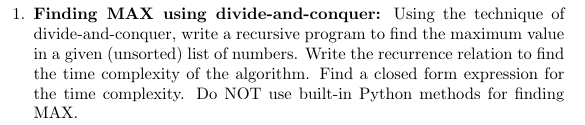
**ASSIGNMENT – 4**

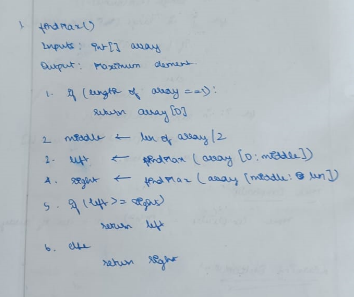
**AIM:**

To solve and implement the given problems using Divide and Conquer Strategies.

**Qn1:**

****

**Psuedo Code:**

****

**Source Code:**

def findMax(arr):

    if(len(arr) == 1):

        return arr[0]

    m = len(arr)//2

    l = findMax(arr[0:m])

    r = findMax(arr[m:len(arr)])

    if l>=r:

        return l

    else:

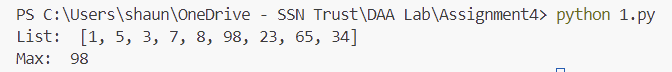
        return r

l = [1,5,3,7,8,98,23,65,34]

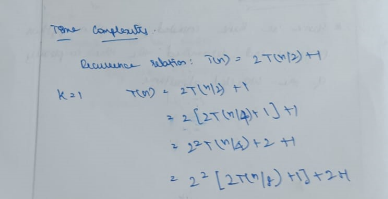
print("List: ", l)

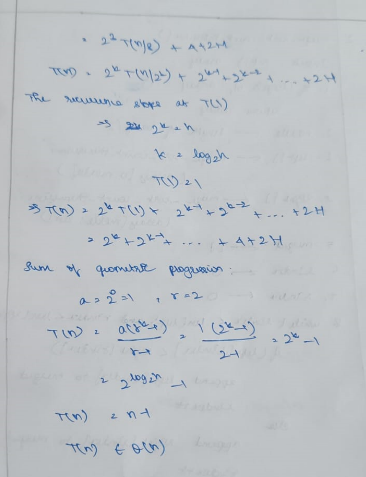
print("Max: ", findMax(l))

**Output:**

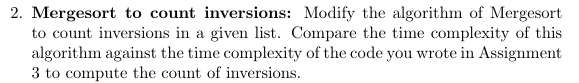
****

**Time Complexity:**

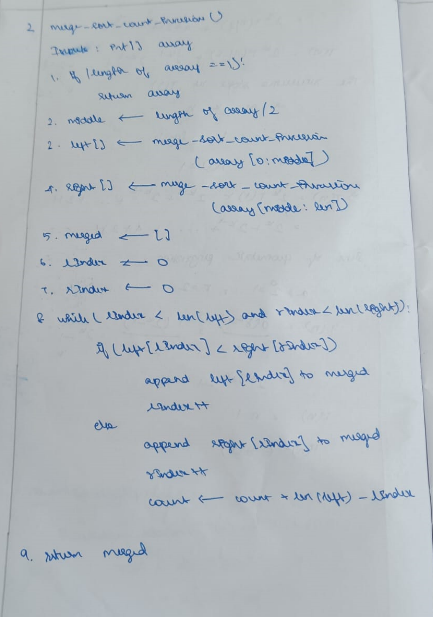
****

****

**Qn2:**

****

**Psuedo Code:**

****

**Source Code:**

def merge(l, r):

    i = 0

    j = 0

    inv\_count = 0

    merged = []

    while i<len(l) and j<len(r):

        if l[i] <= r[j]:

            merged.append(l[i])

            i += 1

        else:

            inv\_count += (len(l) - i)

            merged.append(r[j])

            j += 1

    while i<len(l):

        merged.append(l[i])

        i += 1

    while j<len(r):

        merged.append(r[j])

        j += 1

    return merged, inv\_count

def merge\_sort(arr):

    if len(arr) == 1:

        return arr, 0

    m = len(arr) // 2

    left, left\_inv = merge\_sort(arr[:m])

    right, right\_inv = merge\_sort(arr[m:])

    merged\_arr, merge\_inv = merge(left, right)

    total\_inv = left\_inv + right\_inv + merge\_inv

    return merged\_arr, total\_inv

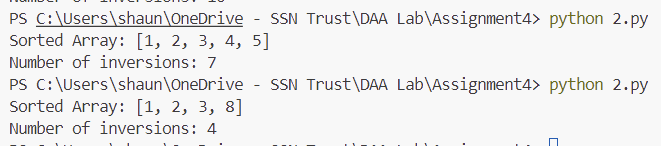
l = [3,2,8,1]

sorted\_l, inversions = merge\_sort(l)

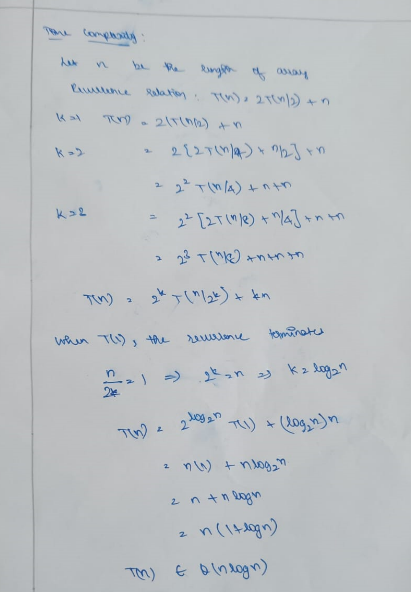
print("Sorted Array:", sorted\_l)

print("Number of inversions:", inversions)

**Output:**

****

**Time Complexity:**

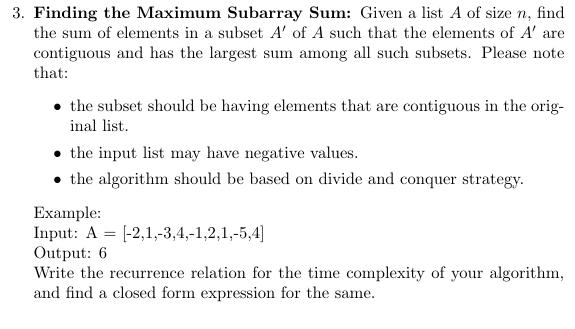
****

**Comparison to Assignment – 3:**

Time Complexity in Assignment – 3: O(n2)

Time Complexity now using merge sort: O(nlogn)

**Qn2:**

****

**Source Code:**

def maxCrossingSum(arr, low, mid, high):

    sm = 0

    left\_sum = float('-inf')

    for i in range(mid, low-1, -1):

        sm += arr[i]

        if sm > left\_sum:

            left\_sum = sm

    sm = 0

    right\_sum = 0

    for i in range(mid, high+1):

        sm += arr[i]

        if sm > right\_sum:

            right\_sum = sm

    return max(left\_sum + right\_sum - arr[mid], left\_sum, right\_sum)

def maxSubarraySum(arr, low, high):

    if low>high:

        return -1 #invalid case

    if low==high:

        return arr[low]

    mid = (low+high)//2

    return max(maxSubarraySum(arr, low, mid-1), maxSubarraySum(arr, mid+1, high), maxCrossingSum(arr, low, mid, high))

arr = [-2,1,-3,4,-1,2,1,-5,4]

n = len(arr)

max\_sum = maxSubarraySum(arr, 0, n-1)

print("Maximum contiguous sum is ", max\_sum)

**Output:**

****

**Learning Outcomes:**

* I learnt to analyse and implement divided and conquer algorithms
* I learnt how to implement various sorting and searching algorithms in Python