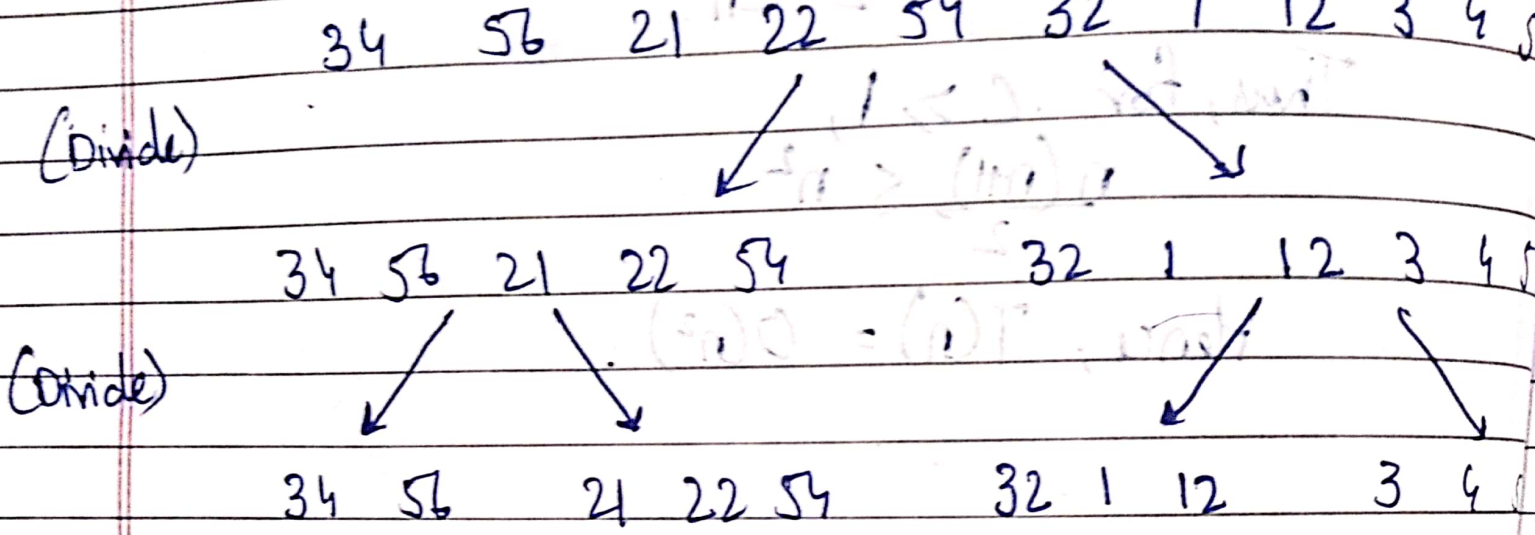
NAME:- KRUNAL RANK

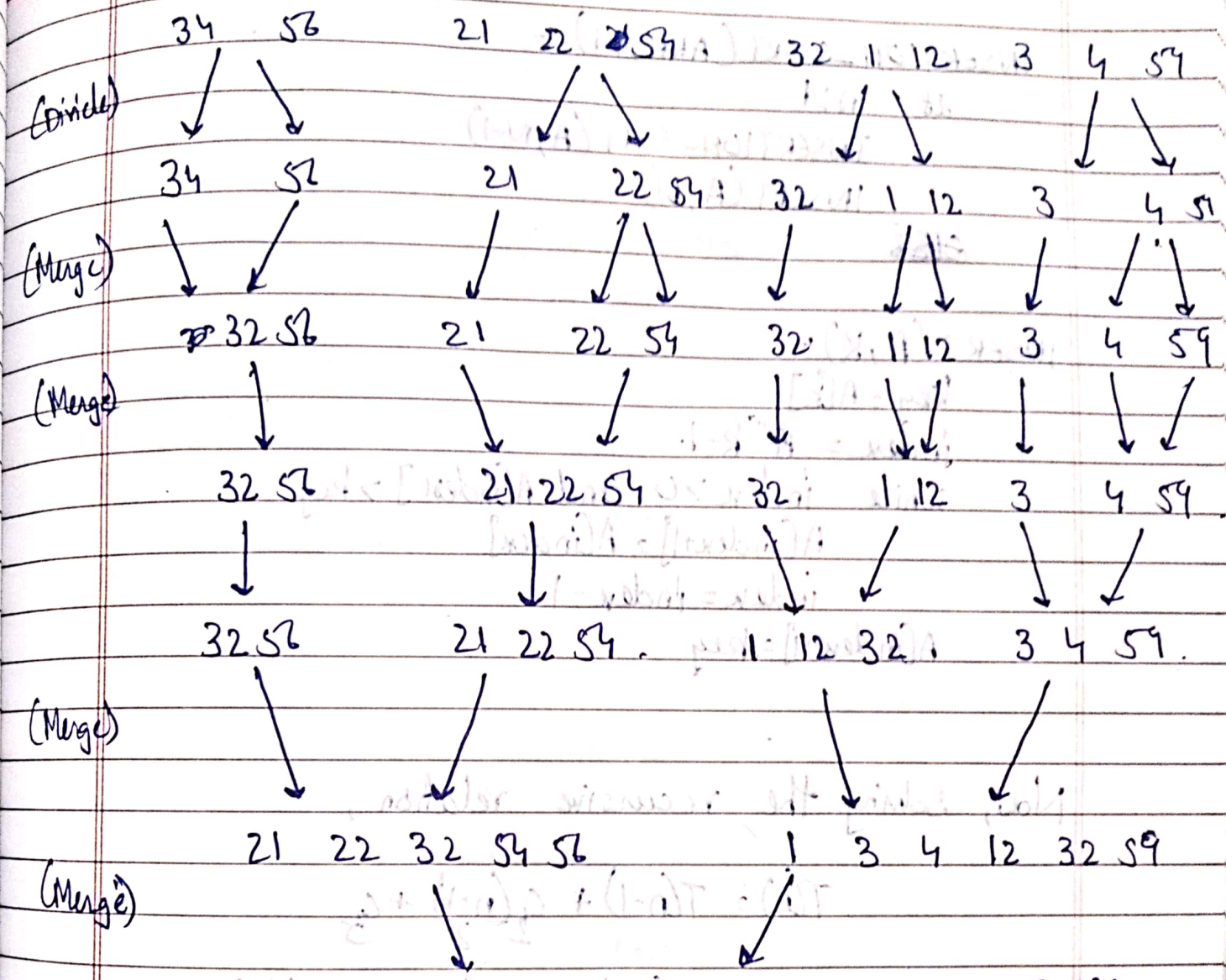
ADM. NO:- U18C0081

B TECH 3RD YEAR

10/9/20 U18CO081 DAA lutori

Ans 4. Given the array A = { 34, 56, 21, 22, 54, 32, 1, 12, 3, 4





Ans: For the given problem,

$$T(n) = \begin{cases} T(n-1) + O(n) & ; n>1 \\ O(1) & ; n=1 \end{cases}$$

Thus, we get,

$$T(n) = \begin{cases} c & ; n=1 \\ T(n-1) + c_2(n-1)/2 + c_3 & ; n>1 \end{cases}$$

This is because of the following recursive procedure:-

```
INSERTION_SORT(ARR, N) :-
    if n>1
        INSERTION-SORT(A,N-1)
        INSERT(A,N)

INSERT(A,K):-
    key = A[K]
    index = k-1.
    while index>0 and A[index] > key
        A[index+1] = A[index]
        index = index-1
    A[index+1] = key
```

Now, solving the recursive relation,

$$T(n) = T(n-1) + C_2\left(\frac{n-1}{2}\right) + C_3$$

$$= T(n-2) + C_2\left(\frac{n-2}{2}\right) + C_2\left(\frac{n-1}{2}\right) + 2C_3$$

$$= C_4 + \frac{C_2}{2}(1+2+3+\dots+n-1) + C_3$$

$$= \Theta(n^2)$$

```
MERGESORT(ARR, LOW, HIGH):
    MID ← (LOW + HIGH)/2
    MERGESORT(ARR, LOW, MID)
    MERGESORT(ARR, MID, HIGH)
    MERGE(ARR, LOW, HIGH, MID)

MERGE(ARR, LOW, HIGH, MID):
    K ← LOW   I ← 0   ARR1 ← {}
    L ← MID
    While K<MID and L<HIGH;
        If ARR[K] < ARR[L],
            ARR1[I++] = ARR[K++]
        Else
            ARR1[I++] = ARR[L++]
    While K<MID,
        ARR1[I++] = ARR[K++]
    While L<HIGH,
        ARR1[I++] = ARR[L++]
    J ← 0
    While J<I
        ARR[J++] = ARR1[J++]
```

Ans 3:

Binary search is a divide and conquer based search technique which requires sorted array.

```
BINARY_SEARCH (ARR, LOW, HIGH, KEY) :-
IF LOW = HIGH AND ARR[LOW] = KEY,
    RETURN LOW
ELSE IF LOW = HIGH
    RETURN -1
MID ← (LOW + HIGH) / 2
IF ARR[MID] >= KEY,
    RETURN BINARY_SEARCH(ARR, LOW, MID-1, KEY)
ELSE IF ARR[MID] < KEY
    RETURN BINARY_SEARCH(ARR, MIDH, HIGH, KEY)
ELSE
    RETURN MID
```

The recurrence for the above function is :-

$$T(N) = T(N/2) + O(1)$$

$$T(N) = O(1) + O(1) + T(N/4)$$

$$T(N) = \underbrace{O(1) + O(1) + \cdots + O(1)}_{\log_2 N \text{ times}}$$

Hence, $O(T(N)) = O(\log_2 N)$

Ans 4:

Example:- Consider the array :-
{8, 9, 13, 17, 21, 25, 28}
Dry run for finding 25 is as follows :-

Binary Search (0, 7, 25) has mid = 3 and arr[mid] = 17
17 < 25, Hence,
Binary Search (4, 7, 25) has mid = 5 and arr[mid] = 25
which is equal to 25. Hence, returns 5.

Ans 4: Given a sorted binary array we need to find number of 1's in it.
Let the sorted array have size N.

```
SEARCH(ARR, LOW, HIGH, N)
If LOW = HIGH, Then,
RETURN N-LOW
MID ← (LOW + HIGH)/2
If ARR[MID] = 1, Then,
    RETURN SEARCH(ARR, LOW, MID-1, N)
ELSE,
    RETURN SEARCH(ARR, MID+1, HIGH, N)
```

T(N) = T(N/2) + O(1) Hence, O(T(N)) = O(log N)

Example:- Given an array:-
{0, 0, 0, 0, 0, 0, 1, 1, 1, 1} N = 10
Call SEARCH(ARR, 0, 10) → MID = 5 → ARR[MID] = 0
Hence, Call SEARCH(ARR, 6, 10) → MID = 8 → ARR[MID] = 1
Hence, Call SEARCH(ARR, 6, 7) → MID = 6 → ARR[MID] = 1
Hence, Call SEARCH(ARR, 6, 6) → LOW = HIGH → Return N-LOW

Ans 5) MAXSUBARRAYSUM(ARR, N):
If N= and ARR[N] >0
Return ARR[N]
Elseif N=
Return 0.
SUM ← MAXSUBARRAYSUM(ARR, N-1)
If SUM + ARR[N] > ARR[N]
Return SUM + ARR[N]
Else
Return MAX(SUM, ARR[N])

$T(N) = T(N-1) + O(1)$
$= \underbrace{O(1) + O(1) + \dots + O(1)}_{N \text{ times}}$
Hence, $O(T(N)) = O(N)$

Here, Example = {-1, 3, 7, 2, -2, -9}
For N=1, It returns 0
For N=2, SUM=0, SUM+ARR[2] > ARR[2], Hence, Returns SUM+ARR[2] = 3
For N=3, SUM = 7, SUM+ARR[3] > ARR[3], Hence, Returns SUM+ARR[3] = 10
For N=4, SUM=10, SUM+ARR[4] > ARR[4], Hence, Returns SUM+ARR[4] = 12
For N=5, SUM=12, SUM+ARR[5] < A. Hence, Returns SUM=12
For N=6, SUM=12, SUM+ARR[6] < SUM, Hence, Returns SUM=12.

Ans 6: SECOND_MAXIMUM (ARR, N):

If N=2 Then,

Return { MAX (ARR[1], ARR[2]), MIN (ARR[1], ARR[2]) }

(FIRSTMAX, SECONDMAX) ← SECOND_MAXIMUM (ARR, N-1)

If ARR[N] > FIRSTMAX Then,

Return { ARR[N], FIRSTMAX }

If ARR[N] > SECONDMAX, Then,

Return { FIRSTMAX, ARR[N] }

Return { FIRSTMAX, SECONDMAX }

Here, $T(N) = T(N-1) + O(1)$

$= \underbrace{O(1) + O(1) + \dots + O(1)}_{N \text{ times}}$

Hence, $O(T(N)) = O(N)$

The method returns maximum and second maximum values.