Agenda of today's live session

Finding of Minima and Maxima in an array

Input : An array of n elements

Output : Maxima and Minima in an array

50 20 15 9 7 30 90

1 2 3 4 5 6 7

Small Problem : If an array is having only single element and if it is having two elements then I consider that problem as small Problem

If i is the lower index and j is the upper index

Then, we can say that if

i==j Single Element in an array

i==j-1 Two Element in an array

Big Problem : If an array is having more than two elements, then I consider that array problem of finding minima and maxima as Big Problem and here basically we use Divide and Conquer to solve them.

Recursive Tree :

a,1,7,_90__,_7__

a,1,4,__50_,_9__ a,5,7,_90__,_7__

a,1,3,__50_,_15__ a,4,4,_9__,_9__ a,5,6,_30__,_7_ a,7,7,_90__,_90__

a,1,2,_50__,_20__ a,3,3,_15__,_15___

Implementation(Psuedocode) of Finding of Minima and Maxima in an array :

```
Maxima_Minima(a,i,j){
    Small Problems          O(1)
```

```
if(i == j){       // Single Element
        max = a[i];
        min = a[i];
}
else if(i == j-1){              //Two Element
        if(a[i] < a[j]){
                max = a[j];
                min = a[i];
        }
        else{
                max = a[i];
                min = a[j];
        }
}
Bigger Problem
else {
int m  = (i + j)/2;                     //Divide      O(1)
// Conquer
max1,min1 = Maxima_Minima(a,i,m)        Left        side        Recursion
T(n/2)
max2,min2 = Maxima_Minima(a,m+1,j)      Right side Recursion
T(n/2)
```

```
        // Combine
        O(1)
        if(max1 < max2){
                max = max2;
        }
        else{
                max = max1;
        }
        if(min1 < min2){
                min = min1;
        }
        else{
                min = min2;
        }
return (max,min)
}
```

Recurrence Relation of Finding of Maxima and Minima in an array :

$T(n) = T(n/2) + T(n/2) + c$

$= 2T(n/2) + c$

Using Master's Theorem, we calculate the overall time complexity :

$a = 2$

$b = 2$

n^(log_b a) = n^log_2 2 = n^1 = n

f(n) = c

O(n) is the overall time complexity of the above algorithm.