Lecture #05

Date: Sept 2, 25 Day: Tuesday

Topic: Merge Sort

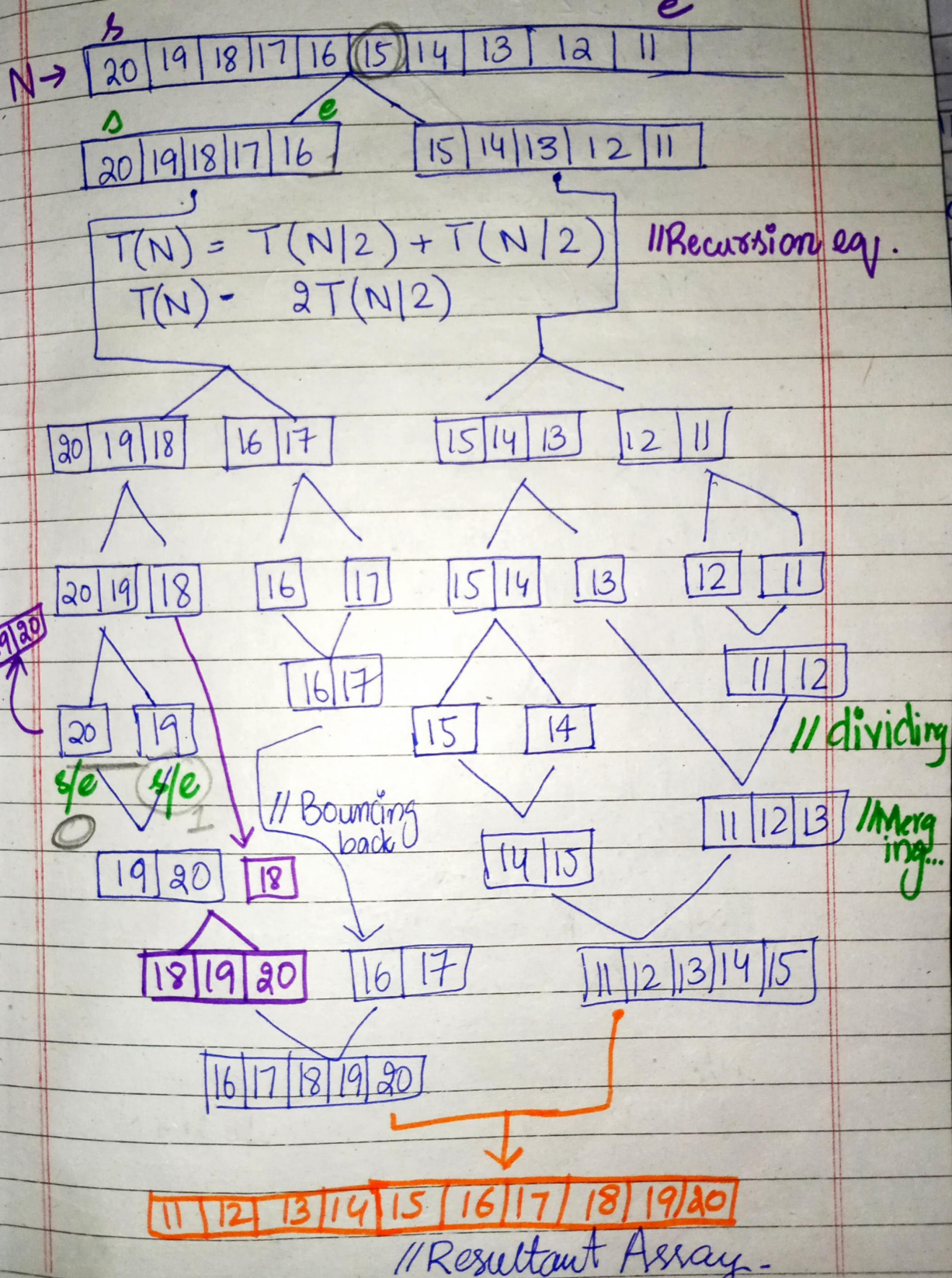
Merge:

```
Merge (Arr1, Arr2, N)

    i = 1, j = 1
  Res [1 .... 2N]
     k = 1
while( i <= N & j <= N ){
 if(Arr1[i] <= Arr2[j])
       Res[k++] = Arr1[i++]
 else
       Res[k++] = Arr2[j++]

        }

// Copying Pending elements:
while( i <= N)
     Res[k++] = Arr1[i++]

while(j <= N)
     Res[k++] = Arr2[j++]
```

Date: Day:

Merge Sort (Divide & Conquer)

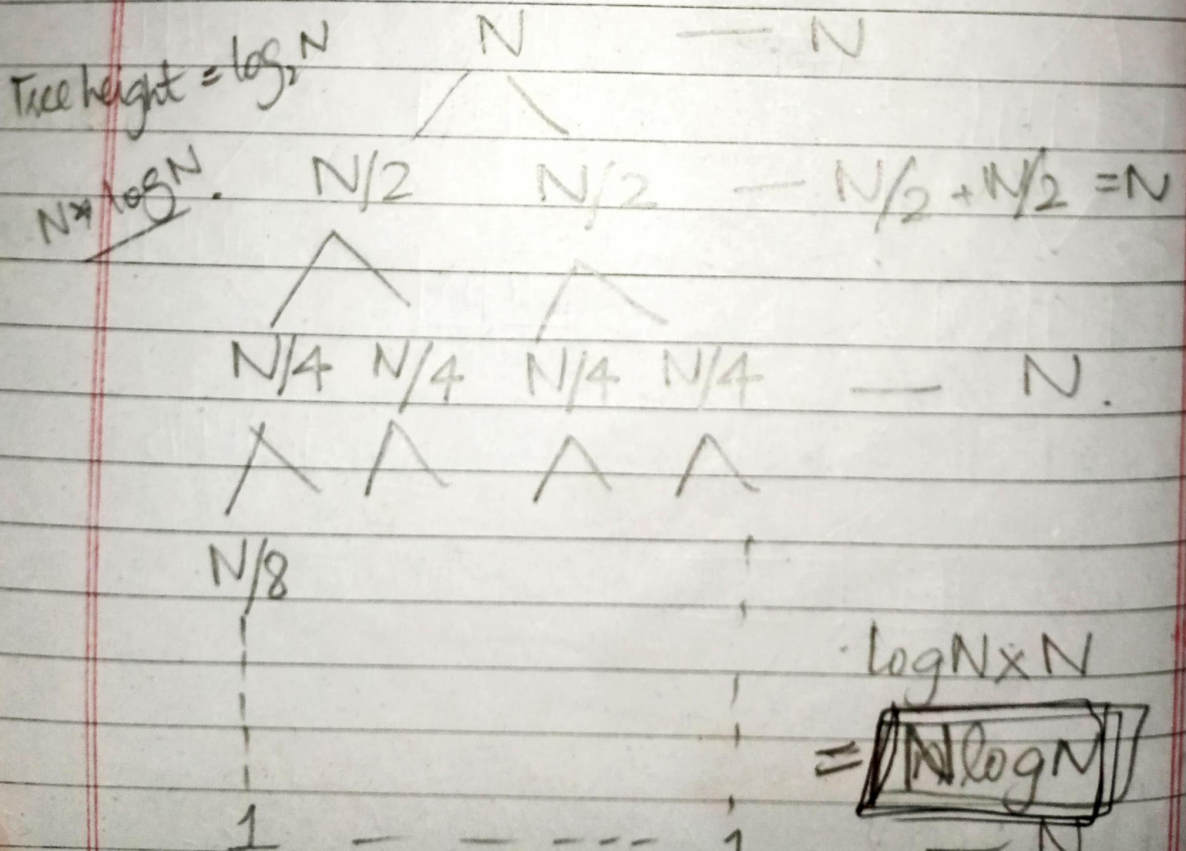
↳ Divides into 2 subproblems.



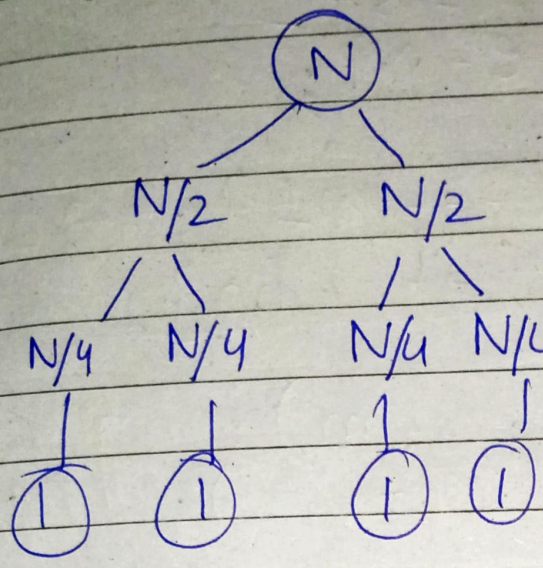
Date:

```
MergeSort(s, e) {
    if(s < e) {
        mid = (s+e)/2;
(N/2)   MergeSort(s, mid);      // Recursive
(N/2)   MergeSort(mid+1, e);    // calls.
(N)     Merge(s, mid, e);
    }
}
```

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



$$T(N) = T(N/2) + T(N/2) + N^2$$
$$= 2T(N/2) + N^2$$



N^2

$2 \times \left(\frac{N^2}{2^2}\right) = \frac{N^2}{2}$

$N^2/4$

$N * 1 = N$

$$N^2 + \frac{N^2}{2} + \frac{N^2}{4} + \cdots \frac{N^2}{2^k}$$

$$N^2\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^k}\right)$$

$r = 1/2$

$r < 1$

$$= N^2 \sum_{i=0}^{\infty} (1/2^i)$$

$$= N^2\left(\frac{1}{1-R}\right)$$

$T(N^2)$

// inplace Algo

Date: Day:

// Now 2 arrays on the same memory address & have to merge.

```
Merge(s, M, e){
    [n1 = M - s + 1;
     n2 = e - M;]   size calculate

    [temp1 [1 --- n1];
     temp2 [1 ---- n2];]   temp arrays

    [Copy(temp1, s, m);
     Copy(temp2, m+1, e);]   copy data

    i = 1, j = 1, k = s;   → starting range.
                           → while dividing into smaller arrays we have to sort them at the same Location; not making any resultant array;

    // Boundary Check:
    while(i <= N1 && j <= N2)
    {
        if(temp1[i] <= temp2[j])
    }

    //                //                //
    // inplace algo   in 'N' times      it inplace-
                                        // so we have to sort it inplace-
```

Lecture #06

Date: 4Sept2025 Day: Thursday

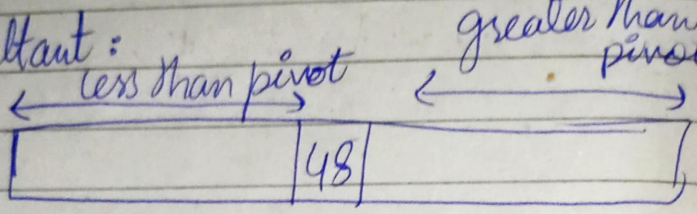
Topic: Quick Sort

Arr | 98 | 13 | 27 | 56 | 32 | 108 | 76 |

Pivot | 48 | (place it to its actual position)

size = N

Resultant:




```
func( Arr[1---N] , N, pivot) {

    Res [1 --- N+1]

    count = 0;
    for ( i=0 ; i < N ; i++)
        if(arr[i] < pivot)
            count++;

    Resultant[count] = pivot

    i = 0 ;  j = count+1;
    for (int k= 0; k < N; k++)
        if( arr[k] <=pivot ){

            Resultant[i++] = arr[k]
```

Date: Day:

```
else
    Resultant[j++] = arr[k]
}
```

Sir's Sol: ~~Merge~~ Task:

```
i = 1 ; j = N+1
for(k = 1 to N) {   or while (i<j)
  if (arr[k]-< = pivot)
      Res[i++] = Arr[K]
  else
      Res[j--] = Arr[k]
  }

Res[i] = pivot
```

Date: Day:

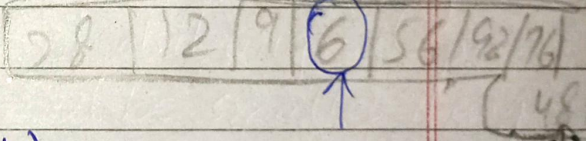
```
QuickSort (left, right) {
    if (left < right) {                          → T(N)
selectpivot //  P_index = partition (left, right)
T(N-1) //       QuickSort (left, P_index-1);
                QuickSort (P_index+1, right);
                → Time complexity of this call will be
                  constant (can be skipped)
    }
}
```

in worst case: $T(N) = T(N-1) + N$

Day: returns the exact position of pivot

Day:

```
Partition ( left , right) {

    pivot = Arr[right]
    i = left , j = right - 1

    while( i < j) {                    // skipping all the values which don't need swaping
        while (Arr[i] <= pivot)        && i<j
            i++ ;

        while (Arr[j] > pivot)         && i<j
            j++ ;                      // skipping values which don't need swapping

    if (i < j) {
        swap ( arr[i] , arr[j])
            i++ ;
            j-- ;
        }

    if (arr[i] > pivot){
        swap(Arr[i] , Arr[right])
        }

    return i ;

}
```




```
if (arr[i] < pivot){
    swap(arr[i], arr[right]
    }.
```

even if a sorted array: cost will be: N^2

Lecture #07

Date: Sept 9, 2025 Day: Tuesday

Time Complexity for Quick Sort:

inplace!!!

$$T(N) = T(N-1) + N$$

(not a stable algo)

(even for a sorted array)

If it is splitting in 9:1 ratio--

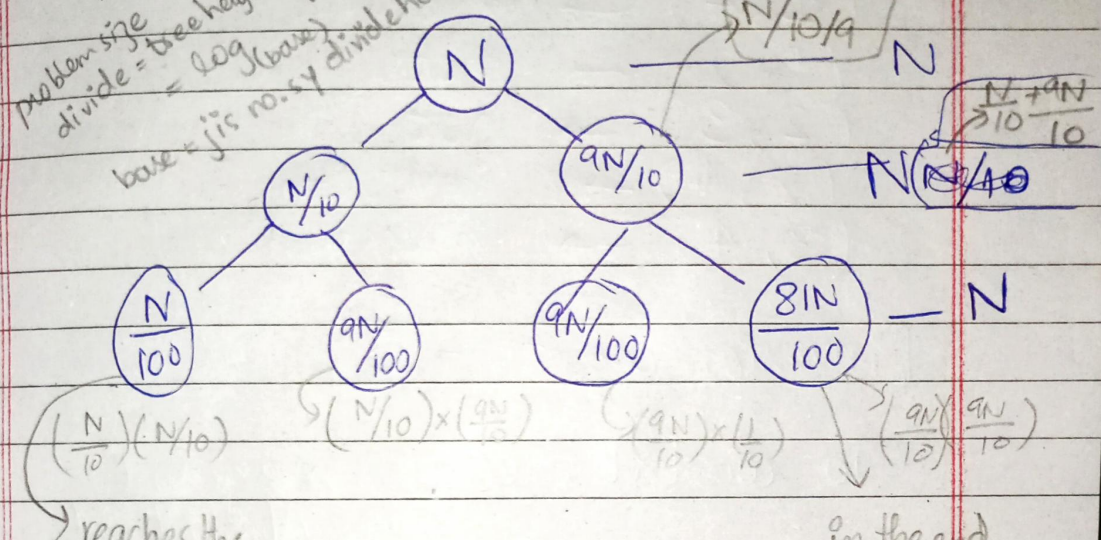
if $T(N) = T(N/10) + T(9N/10) + N$

problem size N

divide = tree height

= log (base)

base = is no. by divide horha.



reaches the base case first

in the end (Overestimation)

height of tree = $\log_{10/9} N$:

$= \log_{10/9} N * N$

$$T(N) = N \log_{10/9} N$$

$$T(N) = N \log N$$ (Average case)

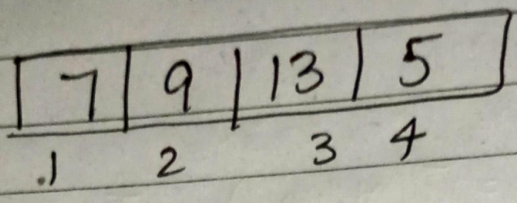
Quick Sort is better than merge sort as it is not consuming extra space (inplace)---

merge sort is stable, Quick sort is unstable.

Date:

"Inversion Count"

Pairs of indexes that are not in sorted position.



(1, 4) → inversion
(1, 2) → not
(2, 3) → not
(2, 4) → inversion
(3, 4) → inversion.

inversion count = 3

To count it; we use merge Sort Algorithm!

→ in this case; first we will count it for left of the array; then right of the array; then overall inversion count...

When we have counted it
→ the data must be sorted
→ So that we can avoid duplicate counts

Date: Day:

(by sorting the data)

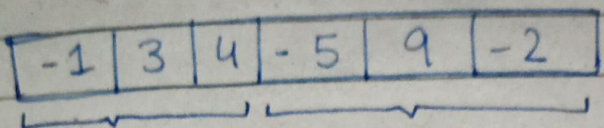
```
Count_inversion ( s , e ){

    inv_count = 0
    if ( s < e )
        M = (s+e) / 2

    inv_count += Count_inversion ( s , M )
    inv_count += Count_inversion ( M+1 , e
    inv_count += Merge (s , M , e)

    return inv_Count ;

}
```

merge sort, inversion count

→ stock investment

Date: Day:

Maximum SubArray Sum : by divide & conquer (N log N)




```
Max-Subarray (s, e) {
    if (s == e)
        return Arr[s];

    Mid = (s+e)/2
max-left = Max-Subarray (s, mid)        T(N/2)
max-right = Max-Subarray (mid+1, e)     T(N/2)

max_cross = crossing_Max(s, mid, e)   // cross sectional subarray ka sum.

    return max of 3
}  =  2T(N/2) + N

Crossing_Max (s, m, e) {        O(N)
    sum = 0;
    max_left = INT_MIN;
    max_right = INT_MIN;
```

ate:_____ Day:_____

```
for i = mid  down to B
        sum += Arr[i]
        if (sum > max_left)
                max_left = sum
sum = 0
for i = mid+1  to e
        sum += Arr[i]
        if (sum > max_right)
                max_right = sum

return  max_left + max_right ;

}
```