

ASSIGNMENT – 4

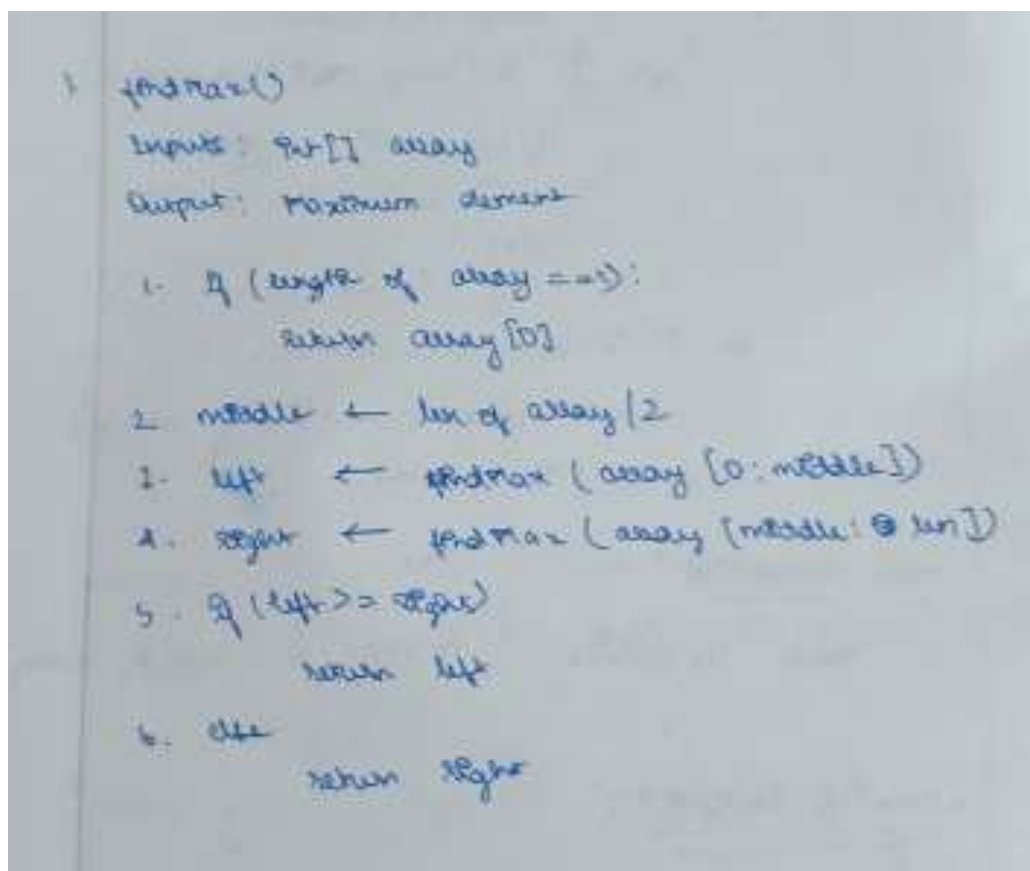
AIM:

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.

Pseudo Code:



```
1. findMax()
   Input: int[] array
   Output: Maximum element

   1. If (length of array == 1):
       return array[0]

   2. middle ← len of array / 2
   3. left ← findMax(array[0:middle])
   4. right ← findMax(array[middle: len])
   5. If (left > right)
       return left
   6. else
       return right
```

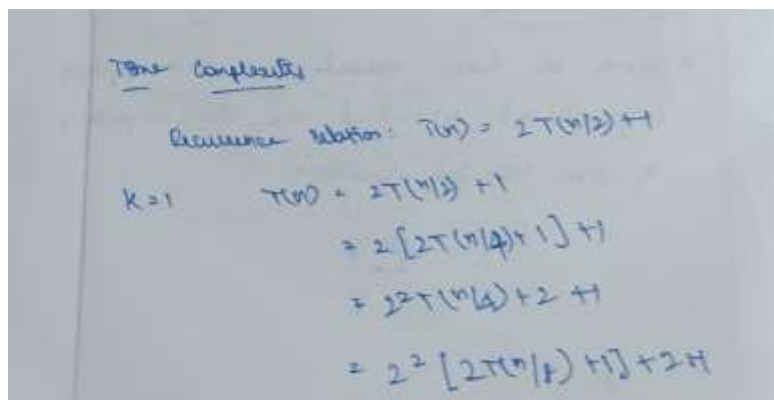
Source Code:

```
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:



Time Complexity

Recurrence relation: $T(n) = 2T(n/2) + 1$

$K=1$ $T(n) = 2T(n/2) + 1$

$$= 2[2T(n/4) + 1] + 1$$
$$= 2^2T(n/4) + 2 + 1$$
$$= 2^2[2T(n/8) + 1] + 2 + 1$$

$$\begin{aligned} &= 2^k T(n/2^k) + 4 + 2H \\ T(n) &= 2^k T(n/2^k) + 2^{k-1} + 2^{k-2} + \dots + 2H \\ \text{The recursion stops at } T(1) \\ \Rightarrow 2^k \cdot 2^k &= n \\ k &= \log_2 n \\ T(1) &= 1 \\ \Rightarrow T(n) &= 2^k T(1) + 2^{k-1} + 2^{k-2} + \dots + 2H \\ &= 2^k + 2^{k-1} + \dots + 4 + 2H \\ \text{Sum of geometric progression} \\ a &= 2^0 = 1, r = 2 \\ T(n) &= \frac{a(r^k - 1)}{r - 1} = \frac{1(2^k - 1)}{2 - 1} = 2^k - 1 \\ &= 2^{\log_2 n} - 1 \\ T(n) &= n - 1 \\ T(n) &\in \Theta(n) \end{aligned}$$

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.

Pseudo Code:

```
2 merge-sort-count-inversions ()
   Input: int[] array
   1. If (length of array == 1):
       return array
   2. middle ← length of array / 2
   3. left[] ← merge-sort-count-inversion
               (array[0:middle])
   4. right[] ← merge-sort-count-inversion
                (array[middle:len])
   5. merged ← []
   6. lIndex ← 0
   7. rIndex ← 0
   8. while (lIndex < len(left) and rIndex < len(right)):
       If (left[lIndex] < right[rIndex])
           append left[lIndex] to merged
           lIndex++
       else
           append right[rIndex] to merged
           rIndex++
       count ← count + len(left) - lIndex
   9. return merged
```

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:

```
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:

The complexity:

Let n be the length of array

Recurrence relation: $T(n) = 2T(n/2) + n$

$k=1$ $T(n/2) = 2T(n/4) + n$

$k=2$ $= 2[2T(n/4) + n/2] + n$

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

$k=3$ $= 2^2 [2T(n/8) + n/4] + n + n$

$= 2^3 T(n/8) + n + n + n$

$T(n) = 2^k T(n/2^k) + kn$

When $T(1)$, the recurrence terminates

$\frac{n}{2^k} = 1 \Rightarrow 2^k = n \Rightarrow k = \log_2 n$

$T(n) = 2^{\log_2 n} T(1) + (\log_2 n)n$

$= n(1) + n \log_2 n$

$= n + n \log n$

$= n(1 + \log n)$

$T(n) \in O(n \log n)$

Comparison to Assignment – 3:

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

Time Complexity now using merge sort: $O(n \log n)$

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
ExNo: 4

Reg No: 3122225001127
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