Documentation Task 4 "Birthday candles"

Documentation for task 4:

____

We will cover main points in this documentation
1-code 7_kh
2-purpose
3-expected outputs
4-define the user_ defined function and its   purpose and parameters and return_define every step
5- Pseudocode and its complexity time
6-The complexity for the whole code (Best//Worst)
7-some test cases to check the validity of code

1-code 7-kh

```
def find_largest (array) :
    largest = array[0]
    for i in range (0,n):
        if array[i]>largest:
            largest=array[i]
    return largest

def count_occurences(array,largest):
    counter=0
    for j in range (0,n):
        if array[j]==largest:
            counter+=1

    return counter
def is_symmetric (array, largest,n):

    if (n%2==0):
        return False
    else:
        for i in range (0,n):
            if array[i]==largest:
                index=i
                break

        if index!=(n//2):
            return False
        else:
            for j in range(0,n):
                if array[j]!=array[n-1-j]:
                    return False
            return True

print("Happy birthday to you")
n= int (input("enter your new age"))
array=[]
for i in range(0,n):
    element =int(input(f"enter element of array which is candles length{i+1}:"))
    array.append(element)
largest=find_largest(array)
print(f"The largest length of candles is: {largest}")
x = count_occurences(array, largest)
print(f"Largest is repeated for {x} times ")
if (is_symmetric(array,largest,n)):
    print ("symmetric")
else:
    print ("not symmetric")
```

2-Purpose:

This problem aims to obtain the age of someone in his birthday and put number of candles ...this number is the same to his age but with different length

For example:

Age=5

So, we will put 5 candles but they can be with different length...and then store these lengths of candles in an array to make some processes on this array which is given by user.

3-expected outputs:

The tester or the user will expect some outputs which is:

Firstly, Output1: is the integer which represent the tallest length of candles.

Secondly, Output2: the number of occurrence of this tallest length in an array. ==how many times this tallest length repeated in an array.

Thirdly, Output3: is Boolean ((True or false))

It returns true if it is symmetric

And false if it is nonsymmetrical

If array size of numbers of candles lengths which is represented by n is even So, It is exactly not symmetric.

But if the n is odd .Now, we can check firstly the position of largest number if it is exactly in the middle, we will continue our check then

Secondly, we will compare the left side of it with the right side of it if…As we will consider this number as a mirror.

But if it is not in the middle we will return and do not continue.

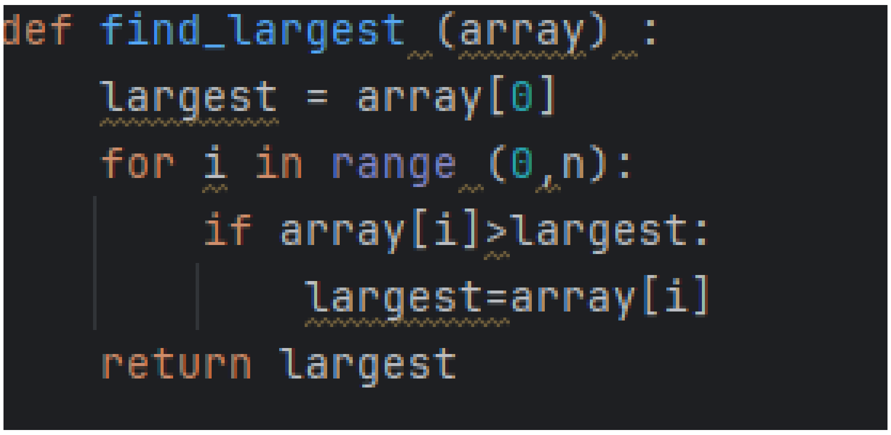
4- Define the main functions

Here, There is the main function as it’s the backbone of code...it cannot run without calling main function to start program execution.

It is simply.

But, there is some user defined functions

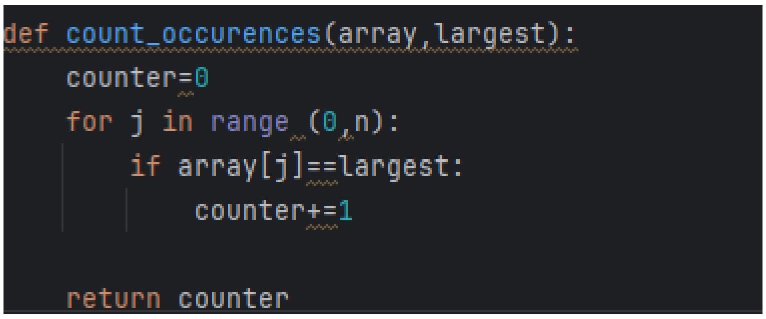
Firstly// (((find _largest)))



```
def find_largest (array) :
    largest = array[0]
    for i in range (0,n):
        if array[i]>largest:
            largest=array[i]
    return largest
```

It obtains the array of candles lengths as parameter...and returns an integer which represent the tallest length which is the same as maximum number of array.

Secondly: (count_occurences ())



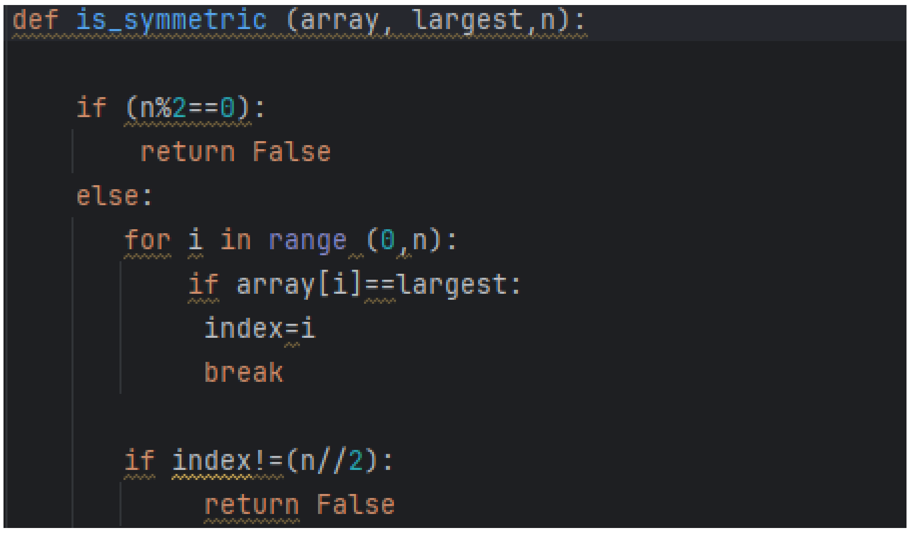
```
def count_occurences(array,largest):
    counter=0
    for j in range (0,n):
        if array[j]==largest:
            counter+=1

    return counter
```

This user defined function gets 2 parameters. They are the array and the largest number which is (((return of function [find_ largest]))).

And

Return an integer number which represents the number of occurrence of this largest value.

Thirdly: is the is_symmetric::



```
def is_symmetric (array, largest,n):

    if (n%2==0):
        return False
    else:
        for i in range (0,n):
            if array[i]==largest:
                index=i
                break

        if index!=(n//2):
            return False
```

It gets 2 parameters

The array and the largest number and the size (n)

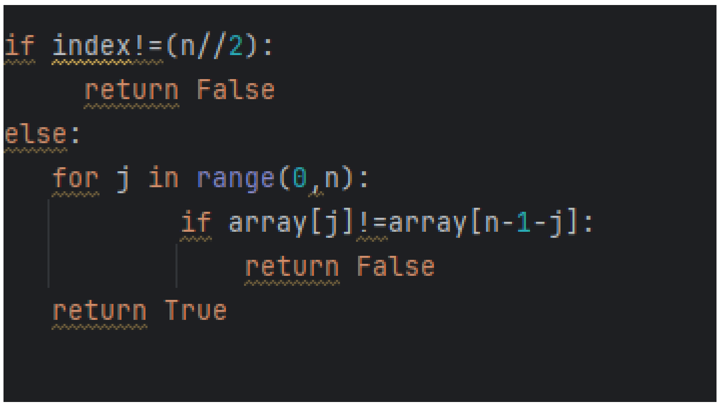
Then we check if it is even or not

If it is even it will print “not symmetric”

If it is odd … Now we have different scenario as we should determine “the largest number position”

If position as index = the median of array (n//2)

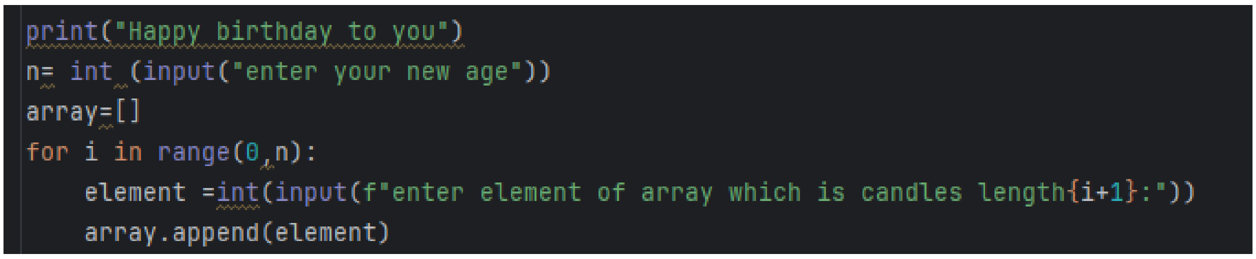
We use this double slash (//) to check that output will be integer. This this the first step to check that is symmetric …..But it is not enough to prove this as we should check that this largest as a mirror as the left side of it matches with the right side.



```
if index!=(n//2):
    return False
else:
    for j in range(0,n):
            if array[j]!=array[n-1-j]:
                return False
    return True
```

We defined all user-defined functions I have used and the rest of functions is

The (((main)))



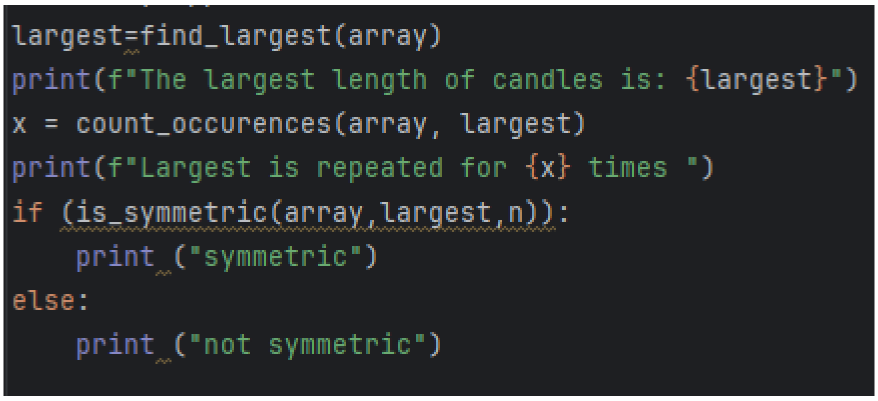
```python
print("Happy birthday to you")
n= int (input("enter your new age"))
array=[]
for i in range(0,n):
    element =int(input(f"enter element of array which is candles length{i+1}:"))
    array.append(element)
```

It is used only in this code to scan the age and store it in variable n and generate an array with size n

Then ask the user to fill this array with candles lengths then start my processes that

I send this array as a parameter to each of functions that I mentioned before to make process on then I just call these functions and print their outputs from main ().

This figure about calling functions from the main to make it return their outputs.



```python
largest=find_largest(array)
print(f"The largest length of candles is: {largest}")
x = count_occurences(array, largest)
print(f"Largest is repeated for {x} times ")
if (is_symmetric(array,largest,n)):
    print ("symmetric")
else:
    print ("not symmetric")
```

5-The complexity of the whole code is:

1. find_largest Function: **Time Complexity:** O(n), where n is the length of the array.

2. count_occurences Function: **Time Complexity:** O(n).

3. is_symmetric Function:

a-Check if n is even: O(1).

b-Find the index of the first occurrence of largest: Time Complexity: O(n).

c-Compare elements for symmetry:Time Complexity: O(n).

Overall Time Complexity: O(n).

4. Main Program: Time Complexity: O(n).

Overall Time Complexity

- **The three key functions (find_largest, count_occurences, and is_symmetric) each takes O(n) time.**
- **The reason the time complexity is O(n) in both the best and worst cases is that the algorithm performs linear scans through the array for finding the largest element, counting its occurrences, and checking symmetry. None of these steps can be skipped or reduced to constant time, so the complexity remains O(n) in both cases.**
- **The input loop also takes O(n).**
- **Thus, the overall time complexity of the program is: O(n)+O(n)+O(n)+O(n)=O(n)**

The program's time complexity is O(n)

5- The pseudocode of this code

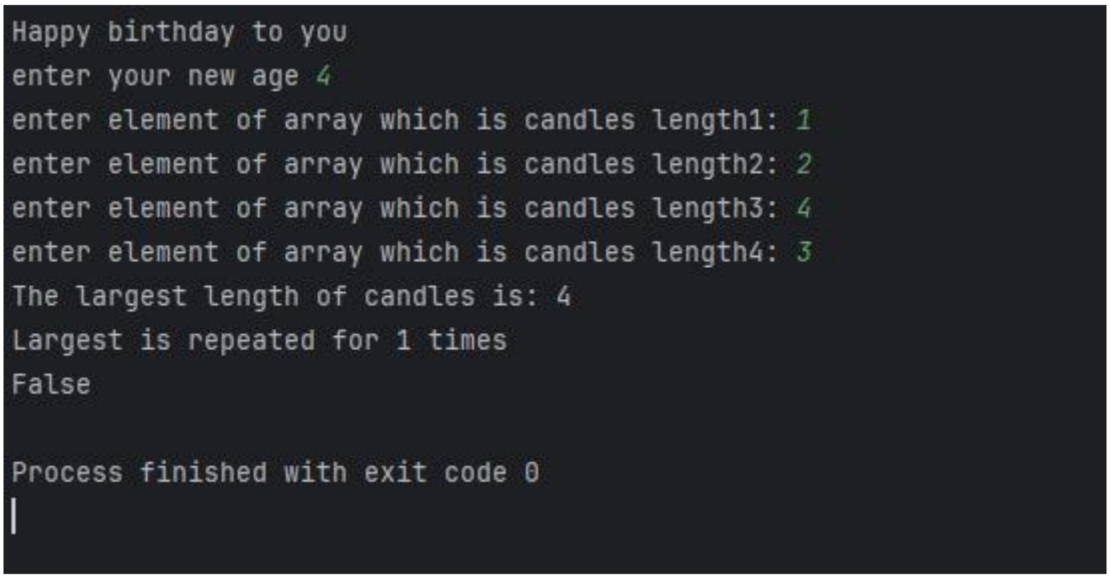
```
1. Print congratulation:
    Output: "Happy birthday to you"
2. Input Data:
    Read integer n (number of candles).
    Initialize empty list array.
    For i from 0 to n-1:
    Prompt: "t Enter he length of candle {i+1}: "
    Read integer value and add it to array.
3. Find the Largest Candle Height:
    Set largest = array[0].
```

```
   For each element height in array:
   If height > largest, update largest.
   Save largest.
4. Count the Occurrences of the Largest Height:
   Set counter = 0.
   For each element height in array:
   If height == largest, increment counter.
   Save counter.
   Check Symmetry:
   If n is even:
           Return "not symmetric".
   Otherwise:
  Find the first index of largest:
  For i from 0 to n-1:
           If array[i] == largest, save index and stop loop.
  If index != n // 2:
  Return "not symmetric".
o Otherwise:
o Compare elements from both ends of array:
  o For j from 0 to n-1:
       If array[j] != array[n - j - 1]:
              Return "not symmetric".
  Return "symmetric".
  Output Results:
```

- **Print the largest candle height: "The largest length of candles is: {largest}"**
- **Print the number of occurrences: "Largest is repeated for {counter} times"**

Print symmetry result: "symmetric" or "not symmetric".

6-Trace for code



```
Happy birthday to you
enter your new age 4
enter element of array which is candles length1: 1
enter element of array which is candles length2: 2
enter element of array which is candles length3: 4
enter element of array which is candles length4: 3
The largest length of candles is: 4
Largest is repeated for 1 times
False

Process finished with exit code 0
```

Input:

The program first displays a greeting: Happy birthday to you.

The program prompts the user to input 4 elements , Final array: [1, 2, 4, 3]

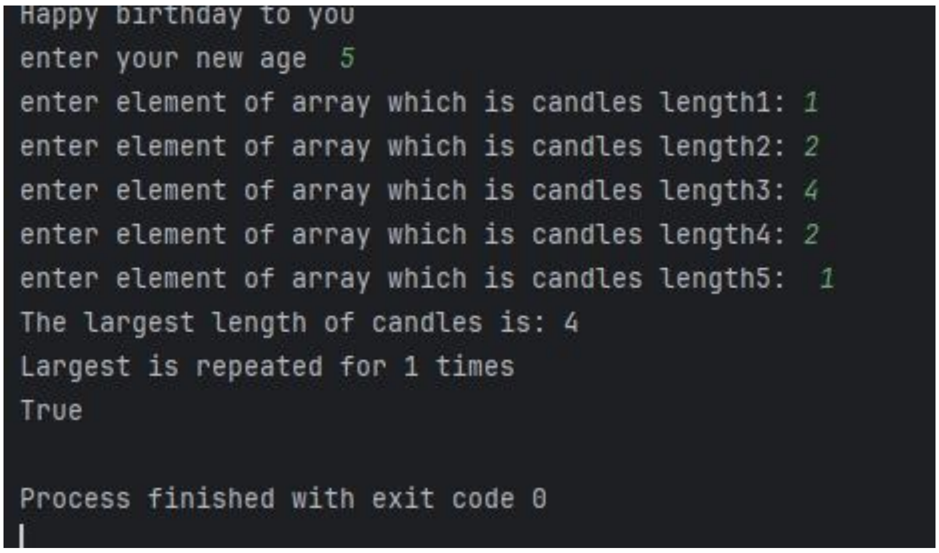
From [1, 2, 4, 3], the largest value is 4.

- The program counts how many times the largest value 4 appears in the array: 4 appears **once**.

Symmetry is checked as follows:

First, the program checks if `n` is even:

Since `n = 4` (even), symmetry is **immediately false** because symmetry is only possible for an odd-length array.



```
Happy birthday to you
enter your new age  5
enter element of array which is candles length1: 1
enter element of array which is candles length2: 2
enter element of array which is candles length3: 4
enter element of array which is candles length4: 2
enter element of array which is candles length5:  1
The largest length of candles is: 4
Largest is repeated for 1 times
True

Process finished with exit code 0
```

The program first displays a greeting: Happy birthday to you.

The program prompts the user to input 5 elements , Final array: [1, 2, 4, 2, 1]

From [1, 2, 4, 2, 1], the largest value is 4.

- The program counts how many times the largest value 4 appears in the array: 4 appears **once**.
- Symmetry is checked as follows:

  First, it verifies that `n` is **odd**:

`n = 5` → **odd** → Symmetry check continues.

Largest value 4 is at index 2, which is the middle index of the array (n // 2 = 2)

Then, the program checks the array is symmetric by

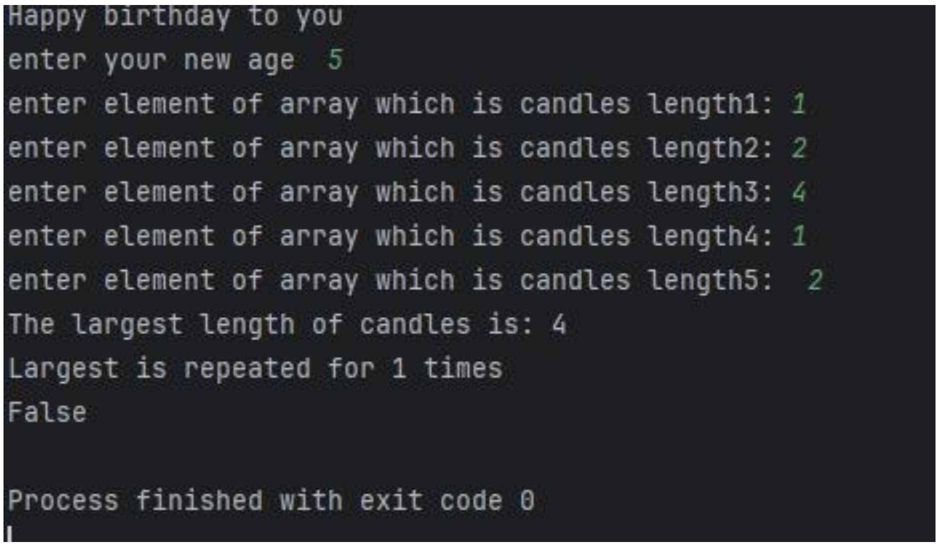
comparing elements from the left and right sides:

- Compare:

4 `array[0] == array[4] → 1 == 1` → True

5 `array[1] == array[3] → 2 == 2` → True

- All corresponding pairs are equal, so the array is **symmetric**.



```
Happy birthday to you
enter your new age  5
enter element of array which is candles length1: 1
enter element of array which is candles length2: 2
enter element of array which is candles length3: 4
enter element of array which is candles length4: 1
enter element of array which is candles length5:  2
The largest length of candles is: 4
Largest is repeated for 1 times
False

Process finished with exit code 0
```

The program first displays a greeting: Happy birthday to you.

The program prompts the user to input 5 elements , Final array: [1, 2, 4, 1, 2]

From [1, 2, 4, 1, 2], the largest value is 4.

- The program counts how many times the largest value 4 appears in the array: 4 appears **once**.
- Symmetry is checked as follows:

    First, it verifies that `n` is **odd**:

`n = 5` → **odd** → Symmetry check continues.

Largest value 4 is at index 2, which is the middle index of the array (n // 2 = 2)

Then, the program checks the array is symmetric by

comparing elements from the left and right sides:

Compare:

`array[0] == array[4] → 1 != 2` → **False**

Since the elements are not symmetric

Documentation Task 7 "Kth Element of Two Sorted Arrays"

Documentation for task 7:

We will cover main points in this documentation
1-code 7_kh
2-purpose
3-expected outputs
4-define the user_ defined function and its purpose and parameters and return_define every step
5- Pseudocode and its complexity time
6-The complexity for the whole code (Best//Worst)
7-some test cases to check the validity of code

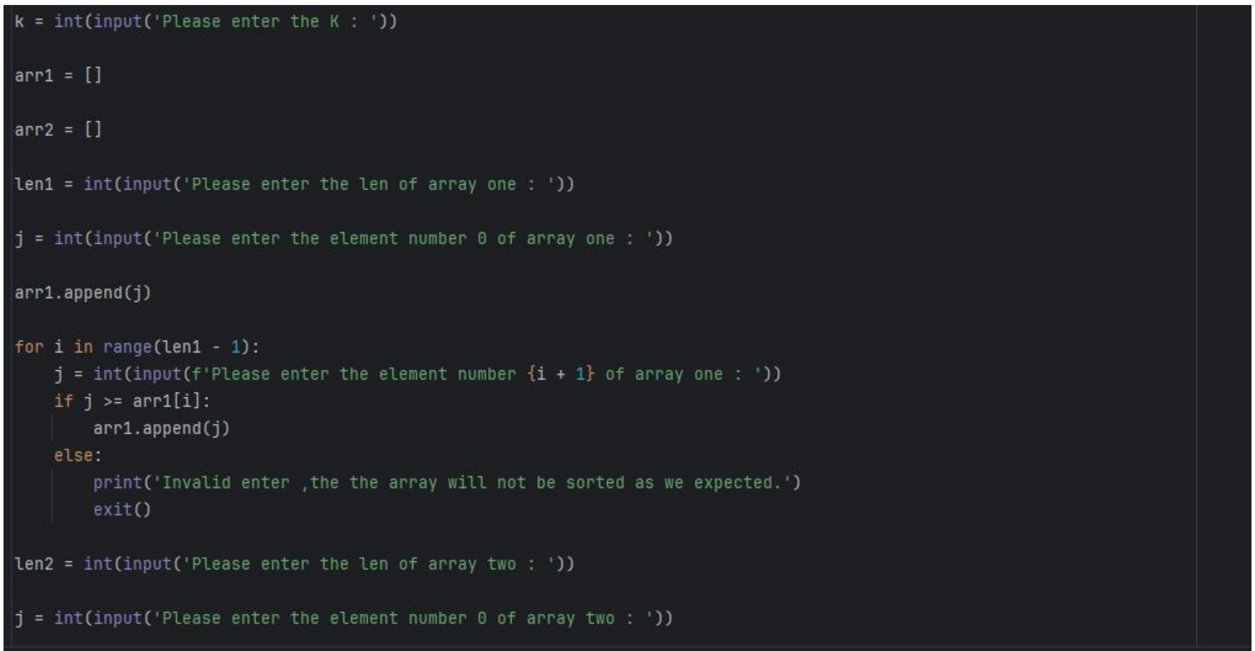
1-Code 7 k_th

```
def k_th_recursive(k_th, arr1, arr2):
    if not arr1:
        return arr2[k_th]
    if not arr2:
        return arr1[k_th]

    len1, len2 = len(arr1), len(arr2)

    if k_th >= len1 + len2 or k_th < 0:
        return -1
    mid1 = len1 // 2
    mid2 = len2 // 2

    if mid1 + mid2 < k_th:
        if arr1[mid1] < arr2[mid2]:
            return k_th_recursive(k_th - mid1 - 1, arr1[mid1+1:], arr2)
        else:
            return k_th_recursive(k_th - mid2 - 1, arr1, arr2[mid2+1:])
    else:
        if arr1[mid1] > arr2[mid2]:
            return k_th_recursive(k_th, arr1[:mid1], arr2)
        else:
            return k_th_recursive(k_th, arr1, arr2[:mid2])

k = int(input('Please enter the K : '))

arr1=[]

arr2=[]

len1 = int (input('Please enter the len of array one : '))
j= int (input('Please enter the element number 0 of array one : '))
arr1.append(j)

for i in range(len1-1):
    j= int (input(f'Please enter the element number {i+1} of array one : '))
    if j>=arr1[i]:
        arr1.append(j)
    else :
        print('Invalid enter ,the the array will not be sorted as we expected.')
        exit()

len2 = int (input('Please enter the len of array two : '))
```

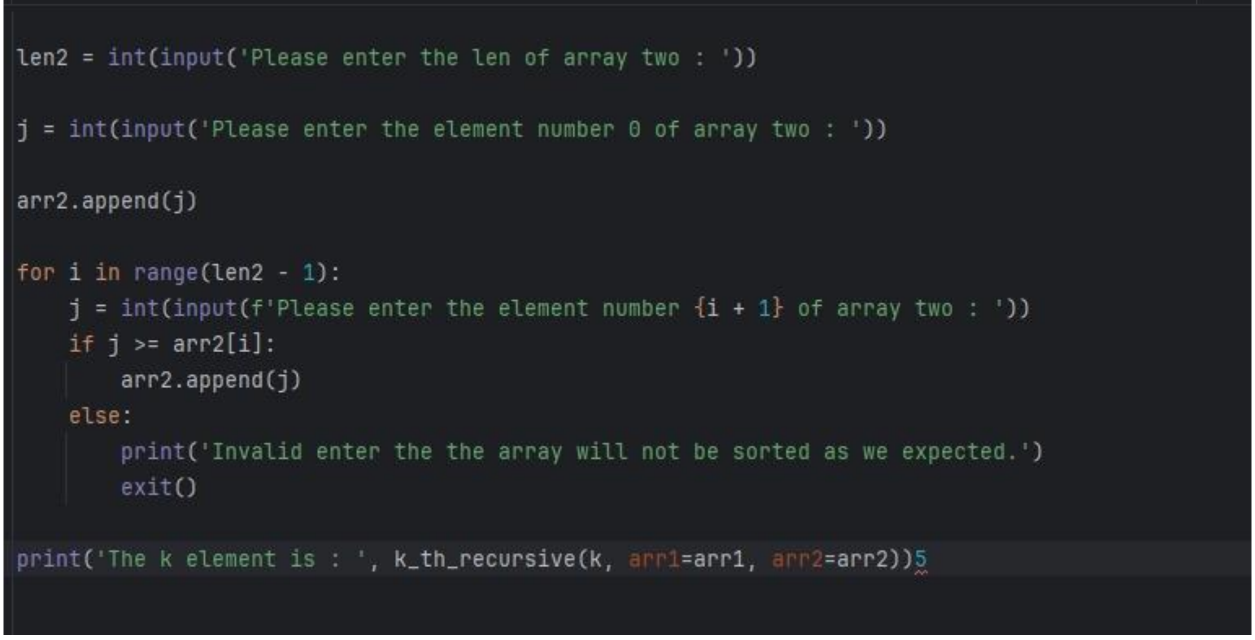
```
j= int (input('Please enter the element number 0 of array two : '))

arr2.append(j)

for i in range(len2-1):
    j= int (input(f'Please enter the element number {i+1} of array two : '))
    if j>=arr2[i]:
        arr2.append(j)
    else :
        print('Invalid enter the the array will not be sorted as we expected.')
        exit()

print('The k element is : ',k_th_recursive(k,arr1=arr1,arr2=arr2) )
```

2-Purpose: To gain 2 sorted arrays then merge them but sort them in another array for example: we have array 1 which is sorted as{1,2,3,4,5} then gain another array which is array2 it is also sorted then merge them with sorting them in another array which is called array 3 ....Array 3 its length = the length of arr1+the length of array 2 if len1 for array1=5,,and len2 for array2=6 then len for array3 is (5+6)=11

3-expected output:: Tester or user expected when he give an index to a program it returns the number which is already in this index...after merging the 2 sorted arrays

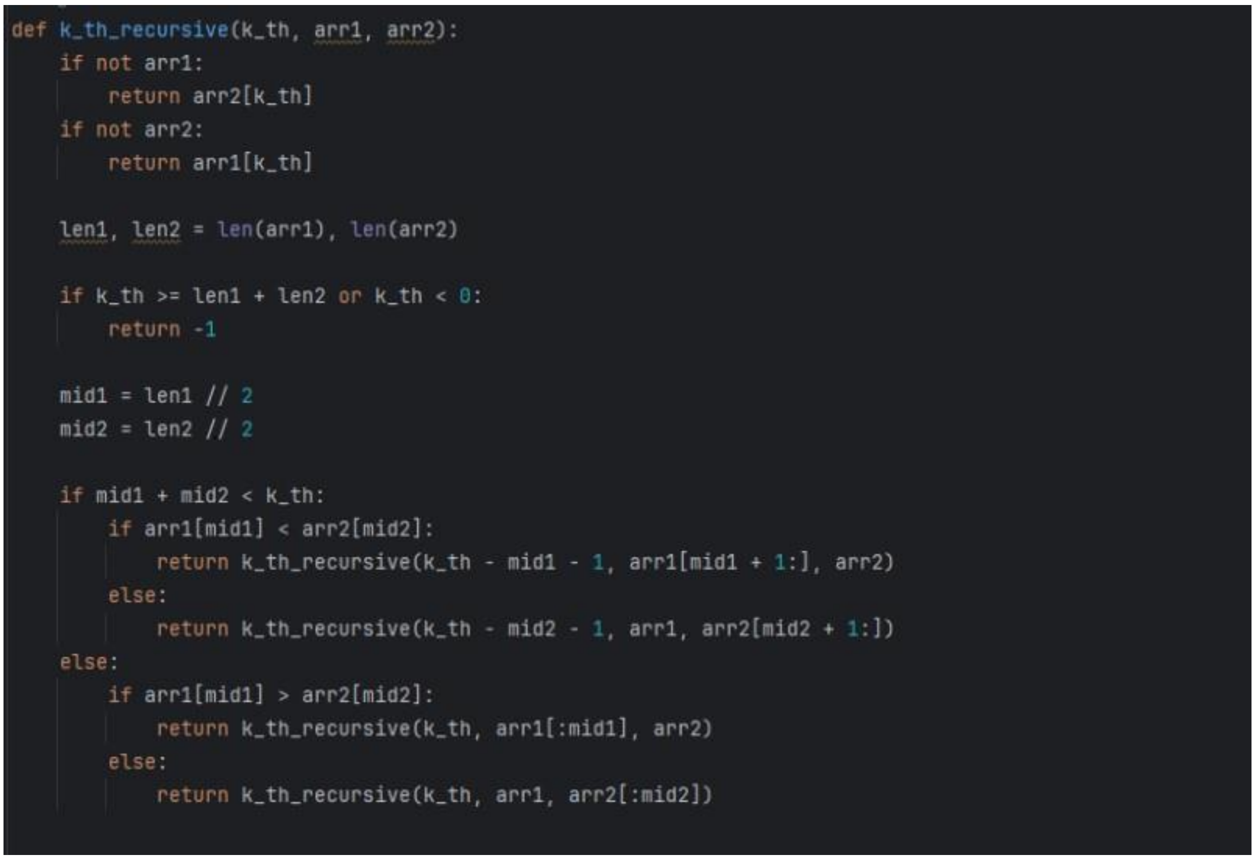


```
k = int(input('Please enter the K : '))

arr1 = []

arr2 = []

len1 = int(input('Please enter the len of array one : '))

j = int(input('Please enter the element number 0 of array one : '))

arr1.append(j)

for i in range(len1 - 1):
    j = int(input(f'Please enter the element number {i + 1} of array one : '))
    if j >= arr1[i]:
        arr1.append(j)
    else:
        print('Invalid enter ,the the array will not be sorted as we expected.')
        exit()

len2 = int(input('Please enter the len of array two : '))

j = int(input('Please enter the element number 0 of array two : '))
```



```
len2 = int(input('Please enter the len of array two : '))

j = int(input('Please enter the element number 0 of array two : '))

arr2.append(j)

for i in range(len2 - 1):
    j = int(input(f'Please enter the element number {i + 1} of array two : '))
    if j >= arr2[i]:
        arr2.append(j)
    else:
        print('Invalid enter the the array will not be sorted as we expected.')
        exit()

print('The k element is : ', k_th_recursive(k, arr1=arr1, arr2=arr2))5
```

4- Define the user _ defined functions and main function ....Firstly in main(): firstly: it accept the K as the index of an array .....then define arr1 is empty and anrr2 is empty to fill them then scan the length to determine the array size then j is the first element of the first array to scan array elements one by one then arr1.append (j) to insert them . Then make a loop to scan array1 elements , as the number is greater than the before to check that is sorted and else print invalid
The same operation will do with arr2 to check it is sorted To terminate or continue the scanning process
As the main terminates its work by scaning K from user and printing the output which is the return of recursive function



```python
def k_th_recursive(k_th, arr1, arr2):
    if not arr1:
        return arr2[k_th]
    if not arr2:
        return arr1[k_th]

    len1, len2 = len(arr1), len(arr2)

    if k_th >= len1 + len2 or k_th < 0:
        return -1

    mid1 = len1 // 2
    mid2 = len2 // 2

    if mid1 + mid2 < k_th:
        if arr1[mid1] < arr2[mid2]:
            return k_th_recursive(k_th - mid1 - 1, arr1[mid1 + 1:], arr2)
        else:
            return k_th_recursive(k_th - mid2 - 1, arr1, arr2[mid2 + 1:])
    else:
        if arr1[mid1] > arr2[mid2]:
            return k_th_recursive(k_th, arr1[:mid1], arr2)
        else:
            return k_th_recursive(k_th, arr1, arr2[:mid2])
```

To define the user_defined function we make a function called K_th recursive which its parameters are k and arr1 and arr2 then check if array1 is empty So exactly the K is in array 2 so search will be in arr2 .....and if array2 is empty we will check only array 1 as exactly K is in array1
then I defined a variable len1 and store in it length of array1 and a variable in array 2 called len 2
Then check the K if it is in array range less than the sum of aar1,arr2 Continue But if its out of bound or less than 0 return -1.......
After checking all of these calculate the median of arr1 and arr2 by dividing the length by 2 ....if K greater than the sum of two middles we handle 2 cases: firstly if the number in middle index of array 1 less than the number in middle of array 2 we will discard the first half of array 1 and start searching as K is (K-mid-1)To and arr1 from mid+1 to the end and arr2 as whole
case2:: if K >mid for array2 so k=mid2-1 and arr1 as whole and arr2 starts from mid+1 to end .....case3:if K is less than sum of len1+len2 and mid arr1>mid arr2 it is already the second half of arr1 .... and arr2 as whole ....case 4 : if K is less than sum and mid arr2>mid arr1 so will discard the first half of arr2 and the whole arr1

4- Pseudocode and its complexity time

```
Function k_th_recursive(k_th, arr1, arr2):
    If arr1 is empty:
        Return arr2[k_th]
    If arr2 is empty:
        Return arr1[k_th]

    len1 ← Length(arr1)
    len2 ← Length(arr2)

    If k_th >= len1 + len2 OR k_th < 0:
        Return -1

    mid1 ← len1 // 2
    mid2 ← len2 // 2

    If mid1 + mid2 < k_th:
        If arr1[mid1] < arr2[mid2]:
            Return k_th_recursive(k_th - mid1 - 1, Subarray(arr1, mid1 + 1,
     len1), arr2)
        Else:
            Return k_th_recursive(k_th - mid2 - 1, arr1, Subarray(arr2, mid2
     + 1, len2))
    Else:
        If arr1[mid1] > arr2[mid2]:
            Return k_th_recursive(k_th, Subarray(arr1, 0, mid1), arr2)
        Else:
            Return k_th_recursive(k_th, arr1, Subarray(arr2, 0, mid2))
Function main():
    Print "Enter value of k:"
    k ← Input()

    arr1 ← []
```

```
arr2 ← []

Print "Enter size of the first sorted array:"
len1 ← Input()

Print "Enter elements of the first sorted array:"
For i from 0 to len1 - 1:
    element ← Input()
    If i > 0 AND element < arr1[i - 1]:
        Print "Invalid input. Array must be sorted."
        Exit()
    arr1.Append(element)

Print "Enter size of the second sorted array:"
len2 ← Input()

Print "Enter elements of the second sorted array:"
For i from 0 to len2 - 1:
    element ← Input()
    If i > 0 AND element < arr2[i - 1]:
        Print "Invalid input. Array must be sorted."
        Exit()
    arr2.Append(element)

result ← k_th_recursive(k, arr1, arr2)
Print "The k-th element is:", result
```

5-Time complexity

Worst case:

This recurrence relation:

$T(n)=aT(b/n)+f(n)$

$T(n)=T(n)=T(n \backslash 2)+O(n)$

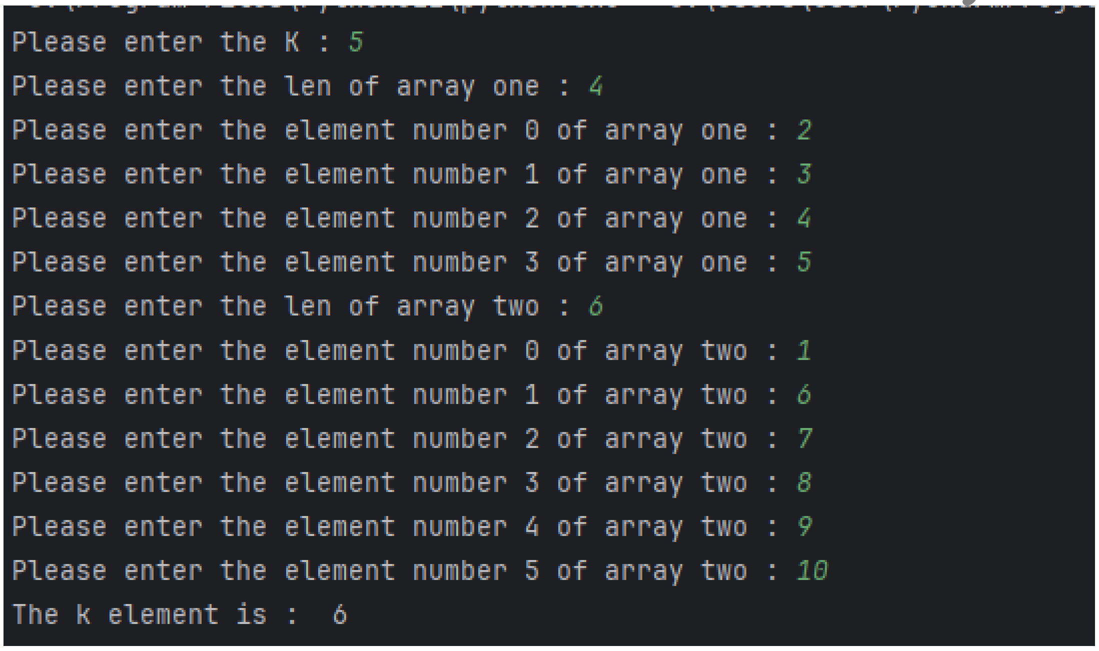
$a=1$

b=2
d=0
logba=log21=0
Thus n^logba =n^0 =O(1)
Since d= logba
Then, Case 2 of the Master Theorem:
T(n)=O(logn)

Best case:

O(1) (base case is reached quickly)

6-some test cases to check the validity of code



```
Please enter the K : 5
Please enter the len of array one : 4
Please enter the element number 0 of array one : 2
Please enter the element number 1 of array one : 3
Please enter the element number 2 of array one : 4
Please enter the element number 3 of array one : 5
Please enter the len of array two : 6
Please enter the element number 0 of array two : 1
Please enter the element number 1 of array two : 6
Please enter the element number 2 of array two : 7
Please enter the element number 3 of array two : 8
Please enter the element number 4 of array two : 9
Please enter the element number 5 of array two : 10
The k element is :  6
```

Input k:
The user is prompted to input the value of k. In this case, k = 5.
Input Array One (arr1):
The program asks for the length of the first sorted array (len1 = 4).
The elements of the first sorted array are entered: [2, 3, 4, 5].
Input Array Two (arr2):
The program asks for the length of the second sorted array (len2 = 6).
The elements of the second sorted array are entered: [1, 6, 7, 8, 9, 10].
Program Output:

The program calculates the k-th smallest element in the merged array of arr1 and arr2.

It outputs: The k element is: 6.

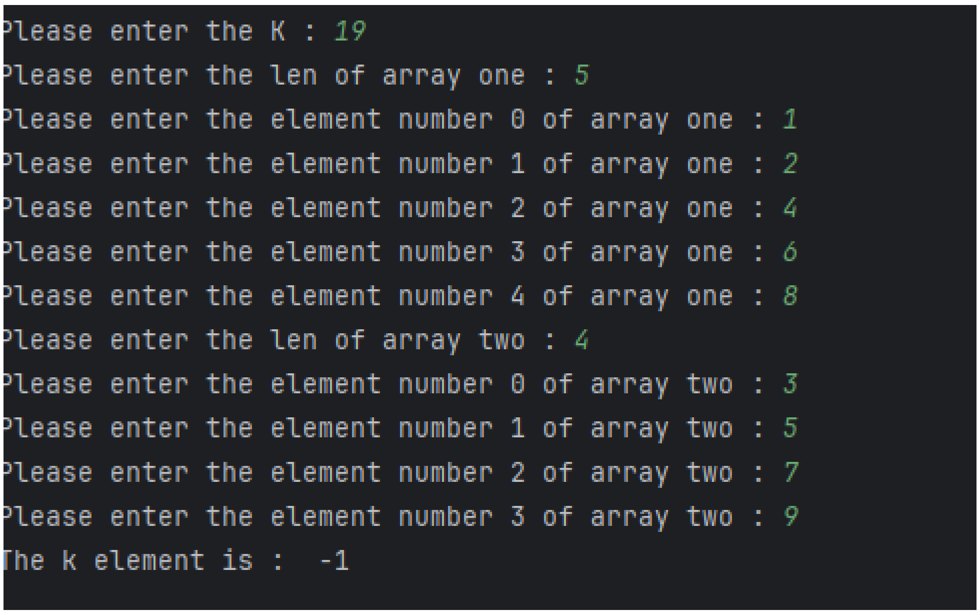
Merged Array and k-th Element:

The program logically merges the two arrays without actually creating a new array:

Merged Array (sorted): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

The k-th smallest element (0-indexed):

k = 5 refers to the 6th element in the merged array, which is 6.



```
Please enter the K : 19
Please enter the len of array one : 5
Please enter the element number 0 of array one : 1
Please enter the element number 1 of array one : 2
Please enter the element number 2 of array one : 4
Please enter the element number 3 of array one : 6
Please enter the element number 4 of array one : 8
Please enter the len of array two : 4
Please enter the element number 0 of array two : 3
Please enter the element number 1 of array two : 5
Please enter the element number 2 of array two : 7
Please enter the element number 3 of array two : 9
The k element is :  -1
```

Input k:

The user is prompted to input the value of k. In this case, k = 19.

Input Array One (arr1):

The program asks for the length of the first sorted array (len1 = 5).

The elements of the first sorted array are entered: [1,2,4,6,8].

Input Array Two (arr2):

The program asks for the length of the second sorted array (len2 = 4).

The elements of the second sorted array are entered: [3,5,7,9].

Program Output:

The program calculates the k-th smallest element in the merged array of arr1 and arr2.

It outputs: The k element is: -1.

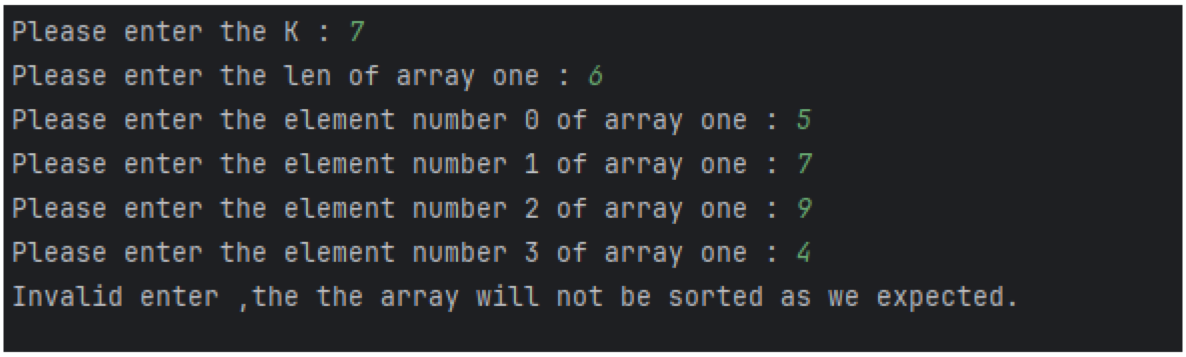
Merged Array and k-th Element:

The program logically merges the two arrays without actually creating a new array:

Merged Array (sorted): [1, 2, 3, 4, 5, 6, 7, 8, 9]

The k-th smallest element (0-indexed):

k = -1 means you're asking for the "20th smallest element" in the merged array, which does not exist.



```
Please enter the K : 7
Please enter the len of array one : 6
Please enter the element number 0 of array one : 5
Please enter the element number 1 of array one : 7
Please enter the element number 2 of array one : 9
Please enter the element number 3 of array one : 4
Invalid enter ,the the array will not be sorted as we expected.
```

third element: 9 .

Forth element: 4

It checks:

if element < arr1[i - 1]: # element = 2, arr1[i - 1] = 3

Since 4< 9, this violates the sorted condition.