Cairo University
Faculty of Engineering
Credit Hour System

SBEN429 Biomedical Data Analytics
Assignment 3

Name: **Sandra Adel Aziz Gebraiel**

ID: **1180059**

Date: **13/12/2021**

Assignment 3 Divide and Conquer
Number of Inversions

1) Explanation of Code:

Naively, to get the number of inversions in array, we pass through the array elements one-by-one, holding each element, we iterate on all the elements after it, comparing each with it, and incrementing a counter that represents the number of times we have found an element whose value is the greater than the one we hold, counting the number of times a small value is placed after a larger value in our array
TIME COMPLEXITY: O(n^2) → two nested loops

To optimize, we count the number of invserions throughout the inner steps of a sorting technique, merge sort here. We do not need to consume the memory with the auxiliary arrays needed to form the merged sorted array, beacuse we only need the count of inversions, not the sorted array.

Therefore, we simulate and keep track of the process of the merge sort algorithm with pointers on the original array, "left" and "right", with the help of a temporary array "b".

To illustrate, the merge function is simulated with get_number_of_inversions, where still the recursion base case is when the length of the sub-array reaches 1, by checking the difference between the array left and right pointers. In each recursion step, the array is divided into 2 halves, through pointers and not using auxillary arrays, where both halves are sorted and merged while counting the number of inversions (number of smaller elements placed after ones larger than them) in the merge function simulated here by get_number_of_pairs and its return value is added on the number_of_inversions variable in this recursion step, adding up on all number_of_inversions values throughout all the recursion steps, reaching total number of inversions in the whole array at the end

get_number_of_pairs function works with the temporary array "b" and left and right for each sub-array, as parameters for each sub-array, a pointer is intialized with the value of the left pointer to iterate through it while both the arrays are non-empty, simulated here with the pointer arrays still not reaching the value of the right pointer, indicating its end, the elements which both pointers of both sub-arrays are pointing are compared.

If left sub-array element is smaller than that of the right, it is firstly copied to its same index in the temporary array using both the pointers of the left sub-array and that of the temporary array, to track it, before they are both incremented.

If right sub-array element is smaller than that of the left, that means that there is a smaller value placed after (a) larger one(s), so the number_of_pairs variable is incremented with the number of elements in the left sub-array, an inversion pair for its element in the right sub-array with each element it is smaller than in the left-subarray. Then, it is also copied to its same index in the temporary array using the pointers of the right sub-array and the temporary array before incremeting them.

Two if conditions check which array is still full, its array pointer still not reaching the right pointer, and its elements are copied according to their indices to the temporary array.

Final result is that the temporary array contains the sorted merged version of the two sub-arrays, and its elements are respectively copied to the original array, containing the 2 sub-arrays, forming the basis of the next recursion step, in which other 2 sub-arrays on array "a" are merged and sorted.

TIME COMPLEXITY: O(nxlogn) → counting the number of inversions without adding any extra complexity to the merge sort algorithm

2) Pseudocode:

GetNumberOfInversions(a, b, left, right):

Initialize NumberOfInversions with zero

If right – left <= one:
    Return NumberOfInversions

Otherwise:

    ave ← (left – right)/2

    Increment NumberOfInversions with the return value of
    GetNumberOfInversions(a, b, left, ave)

    Increment NumberOfInversions with the return value of
    GetNumberOfInversions(a, b, ave, right)

    Increment NumberOfInversions with the return value of
    GetNumberOfPairs(a, b, left, ave, ave, right)

    Return NumberOfInversions

GetNumberOfPairs(a, b, arr1-left, arr1-right, arr2-left, arr2-right):

    Initialize NumberOfPairs with zero
    arr1_ptr ← arr1_left
    arr2_ptr ← arr2_left
    b_ptr ← arr1_left

    While both sub-arrays are not empty:

        If element of arr1_ptr in a <= element of arr2_ptr in a:
            b[b_ptr] ← a[arr1_ptr]
            Increment arr1_ptr
            Increment b_ptr

        Otherwise:
            Increment NumberOfPairs with number of elements in left sub-array
            at this step
            b[b_ptr] ← a[arr2_ptr]
            Increment arr2_ptr
            Increment b_ptr

Copy remaining elements in left or right sub-array to b
using the current value of b_ptr

Copy elements in b between arr1_left to arr2_right to their same indices in a

Return NumberOfPairs

3) Naïve Algorithm and Stress Testing Code:

```
# Naive get_number_of_inversions --> based on 2 for loops
def get_number_of_inversions_naive(a, b, left, right):

    number_of_inversions = 0

    for i in range(len(a)):
        for j in range(i+1, len(a)):
            if a[i] > a[j]:
                number_of_inversions += 1

    return number_of_inversions


if __name__ == '__main__':

    #input = sys.stdin.read()
    #n, *a = list(map(int, input.split()))
    #n = 10 ** 5
    #b = n * [0]
    #a = n * [10**9]
    #print(get_number_of_inversions(a, b, 0, len(a)))


    while 1: # Infinite Loop
        a = []
        n = r.randint(1, 100)
        print(n)
        for i in range(n):
            a.append(r.randint(1, 100))
        print(a)

        b = n * [0]
        res1 = get_number_of_inversions_naive(a, b, 0, len(a))
        res2 = get_number_of_inversions(a, b, 0, len(a))

        if res1 != res2:
            print('wrong answer'+ ' ' + str(res1) + ' ' + str(res2))
            break
        else:
            print('OK')
```

4) Resources:

Counting Inversions In An Array Using Merge Sort - PrepForTech

Inversion Count in an Array Using Merge Sort - Aticleworld