Name: Razin Sufian

ID: 22301219

Sec: 16

Lab Assignment 03

Code Explanation

Task 1:

First task is the implemantaition of merge sort which gives n log n running time in sorting an arra.

Here we used ~~two~~ function. One ~~to~~ divide the arra in small ~~pices~~ peices (merge sort). Here by doing recunsive call we break the arra in small pieies then ~~every~~ there two ~~sort~~ peices are merged into a single sorted arra by merge function.

Task 2:

in this task we found out the max number by ~~deviding~~ the same deviding method we use in merge sort. but here we dont combine the small pices ; we just compare it and keep the max value and return the maxmum one after the full iteration by comparing all the numbers. So; here we dont need any merge function task 3:-

here we are given an array, we have to check for every elements how many small numbers are their of that specific number from the right. and this to the count.

There is alredy $O(n^2)$ solution given in the question. I ampresenting a $O(n \log n)$ solution.

Solution:-

1. first I sorted the arra using merge_sort
= sorted_arra
2.

~~the~~ I ran a loop in to ~~the~~ give arra. ~~find that numt each numbe~~

let it is i.

I will find i in the sorted array using binary search; and the returned index number of it will add ~~to~~ the count.

besically

count += len(arraa) - (len(arra) - indx)

Ex:

2 7 4 1 5 6 8 3

ind=1

1 2 3 4 5 6 7 8

$9 - (9-1)$

$= 1 = ind$

pop after founding each count. So; for 7 the arrays will be

task 4:

→ in this task from the given arra we have to find the maximum possible of

$A[i] + A[j]^2$; $1 <= j <= N$

steps:

first i converted the arra in to taple which contain the square of it and the 2nd element contains the indx number. the i sorted the arra using the 1st element of the taple using marged sort.

Ex:

-5 -2 -6 -7 -1 8 2

tup = (25,0), (4,1), (36,2),

(49,3), (1,4), (64,5)

(4,6)

tup = (1,4), (4,1), (4,6), ~~(36,2)~~.

(25,0), (36,2) (49,3),

(64,5)

then creat two pointer i and j

i = 0

j = len (arr) - 1

while i < j :

if [j] [i] <= i :

j -= 1

it means if the idex of a maximum squared value is less then or equal to i then we move on to the next maximum squared value.

as ~~AD~~ i > j

the saving the maximum each time.

it's running time is O (n log n)

Task 5

~~it~~

this task is the implementation of Quick sort. Here we use two function. one for detmining the area of partition (start, end) and other wam creats the partition.

in partition we considered the leftmost element as the pivot. All the elements to it's left is smaller and to it's right is greater than the pivot.

by recursively doin the partition we get the sorted array, which running time is O(nlogn),

but worst case, (~~so~~ if array is sorted); time is O(n²)

Task 6:

here we are just finding the index number of an element using partition. here the same code as quick sort ~~but here we dont swap~~

Steps:

find the index of K

1. choose a pivot elem from the list

2. Partition the list

3. if the pivot's position is K, return pivot

4. if K is less than the pivot's position recursively apply the the method to the left sub array

5. if k is greater than than the pivot then doing the recursion to the right