Date: 06.03.2024 Name: Mugilkrishna D U
Ex. No: 3 Reg. No.: 3122225001073

UCS 2403 Design & Analysis of Algorithms

Assignment 3

Date of Exercise: 06.03.2024

Aim: To gain understanding and proficiency on how to find and analyse the errors in a code

Question 1:

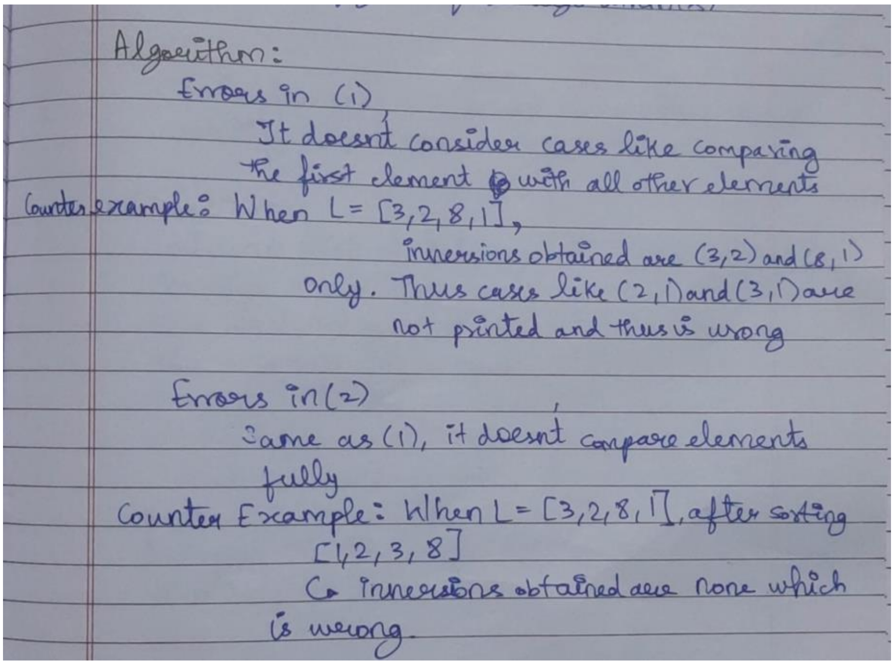
Given a list L of n numbers, an inversion is defined as a pair (L[i], L[j]) such that i < j and L[i] > L[j]. For example, if L = [3, 2, 8, 1], then (3, 2), (8, 1), (2, 1), (3, 1) are the inversions in L. Consider the Python codes given in (1) and (2) below for finding the count of inversions in a list.

```
(1) def count_inversions1(nums):
        count = 0
        for i in range(1, len(nums)):
            if nums[i] < nums[i - 1]:
                count += 1
        return count
```

```
(2) def count_inversions2(nums):
        nums.sort()
        count = 0
        for i in range(1, len(nums)):
            if nums[i] < nums[i - 1]:
                count += 1
        return count
```

Find if there are errors in these codes. If there are, find one counterexample for each incorrect code, fix the errors, and write your own (correct) code to find the count of inversions in a list. Derive the time complexity of your final algorithm(s). Note that a counterexample is an input instance to the algorithm that produces a wrong output

Algorithm:



Algorithm:

Errors in (1)

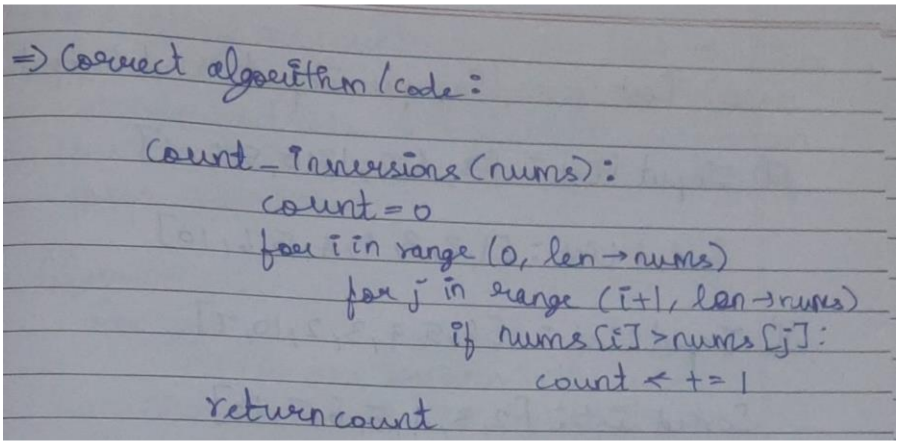
It doesnt consider cases like comparing the first element with all other elements

Counter example: When L = [3,2,8,1],
inversions obtained are (3,2) and (8,1) only. Thus cases like (2,1) and (3,1) are not printed and thus is wrong

Errors in (2)

Same as (1), it doesnt compare elements fully

Counter Example: When L = [3,2,8,1], after sorting [1,2,3,8]
↳ inversions obtained are none which is wrong.

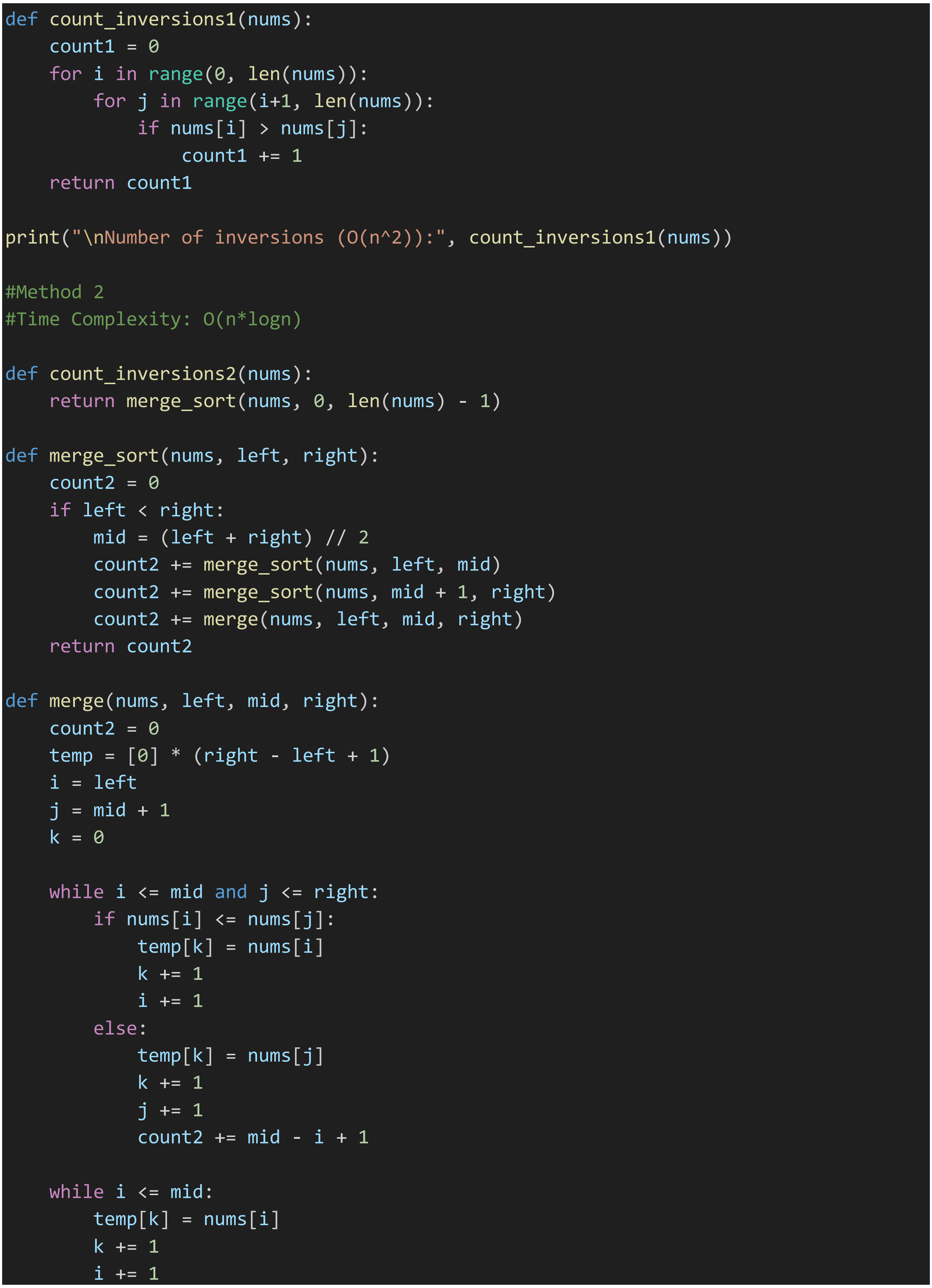


⇒ Correct algorithm / code:

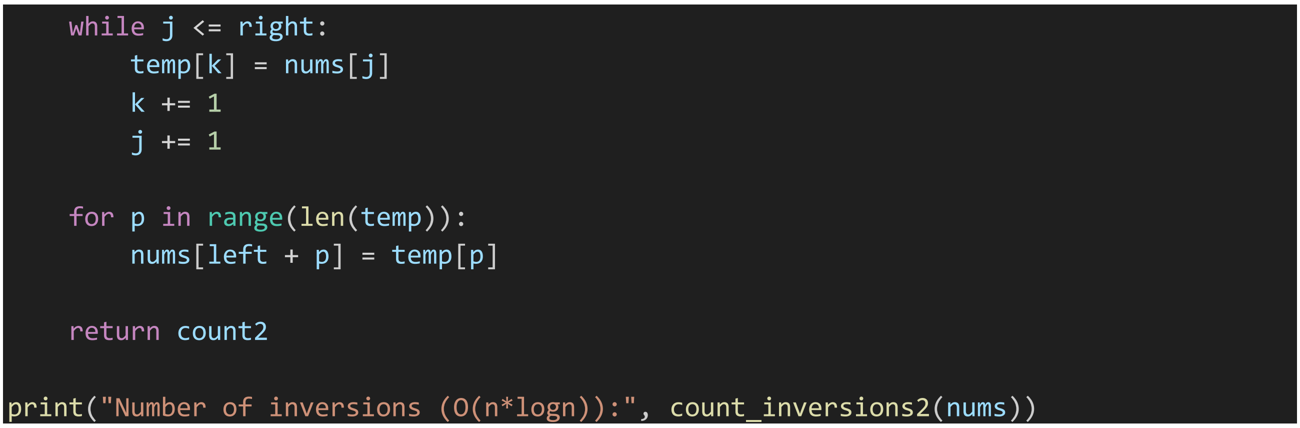
```
Count_Inversions(nums):
    count = 0
    for i in range (0, len→nums)
        for j in range (i+1, len→nums)
            if nums[i] > nums[j]:
                count ← += 1
    return count
```

Code:

```
#INVERSIONS IN A LIST

#Input list
size = int(input("Enter the size of the List: "))
nums = []

for k in range(size):
    nums.append(int(input("Enter element " + str(k+1) + ": ")))

print("\nList:", nums)

#Method 1
#Time Complexity: O(n^2)
```

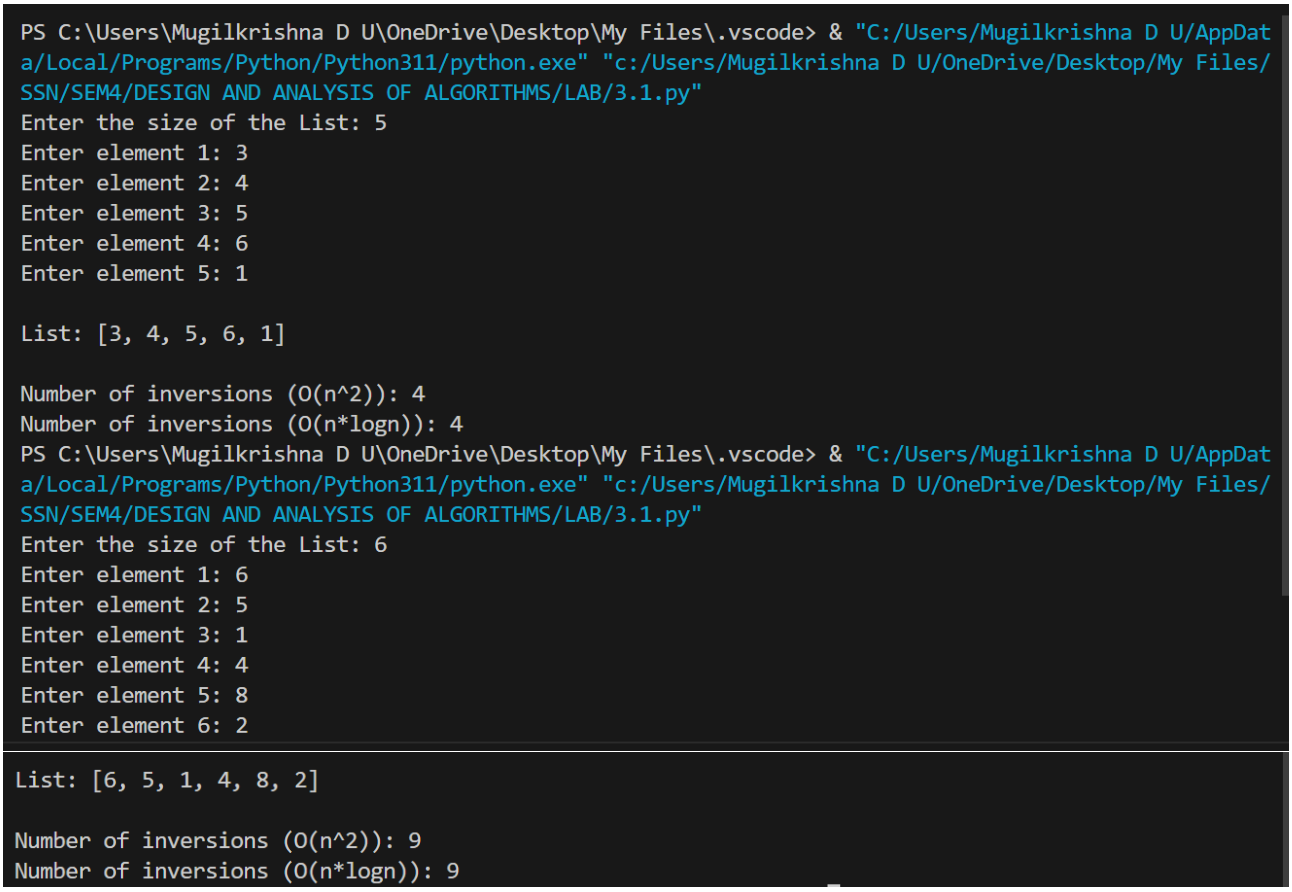


```python
def count_inversions1(nums):
    count1 = 0
    for i in range(0, len(nums)):
        for j in range(i+1, len(nums)):
            if nums[i] > nums[j]:
                count1 += 1
    return count1

print("\nNumber of inversions (O(n^2)):", count_inversions1(nums))

#Method 2
#Time Complexity: O(n*logn)

def count_inversions2(nums):
    return merge_sort(nums, 0, len(nums) - 1)

def merge_sort(nums, left, right):
    count2 = 0
    if left < right:
        mid = (left + right) // 2
        count2 += merge_sort(nums, left, mid)
        count2 += merge_sort(nums, mid + 1, right)
        count2 += merge(nums, left, mid, right)
    return count2

def merge(nums, left, mid, right):
    count2 = 0
    temp = [0] * (right - left + 1)
    i = left
    j = mid + 1
    k = 0

    while i <= mid and j <= right:
        if nums[i] <= nums[j]:
            temp[k] = nums[i]
            k += 1
            i += 1
        else:
            temp[k] = nums[j]
            k += 1
            j += 1
            count2 += mid - i + 1

    while i <= mid:
        temp[k] = nums[i]
        k += 1
        i += 1
```



```
    while j <= right:
        temp[k] = nums[j]
        k += 1
        j += 1

    for p in range(len(temp)):
        nums[left + p] = temp[p]

    return count2

print("Number of inversions (O(n*logn)):", count_inversions2(nums))
```

Output:



```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/3.1.py"
Enter the size of the List: 5
Enter element 1: 3
Enter element 2: 4
Enter element 3: 5
Enter element 4: 6
Enter element 5: 1

List: [3, 4, 5, 6, 1]

Number of inversions (O(n^2)): 4
Number of inversions (O(n*logn)): 4
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/3.1.py"
Enter the size of the List: 6
Enter element 1: 6
Enter element 2: 5
Enter element 3: 1
Enter element 4: 4
Enter element 5: 8
Enter element 6: 2
List: [6, 5, 1, 4, 8, 2]

Number of inversions (O(n^2)): 9
Number of inversions (O(n*logn)): 9
```

Time Complexity:

(i) **Method 1:** O(n^2) - Quadratic Time Complexity
(ii) **Method 2:** O(n*logn) – Linear Logarithmic Time Complexity

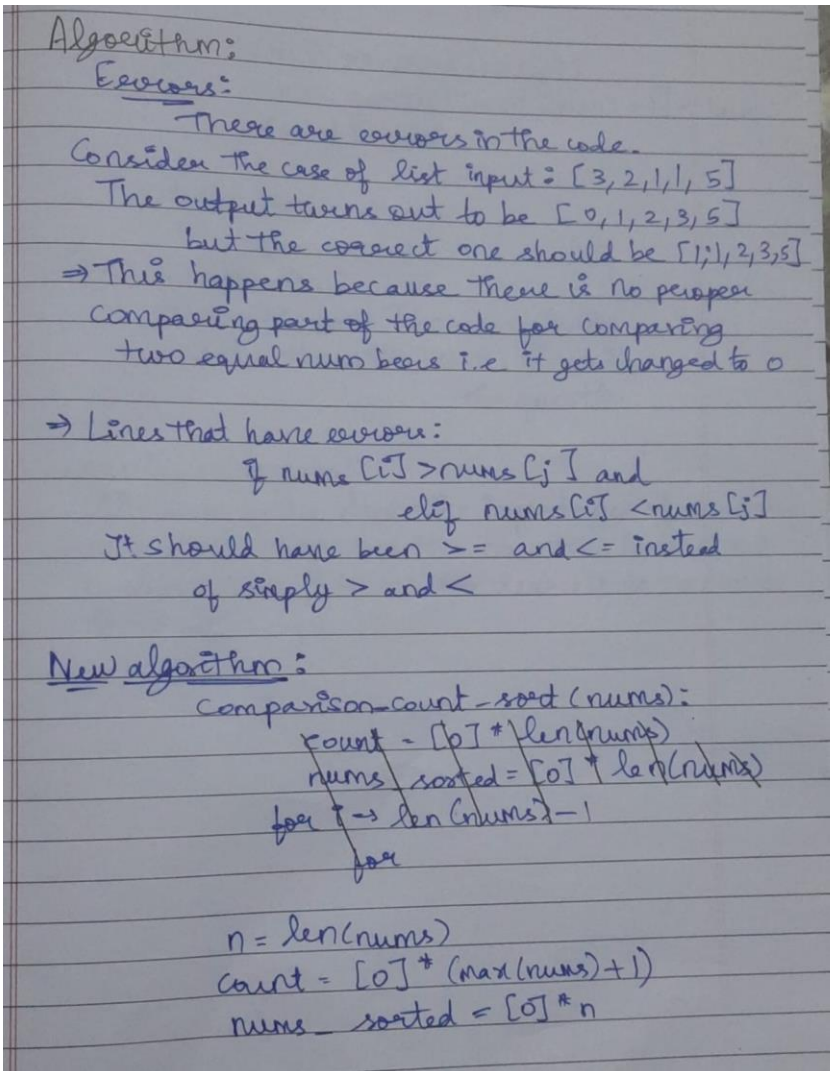
Question 2:

Given a list of integers, the comparison count sorting algorithm sorts the list as follows: For each integer at index i in the list, count the number of integers that are strictly less than it. In the sorted list, place the integer at the index equal to the number of integers that are less than it. For example, there are no integers less than the minimum integer in the list, so the minimum integer is placed at index 0. Now, consider the code given below to sort a list of numbers using comparison count sort.

```
def comparison_count_sort(nums):
    count = [0] * len(nums)
    nums_sorted = [0] * len(nums)
    for i in range(len(nums) - 1):
        for j in range(i + 1, len(nums)):
            if nums[i] > nums[j]:
                count[i] += 1
            elif nums[i] < nums[j]:
                count[j] += 1
    for i in range(len(nums)):
        nums_sorted[count[i]] = nums[i]
    return nums_sorted
```

Find if there are errors in this code. If there are, find at least one counterexample, list the lines of code that have errors, and write your own (correct) code to implement comparison count sort. Derive the time complexity of your algorithm.

Algorithm:



Algorithm:

Errors:

There are errors in the code.
Consider the case of list input: [3, 2, 1, 1, 5]
The output turns out to be [0, 1, 2, 3, 5]
but the correct one should be [1, 1, 2, 3, 5]

⇒ This happens because there is no proper comparing part of the code for comparing two equal num bers i.e it gets changed to 0

⇒ Lines that have errors:

```
if nums[i] > nums[j] and
        elif nums[i] < nums[j]
```

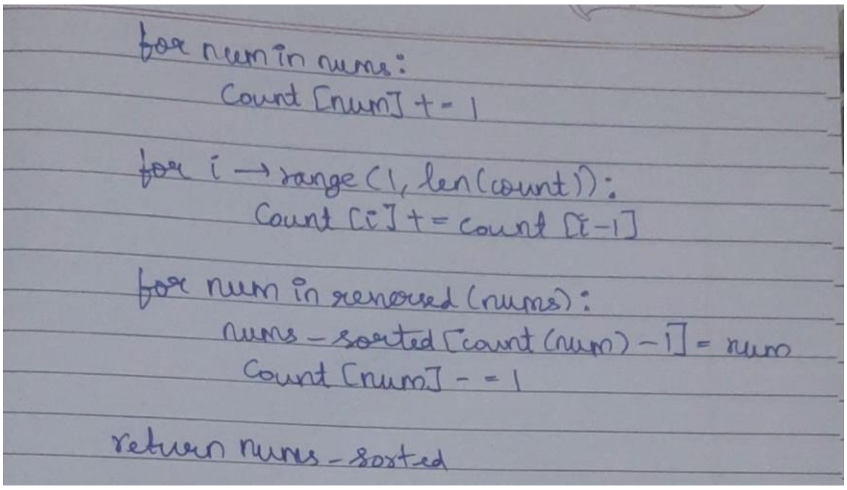
It should have been >= and <= instead of simply > and <

New algorithm:

```
comparison_count_sort(nums):
```

~~count = [0] * len(nums)~~
~~nums_sorted = [0] * len(nums)~~
~~for i → len(nums) - 1~~
~~for~~

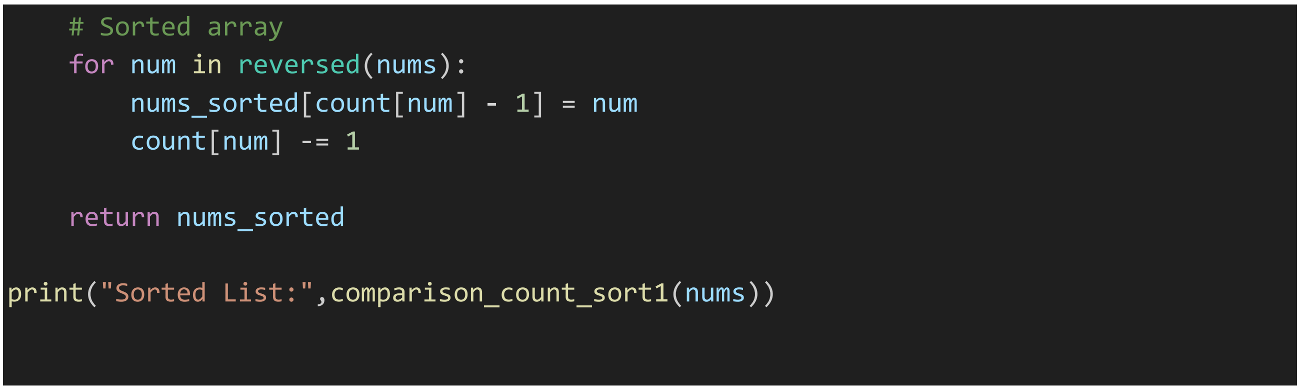
```
    n = len(nums)
    count = [0] * (max(nums) + 1)
    nums_sorted = [0] * n
```



```
    for num in nums:
        Count[num] += 1

    for i → range(1, len(count)):
        Count[i] += count[i-1]

    for num in reversed(nums):
        nums_sorted[count(num) - 1] = num
        Count[num] -= 1

    return nums_sorted
```

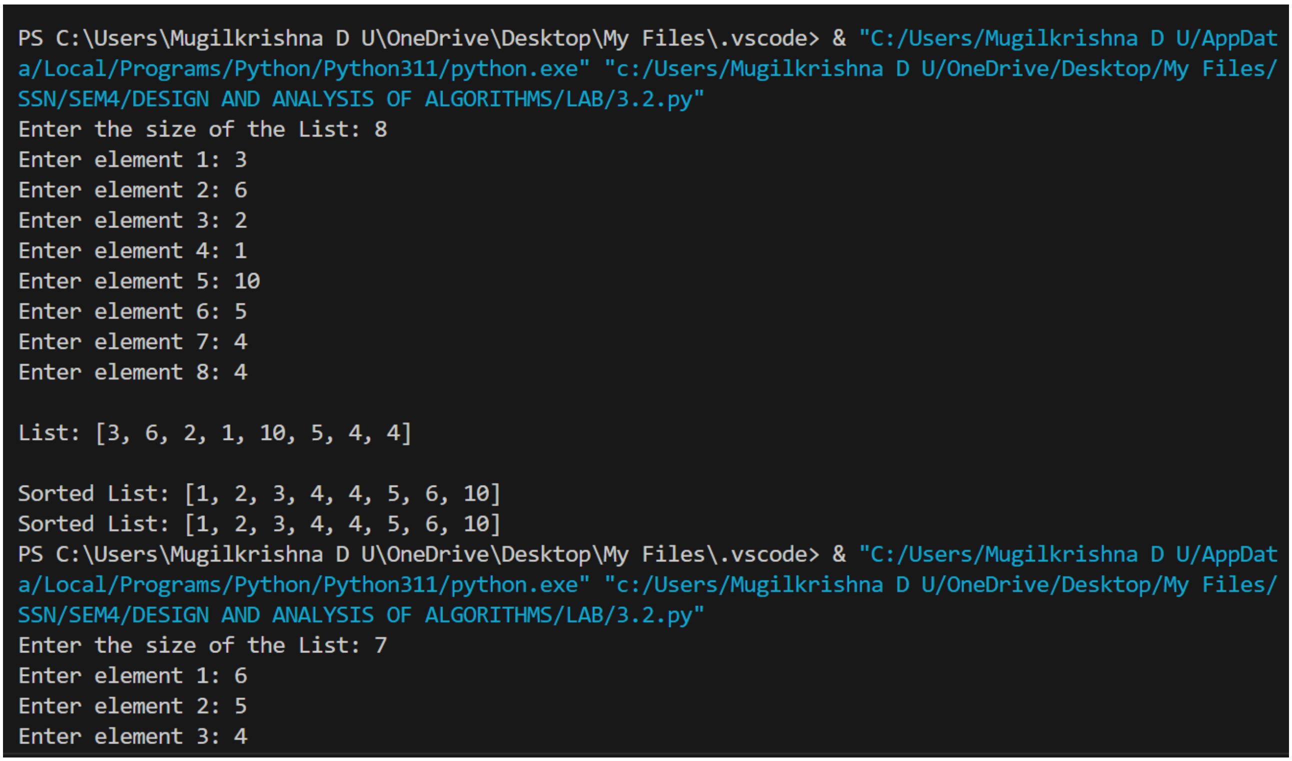
Code:

```
#COUNT SORTING ALGORITHM

#Input list
size = int(input("Enter the size of the List: "))
nums = []

for k in range(size):
    nums.append(int(input("Enter element " + str(k+1) + ": ")))

print("\nList:", nums)

#Method 1
#Time Complexity: O(n^2)

def comparison_count_sort(nums):
    count = [0] * len(nums)
    nums_sorted = [0] * len(nums)
    for i in range(len(nums) - 1):
        for j in range(i + 1, len(nums)):
            if nums[i] >= nums[j]:
                count[i] += 1
            elif nums[i] <= nums[j]:
                count[j] += 1
    for i in range(len(nums)):
        nums_sorted[count[i]] = nums[i]
    return nums_sorted

print("\nSorted List:",comparison_count_sort(nums))

#Method 2
#Time Complexity: O(n)

def comparison_count_sort1(nums):
    n = len(nums)
    count = [0] * (max(nums) + 1)  # Initialize count array with length max(nums) +
1
    nums_sorted = [0] * n

    # Count occurrences of each element
    for num in nums:
        count[num] += 1

    # Calculate Cumulative Sum
    for i in range(1, len(count)):
        count[i] += count[i - 1]
```

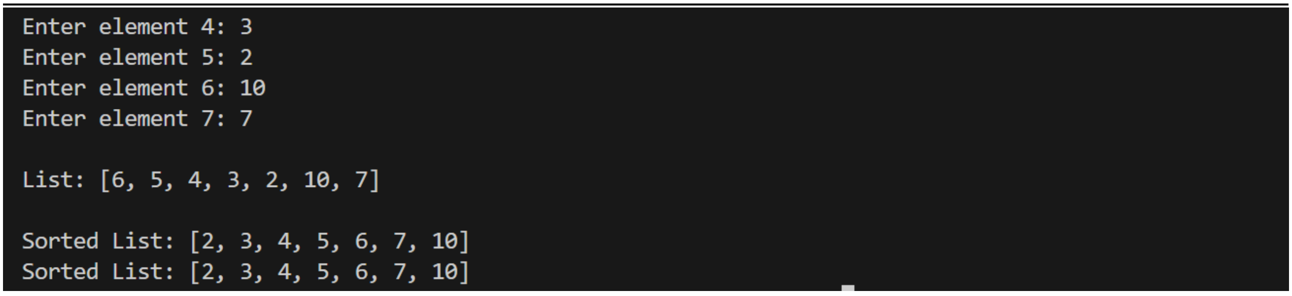


```
    # Sorted array
    for num in reversed(nums):
        nums_sorted[count[num] - 1] = num
        count[num] -= 1

    return nums_sorted

print("Sorted List:",comparison_count_sort1(nums))
```

Output:



```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/3.2.py"
Enter the size of the List: 8
Enter element 1: 3
Enter element 2: 6
Enter element 3: 2
Enter element 4: 1
Enter element 5: 10
Enter element 6: 5
Enter element 7: 4
Enter element 8: 4

List: [3, 6, 2, 1, 10, 5, 4, 4]

Sorted List: [1, 2, 3, 4, 4, 5, 6, 10]
Sorted List: [1, 2, 3, 4, 4, 5, 6, 10]
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppDat
a/Local/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/
SSN/SEM4/DESIGN AND ANALYSIS OF ALGORITHMS/LAB/3.2.py"
Enter the size of the List: 7
Enter element 1: 6
Enter element 2: 5
Enter element 3: 4
```



```
Enter element 4: 3
Enter element 5: 2
Enter element 6: 10
Enter element 7: 7

List: [6, 5, 4, 3, 2, 10, 7]

Sorted List: [2, 3, 4, 5, 6, 7, 10]
Sorted List: [2, 3, 4, 5, 6, 7, 10]
```

Time Complexity:

- **(i)** **Method 1:** O(n^2) – Quadratic Time Complexity
- **(ii)** **Method 2:** O(n) – Linear Time Complexity

Learning Outcome:

Upon completing this exercise, I have understood the importance of counting sort algorithm and the counting inversions algorithm and I have learnt to maximise their efficiency by reducing their time complexities. I have also learnt to check for errors and correct the code.