True:
        try:
            nums[i] += float(input(f"Enter element {i + 1}: "))
            break
        except ValueError:
            print("Please enter a valid number")
if s <= 3:  # best case
    maxp = 1
    for num in nums:
        maxp *= num
    print(f"The maximum product is: {maxp:.1f}")
else:
    print(f"The maximum product is: {MaximumProduct(nums,s):.1
```

```python
def recursive(arr, index=0):
    try:
        VC = arr[index]
        return 1 + recursive(arr, index + 1)
    except IndexError:
        return 0
def mg(nums):
    n = recursive(nums)
    if n > 1:
        mid = n // 2
        l = nums[:mid]
        r = nums[mid:]
        mg(l)
        mg(r)
        i = j = k = 0
        lf = recursive(l)
        lr = recursive(r)
        while i < lf and j < lr:
            if l[i] < r[j]:
                nums[k] = l[i]
                i += 1
            else:
                nums[k] = r[j]
                j += 1
            k += 1
        while i < lf:
            nums[k] = l[i]
            i += 1
            k += 1
        while j < lr:
            nums[k] = r[j]
            j += 1
            k += 1
    return nums
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n%2==0:
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 2
Enter size of array2: 2
Enter element 1 for array1: 1
Enter element 2 for array1: 2
Enter element 1 for array2: 3
Enter element 2 for array2: 4
The middle is: 2.5
PS C:\Users\Dell\OneDrive\Desktop\un>
```

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 2
Enter size of array2: 1
Enter element 1 for array1: 1
Enter element 2 for array1: 3
Enter element 1 for array2: 2
The middle is: 2.0
PS C:\Users\Dell\OneDrive\Desktop\un>
```

```python
    def mg(nums):
            k += 1
        while j < lr:
            nums[k] = r[j]
            j += 1
            k += 1
    return nums
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n%2==0:
        return (arr[mid-1]+arr[mid])/2
    else:
        return arr[mid]
while True:
    try:
        s1 = int(input("Enter size of array1: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
while True:
    try:
        s2 = int(input("Enter size of array2: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
nums1 = [0] * s1
for i in range(s1):
    while True:
        try:
            nums1[i] = float(input(f"Enter element
            break
        except ValueError:
            print("Please enter a valid number")
nums2 = [0] * s2
for i in range(s2):
    while True:
        try:
            nums2[i] = float(input(f"Enter element
            break
        except ValueError:
            print("Please enter a valid number")
print(f"The middle is: {middle(nums1, nums2, s1,
```


un3.py
un4.py
un4.py > mg
def recursive(arr, index=0):
try:
VC = arr[index]
return 1 + recursive(arr, index + 1)
except IndexError:
return 0
def mg(nums):
n = recursive(nums)
if n > 1:
mid = n // 2
l = nums[:mid]
r = nums[mid:]
mg(l)
mg(r)
i = j = k = 0
lf = recursive(l)
lr = recursive(r)
while i < lf and j < lr:
if l[i] < r[j]:
nums[k] = l[i]
i += 1
else:
nums[k] = r[j]
j += 1
k += 1
while i < lf:
nums[k] = l[i]
i += 1
k += 1
while j < lr:
nums[k] = r[j]
j += 1
k += 1
return nums
def middle(nums1, nums2, s1, s2):
arr = mg(nums1 + nums2)
n = s1 + s2
mid = n // 2
if n%2==0:
Python
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 4
Enter size of array2: 4
Enter element 1 for array1: 1
Enter element 2 for array1: 2
Enter element 3 for array1: -3
Enter element 4 for array1: -4
Enter element 1 for array2: 5
Enter element 2 for array2: 6
Enter element 3 for array2: -7
Enter element 4 for array2: -8
The middle is: -1.0
PS C:\Users\Dell\OneDrive\Desktop\un>
TERMINAL
Python
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 4
Enter size of array2: 4
Enter element 1 for array1: 1
Enter element 2 for array1: 2
Enter element 3 for array1: 3
Enter element 4 for array1: 4
Enter element 1 for array2: 5
Enter element 2 for array2: 6
Enter element 3 for array2: 7
Enter element 4 for array2: 8
The middle is: 4.5
PS C:\Users\Dell\OneDrive\Desktop\un>
un4.py un4.py\...
def mg(nums):
k += 1
while j < lr:
nums[k] = r[j]
j += 1
k += 1
return nums
def middle(nums1, nums2, s1, s2):
arr = mg(nums1 + nums2)
n = s1 + s2
mid = n // 2
if n%2==0:
return (arr[mid-1]+arr[mid])/2
else:
return arr[mid]
while True:
try:
s1 = int(input("Enter size of array1: "))
if s1 > 0:
break
print("Please enter a positive number")
except ValueError:
print("Please enter a positive number")
while True:
try:
s2 = int(input("Enter size of array2: "))
if s1 > 0:
break
print("Please enter a positive number")
except ValueError:
print("Please enter a positive number")
nums1 = [0] * s1
for i in range(s1):
while True:
try:
nums1[i] = float(input(f"Enter element
break
except ValueError:
print("Please enter a valid number")
nums2 = [0] * s2
for i in range(s2):
while True:
try:
nums2[i] = float(input(f"Enter element
break
except ValueError:
print("Please enter a valid number")
print(f"The middle is: {middle(nums1, nums2, s1,
UTF-8
CRLF
Python
3.13.1 64-bit (Microsoft Store)
Ln 96, Col 1
Spaces: 4
3.12.8 (Microsoft Store)
External

un3.py | un4.py

un4.py > ...

```python
def recursive(arr, index=0):
    try:
        VC = arr[index]
        return 1 + recursive(arr, index + 1)
    except IndexError:
        return 0
def mg(nums):
    n = recursive(nums)
    if n > 1:
        mid = n // 2
        l = nums[:mid]
        r = nums[mid:]
        mg(l)
        mg(r)
        i = j = k = 0
        lf = recursive(l)
        lr = recursive(r)
        while i < lf and j < lr:
            if l[i] < r[j]:
                nums[k] = l[i]
                i += 1
            else:
                nums[k] = r[j]
                j += 1
            k += 1
        while i < lf:
            nums[k] = l[i]
            i += 1
            k += 1
        while j < lr:
            nums[k] = r[j]
            j += 1
            k += 1
    return nums
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n%2==0:
        return (arr[mid-1]+arr[mid])/2
```

Python

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 5
Enter size of array2: 5
Enter element 1 for array1: -1
Enter element 2 for array1: -2
Enter element 3 for array1: -3
Enter element 4 for array1: -4
Enter element 5 for array1: -5
Enter element 1 for array2: -6
Enter element 2 for array2: 7
Enter element 3 for array2: -8
Enter element 4 for array2: 8.2

Enter element 5 for array2: 2
The middle is: -2.5
PS C:\Users\Dell\OneDrive\Desktop\un>
```

TERMINAL | Python

```
PS C:\Users\Dell\OneDrive\Desktop\un> & C:/Users/Dell/AppData/Local/Microsoft/WindowsApps/python3.12.exe c:/Users/Dell/OneDrive/Desktop/un/un4.py
Enter size of array1: 3
Enter size of array2: 3
Enter element 1 for array1: 1.2
Enter element 2 for array1: 1.3
Enter element 3 for array1: 4.5
Enter element 1 for array2: 7.8
Enter element 2 for array2: 9
Enter element 3 for array2: -7
The middle is: 2.9
PS C:\Users\Dell\OneDrive\Desktop\un>
```

un4.py un4.py\...

```python
def mg(nums):
            k += 1
        while j < lr:
            nums[k] = r[j]
            j += 1
            k += 1
    return nums
def middle(nums1, nums2, s1, s2):
    arr = mg(nums1 + nums2)
    n = s1 + s2
    mid = n // 2
    if n%2==0:
        return (arr[mid-1]+arr[mid])/2
    else:
        return arr[mid]
while True:
    try:
        s1 = int(input("Enter size of array1: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
while True:
    try:
        s2 = int(input("Enter size of array2: "))
        if s1 > 0:
            break
        print("Please enter a positive number")
    except ValueError:
        print("Please enter a positive number")
nums1 = [0] * s1
for i in range(s1):
    while True:
        try:
            nums1[i] = float(input(f"Enter element
            break
        except ValueError:
            print("Please enter a valid number")
nums2 = [0] * s2
for i in range(s2):
    while True:
        try:
            nums2[i] = float(input(f"Enter element
            break
        except ValueError:
            print("Please enter a valid number")
print(f"The middle is: {middle(nums1, nums2, s1,
```