Ex-no:5 3122 21 5001 085
Date: 29-03-2023 Name: M.Rohith

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam - 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)

UCS2403: DESIGN & ANALYSIS OF ALGORITHMS

Assignment 5

1. Recall the problem statement for counting inversions in a list.
   (a) Modify the algorithm of Mergesort to count inversions in a given list.
   (b) Compare the time complexity of this algorithm against the time complexity of the code you wrote in Assignment 3 to compute the count of inversions.

2. (a) Given a list A of size n, find the sum of elements in a subset A' of A such that the elements of A' are contiguous and has the largest sum among all such subsets. Please note that:
   - the subset should be having elements that are contiguous in the original list.
   - the input list may have negative values.
   - the algorithm should be based on divide and conquer strategy.

   Example:
   Input: A = [-2,1,-3,4,-1,2,1,-5,4]
   Output: 6

   (b) Write the recurrence relation for the time complexity of your algorithm, and find a closed form expression for the same.

Ex-no:5 3122 21 5001 085
Date: 29-03-2023 Name: M.Rohith

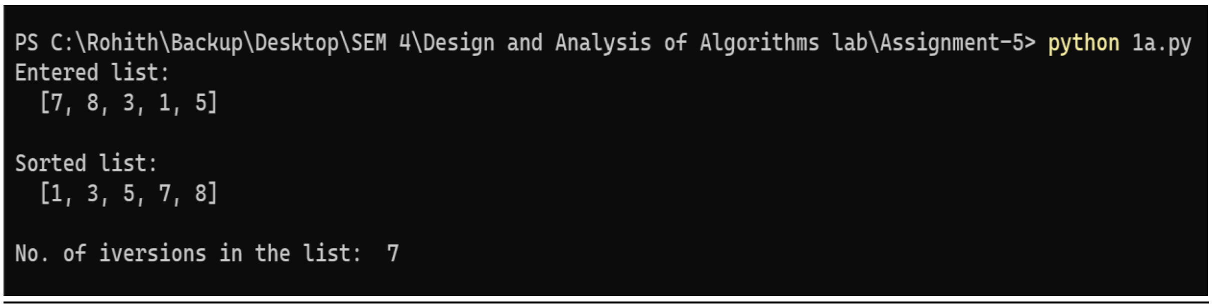
1. **A)**

Program code:

```python
# Python program to count inversions in an array

# Function to Use Inversion Count
def mergeSort(arr, n):
    # A temp_arr is created to store
    # sorted array in merge function
    temp_arr = [0]*n
    return _mergeSort(arr, temp_arr, 0, n-1)


# This Function will use MergeSort to count inversions
def _mergeSort(arr, temp_arr, left, right):
    # A variable inv_count is used to store
    # inversion counts in each recursive call
    inv_count = 0

    # We will make a recursive call if and only if
    # we have more than one elements
    if left < right:

        # mid is calculated to divide the array into two subarrays
        # Floor division is must in case of python
        mid = (left + right)//2

        # It will calculate inversion
        # counts in the left subarray
        inv_count += _mergeSort(arr, temp_arr,left, mid)

        # It will calculate inversion
        # counts in right subarray
        inv_count += _mergeSort(arr, temp_arr,mid + 1, right)

        # It will merge two subarrays in
        # a sorted subarray
        inv_count += merge(arr, temp_arr, left, mid, right)
    return inv_count
```

```python
# This function will merge two subarrays
# in a single sorted subarray


def merge(arr, temp_arr, left, mid, right):
    i = left     # Starting index of left subarray
    j = mid + 1  # Starting index of right subarray
    k = left     # Starting index of to be sorted subarray
    inv_count = 0

    # Conditions are checked to make sure that
    # i and j don't exceed their
    # subarray limits.
    while i <= mid and j <= right:

        # There will be no inversion if arr[i] <= arr[j]
        if arr[i] <= arr[j]:
            temp_arr[k] = arr[i]
            k += 1
            i += 1
        else:
            # Inversion will occur.
            temp_arr[k] = arr[j]
            inv_count += (mid-i + 1)
            k += 1
            j += 1

    # Copy the remaining elements of left
    # subarray into temporary array
    while i <= mid:
        temp_arr[k] = arr[i]
        k += 1
        i += 1

    # Copy the remaining elements of right
    # subarray into temporary array
    while j <= right:
        temp_arr[k] = arr[j]
        k += 1
        j += 1

    # Copy the sorted subarray into Original array
    for loop_var in range(left, right + 1):
        arr[loop_var] = temp_arr[loop_var]

    return inv_count


# Driver Code
# Given array is
arr = [7,8,3,1,5]
n = len(arr)
print("Entered list:\n ",arr)
result = mergeSort(arr, n)
print("\nSorted list:\n ",arr)
print("\nNo. of iversions in the list: ",result,"\n")
```

Output:

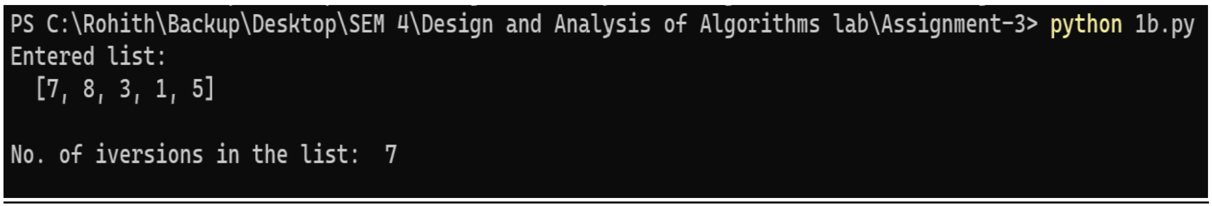


```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-5> python 1a.py
Entered list:
  [7, 8, 3, 1, 5]

Sorted list:
  [1, 3, 5, 7, 8]

No. of iversions in the list:  7
```

Program code (Assignment-3) :

```python
def count_inversion1(nums):
    count=0
    for i in range(1,len(nums)):
        for j in range(i,-1,-1):
            if(nums[i]<nums[j]):
                count+=1
    return count

nums=[7,8,3,1,5]
print("Entered list:\n ",nums)
print("\nNo. of iversions in the list:
",count_inversion1(nums),"\n")
```

Output:



```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-3> python 1b.py
Entered list:
  [7, 8, 3, 1, 5]

No. of iversions in the list:  7
```

b) Time complexity of Program code-1:

The time complexity of the given code is O(n*log(n)), where n is the size of the input array.

This is because the mergeSort() function uses the merge-sort algorithm which has a time complexity of O(n*log(n)) in the average and worst case scenarios.

In each recursive call of _mergeSort(), the array is divided into two halves and then merged using the merge() function. The time complexity of the merge() function is O(n), where n is the size of the two sub-arrays being merged. Therefore, the time complexity of _mergeSort() is also O(n*log(n)).

Overall, the time complexity of the entire program is dominated by the time complexity of the mergeSort() function, which is O(n*log(n)).

Time complexity of Program code-2 (Assignment-3):

The time complexity of the count_inversion1 function is O(n^2), where n is the length of the input list nums.

The outer loop runs n-1 times, and the inner loop runs from i-1 to 0, which takes i iterations. Therefore, the total number of iterations of the inner loop is the sum of the integers from 1 to n-1, which is (n-1)n/2. This gives us a time complexity of O(n(n-1)/2), which simplifies to O(n^2).

Since the rest of the code is just a constant-time operation, the overall time complexity of the program is O(n^2).

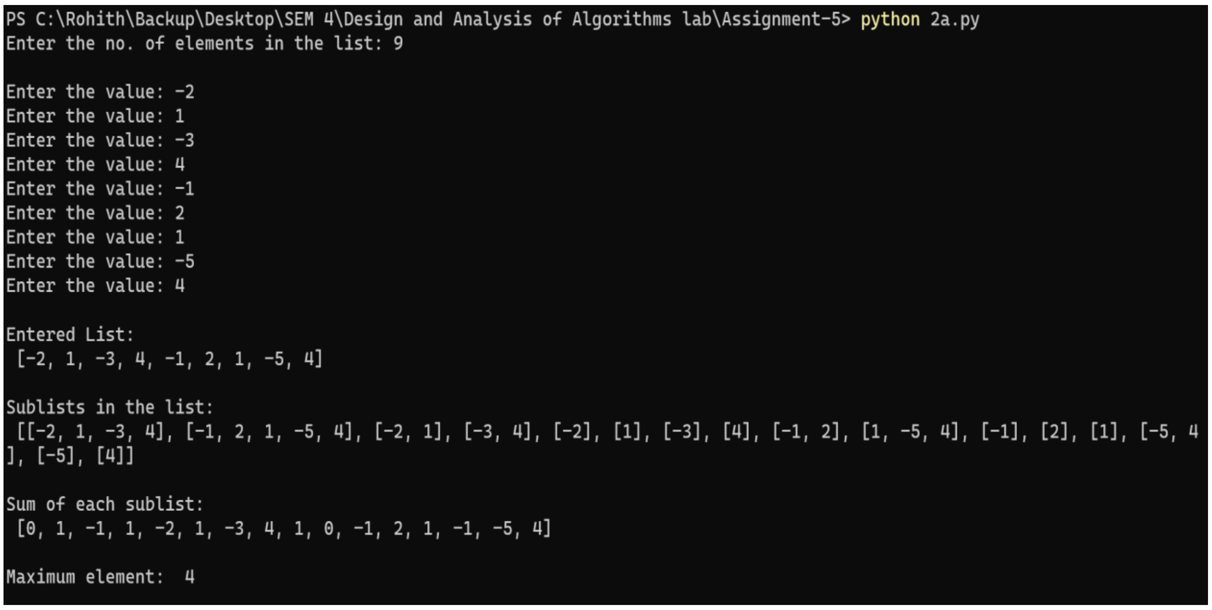
2. A) Program code:

```python
#Python program to find the largest sum of all subsets in the list

#Divide and conquer strategy to find the contiguous subsets
#in the list A and storing it in A1
def Divide_and_Conquer_sublist(A,A1):
    l=len(A)
    if l>=2:
        mid=l//2
        left_list=[]
        for i in range(mid):
            left_list.append(A[i])
        A1.append(left_list)

        right_list=[]
        for i in range(mid,l):
            right_list.append(A[i])
        A1.append(right_list)

        Divide_and_Conquer_sublist(left_list,A1)
        Divide_and_Conquer_sublist(right_list,A1)

#Divide and conquer strategy to find the max element
#in A1 by providing starting index and length of A1
def Divide_and_Conquer_findmax(A1,index,length):
    max=-1
    if(length-1==0):
        return A1[index]
    if(index>=length-2):
        if(A1[index]>A1[index+1]):
            return A1[index]
        else:
            return A1[index+1]
    max=Divide_and_Conquer_findmax(A1,index+1,length)
    if(A1[index]>max):
        return A1[index]
    else:
        return max
```

Ex-no:5 3122 21 5001 085
Date: 29-03-2023 Name: M.Rohith

```
#Function to calculate the sum of elements in
#each subset of list A1
def calc_sum(A1):
    for i in range(len(A1)):
        sum=0
        for j in range(len(A1[i])):
            sum+=A1[i][j]
        A1[i]=sum

#Driver code
A=[]
n=int(input("Enter the no. of elements in the list: "))
print()

#Getting input list
for i in range(n):
    A.append(int(input("Enter the value: ")))

print("\nEntered List: \n",A,"\n")

A1=[]
Divide_and_Conquer_sublist(A,A1)
print("Sublists in the list: \n",A1,"\n")

calc_sum(A1)
print("Sum of each sublist: \n",A1)

max=Divide_and_Conquer_findmax(A1,0,len(A1))
print("\nMaximum element: ",max,"\n")
```

Output:



```
PS C:\Rohith\Backup\Desktop\SEM 4\Design and Analysis of Algorithms lab\Assignment-5> python 2a.py
Enter the no. of elements in the list: 9

Enter the value: -2
Enter the value: 1
Enter the value: -3
Enter the value: 4
Enter the value: -1
Enter the value: 2
Enter the value: 1
Enter the value: -5
Enter the value: 4

Entered List:
 [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Sublists in the list:
 [[-2, 1, -3, 4], [-1, 2, 1, -5, 4], [-2, 1], [-3, 4], [-2], [1], [-3], [4], [-1, 2], [1, -5, 4], [-1], [2], [1], [-5, 4
], [-5], [4]]

Sum of each sublist:
 [0, 1, -1, 1, -2, 1, -3, 4, 1, 0, -1, 2, 1, -1, -5, 4]

Maximum element:  4
```

b) Time complexity:

The time complexity of the code can be analyzed as follows:

Divide_and_Conquer_sublist function:

The function divides the list into two halves and recursively calls itself on each half. The time complexity for dividing the list takes O(1) time, and the recursion depth is log(n), where n is the number of elements in the list.

At each recursive call, the function appends the left and right halves to the A1 list. The time complexity for each append operation is O(n), where n is the length of the list being appended.

Therefore, the time complexity of Divide_and_Conquer_sublist function is O(n*log(n)).

Divide_and_Conquer_findmax function:

The function finds the maximum element in the list using divide and conquer approach. The function recursively calls itself on a sub-list until it reaches the base case.

At each recursive call, the function checks two elements and returns the maximum of the two. The function makes n-1 comparisons to find the maximum element in the list, where n is the length of the list.

Therefore, the time complexity of Divide_and_Conquer_findmax function is O(n).

calc_sum function:

The function calculates the sum of elements in each sublist. It iterates over each sublist and sums up its elements. The time complexity of the function is O(n^2), where n is the total number of elements in all sublists.

The driver code:

It takes input from the user and creates the list A. The time complexity for taking input from the user is O(n).

The driver code calls Divide_and_Conquer_sublist function, which takes O(n*log(n)) time.

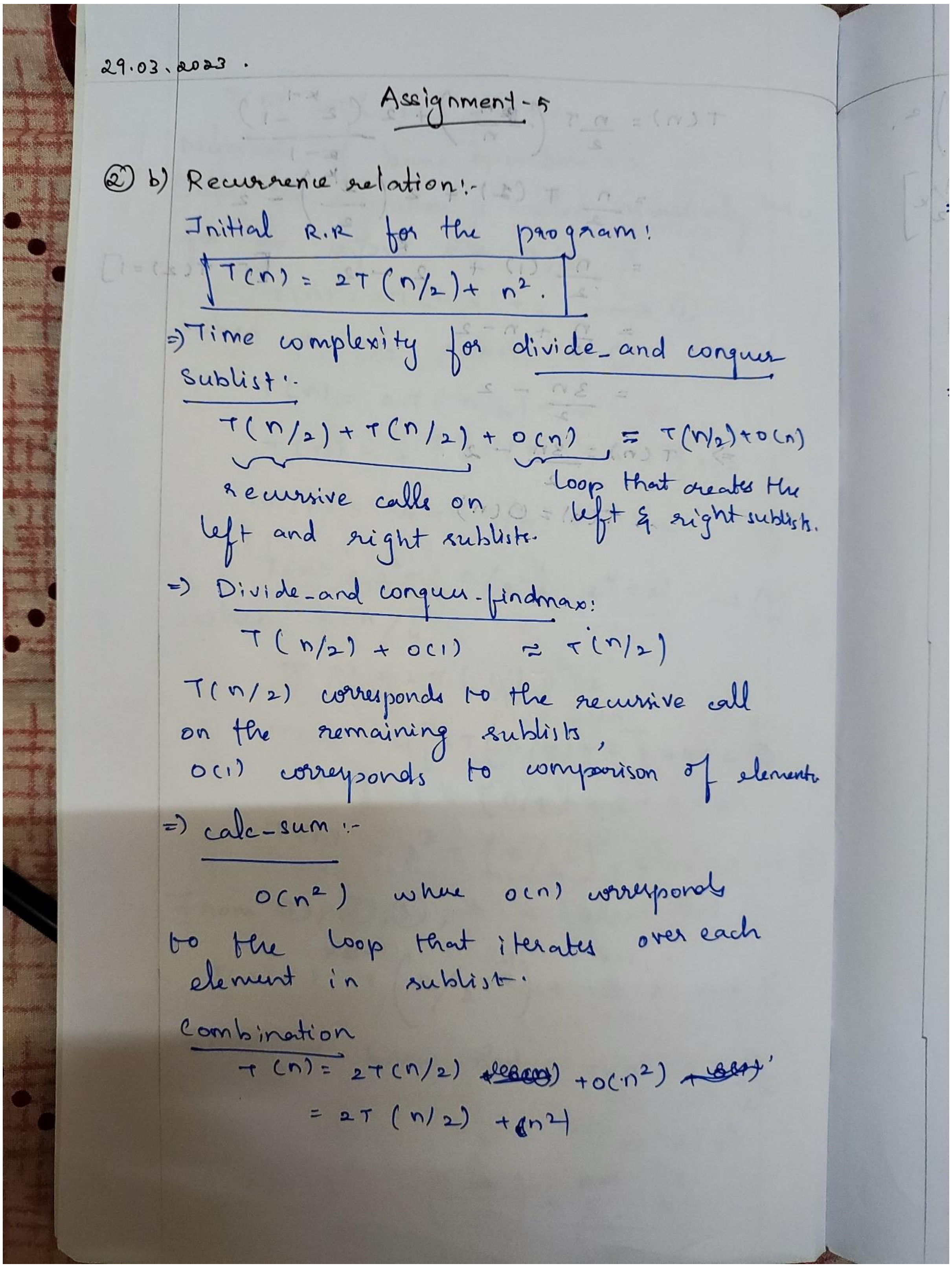
The driver code calls calc_sum function, which takes O(n^2) time.

The driver code calls Divide_and_Conquer_findmax function, which takes O(n) time.

Therefore, the time complexity of the driver code is O(n*log(n)) + O(n^2) + O(n) = O(n^2).

Therefore, the overall time complexity of the code is O(n*log(n)) + O(n^2) + O(n) = O(n^2).

Ex-no:5 3122 21 5001 085
Date: 29-03-2023 Name: M.Rohith



29.03.2023.

Assignment-5

② b) Recurrence relation:-

Initial R.R for the program:

$$\boxed{T(n) = 2T(n/2) + n^2.}$$

⇒ Time complexity for divide_and conquer sublist:-

$\underbrace{T(n/2) + T(n/2)}_{\text{recursive calls on left and right sublists.}} + \underbrace{O(n)}_{\text{Loop that creates the left \& right sublists.}} = T(n/2) + O(n)$

⇒ Divide_and conquer_findmax:

$T(n/2) + O(1) \approx T(n/2)$

$T(n/2)$ corresponds to the recursive call on the remaining sublists,
$O(1)$ corresponds to comparison of elements.

⇒ calc_sum:-

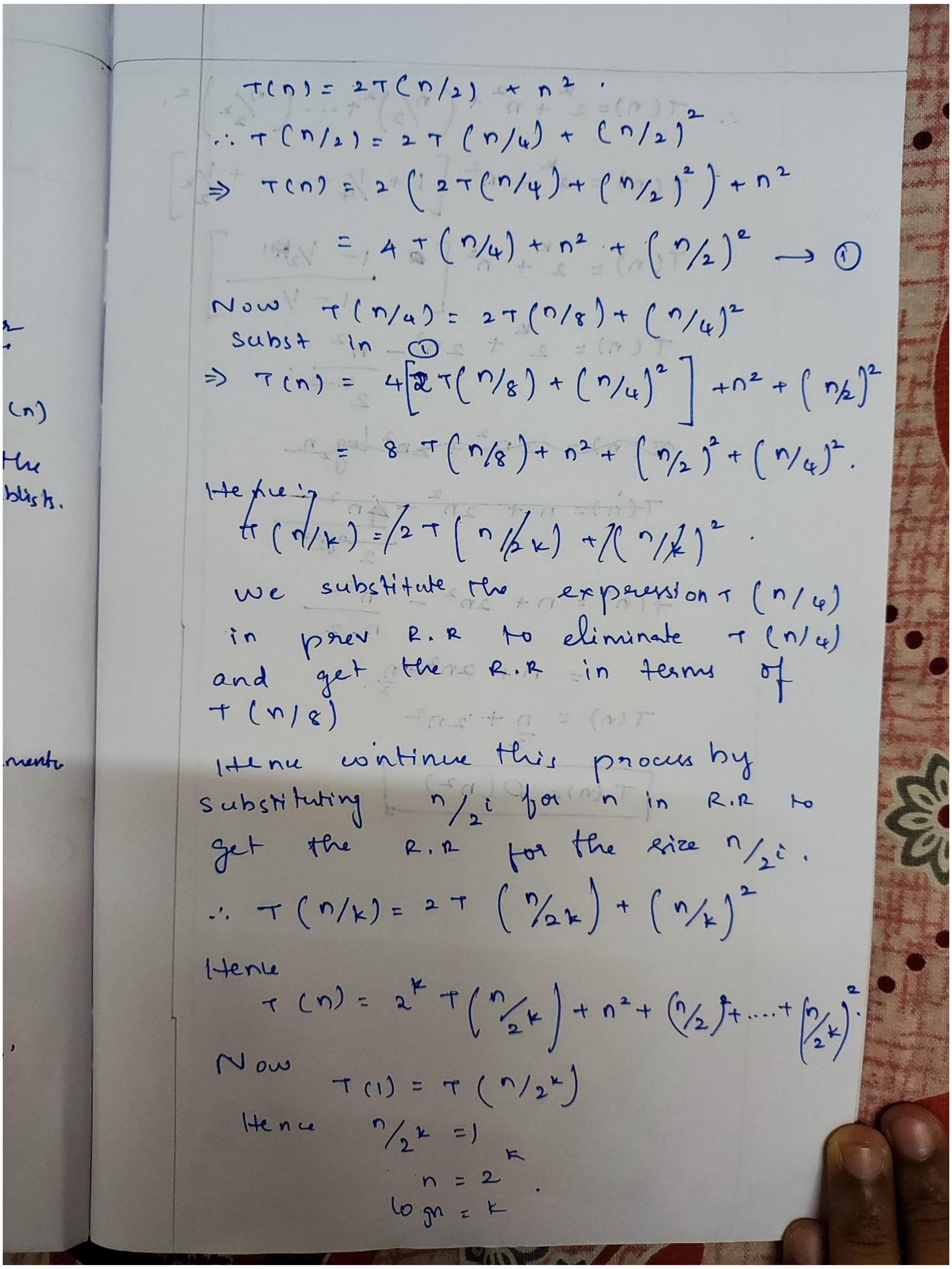
$O(n^2)$ where $O(n)$ corresponds to the loop that iterates over each element in sublist.

Combination

$T(n) = 2T(n/2)$ ~~[illegible]~~ $+ O(n^2)$ ~~[illegible]~~

$= 2T(n/2) + O(n^2)$

Ex-no:5 3122 21 5001 085
Date: 29-03-2023 Name: M.Rohith



$T(n) = 2T(n/2) + n^2$.

$\therefore T(n/2) = 2T(n/4) + (n/2)^2$

$\Rightarrow T(n) = 2\left(2T(n/4) + (n/2)^2\right) + n^2$

$= 4T(n/4) + n^2 + (n/2)^2 \longrightarrow ①$

Now $T(n/4) = 2T(n/8) + (n/4)^2$
subst in ①

$\Rightarrow T(n) = 4\left[2T(n/8) + (n/4)^2\right] + n^2 + (n/2)^2$

$= 8T(n/8) + n^2 + (n/2)^2 + (n/4)^2$.

Hence in
$T(n/k) = 2T(n/2k) + (n/k)^2$.

we substitute the expression $T(n/4)$ in prev R.R to eliminate $T(n/4)$ and get the R.R in terms of $T(n/8)$

Hence continue this process by substituting $n/2^i$ for n in R.R to get the R.R for the size $n/2^i$.

$\therefore T(n/k) = 2T(n/2k) + (n/k)^2$

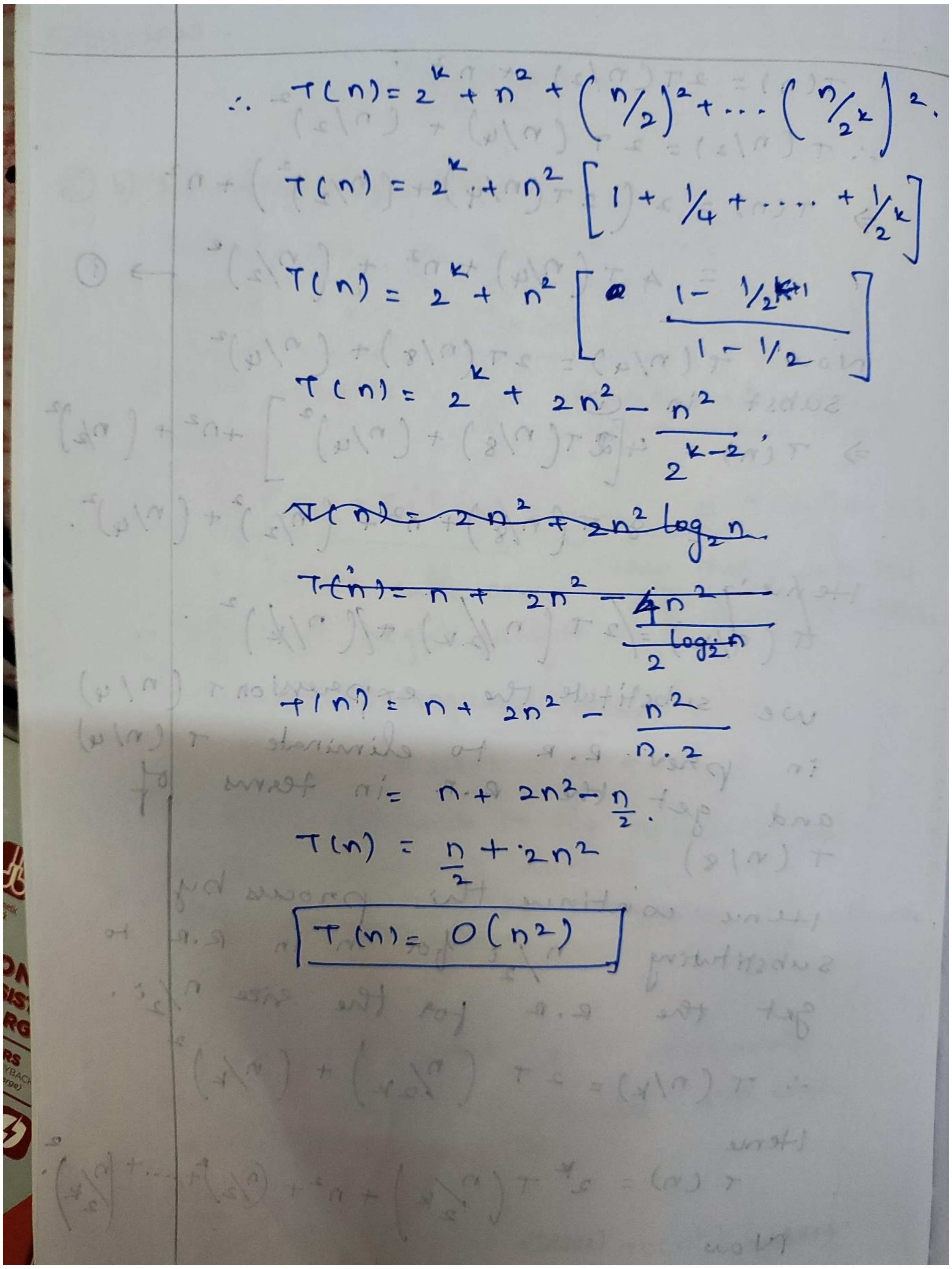
Hence
$T(n) = 2^k T\left(\frac{n}{2^k}\right) + n^2 + (n/2)^2 + \dots + \left(\frac{n}{2^k}\right)^2$.

Now
$T(1) = T(n/2^k)$

Hence $n/2^k = 1$

$n = 2^k$.

$\log_2 n = k$



$\therefore\ T(n) = 2^k + n^2 + \left(n/2\right)^2 + \dots \left(n/2^k\right)^2.$

$T(n) = 2^k + n^2 \left[1 + 1/4 + \dots + 1/2^k\right]$

$T(n) = 2^k + n^2 \left[\frac{1 - 1/2^{k+1}}{1 - 1/2}\right]$

$T(n) = 2^k + 2n^2 - \frac{n^2}{2^{k-2}},$

~~$T(n) = 2n^2 + 2n^2 \log_2 n$~~

~~$T(n) = n + 2n^2 - \frac{4n^2}{2^{\log_2 n}}$~~

$T(n) = n + 2n^2 - \frac{n^2}{n \cdot 2}$

$= n + 2n^2 - \frac{n}{2}.$

$T(n) = \frac{n}{2} + 2n^2$

$\boxed{T(n) = O(n^2)}$