Date: 16.03.24 Reg No: 3122225001127
ExNo: 4 Name: Shaun Allan H

ASSIGNMENT – 4

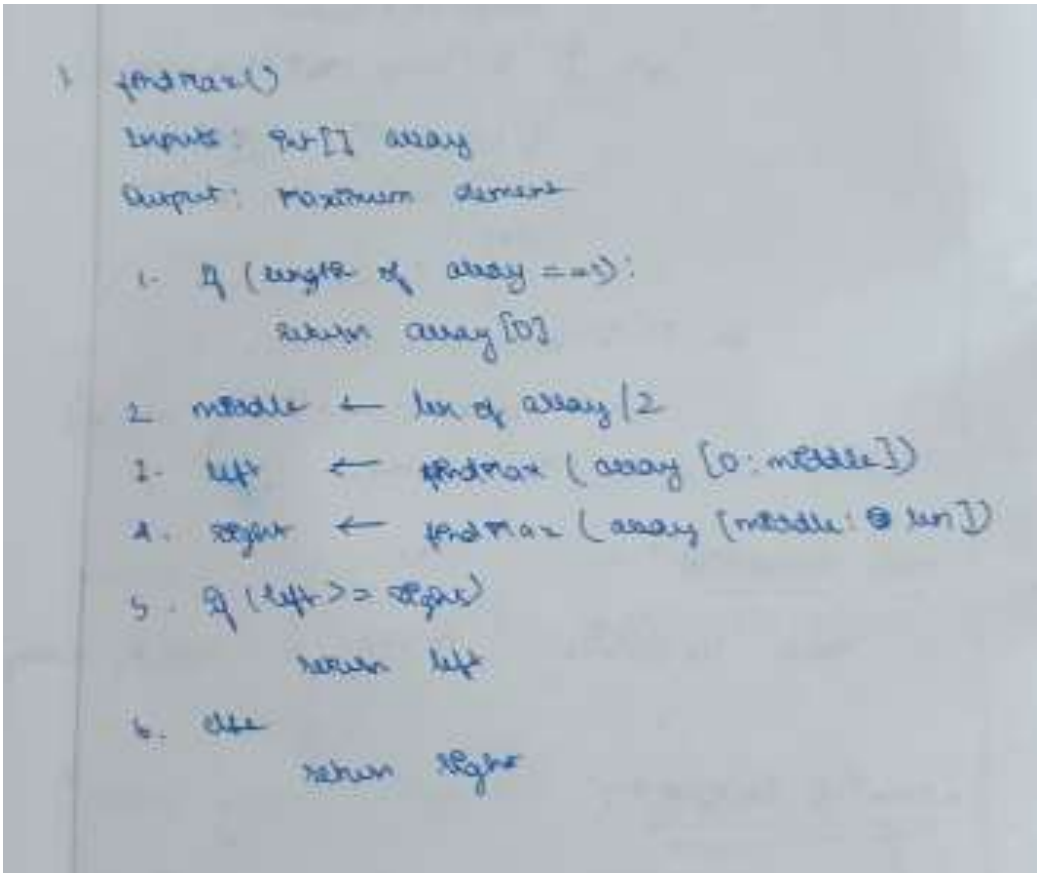
AIM:

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

Qn1:

1. **Finding MAX using divide-and-conquer:** Using the technique of divide-and-conquer, write a recursive program to find the maximum value in a given (unsorted) list of numbers. Write the recurrence relation to find the time complexity of the algorithm. Find a closed form expression for the time complexity. Do NOT use built-in Python methods for finding MAX.

Psuedo Code:



Date: 16.03.24
ExNo: 4

Reg No: 3122225001127
Name: Shaun Allan H

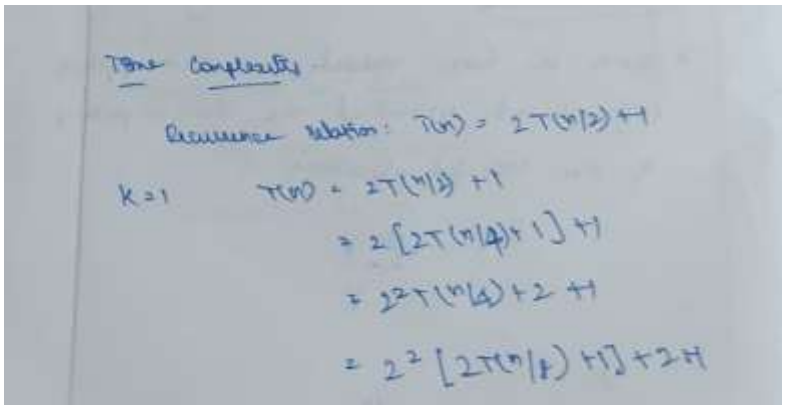
Source Code:

```
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:

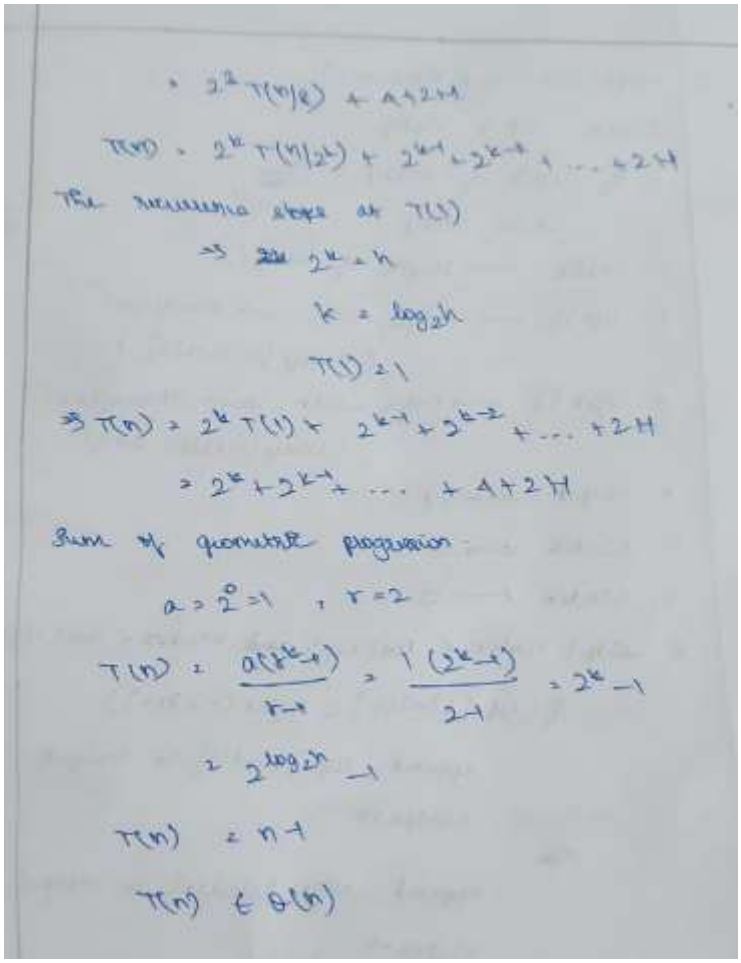
```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment4> python 1.py
List:  [1, 5, 3, 7, 8, 98, 23, 65, 34]
Max:  98
```

Time Complexity:



Date: 16.03.24
ExNo: 4

Reg No: 3122225001127
Name: Shaun Allan H

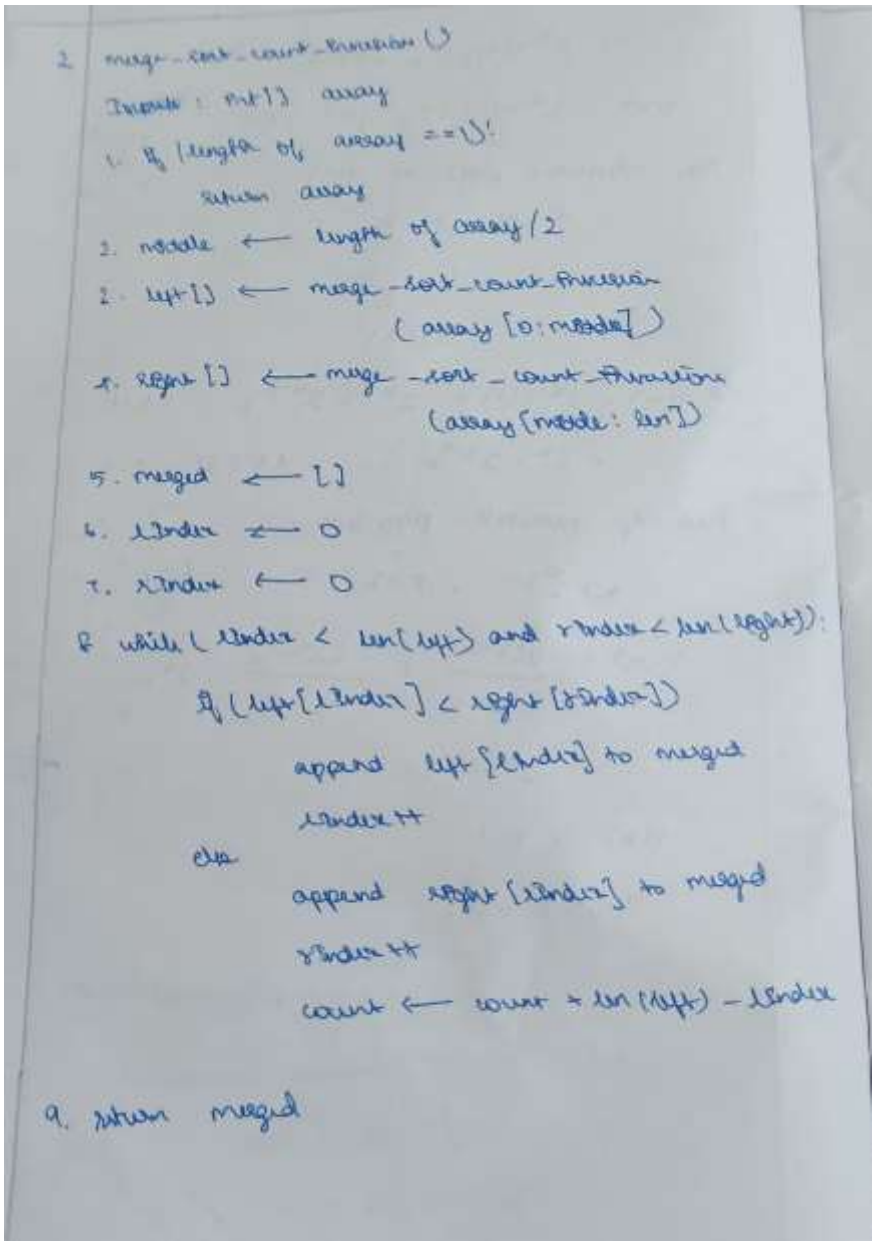


Date: 16.03.24 Reg No: 3122225001127
ExNo: 4 Name: Shaun Allan H

Qn2:

2. **Mergesort to count inversions:** Modify the algorithm of Mergesort to count inversions in a given list. 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.

Psuedo Code:



Date: 16.03.24 Reg No: 3122225001127
ExNo: 4 Name: Shaun Allan H

Source Code:

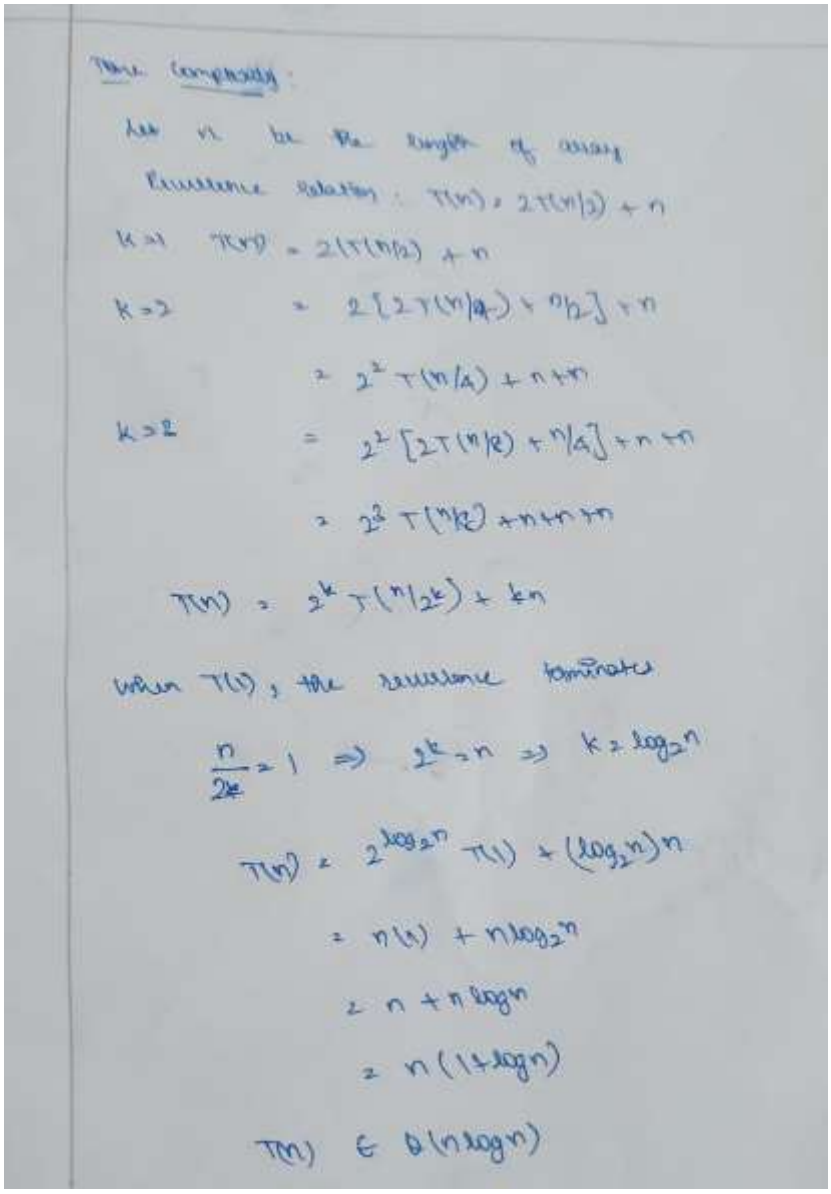
```
def merge(l, r):
    i = 0
    j = 0
    inv_count = 0
    merged = []
    while i<len(l) and j<len(r):
        if l[i] <= r[j]:
            merged.append(l[i])
            i += 1
        else:
            inv_count += (len(l) - i)
            merged.append(r[j])
            j += 1
    while i<len(l):
        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)
```

Date: 16.03.24 Reg No: 3122225001127
ExNo: 4 Name: Shaun Allan H

Output:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment4> python 2.py
Sorted Array: [1, 2, 3, 4, 5]
Number of inversions: 7
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment4> python 2.py
Sorted Array: [1, 2, 3, 8]
Number of inversions: 4
```

Time Complexity:



Comparison to Assignment – 3:

Time Complexity in Assignment – 3: $O(n^2)$

Time Complexity now using merge sort: O(nlogn)

Date: 16.03.24
ExNo: 4

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:

3. **Finding the Maximum Subarray Sum:** 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
   Write the recurrence relation for the time complexity of your algorithm, and find a closed form expression for the same.

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))
```

Date: 16.03.24 Reg No: 3122225001127
ExNo: 4 Name: Shaun Allan H

```
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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment4> python 3.py
Maximum contiguous sum is  6
```

Learning Outcomes:

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